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
xijie_wang@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". To address these concerns, machine unlearning has been proposed to efficiently eliminate the influence of specific training data from trained DNNs. However, existing machine unlearning methods struggle with the large number of parameters in trained DNNs, which lead to slow execution and high memory consumption, making them impractical for large-scale models. In this paper, we shift our focus to the small set of weights in the final classification layer of DNNs, which are defined as as "prototypes" for different classes. Our key observation is that the prototype associated with the unlearned training data undergoes a significant shift, whereas prototypes of unrelated classes exhibit only minor changes when comparing the prototypes of original and retrained models. Based on this observation, we propose a novel machine unlearning approach that efficiently achieves machine unlearning by directly adjusting the prototypes of DNNs. We first introduce Naive Prototype Surgery (Naive PS), a fast and simplified method that uses a closed-form solution to approximate unlearning effect by directly adjusting the prototype associated with the unlearned data. Next, we propose Prototype Surgery (PS), which incorporates soft label information to fine-tune the prototypes of all classes, to achieve a more effective unlearning. Both methods achieve data unlearning by only modifying the prototypes in the DNNs, thus avoiding the challenges posed by the large number of model parameters. 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 Network, 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 the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3719027.3744827

1 Introduction

Driven by large-scale training datasets and sophisticated architectures [19], the rapid advancements and widespread application of machine learning models, particularly Deep Neural Networks (DNNs), have achieved near-human performance on various tasks. However, the reliance on collecting large-scale training data, which usually including sensitive and private information, has raised significant privacy concerns of the data providers and owners [11, 40]. Existing works have shown that the DNNs 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 data owners the "right to be forgotten" and the ability to request the removal of their data from trained DNNs. Retraining models from scratch without the data that needs to be unlearned is a theoretically effective

*Corresponding author.

ACM CCS'25, Taipei, Taiwan

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3744827

approach, but it is computationally prohibitive. This has catalyzed growing interest in machine unlearning techniques, which aim to efficiently remove the influence of specific data from trained models [33, 44].

Recently, lots of machine unlearning methods have been proposed. In this paper, we focus on sample-level unlearning, which aims to remove the influence of specific data samples from trained models. Specifically, existing unlearning methods for DNNs mainly rely on modifying model parameters. A part of methods rely on matrix-based computations, such as the Fisher Information Matrix, to identify the parameters most relevant to the data to be unlearned [13–15], and then modify these parameters accordingly to achieve unlearning. Beyond matrix-based approaches, fine-tuning methods aim to adjust model parameters by either optimizing all parameters [5] or selectively updating specific ones to improve unlearning efficiency [4]. Although these approaches can address requirements of machine unlearning, they face two critical limitations: (1) prolonged unlearning times due to high computational complexity, and (2) substantial memory overhead during parameter updates. These constraints significantly hinder their practical deployment on large-scale models and datasets, as summarized in Table 1.

To address the limitations of computational and memory overhead, we focus on the classification layer of DNNs, which typically maps the features extracted by the preceding layers to class-specific predictions. Importantly, the objective of machine unlearning is to ensure that the model's outputs no longer reflect the influence of specific data. Since the final classification layer directly governs these outputs, we naturally take it into consideration as a target for unlearning. Based on this insight, we concentrate on the weights of the classification layer, which we define as "prototypes" representing each class. During training, samples from a given class collectively shape the corresponding prototype, allowing it to serve as an abstract representation of the whole training data of a class. Ideally, if a portion of the training data is removed, the prototype should exhibit a shift compared to the one trained on the full dataset—capturing the absence of the unlearned data's influence.

Focusing on the prototypes in DNNs, we explore the possibility of achieving machine unlearning by directly adjusting these prototypes. Correspondingly, we first observe the prototypes in DNNs by comparing the prototypes of multiple original models and retrained models after removing a part of training data from one certain class. For clarity, we refer the class corresponding the unlearned data as the target class. As visualized in Figure 1, two key insights emerge from this analysis. Firstly, the prototypes for the target class undergo significant shifts in the retrained models compared to their counterparts in the original models. 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. These observations suggest that the influence of unlearned data is primarily reflected in the prototype of the target class, while the impact on non-target classes is limited.

In light of the above observations, we propose directly modifying the prototypes (i.e., the weights of the final classification layer) to effectively eliminate the influence of specific training data from

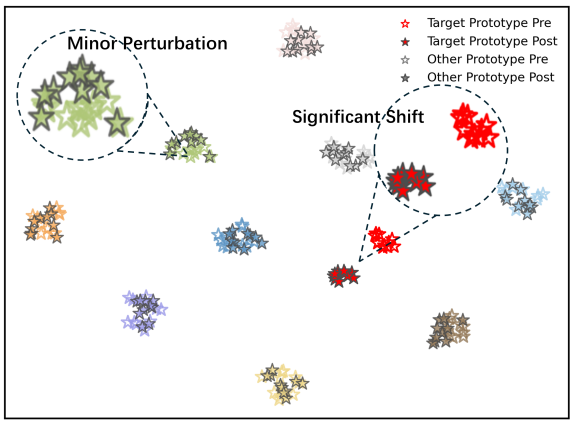


Figure 1: The t-SNE visualization of prototype shifts reveals two key observations. Prototypes before unlearning (pre) and after unlearning (post) are color-coded by class, with the target class highlighted in vibrant red. First, the prototype of the target class undergoes a significant shift from pre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

a trained DNN. This approach avoids the need for costly updates across the entire network, offering a lightweight and scalable alternative. In this paper, we propose two novel prototype-based machine unlearning methods: Naive Prototype Surgery (Naive PS) and Prototype Surgery (PS). Naive PS utilizes a closed-form solution to efficiently approximate the prototype shift for the target class, enabling selective removal of the unlearned data's influence with minimal computational cost. In contrast, PS introduces a more fine-grained adjustment strategy by leveraging soft-labeled unlearned data. This allows for precise and robust updates to both target and non-target class prototypes, further enhancing unlearning effectiveness while preserving model performance.

Despite the apparent simplicity of directly modifying prototypes, our unlearning approaches confront two key challenges. The first challenge stems from the inherent complexity of operating in high-dimensional prototype spaces. Unlike low-dimensional visualizations where updates are intuitive, prototype adjustment in high-dimensional spaces is complicated by the large number of dimensions and their interdependencies. To address this, we conduct a detailed analysis of the prototype update process and formulate it mathematically. This modeling enables us to approximate the required changes for unlearning using a closed-form solution, as implemented in Naive PS, thereby allowing efficient and direct prototype updates that effectively remove the influence of unlearned data.

The second challenge arises from the observation that, although unlearning predominantly affects the prototype of the target class, it also induces subtle yet non-negligible shifts in non-target class prototypes. If left uncorrected, these shifts may lead to inconsistencies between the unlearned model and a model retrained from scratch. To mitigate this, PS leverages the rich semantic information contained in soft-labeled unlearned data to guide the adjustments of both target and non-target prototypes. This strategy ensures

Table 1: Comparison of Machine Unlearning Methods

Comparison Method	Ratio of Modified Model Parameters	Remaining Training Data	Extra Data	Additional 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: ×

that the updated model remains well-aligned with the behavior of a fully retrained model, achieving more precise and robust machine unlearning.

We summarize our contributions as follows:

- We introduce a novel prototype-centric perspective for machine unlearning in DNNs, shifting the focus from traditional parameter-centric approaches, which aim to identify and modify large-scale model parameters, to the modification of the final classification layer. By directly adjusting the prototypes (i.e., the weights of the last classification layer) associated with the unlearned data's class, we offer a more efficient alternative with potential advantages in terms of computational cost and scalability. To the best of our knowledge, this is the first work to explore machine unlearning through a prototype-centric lens.
- We propose Naive PS, a method that efficiently approximates the removal of unlearned data's influence by directly adjusting the prototype of the corresponding class using a closed-form solution. By directly adjusting the prototypes, Naive PS enables fast unlearning with minimal computational complexity.
- We also introduce PS, a method that refines the prototypes of all data classes through fine-tuning the last classification layer on the soft-labeled unlearned data. This approach ensures precise adjustments to all prototypes while maintaining computational efficiency. In addition to updating the prototype of the unlearned data's class, PS carefully accounts for the impact on other prototypes, aligning the unlearned model closely with a retrained model.
- 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://github.com/SPHelixLab/PS_Unlearn.

2 Related Works

In this section, we first provide an overview of machine unlearning techniques for traditional machine learning models, followed by a focus on unlearning methods specifically designed for DNNs.

2.1 Unlearning for Machine Learning Models

Machine unlearning techniques have already 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 models, updating only the relevant clusters to avoid full retraining. Similarly, Nguyen et al. [31] use variational inference to enable efficient unlearning in Bayesian models by minimizing the Kullback-Leibler divergence between the approximate posterior belief of model parameters (after unlearning from erased data) and the exact posterior belief (from retraining with the remaining data).

For other machine learning models, Brophy et al. [2] introduce DaRE, which randomly updates subtrees in random forest models to achieve machine unlearning. Other methods, such as the Markov chain Monte Carlo-based algorithm by Nguyen et al.[32], approximate retrained models to reduce unlearning overhead.

Brophy et al. [2] introduce DaRE, which randomly updates subtrees of random forest models to achieve machine unlearning. 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 techniques are tailored to specific models and cannot be directly applied to DNNs.

2.2 Unlearning for DNNs

In DNNs, the naive approach of retraining models from scratch to exclude unlearned data is exact but computationally prohibitive, particularly for large-scale models. As a result, many machine unlearning methods for DNNs have been proposed. Existing work in DNNs can be categorized into two main approaches: exact unlearning and approximate unlearning.

2.2.1 Exact Machine Unlearning. Exact machine unlearning methods aim to accelerate retraining by modifying the training pipeline of DNNs. Bourtoule et al. [1] propose the SISA framework, which partitions the training data into chunks and trains submodels independently, enabling retraining of only a subset of submodels and accelerating the overall retraining process. 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] introduce GraphEraser, a shard-based partitioning approach to efficiently unlearn specific data.

In contrast to the 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. While these methods can accelerate the retraining process, they either fundamentally alter the original training pipeline or require information from the training process, which may compromise model performance and cannot be applied to already trained models.

2.2.2 Approximate Machine Unlearning. Approximate machine unlearning methods aim to remove the influence of unlearned data without requiring full retraining, instead directly modifying the model parameters. Golatkar et al. [13] use the Fisher Information Matrix (FIM) to identify and neutralize parameters that are heavily influenced by the unlearned data. Similarly, Guo et al. [15] apply influence functions to approximate the impact of specific data samples and adjust the corresponding model parameters. To further reduce computational complexity, Suriyakumar et al. [38] propose an online unlearning algorithm that avoids recalculating the Hessian matrix. Mehta et al.[29] identify a Markov Blanket of parameters for efficient unlearning, limiting updates to a small subset of parameters. In federated learning settings, FedEraser [24] and SIFU [10] utilize stored historical updates to efficiently unlearn the training data of a federated client.

In addition, a part of works leverage fine-tuning techniques to refine specific aspects of the model in order to achieve machine unlearning. For instance, Boundary Unlearning [5] fine-tunes DNNs to remove the influence of unlearned data from the perspective pf decision boundaries rather than the model parameters. Similarly, Cha et al. [4] employ adversarial examples and importance weights to adjust the model's decision boundaries and selectively erase specific instance information. Chundawat et al.[7] propose a teacher-student framework for selective knowledge transfer as a method for achieving machine unlearning.

Despite the variety of methods introduced above, they all share a common underlying approach: removing the influence of unlearned data by modifying model parameters. While these parameter-centric approaches have driven significant progress in machine unlearning, they also come with two remarkable limitations. First, parameter-centric methods often entail significant computational costs. Even when only a subset of middle parameters is modified, the inherent update mechanism of DNNs still requires computing gradient information for most of the model's parameters, leading to substantial memory and computational overhead. This undermines the goal of reducing resource consumption, particularly for large-scale models and datasets. Second, these methods typically rely on access to the remaining datasets or stored intermediate states, which can be impractical in real-world scenarios due to privacy concerns or data unavailability. These limitations motivate us to propose a more parameter-efficient unlearning method. In this paper, we focus on a prototype-centric perspective, where only the weights of the last classification layer are adjusted. Our approach significantly reduces resource consumption while still achieving the objectives of machine unlearning.

3 Preliminaries and Notation

In this section, we establish the notation and foundational concepts related to 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 samples of training data. Then, we define the unlearning set $\mathcal{D}_u \subseteq \mathcal{D}$ as a subset of the original training data, 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 our paper, we focus on sample-level machine unlearning, where the unlearning set $\mathcal{D}_u$ is constructed by training samples from one specific class.

Next, given the critical role of prototypes in our approach, we formally define the prototypes within DNN classification models. Generally, a DNN classification model $\mathcal{M}$ can be divided into two components: a feature extractor $\mathcal{F}$ and a final classification layer. $\mathcal{F}$ 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 number of feature embedding dimensionality.

With the embedding vector for an input sample, the final classification layer then computes the logits as follows:

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

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. The dot product $\mathbf{e}\mathbf{W}^T$ essentially captures the similarly between the embedding vector $\mathbf{e}$ and each row $\mathbf{w}_c \in \mathbb{R}^{1\times E}$ of $\mathbf{W}$, which corresponds to the prototype for the c-th class. In other words, $\mathbf{w}_c$ represents the class prototype for class c, and thus we regard $\mathbf{w}_c$ as the prototype for the c class. Besides, we define the weight matrix $\mathbf{W} \in \mathbb{R}^{C\times E}$ as the prototype matrix,

After that, 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:

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

Through the softmax operation, the logits are transformed into normalized probabilities. Generally speaking, for a trained model, the prediction of the classification model is essentially based on calculating the similarity between the sample's embedding and each column of the classification layer's weights. Then the model's prediction is determined by comparing the similarity scores.

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 the prototype matrix $\mathbf{W}^{\text{ori}}$ 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 model parameters unchanged. This ensures that the unlearned model's outputs closely match those of the retrained model, achieving the unlearning objective with a fewer computational cost.

4 Naive Prototype Surgery

In this section, we begin with the simple one-layer linear classification model and gradually extend our prototype concept to more complex DNN models.

4.1 Prototypes in One-Layer Classification Models

The fundamental idea of our machine unlearning approach is derived from the simple one-layer linear classification model. To build a foundation for understanding more complex architectures, we begin with this simple yet powerful model. Within the one-layer linear classification model, the input $\mathbf{x}$ is mapped to C classes using a set of prototype vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_C$ as follows:

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

where $\mathbf{z} = [z_1, z_2, \ldots, z_C]$ represents the logits for each class. Intuitively, each logit z_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 with Eq. (3).

The training process of this model ensures that each prototype $\mathbf{w}_c$ evolves into a representative embedding for its corresponding class through interactions with the training samples. During training, the model adjusts the weight matrix $\mathbf{W}$ to maximize the dot product between a sample's features and the weight corresponding to its class. This process enables each class's weight vector $\mathbf{w}_c$ (i.e., prototype) to aggregate the typical features of that class's data. In other words, the weight vector $\mathbf{w}_c$ for each class can be regarded as the weighted average of the training samples, making it the prototype for that class's data. Understanding how this process operates is crucial, especially when data distributions shift, as it ensures that prototypes adapt appropriately and continue to accurately represent their respective classes.

4.2 Prototypes in DNNs

Now, we extend our prototype concept to more complex DNN models. Specifically, we focus exclusively on the final classification layer. By doing so, we can treat the parameter matrix of the classification layer $\mathbf{W}$[1] as the set of prototypes. Note that we modify only the classification layer while keeping the feature extractor fixed. Thus the embedding vector $\hat{\mathbf{x}} = \mathcal{F}(\mathbf{x})$ for the input data $\mathbf{x}$ is fixed. This allows us to approximate the final classification layer as a one-layer linear classification model. To quantify the prediction error when predicting the probabilities $\mathbf{y}$ from an input $\mathbf{x}$, a commonly used metric is the cross-entropy (CE) loss function $\mathcal{L}_{\text{CE}}$, expressed as:

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

[1]For clarity and consistency with the previous one-layer model, we also denote the parameter matrix of the classification layer as $\mathbf{W}$.

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

Due to the non-convex nature of DNNs, iterative optimization methods are typically employed to minimize the loss function $\mathcal{L}_{\text{CE}}$. One of the most widely used optimization strategies is stochastic gradient descent (SGD), which updates the model parameters iteratively. the update rule for optimizing the classification layer's parameter matrix $\mathbf{W}$ using SGD is given by:

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

where $\mathbf{W}_t$ represents the prototypes at iteration t, $\mathcal{D}$ is the training data, 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:

$$\left.\frac{\partial L}{\partial \mathbf{W}}\right|_{\mathbf{W}_t, \mathcal{D}} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{p} - \mathbf{y})\hat{\mathbf{x}}_i, \tag{7}$$

where $\mathbf{p}$ is the predicted probability vector obtained through the softmax function, and $\mathbf{y}$ is the one-hot encoded ground truth label. $\hat{\mathbf{x}}_i$ denotes the embedding vector obtained from the feature extractor of the DNN model for the input sample $\mathbf{x}_i$, where $i \in [1, 2, \ldots, N]$. Then integrating this gradient into the SGD update rule yields the following iterative formula for updating $\mathbf{W}$:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta \frac{1}{N}\sum_{i=1}^{N}(\mathbf{p} - \mathbf{y})\hat{\mathbf{x}}_i. \tag{8}$$

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

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

Here, $\mathbf{W}_0$ represents the initial parameters, $\hat{\mathbf{X}}$ denotes the embeddings of the trianing data $\mathcal{D}$, and Ω is the cumulative error term, which represents the weight adjustments accumulated over all iterations. Specifically, Ω accumulates the gradient contributions from the predicted errors over all past steps.

While DNNs are significantly more complex than one-layer classification models, the core principle of prototypes remains consistent across both. In both models, prototypes are adjusted to better align with the training samples from the same class and to distance themselves from training samples from other classes. This key mechanism persists despite the deeper and more intricate structure of DNNs, emphasizing the foundational role of prototypes in linking data with predictions.

From a broader perspective, Ω accumulates the influence of all training samples during optimization, illustrating how each training sample's embedding contributes to the corresponding prototype. Notably, the updates to $\mathbf{W}$ at each training step are driven solely by the training data and its prediction errors. Therefore, the trained prototypes $\mathbf{W}$ can be viewed as a weighted average of the training data, with the weights corresponding to each sample's prediction By connecting prototypes with the training data in this way, we enable a mechanism for effective machine unlearning, where adjustments to the prototypes can remove or modify the influence of specific data points in a computationally efficient manner.

4.3 Implementation of Naive PS

Through our observations, we identify the phenomenon of prototype shift that occurs before and after unlearning specific data. This shift arises directly from the removal of unlearned data, which disrupts the cumulative contributions to the prototype. Since prototypes can be regarded as weighted aggregations of the training data, this shift can be intuitively corrected by isolating and explicitly removing the contribution of the unlearned data from the corresponding prototype.

To implement the 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 $\hat{\mathbf{X}}_r$, which represent the feature embeddings of the remaining data. Next, we need to update the cumulative error term Ω to reflect the changes in $\hat{\mathbf{X}}_r$. However, Ω is not directly observable. To compute it, we leverage the relationship outlined in Eq. (9), utilizing the parameters from the original model and the training initialization. Once Ω is determined, we remove the columns corresponding to the unlearned samples, resulting in the adjusted term Ω_r. Finally, the unlearned prototypes can be computed using the following equation:

$$\mathbf{W}^{\text{un}} = \mathbf{W}_0 - \Omega_r \hat{\mathbf{X}}_r, \tag{10}$$

where $\mathbf{W}^{\text{un}}$ denotes the parameter of the unlearned classification layer and $\mathbf{W}_0$ represents the parameter initialization of the classification layer.

We refer to this unlearning method as Naive PS. It effectively captures the observed changes and allows the model to efficiently transition to its unlearned state. By aligning the prototypes with their expected behavior after unlearning, this method provides a computationally efficient solution, ensuring minimal disruption to the representation of the remaining data.

5 Prototype Surgery

Although the Naive PS method can effectively achieve machine unlearning with low computational cost, it has two key limitations that constrain its practical applicability. First, it relies on pre-saved feature embeddings of the training data. In practical scenarios, however, these embeddings are often not deliberately stored during training. Recomputing them by loading the entire training dataset is neither efficient nor feasible, especially given today's large-scale datasets, making this approach difficult to scale in practice. Second, Naive PS overlooks the broader impact of unlearning on classes unrelated to the unlearned data. While it adjusts the class prototype corresponding to the unlearned data, it does not account for the influence that unlearning certain samples may have on other class prototypes. However, our previous analysis shows that removing data from a single class can lead to prototype shifts in multiple classes, not just in the one directly associated with the unlearned data.

As discussed in the previous section, during training, the contribution of a sample to all class prototypes in a DNN is influenced by its prediction loss, which is directly determined by the model's predicted probability for that sample. To explore this further, we conduct experiments analyzing how the retrained model (without the unlearned data) predicts on the unlearned samples. As shown in Figure 2, we observe a key phenomenon: in the original model, the prediction probabilities of unlearned samples tend to naturally align them with classes that are semantically or structurally close to their original category.

As shown in Figure 2, our observations highlight a key phenomenon: in the retrained model, the prediction probability for the corresponding class of the unlearned data significantly decreases, while the probabilities assigned to other classes increase accordingly. However, the overall prediction probability distribution of the unlearned samples across all classes in the original model naturally aligns with that of the retrained model. This observation inspires our approach: by assigning a carefully designed prediction probability vector to the unlearned data and training the model exclusively on these samples, we can steer the adjustment of all class prototypes in the original model. This enables a fine-grained, global correction of prototype shifts induced by unlearning.

5.1 Soft-Labeling Unlearned Data

Based on our observations, we adopt the soft label technique from prior research [30, 43] to relabel the unlearned data. By incorporating the model's predicted probability distribution, our soft-label method enables finer-grained adjustments and facilitates prototype updates across all classes.

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

$$\mathbf{z}' = \mathbf{z} - (\mathbf{z} \circ \mathbf{y})\mathbf{y}, \tag{11}$$

where $(\mathbf{z} \circ \mathbf{y})$ isolates the logit value of the corresponding class of the given unlearned sample.

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

$$p_c = \frac{z'_c}{\sum_{k=1}^{C} z'_k}, \tag{12}$$

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.

5.2 Refining Prototypes with Soft-Labeled Data

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

$$\mathbf{W}^{\text{un}} = \arg\min_{\mathbf{W}^{\text{ori}}} \sum_{\mathbf{x}_i \in \mathcal{D}_u, \mathbf{p}_i} \mathcal{L}_{\text{CE}}(\mathbf{x}_i, \mathbf{p}_i, \mathbf{W}^{\text{ori}}). \tag{13}$$

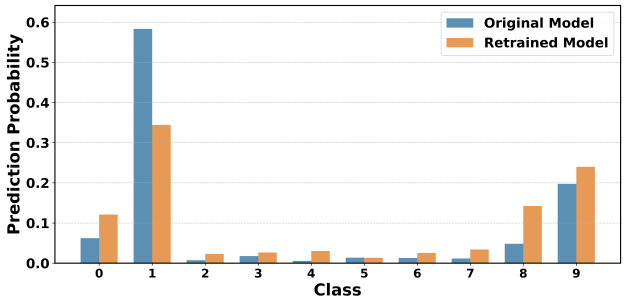


Figure 2: Consistency in prediction distributions between original and retrained models on the unlearned data. The alignment between the original and the retrained model's predictions highlights the valuable role of the original predictions in guiding prototype adjustments, especially for the classes that are not corresponding to the unlearned data.

Note that the prototype refinement process involves only the unlearned data and does not require access to either the remaining data or the test data.

This soft-label-based adjustment ensures that the prototypes in the unlearned model closely align with those of the retrained model. By leveraging only the soft-labeled unlearned samples, our method enables efficient unlearning without requiring access to the remaining training data. In addition, our PS method updates only the prototype parameters in the final classification layer of the original model, which significantly reduces computational overhead by restricting gradient updates to a single layer. Furthermore, since only the prototypes are updated while the rest of the model remains frozen, the impact on non-unlearned data is minimal, thereby preserving overall model performance. By incorporating the nuanced information captured in the soft labels, our method refines the prototypes to more accurately adjust the original model, making them consistent with those of a fully retrained model. This approach effectively minimizes the influence of the unlearned data while maintaining both robustness and efficiency.

6 Performance Evaluation

We evaluate our Naive PS and PS on multiple machine unlearning benchmarks, using experimental settings consistent with those employed in previous machine unlearning studies [4, 5, 13]. The performance of our methods is assessed from several perspectives, including unlearning performance, privacy guarantees, unlearning efficiency, and prototype similarity.

6.1 Experimental Settings

Datasets. We conduct our evaluation experiments on four widely used datasets in the field of machine unlearning: MNIST [9], CIFAR-10, CIFAR-100 [21], and TinyImageNet [8], which are consistent with existing works [4, 5, 13].

Models. We demonstrate the versatility of our methods by applying different model architectures to each dataset (c.f. Table 2). Specifically, for the MNIST dataset, we use the LeNet-5 model [23]. For the CIFAR-10 and CIFAR-100 datasets, we respectively employ the ResNet-20 and ResNet-50 models [17]. For TinyImageNet, we utilize the VGG-16 model [37].

Table 2: Datasets and Corresponding Model Architectures

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

Comparison Baselines. We compare our Naive PS and PS methods with six existing machine unlearning methods.

- `Retrain`. This method retrains the model from scratch using only the remaining data. Typically, the retrained model is considered the gold standard for machine unlearning.
- `Fine-Tune`. This method fine-tunes the original model using the remaining data with a larger learning rate for multiple epochs. Continuing to train on the remaining set induces catastrophic forgetting in the unlearning set, thereby achieving unlearning.
- `Random Label` [16]. This method fine-tunes the original model on the modified unlearned data, in which each unlearned sample is assigned a random label. Since the unlearned set is randomly labeled, the model will be misguided and perform worse on unlearning set, thus remove the corresponding influence from the original model.
- `Negative Gradient` [13]. This method fine-tunes the original model on both the unlearned data and the remaining data. For the unlearned data, it uses the negative loss gradient for parameter updates via gradient ascent. This approach intentionally degrades the model's performance on the unlearning set to achieve unlearning.
- `Fisher` [13]. The Fisher unlearning method first uses gradient information to identify the parameters most relevant to the unlearned data, and then adds noise to these parameters to interfere with the model's performance on the unlearned data, thereby facilitating unlearning.
- `L2UL` [4]. The Learning to Unlearn (L2UL) method identifies important parameters associated with the unlearned data using an importance metric. It then leverages adversarial samples generated from the unlearned data to 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. All experiments are conducted on a workstation equipped with two NVIDIA GeForce RTX 4080 GPUs. For all datasets, we use the SGD optimizer with a weight decay of 10^{-4}, a momentum of 0.9, and a batch size of 256 to train the original model. The learning rates for MNIST, CIFAR-10, CIFAR-100, and TinyImageNet are set to 10^{-3}, 0.2, 0.01, and 0.02, respectively. Unless otherwise specified, these configurations are used by default. For unlearning methods based on retraining or fine-tuning, we employ the SGD optimizer with 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 specified in the original papers.

Table 3: Average accuracy (%) of our PS and comparison methods. The prediction accuracy gap relative to the Retrain method is indicated in parentheses. Both Original and Retrain methods are not directly comparable in terms of prediction accuracy gap, hence denoted by a "-" symbol. Experiments are conducted across 4 datasets. The best results among the all unlearning methods are highlighted in bold. The Avg. Gap column represents the average performance gap compared to the Retrain method across the unlearned data, remanining data, and test data.

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)**	3.85 (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)	23.78 (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(1.40)**
	Avg. Gap	-	-	4.61	4.94	6.99	28.13	3.06	27.13	**1.02**

Specifically, for all experiments, we repeat each trial 10 times within a single run, and 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 performance on the specific data. Note that we focus on sample-level machine unlearning, where we randomly select images from a specific class to construct the unlearning dataset. In the main experiment table, consistent with the settings of existing unlearning studies [22, 26], we randomly choose 10% of the data from a particular class as the unlearned dataset, with the remaining samples from the training data forming the remaining dataset.

Metrics. We evaluate our unlearning methods across four key dimensions: Unlearning Performance, Privacy Guarantee, Unlearning Efficiency, and Prototype Similarity.

Specifically, for evaluating **Unlearning Performance**, we use four metrics: prediction accuracy on the unlearned data $\mathcal{D}_u$, the remaining data $\mathcal{D}_r$, and the original test set $\mathcal{D}_t$. Additionally, we calculate the average accuracy gap on $\mathcal{D}_u$, $\mathcal{D}_r$, and $\mathcal{D}_t$ between the unlearned model and the retrained model, which serves as the gold standard for all unlearning methods. Additionally, we use 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:

$$\mathrm{ED}(P_r \parallel P_u) = \frac{1}{2}\left(\mathrm{KL}(P_r \parallel \frac{P_r + P_u}{2}) + \mathrm{KL}(P_u \parallel \frac{P_r + P_u}{2})\right), \quad (14)$$

where the Kullback-Leibler divergence (KL) is:

$$\mathrm{KL}(P \parallel Q) = \sum P(x) \log \frac{P(x)}{Q(x)}. \quad (15)$$

A lower ED indicates that the output distribution of the unlearned model is more similar to that of the retrained model, suggesting effective unlearning. For **Unlearning Efficiency**, we measure the total time required by each unlearning method, excluding time spent on operations such as data loading and model loading.

For evaluating the performance from the **Privacy Guarantee** perspective, we follow existing works [5, 13] and employ the widely used Membership Inference Attack (MIA) [36] to assess the unlearned models. By analyzing predicted labels and probability outputs of the unlearned models, we can measure the residual influence of the unlearned data within the model. With the retrained model serving as the gold standard, the ideal outcome is that the prediction behavior of the unlearned model is indistinguishable from that of a retrained model [18, 25, 35]. Thus the MIA can achieve a similar performance on both the unlearned and retrained models, indicating that the unlearned data no longer has a detectable influence on the unlearned model.

Finally, since our method attempts to achieve machine unlearning by adjusting the prototypes of the original model to resemble those of the retrained model, the similarity between our unlearned prototypes and the retrained prototypes serves as an indicator of unlearning effectiveness. Therefore, we define the **Prototype Similarity Score** (PSC) as a metric to assess whether our unlearned prototypes are effectively aligned with the retrained ones. 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 an unlearned model. Formally, the PSC is calculated using cosine similarity, defined as:

$$\mathrm{PSC}(\mathbf{v}_r, \mathbf{v}_u) = \frac{\mathbf{v}_r \cdot \mathbf{v}_u}{|\mathbf{v}_r||\mathbf{v}_u|}, \quad (16)$$

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, suggesting effective unlearning.

6.2 Unlearning Performance

The primary objective of machine unlearning is to effectively remove the influence of the unlearned dataset $\mathcal{D}_u$ while preserving model performance on the retained dataset $\mathcal{D}_r$ and unseen test data $\mathcal{D}_t$. To evaluate the effectiveness of various unlearning methods, including our proposed approaches, we present a comparison experiment and the results are shown 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.

From the experimental results, we observe that models trained on the TinyImageNet dataset present the most challenging scenario in our experiments. In this case, our PS method achieves an exceptionally low average gap of 1.02%, representing a significant improvement compared to the latest L2UL method, which achieves a gap of 3.06%. The L2UL method uses high-level features from $\mathcal{D}_u$ to guide fine-tuning, resulting in a small gap of 0.71% on $\mathcal{D}_u$. However, it would obtain a significant gap of 5.71% on $\mathcal{D}_r$ and a 2.76% on $\mathcal{D}_t$. In contrast, our PS method fine-tunes only a small subset of parameters with the soft-labeled unlearned data, focusing specifically on prototype updates. This targeted approach minimizes disruptions to the model's knowledge, allowing it to preserve 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 gap of 1.40%). Compared to simpler methods like Random Label, which performs well on $\mathcal{D}_u$ but struggles with a higher average gap of 4.94%, our approach outperforms in retaining model integrity across datasets. This improvement can be attributed to our usage of soft-label mechanism. By leveraging soft labeled unlearned data, our method offers more precise guidance for the unlearned model, ensuring that it aligns closely with the retrained version. As a result, it retains overall model performance while effectively achieving unlearning on the unlearned data.

Next, we analyze the performance of our proposed Naive PS, which, like Fisher, avoids additional fine-tuning by directly computing the necessary parameter adjustments for unlearning. From the experimental results, we can see that Naive PS demonstrates performance comparable to the Fisher unlearning method. For models trained on simpler datasets like MNIST, Naive PS demonstrates superior unlearning performance, with a smaller average gap of 2.68%, compared to Fisher's gap of 3.11%. As noted in the original Fisher paper [13], while Fisher is effective for simple architectures like LeNet5, it struggles as model complexity increases. In contrast, Naive PS extends its effectiveness to more complex models, such as those trained on CIFAR-10, consistently outperforming Fisher and demonstrating greater robustness in unlearning. However, the limitations of Naive PS become apparent as the complexity of the models increases. On more complex models trained on datasets like CIFAR-100 and TinyImageNet, Naive PS shows a decline in unlearning performance. This can be attributed to the fact that, for models trained on CIFAR-100 and TinyImageNet, the prediction probabilities often show high values across multiple classes. As a result, during training, the unlearned data not only contributes significantly to the prototype of its own class but also to the prototypes of other classes. Therefore, when Naive PS only modifies the prototype corresponding to the unlearned data, it causes a significant drop in performance for other classes. Despite this, Naive PS still consistently outperforms Fisher across all datasets.

Overall, our proposed methods consistently achieve competitive results across all datasets, showcasing robust unlearning performance. Even on challenging datasets like TinyImageNet, our PS method achieves a significantly lower average gap compared to existing approaches, validating its effectiveness. While Naive PS exhibits some limitations on more complex datasets, it consistently outperforms Fisher unlearning. These results demonstrate that our methods are on par with existing approaches, confirming their adaptability and reliability in diverse unlearning scenarios.

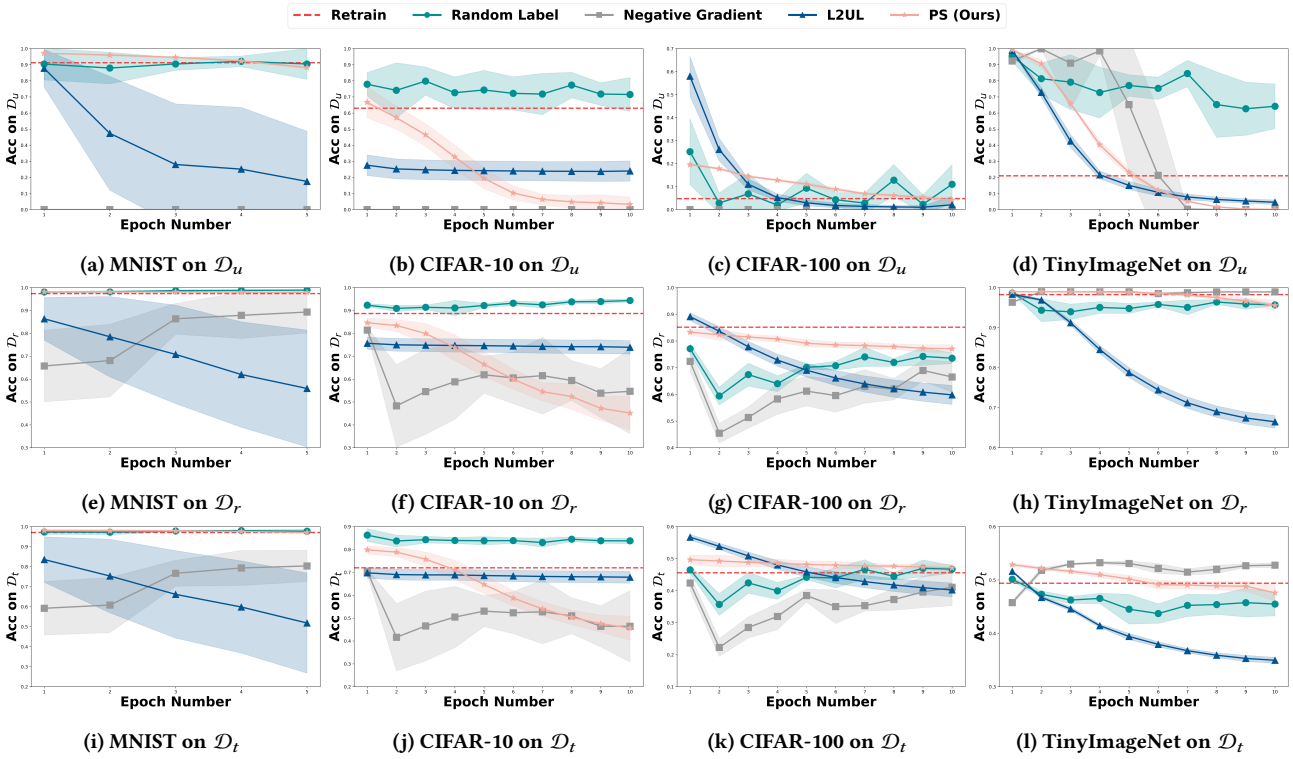
6.3 Unlearning Efficiency

Unlearning efficiency is particularly critical in applications where quick updates are required. Therefore, we compare the time required for performing unlearning using our methods against existing machine unlearning approaches. The experimental results are shown in Table 4. From the results we can see that both of our proposed methods offer significant efficiency advantages over other unlearning techniques. For instance, on the CIFAR-100 and TinyImageNet datasets, our PS and Naive PS respectively are 2,360 times and 9,176 times faster than retraining from scratch. Furthermore, compared to the latest L2UL method, our PS achieves a speedup of 1.84 times on the models trained on CIFAR-100 and 3.11 times on those on TinyImageNet. The remarkable efficiency of our approach can be attributed to two key factors. First, by focusing updates solely on the unlearned dataset, we dramatically reduce the number of samples processed in each batch, accelerating the unlearning process. Second, our method updates only the prototypes in the final classification layer of the given model, limiting gradient computations to a single layer rather than propagating through the entire model. In contrast to methods like L2UL, which also update only a subset of model parameters, they still require the gradient computations across multiple layers. Moreover, L2UL relies on adversarial samples, further increasing the computational cost. By leveraging the unique position of prototypes in the final classification layer, our prototype-based method ensures efficient updates with minimal computational effort.

Next, based on the results shown in Table 4, we compare two distinct methods (i.e. Fisher and Naive PS), which are notably faster than other methods. Neither of these two methods requires additional training. The table clearly shows that our Naive PS method significantly outperforms Fisher in terms of unlearning efficiency. This performance gap can be attributed to the fundamental differences in the computational processes of the two methods. Our Naive PS approach simply deletes selected data from a pre-stored matrix and uses basic matrix multiplication to update the model parameters. In contrast, Fisher requires the estimation of the Fisher information matrix using all the training data, a process that is both computationally intensive and time-consuming. Therefore, the substantial time advantage of our method stems directly from these

Table 4: Total time required for different machine 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) 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 $\mathcal{D}_r$

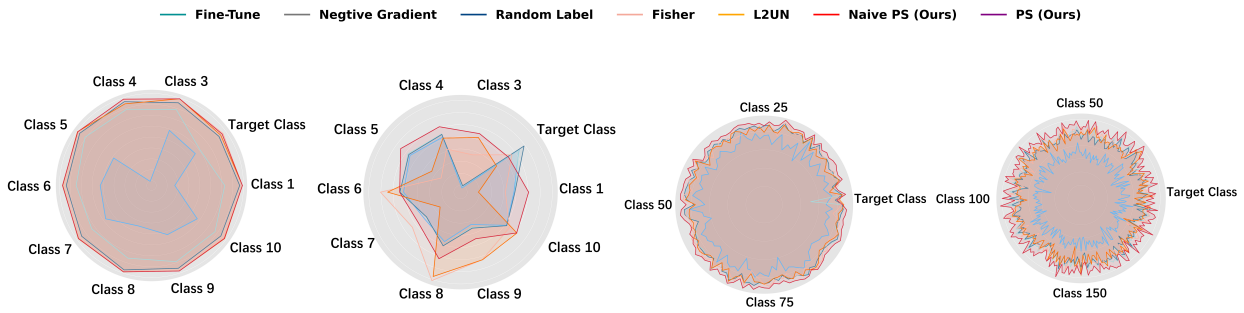
(i) MNIST on $\mathcal{D}_t$ (j) CIFAR-10 on $\mathcal{D}_t$ (k) CIFAR-100 on $\mathcal{D}_t$ (l) TinyImageNet on $\mathcal{D}_t$

Figure 3: 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.

computational differences. Overall, our Naive PS method demonstrates exceptional efficiency compared to Fisher, making it a more viable solution for time-sensitive unlearning scenarios.

After exploring the efficiency of different unlearning methods, we now examine their stability, which is critical for ensuring consistent model performance. An effective unlearning method should maintain stability across repeated experiments, meaning that the model's behavior should remain consistent with the same parameters. Figure 3 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 do not include Fine-Tune results in Figure 3, as it requires an excessive number of epochs for updates, making it difficult to clearly represent its performance changes.

Overall, our proposed PS method exhibits relatively low variance in prediction accuracies as the number of training epochs increases, quickly converging to the performance of the retrained model. Specifically, our method demonstrates minimal variance across all datasets, suggesting that the carefully soft-labeled samples enable stable and precise adjustments to the original model. On the TinyImageNet dataset, our method's unlearning speed ranks 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 shows larger discrepancies between the remaining set and the test set. In contrast, our approach minimizes the impact on unrelated data, ensuring more consistent and stable performance.

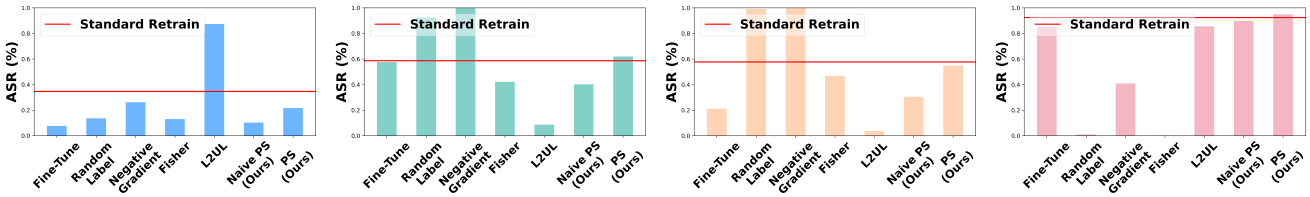


(a) MNIST (b) CIFAR-10 (c) CIFAR-100 (d) TinyImageNet

Figure 4: Prototype similarities between retrained models and unlearned models

Table 5: Ablation study with varying unlearn ratios from 5% to 15%. The table presents the prediction 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) MNIST (b) CIFAR-10 (c) CIFAR-100 (d) TinyImageNet

Figure 5: The results of Membership inference attacks against unlearned models. The red line represents the MIA performance against the retrained model. An effective unlearning method should produce attack results that are close to this red line, indicating minimal information retention from the unlearned data.

6.4 Privacy Guarantee

Our approaches focus on modifying only the final layer of the original model, which means that some information about the target unlearned data might still be retained in the intermediate layers. However, in current DNN deployment scenarios, most models are deployed via APIs or as Machine Learning-as-a-Service (MLaaS). In these settings, the internal model information is inaccessible, and only the predictions are exposed. Therefore, the key requirement in such cases is to ensure that no information about the target unlearned data is leaked through the model's output.

To evaluate whether the unlearned model, after applying our proposed methods, produces predictions that contain information about the unlearned data, we use a prediction-based MIA attack [36**?**]. This technique infers whether the unlearned data was included in the model's training process. If the MIA results on the unlearned models are consistent with those of the retrained models, it indicates that our method successfully achieves machine unlearning.

Table 6: 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

We construct an attacked dataset that contains both the unlearned data and an equal number of remaining samples. We record the Attack Success Rate (ASR) of MIA attacks for all unlearning methods across four datasets, with the results presented in Figure 5. The red line in this figure represents the ASR of the retrained model. The experimental results show that our PS method performs closest to the retrained model on the more challenging datasets, CIFAR-100 and TinyImageNet. On the simpler datasets, MNIST and CIFAR-10, although our method is not the closest to the retrained model, the difference is minimal. These results indicate that our approach effectively prevents the unlearned model from leaking private information about the forgotten samples.

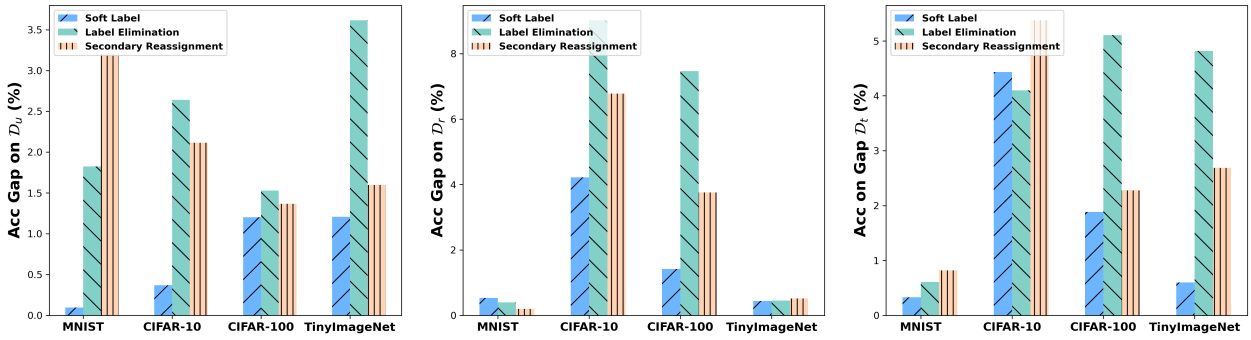
Specifically, for the CIFAR-10 dataset, Fine-Tune yields the best result in terms of ASR, but it generally exhibits less consistency across the other datasets. We believe this is because the Fine-Tune method primarily relies on the catastrophic forgetting effect to achieve unlearning. However, this effect is difficult to control and can result in significant variation in performance across different datasets. Interestingly, the L2UL method performs poorly on smaller datasets, which we attribute to its reliance on high-level features from $\mathcal{D}_u$. The guidance from these 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 information leakage during the MIA.

Next, we further analyze the prediction differences between the unlearned models and the retrained models. We present the results of the ED, which measures the JS divergence between the entropy distributions of the retrained model and the unlearned model. A smaller ED indicates a closer alignment between the retrained model and the unlearned model, indicating a better unlearning performance. The experimental results are shown in Table 6, and they align with the MIA results presented earlier. Specifically, our PS method achieves the best performance on the unlearning set, showing the lowest ED values of 0.005, 0.010, and 0.034 for CIFAR-10, CIFAR-100, and TinyImageNet, respectively. However, we are surprised to observe the high ED values for all unlearning methods on the $\mathcal{D}_r$ of TinyImageNet. We attribute this to the intrinsic complexity of the TinyImageNet dataset. Both the retrained model and all unlearned models seem to "force" themselves to memorize examples from the remaining set, rather than converging to a unified, generalized solution. Nevertheless, our approaches still achieve the best results in terms of both ASR and ED, further demonstrating the unlearning effectiveness of our Naive PS and PS methods.

6.5 Prototype Similarity

Since our method primarily adjusts the prototypes, it is essential to assess how well the unlearning process preserves the consistency of the unlearned prototypes with the retrained ones. To evaluate the similarity between the prototypes of the unlearned and retrained models, we focus on analyzing shifts in the prototype vectors. By comparing the prototype similarity before and after unlearning, we can validate whether our approach is achieving the intended outcome. To visually represent this similarity, we use radar charts, as shown in Figure 4. Each point on the chart reflects the PSC values between the unlearned model and the retrained model for a given data class. Ideally, a high PSC would indicate that the prototypes of unlearned models closely matches those of retrained models, indicating the effectiveness of our unlearning methods.

The experimental results show that our proposed methods achieve superior performance across multiple datasets, including MNIST, CIFAR-100, and TinyImageNet datasets. On the CIFAR-10 dataset, while our 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 with our expectation of more pronounced prototype shifts in the target class, further supporting the effectiveness of our approach in managing prototype dynamics during the unlearning process.



(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 Soft 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.

6.6 Impact of Unlearn Ratios

We investigate the effect of varying the unlearn ratio on the performance of our methods. Although our main results use a 10% unlearn ratio for samples from a single target class, we also evaluate performance at unlearn ratios of 5% and 15%. As shown in Table 5, our methods exhibit a high level of performance consistency across these different unlearn ratios.

The Naive PS method performs well on models trained with smaller datasets like MNIST and CIFAR-10. However, its performance shows significant variation under certain conditions. For example, when the unlearn ratio is set to 15% on CIFAR-10, there is a noticeable accuracy drop on the unlearning set, indicating the limitations of its simplified approach. This behavior suggests that Naive PS struggles to maintain effective unlearning as the unlearning ratio increases, likely due to its inadequate parameter update strategy for handling higher unlearning ratios. Similarly, while Naive PS performs reasonably on CIFAR-100 and TinyImageNet, it exhibits more pronounced performance gaps, particularly on $\mathcal{D}_u$, as the unlearn ratio grows.

In contrast, the improved PS method consistently delivers strong performance across all conditions. For instance, on the models trained on the TinyImageNet dataset, the PS method maintains minimal accuracy gaps of 14.57%, 1.21%, and 6.00% on $\mathcal{D}_u$ at unlearn ratios of 5%, 10%, and 15%, respectively. Additionally, the PS method retains high accuracy on $\mathcal{D}_r$ and $\mathcal{D}_t$ across all datasets and unlearn ratios, showcasing its robustness and adaptability. These results demonstrate that the PS method remains reliable and stable even under higher unlearn ratios, making it a robust choice for various unlearning scenarios."

6.7 Ablation Study on Labeling Methods

To assess the impact of our labeling method on machine unlearning performance, we conduct an ablation study of our approach. We evaluate three strategies: our Soft Label, Label Elimination, and Secondary Reassignment. Label Elimination involves setting the highest prediction probability of the unlearned sample to zero, and then using the remaining probabilities as the new labels. Secondary Reassignment directly assigns the second-highest predicted class as the new label. For example, if a sample's prediction probabilities are [0.6, 0.3, 0.1] across three classes, the Label Elimination method Secondary Reassignment would remove the 0.6 probability, resulting in a new soft label of [0, 0.3, 0.1]. In contrast, the Secondary Reassignment method would assign the class corresponding to the 0.3 probability as the new hard label.

Our results for the different strategies are summarized in Figure 6. These findings show that our Soft Label method closely matches the performance of the retrained model across all datasets. This is particularly evident in the accuracy gap on the unlearned dataset, where the Soft Label method consistently outperforms the other two strategies.

We attribute this success to the additional information provided by the soft labels in the our weighted approach, 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 introduce a novel prototype-based perspective for addressing the machine unlearning problem, proposing two methods: Naive PS and PS. Naive PS employs a direct adjustment strategy to approximate the unlearning process efficiently, while PS achieves machine unlearning by leveraging soft-labeled unlearned data to guide more fine-grained prototype adjustments. Since our methods perform unlearning exclusively by modifying the prototypes in the final classification layer of the DNN while keeping the rest of the model parameters unchanged, they significantly reduce computational overhead and accelerate the unlearning process. Specifically, PS further improves our naive method by fine-tuning only the prototype parameters using the unlearned data, thereby avoiding the need for additional access to the training or test data. Experimental results on four benchmark datasets and four neural network architectures demonstrate that our methods can achieve

effective unlearning while drastically reducing time and computation requirements compared to existing unlearning techniques, while maintaining performance comparable to state-of-the-art techniques. Compared with existing approaches that either relabel the training data for the entire model or apply gradient ascent to a subset of important parameters, our method decouples the relationship between training data and model parameters and enables more efficient and fine-grained unlearning. We believe our work establishes a new paradigm for machine unlearning, laying a solid foundation for future research and contributing significantly to the advancement of the machine unlearning field.

Acknowledgments

We thank the anonymous reviewers for their constructive comments. 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 GJHZ202500049, 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/CVF CVPR*. 7766–7775.

[6] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. 2022. Graph unlearning. In *Proceedings of ACM CCS*. 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/CVF 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/CVF 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 IEEE/CVF 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. *Comput. Surveys* 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/ACM 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/CVF 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 ASIACCS*. 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/CVF 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. 2024. *The EU general data protection regulation (GDPR)-A Practical Guide*. Springer Cham.

[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, 337 (2023).

[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/CVF CVPR*. 253–261.