machine learning & knowledge extraction

MDPI

Article

Certifiable Unlearning Pipelines for Logistic Regression: An Experimental Study

Ananth Mahadevan * and Michael Mathioudakis

Department of Computer Science, University of Helsinki, 00014 Helsinki, Finland; michael.mathioudakis@helsinki.fi

* Correspondence: ananth.mahadevan@helsinki.fi

Abstract: Machine unlearning is the task of updating machine learning (ML) models after a subset of the training data they were trained on is deleted. Methods for the task are desired to combine *effectiveness* and *efficiency* (i.e., they should effectively "unlearn" deleted data, but in a way that does not require excessive computational effort (e.g., a full retraining) for a small amount of deletions). Such a combination is typically achieved by tolerating some amount of approximation in the unlearning. In addition, laws and regulations in the spirit of "the right to be forgotten" have given rise to requirements for *certifiability* (i.e., the ability to demonstrate that the deleted data has indeed been unlearned by the ML model). In this paper, we present an experimental study of the three state-of-the-art approximate unlearning methods for logistic regression and demonstrate the trade-offs between efficiency, effectiveness and certifiability offered by each method. In implementing this study, we extend some of the existing works and describe a common unlearning pipeline to compare and evaluate the unlearning methods on six real-world datasets and a variety of settings. We provide insights into the effect of the quantity and distribution of the deleted data on ML models and the performance of each unlearning method in different settings. We also propose a practical online strategy to determine when the accumulated error from approximate unlearning is large enough to warrant a full retraining of the ML model.

Keywords: machine unlearning; pipelines; logistic regression

Citation: Mahadevan, A.; Mathioudakis, M. Certifiable Unlearning Pipelines for Logistic Regression: An Experimental Study. *Mach. Learn. Knowl. Extr.* **2022**, *4*, 591–620. https://doi.org/10.3390/make4030028

Academic Editor: Andreas Holzinger

Received: 27 May 2022
Accepted: 19 June 2022
Published: 22 June 2022

1. Introduction

Machine unlearning is the task of updating a machine learning (ML) model after the partial deletion of data on which the model had been trained so that the model reflects the remaining data. The task arises in the context of many database applications that involve training and using an ML model while allowing data deletions to occur. For example, consider an online store that maintains a database of ratings for its products and uses the database to train a model that predicts customer preferences (e.g., a logistic regression model that predicts what rating a customer would assign to a given product). If part of the database is deleted (e.g., if some users request their accounts to be removed), then a problem arises: how to update the ML model to "unlearn" the deleted data. It is crucial to address this problem appropriately so that the computational effort for unlearning is in proportion to the effect of the deletion. A tiny amount of deletion should not trigger a full retraining of the ML model, leading to potentially huge data-processing costs, but at the same time, data deletions should not be ignored to such extent that the ML model does not reflect the remaining data anymore.

In this work, we perform a comparative analysis of the existing methods for machine unlearning. In doing so, we are motivated both by the practical importance of the task and the lack of a comprehensive comparison in the literature. Our goal is to compare the performance of existing methods in a variety of settings in terms of certain desirable qualities.

What are those qualities? First, machine unlearning should be **efficient** (i.e., achieving small running time) and **effective** (i.e., achieving good accuracy). Moreover, machine

unlearning is sometimes required to be **certifiable** (i.e., guarantee that after data deletion the ML model operates *as if* the deleted data had never been observed). Such a requirement may be stipulated by laws (e.g., in the spirit of the *right to be forgotten* [1] or the *right of erasure* [2] in EU laws) or even offered voluntarily by the application in order to address privacy concerns. In the example of the online store, consider the case where some users request their data to be removed from its database. The online store should not only delete the data in the hosting database but also ensure that the data are unlearned by any ML model that was built from them. Essentially, if an audit was performed, the employed ML models should be found to have unlearned the deleted data as well as a model that is obtained with a brute-force full retraining of the remaining data, even if full retraining was not actually performed to unlearn the deleted data.

The aforementioned qualities exhibit pairwise trade-offs. There is a trade-off between efficiency on one hand and effectiveness or certifiability on the other because it takes time to optimize a model so as to reflect the underlying data or unlearn the deleted data. Moreover, there is a trade-off between certifiability and effectiveness. That's because unlearning the deleted data, thus ensuring certifiability, corresponds to learning from fewer data, thus decreasing accuracy. In this study, we observe the three trade-offs experimentally and find that because the compared methods involve different processing costs for different operations, they offer better or worse trade-offs in different settings.

For the experimental evaluation, we implement a common *unlearning pipeline* (Figure 1) for the compared methods. The first stage trains an *initial* ML model from the data. To limit the variable parts of our experimentation, we will be focusing on **logistic regression**, which represents a large class of models that are commonly used in a wide range of settings. In addition, we will be assuming that the initial model is trained with **stochastic gradient descent** (SGD), since SGD and its variants are the standard algorithms for training general ML models. The second stage *employs* the initial ML model for inference (i.e., for classification). During this stage, if data deletion occurs, then the pipeline proceeds to the third stage to unlearn the deleted data and produce an updated model. After every such model update, the updated model is evaluated for certifiability. If it fails, then the pipeline restarts and trains a new model from scratch on the remaining data; otherwise, it is employed in the inference stage, and the pipeline resumes. When an audit is requested by an external auditor (not shown in Figure 1), a full retraining of the ML model is executed on the remaining data, and the *fully retrained* model is compared to the currently employed model. If the employed model is found to have unlearned the deleted data as well as the fully retrained model (within a threshold of disparity), then the audit is successful, meaning the pipeline has certifiably unlearned the deleted data thus far and is allowed to resume.

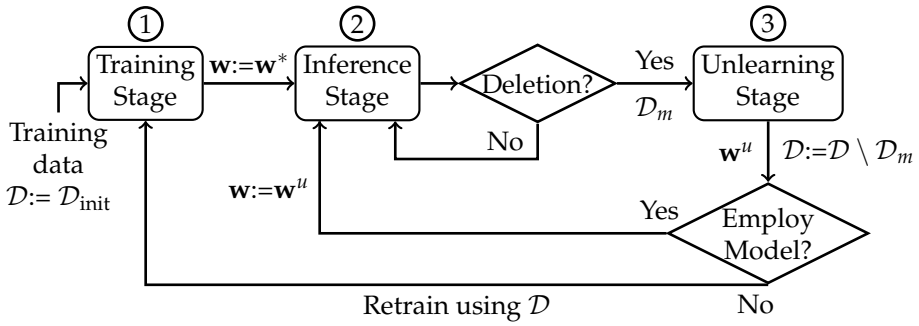


Figure 1. The common unlearning pipeline with the three stages of *training*, *inference* and *unlearning*. First, an initial model $\mathbf{w}^*$ is trained on all data and used for inference. Subsequently, whenever a part $\mathcal{D}_m$ of the data is deleted, an updated model $\mathbf{w}^u$ is obtained via machine unlearning. The pipeline restarts if the updated model is deemed inadequate.

Given this pipeline, we evaluate three methods, namely FISHER [3], INFLUENCE [4], and DELTAGRAD [5], that follow largely different approaches for machine unlearning and represent the state of the art for our setting (linear classification models trained with SGD).

FISHER, proposed by Golatkar et al. [3], updates the initial ML model using the *remaining* data to perform a corrective Newton step. INFLUENCE, proposed by Guo et al. [4], updates the initial ML model using the *deleted* data to perform a corrective Newton step. Finally, DELTAGRAD, proposed by Wu et al. [5], updates the initial ML model by correcting the SGD steps that led to the initial model. Note that, in this work, we extend the original algorithms of [3,5] to ensure that all the evaluated methods are equipped with mechanisms to control the trade-offs between efficiency, effectiveness and certifiability.

For the experimental evaluation, we implement the three methods and compare them in a large range of settings that adhere to the pipeline described above. The aim of the experiments is to demonstrate the trade-offs that the three methods offer in terms of efficiency, effectiveness and certifiability. First, we demonstrate that the trade-offs are much more pronounced for certain worst-case deletion distributions than for random deletions. Subsequently, we observe that FISHER offers the overall best certifiability, along with good effectiveness at a lower efficiency than INFLUENCE, especially for larger datasets. In addition, INFLUENCE offers the overall best efficiency, along with good effectiveness at lower levels of certifiability, and DELTAGRAD offers stable albeit lower performance across all qualities. Moreover, we observe that the efficiency of FISHER and INFLUENCE is much higher for datasets of lower dimensionality. The patterns we observe in these experiments have a beneficial by-product: they allow us to define a practical approach to determine in online fashion (i.e., as the pipeline unfolds) when the accumulated error from approximate unlearning is large enough to require restarting the pipeline to perform a full retraining of the ML model.

To summarize, we make the following contributions:

- We define a novel framework to compare machine unlearning methods in terms of effectiveness, efficiency, and certifiability.
- We extend the methods of [3,5] with mechanisms to control performance trade-offs.
- We offer the first experimental comparison of the competing methods in a large variety of settings. As an outcome, we obtain novel empirical insights about (1) the effect of the deletion distribution on the performance trade-offs and (2) the strengths of each method in terms of performance trade-offs.
- We propose a practical online strategy to determine an optimal time for when to restart the training pipeline.

As for future work, a similar experimental study would address model updates for data addition rather than deletion. For this work, we opted to focus on deletion to keep the paper well-contained, because certifiability is typically required in the case of deletion (e.g., when users request their data to be deleted from an application), and the methods we evaluate are tailored to certifiable deletion.

2. Related Work

Unlearning methods are classified as *exact* or *approximate*.

Exact unlearning methods produce ML models that perform as fully retrained models. By definition, these methods offer the highest certifiability, as the produced models are effectively the same as ones obtained with retraining. There exist several exact unlearning methods, typically for training algorithms that are model-specific and deterministic in nature. For instance, ML models such as support vector machines [6–8], collaborative filtering, naive Bayes [9,10], k-nearest neighbors and ridge regression [9] possess exact unlearning methods. The efficiency for such exact methods varies.

For stochastic training algorithms such as SGD, an exact unlearning approach, under the assumption that learning is performed in federated fashion, was proposed in [11]. In federated learning, separate ML models are trained on separate data partitions, and their predictions are aggregated during inference. This partitioning of data allows for efficient retraining of ML models on smaller fragments of data, leading to efficient unlearning when data are deleted. However, for general ML models trained with SGD, the setting of

federated learning comes with a potential cost of effectiveness that is difficult to quantify and control because model optimization is not performed jointly on the full dataset.

Approximate unlearning methods produce ML models that are an approximation of the fully retrained model. These methods typically aim to offer much larger efficiency through relaxation of the effectiveness and certifiability requirements. Most of them can be categorized into one of three groups.

The *first group* [3,12,13] uses the remaining data of the training dataset to update the ML model and control certifiability. These methods use Fisher information [14] to retain the information of the remaining data and inject optimal noise in order to unlearn the deleted data. The *second group* [4,15,16] uses the deleted data to update ML models during unlearning. They perform a Newton step [17] to approximate the influence of the deleted data on the ML model and remove it. To trade off effectiveness for certifiability, they inject random noise into the training objective function [16]. The *third group* [5,18–20] stores data and information during training and then utilizes this when deletion occurs to update the model. Specifically, these methods focus on approximating the SGD steps that would have occurred if full retraining was performed. To aid in this approximation, they store the intermediate quantities (e.g., gradients and model updates) produced by each SGD step during training. The amount of stored information and the approximation process raise an effectiveness vs. efficiency trade-off.

Methods from the above three groups can be used to perform unlearning for classification models with SGD, as long as the relevant quantities (e.g., the model gradients) are easy to compute for the model at hand. Apart from the above three groups, there are other approximate unlearning methods that do not fit the same template (e.g., methods for specific ML models, such as [21] for random forest models, or for Bayesian modeling, such as [22] for Bayesian learning), and so we consider them outside the scope of this paper.

In this paper, we focus on approximate unlearning methods, because they are applicable to general ML models, when training is performed with general and widely used optimization algorithms such as SGD. We implement three methods—FISHER, INFLUENCE and DELTAGRAD—which correspond to state-of-the-art unlearning methods from each of the aforementioned groups ([3–5], respectively).

3. Machine Unlearning

Section 3.1 below will present the stages of the unlearning pipeline and how each method implements them. Each unlearning method we consider is equipped with mechanisms to navigate trade-offs between efficiency, effectiveness and certifiability. Therefore, to facilitate presentation, let us first provide a high-level description of those mechanisms along with some related terms and notation.

In what follows, effectiveness is measured as the model's accuracy on the test dataset $\texttt{Acc}_{\text{test}}$ (i.e., the fraction of the test data that it classifies correctly). Certifiability is measured as the disparity `AccDis` in accuracy of the deleted data between the updated model (i.e., the model that results from the (possibly approximate) unlearning of the deleted data) and the fully retrained ML model. This metric is a normalized version of the "error on the cohort to be forgotten" metric seen in Golatkar et al. [3]. Intuitively, `AccDis` quantifies how well the updated model "remembers" the deleted data. If it is small, then the updated model has "unlearned" the deleted data almost as well as if fully retrained.

The first mechanism trades efficiency for effectiveness on one hand and certifiability on the other via an *efficiency parameter* τ, which is specified separately for each unlearning method. Lower values of τ indicate lower efficiency, thus allowing longer running times to improve the effectiveness and certifiability of the updated model.

The second mechanism trades effectiveness (high accuracy $\texttt{Acc}_{\text{test}}$) for certifiability (low disparity `AccDis`) via noise injection. Simply expressed, noise injection deliberately

adds randomness to an ML model both during training and unlearning. A *noise parameter* σ controls the amount of injected noise, defined for a model of d features as

$$\sigma\mathbf{b}, \text{ where } \mathbf{b} \sim \mathcal{N}(0,1)^d. \tag{1}$$

On one end of this trade-off, large amounts of noise ensure that the fully retrained and unlearned model are both essentially random, thus leading to high certifiability (low disparity `AccDis`) at the cost of low effectiveness (low accuracy). On the other end, when no noise is injected, the unlearning method strives to readapt the model to the remaining data, thus possibly sacrificing in favor of effectiveness some of the certifiability it would achieve on the first end of this trade-off. We note that this trade-off, along with noise injection as a control mechanism, has already been introduced as concepts in the differential privacy literature [4,16,23].

3.1. Unlearning Pipeline and Methods

The unlearning pipeline represents the lifecycle of the ML models in our study (Figure 1). Below, we describe how each unlearning method we consider executes each stage of the pipeline, with a summary shown in Table 1.

Table 1. Overview of the chosen unlearning methods.

Method	Training Algorithm	Unlearning Algorithm	Efficiency Parameter τ
FISHER (Appendix A.1)	$\mathbf{w}^* := \mathbf{w}^{opt} + \sigma F^{-1/4}\mathbf{b}$ $\mathbf{w}^{opt}$=arg,min$_{\mathbf{w}}$ $L(\mathbf{w},\mathcal{D})$, F=$\nabla^2 L(\mathbf{w}^{opt},\mathcal{D})$.	$\mathbf{w}^u := \mathbf{w} - F^{-1}\Delta + \sigma F^{-1/4}\mathbf{b}$ Δ=$\nabla L(\mathbf{w},\mathcal{D}\setminus\mathcal{D}_m)$, F=$\nabla^2 L(\mathbf{w},\mathcal{D}\setminus\mathcal{D}_m)$.	Unlearning mini-batch size m'
INFLUENCE (Appendix A.2)	$\mathbf{w}^* := \arg,\min_{\mathbf{w}} L_\sigma(\mathbf{w},\mathcal{D})$ $L_\sigma(\mathbf{w},\mathcal{D})$=$L(\mathbf{w},D)$+$(\sigma\mathbf{b}^\top\mathbf{w})/\lvert\mathcal{D}\rvert$	$\mathbf{w}^u := \mathbf{w} + F^{-1}\Delta^{(m)}$ $\Delta^{(m)}$=$\nabla L(\mathbf{w},\mathcal{D}_m)$, F=$\nabla^2 L(\mathbf{w},\mathcal{D}\setminus\mathcal{D}_m)$.	Unlearning mini-batch size m'
DELTAGRAD (Appendix A.3)	$\mathbf{w}^* := \mathbf{w}^{opt} + \sigma\mathbf{b}$ Store $\mathbf{w}_t$ and $\nabla L(\mathbf{w}_t,\mathcal{D})$	$\mathbf{w}^u :=$ DGAPPROX$(\{\mathbf{w}_t\}) + \sigma\mathbf{b}$ Approx $\nabla L(\mathbf{w}_t,\mathcal{D}\setminus\mathcal{D}_m)$	Periodicity T_0

3.2. Training Stage

This stage produces an ML model from the training dataset $\mathcal{D}$. In what follows, we assume that the ML model is a logistic regression, a simple and widely used model for classification. Each data point $\mathbf{x} \in \mathbb{R}^d$ is accompanied by a categorical label $\mathbf{y}$, and there are n_{init} data points initially. At any time, $\mathcal{D}$ will denote the *currently available* training dataset, which is a subset of the initial training dataset $\mathcal{D}_{\text{init}}$ due to possible deletions (i.e., $\mathcal{D} \subseteq \mathcal{D}_{\text{init}}$). The fitness of an ML model's parameters $\mathbf{w}$ on a dataset $\mathcal{D}$ is measured via an *objective function*, such as

$$L = L(\mathbf{w},\mathcal{D}) = \frac{1}{|\mathcal{D}|}\sum_{i\in\mathcal{D}} \ell(\mathbf{w}^T\mathbf{x}_i,\mathbf{y}_i) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2, \tag{2}$$

with binary cross entropy ℓ in the first term and ridge regularization in the second term for a fixed value λ.

Moreover, we use SGD for training [24]. SGD iteratively minimizes the objective function over the training data. First, it initializes the model parameters to a random value $\mathbf{w} := \mathbf{w}_0$, and it improves them iteratively:

$$\mathbf{w}_{t+1} := \mathbf{w}_t - \eta_t \nabla L(\mathbf{w}_t,\mathcal{D}) \tag{3}$$

where η_t is the learning rate at iteration t and the iterations repeat until convergence. Following common practice, we execute SGD in mini-batch fashion (i.e., only a subset of $\mathcal{D}$ is used in each execution of Equation (3)).

The training algorithms for each unlearning method are presented in the second column of Table 1. The FISHER and DELTAGRAD methods obtain the model $\mathbf{w}^{opt}$ that optimizes L and subsequently adds to it random noise proportionally to the noise parameter σ. Here, the random noise is directly added to $\mathbf{w}^{opt}$ by the DELTAGRAD method, while the FISHER method adds noise in the direction of the fisher matrix F, which for logistic regression is the Hessian of the objective L. On the other hand, the INFLUENCE method optimizes a modified objective function L_σ, which adds random noise to the standard objective L rather than directly adding noise to the model parameters. Furthermore, in the DELTAGRAD method, after every iteration of the SGD algorithm (Equation (3)), the parameters $\mathbf{w}_t$ and the objective function gradients $\nabla L(\mathbf{w}_t, \mathcal{D})$ are *stored to the disk*.

A model obtained from the training stage of the pipeline is denoted by $\mathbf{w}^*$. When it is obtained using the initial dataset $\mathcal{D}_{\text{init}}$, then $\mathbf{w}^*$ is referred to as the *initial trained* model, and when it is obtained using a subset $\mathcal{D} \subseteq \mathcal{D}_{\text{init}}$ of the training dataset, it is referred to as the *fully retrained* model. This model $\mathbf{w} := \mathbf{w}^*$ is sent to the second stage to be employed for inference.

For further details on the training algorithm, see from Appendices A.1–A.3.

3.3. Inference Stage

This stage uses the available model $\mathbf{w}$ to predict the class $\mathbf{y}$ of arbitrary data points $\mathbf{x}$ specified as queries. As soon as a subset of the data is deleted, the pipeline moves to the third stage.

3.4. Unlearning Stage

This stage executes the *unlearning algorithm* so as to "unlearn" the deleted data $\mathcal{D}_m$ and produce an updated model $\mathbf{w}^u$ (third column of Table 1).

The FISHER and INFLUENCE methods use corrective Newton steps in their unlearning algorithms (terms $-F^{-1}\Delta$ and $F^{-1}\Delta^{(m)}$, respectively). These Newton steps compute the inverse Hessian of the objective L on the remaining data $\mathcal{D} \setminus \mathcal{D}_m$. However, while the FISHER method also computes the gradient Δ on the remaining data, the INFLUENCE method computes the gradient $\Delta^{(m)}$ on the deleted data. Furthermore, the FISHER method adds random noise in the direction of the Fisher matrix, similar to the training algorithm. Both the FISHER and INFLUENCE unlearning algorithms can be performed in mini-batches of size $m' \leq m$, which leads to multiple smaller corrective Newton steps. Smaller unlearning mini-batch sizes lead to a more effective ML model at the cost of efficiency, and therefore, m' serves as the efficiency parameter τ for the FISHER and INFLUENCE methods. Refer to Algorithms A1 and A2 for the mini-batch versions of the unlearning algorithms.

The DELTAGRAD unlearning algorithm proceeds in two steps. The first step of the algorithm aims to obtain the approximate ML model that would have resulted from SGD if the subset $\mathcal{D}_m$ had never been used for training. This is achieved by approximating the objective function gradient on the remaining data points $\nabla L(\mathbf{w}_t, \mathcal{D} \setminus \mathcal{D}_m)$ using the L-BFGS algorithm [25–27] and the stored terms $\mathbf{w}_t$ and $\nabla L(\mathbf{w}_t, \mathcal{D})$ from the training stage. However, to reduce the error from consecutive L-BFGS approximations, the gradient $\nabla L(\mathbf{w}_t, \mathcal{D} \setminus \mathcal{D}_m)$ is computed explicitly after every T_0 SGD iterations. Larger values of T_0 lead to more consecutive approximations steps, resulting in higher efficiency, though at the cost of a less effective model. Therefore, the periodicity T_0 serves as the efficiency parameter τ. The second step adds random noise similar to the training algorithm. Please see Appendix A.3 and Algorithm A3 for further details.

3.5. Evaluation

During evaluation the updated model $\mathbf{w}^u$ is assessed for certifiability as soon as it is produced using the test dataset $\mathcal{D}_{\text{test}}$. If the disparity `AccDis` is below some threshold (e.g., determined by the administrator of this pipeline), then the pipeline returns to the second stage and employs the updated model $\mathbf{w}$:=$\mathbf{w}^u$ for inference; otherwise, it returns to the

first stage for a full retraining over the remaining data. In Section 7, we discuss a practical online strategy for evaluation.

A properly conducted evaluation ensures that the pipeline will successfully pass a certifiability audit at any point. In other words, if an auditor were to perform a full retraining on the available data and compare the resulting model to the currently employed model, then the two models should be found to be within the threshold of disparity.

4. Experimental Setup

In this section, we describe the datasets, the implementation of the pipeline and the metrics used for evaluation.

4.1. Datasets

We performed experiments over six datasets retrieved from the public LIBSVM repository [28]. The datasets cover a large range of size and dimensionality, as summarily shown in Table 2. In addition, we focused on the task of binary classification, and therefore, most datasets were chosen to include two target classes (or, if the original dataset contained more classes, the experiments focused on two of them, as reported in Table 2). Nevertheless, we also included one multi-class dataset (MNIST, with 10 classes).

Table 2. Datasets.

Dataset	Dimensionality		Classes	Train	Test
	d	Level		n_{init}	n_{test}
MNIST[b]	784	medium	2	11,982	1984
CIFAR2	3072	high	2	20,000	2000
MNIST	784	medium	10	60,000	10,000
COVTYPE	54	low	2	522,910	58,102
EPSILON	2000	high	2	400,000	100,000
HIGGS	28	low	2	9,900,000	1,100,000

In more detail, MNIST [29] consists of 28×28 black and white images of handwritten digits (0–9), each digit corresponding to one class. MNIST[b] is the binary class subset of the MNIST dataset, consisting only of digits "3" and "8" for both the training and test data. CIFAR2 consists of 32×32 RGB color images, belonging to the "cat" or "ship" categories from the original 10-category CIFAR-10 [30] dataset. COVTYPE [31] consists of 54 features used to categorize forest cover into 2 types. The HIGGS [32] dataset consists of kinematic features from the Monte Carlo simulation of particle detectors for binary classification. EPSILON [33] was obtained from the PASCAL Large Scale Learning Challenge 2008.

4.2. Unlearning Pipeline

We now provide implementation details for the pipeline (Figure 1). The pipeline is designed so that it is suitable for all the three chosen unlearning methods discussed in Section 3. The pipeline was implemented in Python 3.6 using PyTorch 1.8 [34]. All experiments were run on a machine with 24 CPU cores and 180 GB RAM. Our full code base is publicly available (https://version.helsinki.fi/mahadeva/unlearning-experiments (accessed on 26 May 2022)).

4.2.1. Preprocessing

The INFLUENCE unlearning method requires all data points $\mathbf{x}_i$ of a dataset to have a Euclidean norm of at most one (i.e., $\|\mathbf{x}_i\|_2 \leq 1$) (see Guo et al. [4]). To satisfy this requirement, we performed a max-L_2 normalization for all datasets as a preprocessing step, where we divided each data point $\mathbf{x}_i$ by the largest L_2 norm of any data point in the dataset. This normalization does not affect the performance of other unlearning methods.

4.2.2. Training

As mentioned in Section 3, we used the mini-batch SGD algorithm for training. In all cases, we use a fixed learning rate $\eta = 1$ and ridge regularization parameter $\lambda = 10^{-4}$. Moreover, we used standard SGD since DELTAGRAD does not support momentum-based SGD algorithms such as Adam [35]. Note that both the FISHER and INFLUENCE unlearning algorithms require a positive definite Hessian matrix to compute its inverse (see Table 1). This is ensured by a sufficiently large number of SGD iterations during training to achieve convergence. Toward this end, we used a small subset of the training data as a validation dataset to identify an optimal mini-batch size and the total number of SGD iterations. Moreover, we controlled the data points selected in each mini-batch by fixing the random seed used to produce mini-batches in the SGD algorithm. This ensured reproducibility of the experiments across various unlearning methods. Finally, for multi-class classification with $k > 2$ classes on the MNIST dataset, we trained k independent binary logistic regression classifiers in a One vs. Rest (OVR) fashion.

4.2.3. Unlearning

We modified and extended the code of Guo et al. [4] to implement the INFLUENCE and FISHER mini-batch unlearning algorithms as described in Algorithms A1 and A2. For the DELTAGRAD method, we used the code of Wu et al. [5] and modified it to add the noise injection mechanism seen in Table 1 to trade off effectiveness for certifiability.

4.3. Evaluation Metrics

In this section, we define the metrics we used to report the performance of different unlearning methods in terms of effectiveness, certifiability and efficiency. For uniformity of presentation, we reported the performance achieved by a given model as relative to the performance of a baseline model. Toward this end, we used the *Symmetric Absolute Percentage Error* (SAPE), defined as

$$\texttt{SAPE}(a, b) := \frac{|b - a|}{|b| + |a|} \cdot 100\%, \tag{4}$$

where a is the baseline value and b is the value compared against the baseline. Furthermore, we defined $\texttt{SAPE}(0, 0) = 0$.

4.3.1. Effectiveness

It is measured in terms of the predictive *accuracy* of a given ML model on a particular dataset. Let $\texttt{Acc}_{\text{test}}$ denote the accuracy on the test dataset $\mathcal{D}_{\text{test}}$. Specifically, let $\texttt{Acc}^{u}_{\text{test}}$ be the accuracy of the updated model $\mathbf{w}^u$ on the test dataset and $\texttt{Acc}^{*}_{\text{test}}$ be the optimal test data accuracy of a fully trained model, with $\sigma = 0$ obtained via logistic regression on the available data. We reported AccErr as the error in the test accuracy of the updated model $\mathbf{w}^u$ compared with the optimal one, where

$$\texttt{AccErr} := \texttt{SAPE}(\texttt{Acc}^{*}_{\text{test}}, \texttt{Acc}^{u}_{\text{test}}). \tag{5}$$

A low value for AccErr implies that the updated model $\mathbf{w}^u$ is more effective (i.e., the predictive accuracy of the updated model is close to optimal for the available data).

4.3.2. Certifiability

It is measured in terms of how well the updated model has unlearned the deleted data relative to a fully retrained model with the same method. Let $\texttt{Acc}_{\text{del}}$ be the accuracy of the deleted data $\mathcal{D}_m$. Specifically, let $\texttt{Acc}^{u}_{\text{del}}$ and $\texttt{Acc}^{*}_{\text{del}}$ be the accuracy of the deleted data for the updated model and the fully retrained model, respectively, for the same noise value σ. We reported AccDis as the disparity in accuracy of the two models, where

$$\texttt{AccDis} := \texttt{SAPE}(\texttt{Acc}^{*}_{\text{del}}, \texttt{Acc}^{u}_{\text{del}}). \tag{6}$$

The deleted data are the union of all the subsets of the initial dataset $\mathcal{D}_{\text{init}}$ that were deleted since the initialization of the pipeline. A lower value of `AccDis` implies that the updated model $\mathbf{w}^u$ has higher certifiability (i.e., the updated model is more similar to the fully retrained model, which never saw the deleted data). Note that the symmetry of `SAPE` is essential here, because both under- and over-performance of the updated model contributes toward a disparity with respect to the fully retrained model.

4.3.3. Efficiency

It is measured as the speed-up in running time to obtain the updated model $\mathbf{w}^u$ relative to the running time to obtain the fully retrained model $\mathbf{w}^*$. A speed-up of 2× indicates that the unlearning stage is able to produce an updated model twice as fast as it takes for the training stage to produce a fully retrained model.

5. Effect of Deletion Distribution

Before we compare the unlearning methods, let us explore how the volume and distribution of the deleted data affect the accuracy of fully trained models. This will allow us to separate the effects of data deletion from the effects of a specific unlearning method.

We implemented a two-step process to generate different deletion distributions. The process was invoked once for each deleted data point for a predetermined number of deletions. In the first step, one *class* is selected. For example, for binary class datasets (see Table 2), the first step selects one of the two classes. The selection may be either *uniform*, where one of the k classes is selected at random, each with a probability $1/k$, or *targeted*, where one class is randomly predetermined and subsequently always selected.

The second step selects a data point from the class chosen in the previous step. The selection may be either *random*, where one data point is selected uniformly at random, or *informed*, where the point that decreases the model's accuracy the most is selected. Ideally, for the *informed* selection and for each data point, we would compute the exact drop in the accuracy of a fully trained model on the remaining data after the single-point removal, and we would repeat this computation after every single selection. In practice, however, such an approach is extremely heavy computationally, even for experimental purposes. Instead, for the *informed* selection, we opted to heuristically select the outliers in the dataset, as quantified by the L_2 norm of each data point. This heuristic is inspired by [15], who stated that deleting data points with a large L_2 norm negatively affects the approximation of Hessian-based unlearning methods. We note that the *informed* selections are highly adversarial and are not practically feasible. We included it in our experiments to study the worst-case effects of data deletion on unlearning methods.

The two-step process described above yields four distinct deletion distributions, namely `uniform-random`, `uniform-informed`, `targeted-random` and `targeted-informed`. In the experiments that follow, we vary the distribution and volume of deletions performed. For each set of deleted data, we report the accuracy of the fully trained model after deletion (i.e., the accuracy achieved by the model that optimizes Equation (2) using SGD).

The results are shown in Figure 2. Each plot in the figure corresponds to one dataset. The first row of plots reports the accuracy on the test dataset, and the second row shows the accuracy on the deleted data. Accuracy values correspond to the y-axis, while the volume of deletion (as a fraction of the original dataset size) corresponds to the x-axis. Different deletion distributions are indicated with different markers and colors. The variance seen in Figure 2 is a consequence of the randomness in the selection of deleted points (two random runs were performed). There are three main takeaways from these results.

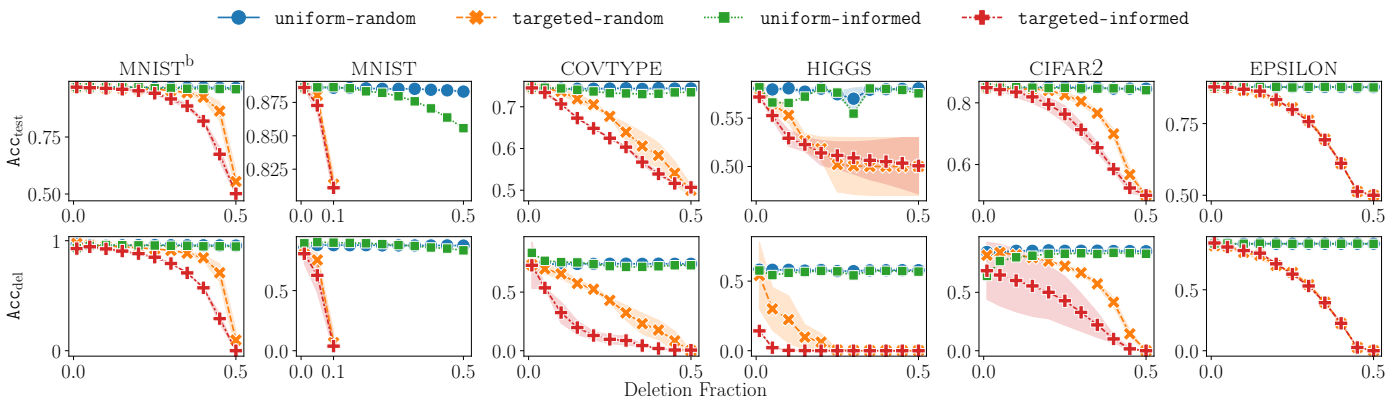


Figure 2. Effect of different deletion distributions on the test accuracy ($\texttt{Acc}_{\text{test}}$) and deleted data accuracy ($\texttt{Acc}_{\text{del}}$), as the fraction of deleted data was varied for different datasets.

1. **Uniform deletion distributions are unimpactful**
   `uniform-random` and `uniform-informed` do not adversely affect the test accuracy of a fully retrained ML model even at deletion fractions close to 0.5. This is due to the clear separation of classes encountered in many real-world datasets. Therefore, evaluating unlearning methods on deletions from uniform distributions will not offer significant insights into the effectiveness and efficiency trade-offs.
2. **Targeted deletion distributions are the worst case**
   `targeted-random` and `targeted-informed` deletions lead to large drops in test accuracy. This is because deleting data points from one targeted class eventually leads to class imbalance, causing ML models to be less effective in classifying data points from that class. Therefore, to validate the performance of unlearning methods, one should test them on targeted distributions, where data deletions reduce the accuracy of the learned model.
   In addition, we observed that the variance resulting from the selection of the deleted class was low in all datasets apart from HIGGS. We postulate that this is because of the particular way that missing values were treated for this dataset: data points that had missing feature values disproportionately belonged to class 1. Therefore, this tended to cause a steeper drop in accuracy when data from class 1 was targeted. Moreover, between the two targeted distributions, we observed that `targeted-informed` led to quicker accuracy drops.
3. **Metrics $\texttt{Acc}_{\text{test}}$ and $\texttt{Acc}_{\text{del}}$ are correlated**
   We see across deletion fractions that the values of the accuracy on the test and deleted dataset were highly correlated. Hence, the test accuracy $\texttt{Acc}_{\text{test}}$, which can always be computed for a model on the test data, can be used as a good proxy for the $\texttt{Acc}_{\text{del}}$ of an ML model, which may be impossible to compute after data deletion but is required in order to assess certifiability. This observation will be useful in deciding when to trigger a model retrained in the pipeline (Section 7).
   Additionally, we note that the drop rate in $\texttt{Acc}_{\text{test}}$ and $\texttt{Acc}_{\text{del}}$ with respect to the deletion fraction varied with the dataset.

6. Experimental Evaluation

In this section, we demonstrate the trade-offs exhibited by the unlearning methods in terms of the qualities of interest (effectiveness, efficiency and certifiability) for different values of their parameters τ and σ. For each dataset, we experimented with three volumes of deleted data points—*small*, *medium* and *large*—measured as a fraction of the initial training data. The deletion volumes corresponded to a 1%, 5% and 10% drop in $\texttt{Acc}_{\text{test}}$ for a fully retrained model when using a `targeted-informed` deletion distribution, with class 0 as the deleted class (see Figure 2). Here, we present the results for the *large* volumes of deletion, which were 4493, 2000, 4500, 78,436, 100,000 and 990,000 deletions for the

datasets in Table 2, respectively. Furthermore, we grouped the datasets presented in Table 2 into three categories (*low*, *moderate* and *high*) based on their dimensionality. Please see Appendix D for the results corresponding to the *small* and *medium* deletion volumes.

6.1. Efficiency vs. Certifiability

In this experiment, we studied how varying the efficiency parameter τ trades off certifiability and efficiency for a fixed noise parameter σ. The efficiency parameter τ for FISHER and INFLUENCE was the size of the unlearning mini-batch, and we varied $m' \in \{m, \lfloor m/2 \rfloor, \lfloor m/4 \rfloor, \lfloor m/8 \rfloor\}$, where m is the volume of deleted data. For DELTAGRAD, the efficiency parameter is the periodicity T_0 of the unlearning algorithm, and we varied $T_0 \in \{2, 5, 50, 100\}$. The noise parameter was set at $\sigma = 1$ for all methods, and we obtained the updated model $\mathbf{w}^u$ and the fully retrained model $\mathbf{w}^*$ for each unlearning method as described in Table 1.

The results are shown in Figure 3a. For each plot in the figure, the y-axis reports certifiability (`AccDis`), and the x-axis reports efficiency (speed-up). Different unlearning methods and values of τ are indicated with different colors and markers, respectively, in the legend.

We observed two main trends. First, for the general trade-off between efficiency and certifiability, higher efficiency (i.e., a higher speed-up) was typically associated with lower certifiability (i.e., a higher `AccDis`) in the plots. Some discontinuity in the plotlines, especially for DELTAGRAD, was largely due to the convergence criteria, particularly since DELTAGRAD employs SGD not only for training but also for unlearning.

Second, the efficiency of INFLUENCE and FISHER had a roughly similar trend for each dataset. For the low-dimensional datasets, INFLUENCE and FISHER provided large speed-ups of nearly 200× and 50× for each dataset, respectively, when $m' = m$, while DELTAGRAD provided a speed-up of less than 1× (i.e., requiring more time than the fully retrained model). This was because when the dimensionality was low, the cost of computing the inverse Hessian matrix for INFLUENCE and FISHER (see Section 3) was much lower compared with the cost of approximating a large number of SGD iterations for the DELTAGRAD method. Conversely, for the high-dimensional datasets, INFLUENCE and FISHER provided a smaller speed-up. When τ was decreased to $m' = \lfloor m/8 \rfloor$, the efficiency reduced to 2.2× and 1× for CIFAR2 and 0.3× and 1.2× for EPSILON, respectively, whereas DELTAGRAD at $T_0 = 50$ provided better speed-ups of 1.3× and 1.1× , respectively, along with comparable values of `AccDis`.

6.2. Efficiency vs. Effectiveness

In this experiment, we studied how varying the efficiency parameter τ trades off efficiency and effectiveness for a fixed σ and volume of deleted data. The range of τ for each unlearning method was the same as the previous experiment and $\sigma = 1$. In Figure 3b, each plot reports effectiveness as the test accuracy error `AccErr` and efficiency as the speed-up in running time. We observed the following trends. First, there was the general trade-off: higher efficiency (i.e., a higher speed-up) was associated with slightly higher accuracy error (i.e., `AccErr`) for each method. Furthermore, for INFLUENCE, decreasing τ in the unlearning algorithm led to lower test accuracy error because the noise was injected only in the training algorithm (see Table 1).

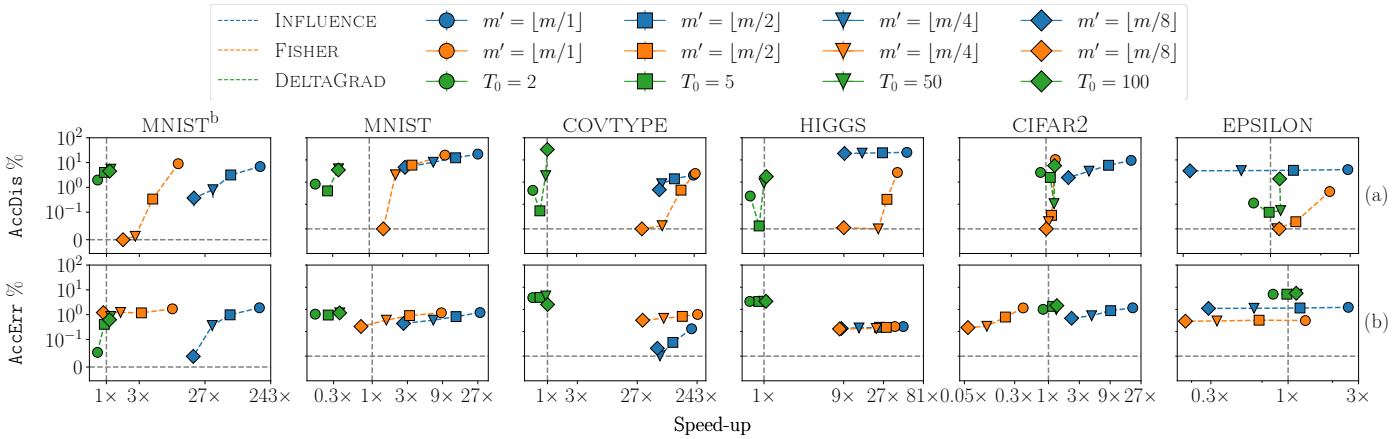


Figure 3. Trade-offs between efficiency and (**a**) certifiability and (**b**) effectiveness for $\sigma = 1$ at the largest volume of deletion as τ is varied. The y-axis reports (**a**) certifiability (`AccDis`) and (**b**) effectiveness (`AccErr`), while the x-axis reports the efficiency (speed-up).

Second, INFLUENCE offered the best efficiency and effectiveness trade-off among all the methods. Especially for the high-dimensional datasets, the highest efficiency offered was 20× and 2.5× compared with 0.4× and 1.3× for FISHER, respectively, at a slightly larger test accuracy error. For the low-dimensional datasets, INFLUENCE and FISHER offered similar efficiency while the former had a lower `AccErr`. Lastly, for the moderate dimensional datasets, the largest efficiency INFLUENCE offered was 168× and 29× compared with 9× and 8.5× for FISHER, respectively, at a lower test accuracy error.

Third, we saw that DELTAGRAD was mostly stable in terms of both efficiency and effectiveness. However, note that the test accuracy error for all datasets was larger compared with the other methods due to the direct noise injection and hence offered lower effectiveness even at $\sigma = 1$.

6.3. Effectiveness vs. Certifiability

In this experiment, we studied how varying the noise parameter σ traded off effectiveness and certifiability for a fixed efficiency parameter τ.

The efficiency parameter τ was set as follows: for INFLUENCE and FISHER, we set the size of the unlearning mini-batch to $m' = m$, and for DELTAGRAD, we set the periodicity $T_0 = 100$. For different values of the noise parameter σ, we obtained the updated models $\mathbf{w}^u$ corresponding to each unlearning method as described in Section 3. For the baselines, we first obtained the fully retrained model $\mathbf{w}^*$ at the same σ to measure certifiability and a second fully retrained model $\mathbf{w}^*$ at $\sigma = 0$ to measure effectiveness, as per Section 4.3.

The results are shown in Figure 4, where for each plot the left y-axis reports the certifiability (`AccDis`), and the right y-axis reports effectiveness (`AccErr`) as σ is varied from 10^{-2} to 10^2 for the different unlearning methods. We observed the trade-off between effectiveness and certifiability: higher effectiveness (lower `AccErr`) was typically associated with lower certifiability (higher `AccDis`).

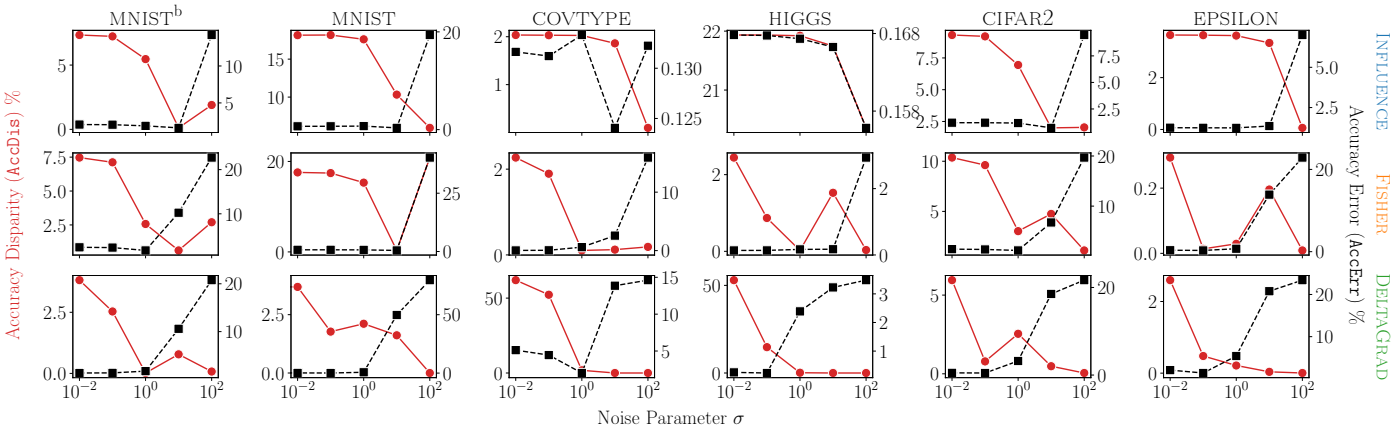


Figure 4. Effectiveness and certifiability trade-offs for largest volume of deletion. Each row corresponds to an unlearning method. Efficiency parameter is fixed at $m' = m$ for INFLUENCE and FISHER and at $T_0 = 100$ for DELTAGRAD. The left y-axis reports certifiability (AccDis), the right y-axis reports effectiveness (AccErr), and the x-axis varies the noise parameter σ. Lower is better for both y-axes.

Another clear observation is that for the INFLUENCE method, the test accuracy error AccErr increased only at higher values of σ ($\geq 10^1$). Moreover, we saw that its largest AccErr was lower than that for other methods across all datasets. For example, in the MNIST dataset, the maximum AccErr (at $\sigma = 100$) was approximately 19%, 40% and 79% for INFLUENCE, FISHER and DELTAGRAD, respectively. At the same time, however, improved certifiability (i.e., decreased AccDis) was achieved for high values of σ. Therefore, to obtain a good combination of effectiveness and certifiability, one must select higher values of σ based on the dataset.

Moreover, for FISHER, near $\sigma = 1$, the trade-off between AccErr and AccDis was the best among all methods, having values of at most 1.7% and 15%, respectively, across all datasets, as seen in Figure 4. If a good effectiveness and certifiability trade-off is required, then the FISHER method appears to be a very suitable method.

Note that because INFLUENCE and FISHER shared the same efficiency parameter τ, their results in this section are directly comparable. However, that is not the case with DELTAGRAD. As we saw earlier in this section, DELTAGRAD was typically quite slower than the other methods. Therefore, for these experiments, we used it with the largest value of $T_0 := 100$ so that its running time was small and closer to the running time of the other two methods (being, in fact, comparable for the high-dimensional datasets).

7. Online Strategy for Retraining

When the updated model $\mathbf{w}^u$ is obtained after data deletion (see Figure 1), a decision is made on whether to employ the model for inference. Specifically, if the disparity AccDis of the updated model is below a certain predetermined threshold for certifiability, then the model is employed for inference; otherwise, the pipeline restarts, training a new model on the remaining data. However, measuring AccDis using Equation (6) would require the full retrained model $\mathbf{w}^*$, which was not readily available. In fact, computing $\mathbf{w}^*$ after every batch of deletions would defeat the purpose of using an approximate unlearning method in the first place.

Therefore, in practice, AccDis needs to be estimated. To this end, we propose an online estimation strategy based on the empirical observation (see Table A5) that, as more data are deleted, the disparity AccDis grows proportionally to the drop $\texttt{AccErr}_{\text{init}}$ in test accuracy relative to the initial model $\mathbf{w}^*$:

$$\texttt{AccErr}_{\text{init}} = \texttt{SAPE}(\texttt{Acc}^{\text{init}}_{\text{test}}, \texttt{Acc}^{u}_{\text{test}}), \tag{7}$$

where $\texttt{Acc}^{\text{init}}_{\text{test}}$ and $\texttt{Acc}^{u}_{\text{test}}$ are the test accuracies for the initial and updated model, respectively. In more detail, we measured the correlations between AccDis and $\texttt{AccErr}_{\text{init}}$ for

`targeted-random` deletions while varying the deletion fraction from 0.01 to 0.45 (0.095 for MNIST) while utilizing the FISHER unlearning method ($m' = m$). We observed that, apart from the HIGGS dataset, the Pearson correlation [36] for all other datasets was greater than 87%, suggesting a strong correlation between `AccDis` and $\texttt{AccErr}_{\text{init}}$. Note that this observation is related to the correlation of accuracy on the test and deleted data that we mentioned in Section 5.

Building upon this observation, we estimated `AccDis` as

$$\overline{\texttt{AccDis}} = c \cdot \texttt{AccErr}_{\text{init}}. \tag{8}$$

where c is a constant proportion learned from the data *before* the pipeline starts as follows. First, we obtained the test accuracy $\texttt{Acc}^{\text{init}}_{\text{test}}$ for the model trained on the initial dataset. Second, we obtained an updated model $\mathbf{w}^u$ and a fully retrained model $\mathbf{w}^*$ for a large deletion fraction θ, such as $\theta = 0.45$. Third, the proportion c was calculated as follows:

$$c = [\texttt{AccDis}/\texttt{AccErr}_{\text{init}}]_{\text{deletion fraction}=\theta} \tag{9}$$

An example run of the pipeline using the estimation strategy for the threshold $\kappa = 1.0$ is shown in Figure 5. At each timestep, batches of $m = 100$ data points from a `targeted-informed` distribution were chosen for deletion. Then, the FISHER unlearning algorithm (see Table 1) was used with the efficiency parameter set as $\tau = m' := m$ to obtain an updated model $\mathbf{w}^u$. Next, the $\texttt{AccErr}_{\text{init}}$ of $\mathbf{w}^u$ was measured using the stored $\texttt{Acc}^{\text{init}}_{\text{test}}$, and the certifiability disparity was estimated using Equation (8). If the estimated $\overline{\texttt{AccDis}}$ exceeds the threshold κ, then the pipeline restarts; a new initial model is trained from scratch on the remaining data, and the $\texttt{Acc}^{\text{init}}_{\text{test}}$ is updated, while c remains unchanged. The dotted green lines in Figure 5 correspond to the times when the pipeline restarted. Then, the pipeline resumes receiving further batches of deletions until the next restart is determined by the estimation strategy or the pipeline ends.

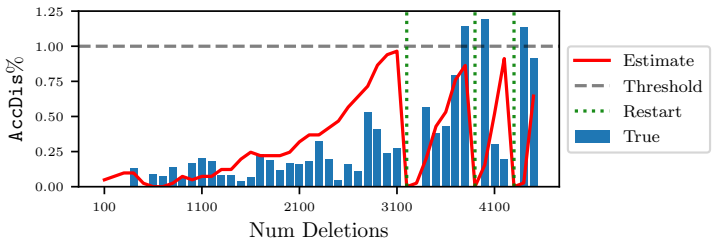


Figure 5. Pipeline run at $\kappa = 1$ for the MNIST$^{\text{b}}$ dataset. FISHER unlearning with $\sigma = 1$ and `targeted-informed` deletions.

To evaluate the strategy, we selected three thresholds κ for each dataset: 10, 20 and 50 for HIGGS and 1, 2 and 5 for the other datasets. Next, the number of deletions m was 100 for the MNIST$^{\text{b}}$, MNIST and CIFAR2 datasets and 500, 1000 and 10,000 for the for the EPSILON, COVTYPE and HIGGS datasets, respectively. In Figure 5, we observe that the initial $\overline{\texttt{AccDis}}$ estimate was larger than the true `AccDis`, and as more data were deleted, the true `AccDis` exceeded the threshold, leading to errors in the estimation. We define the relative percentage estimation error of the true `AccDis` for a given threshold κ at a given time as

$$EstDis_{\kappa} = \max(\texttt{AccDis} - \kappa, 0)/\kappa \cdot 100\%, \tag{10}$$

where the true `AccDis` is computed by obtaining a fully retrained model $\mathbf{w}^*$ on the remaining data at that time. For each individual run of the pipeline, we computed the mean *EstDis* across the duration of the pipeline (e.g., in Figure 5, the mean *EstDis* is 1.039%). Then, we collected multiple individual runs corresponding to different random seeds (six for σ and two for the random deletion distributions) and computed the *pooled mean* (i.e., the mean of the mean *EstDis* for each threshold κ).

In Table 3, we report the pooled mean of *EstDis* for different datasets and deletion distributions at different thresholds. We observe that the pooled mean *EstDis* for all thresholds was lower than 100%. The larger estimation errors for the HIGGS dataset were a result of the lower Pearson correlation of 47.8% discussed earlier. However, the Spearman correlation [36] was 89.2%, indicating that the estimation error may have been reduced by using a non-linear estimation strategy. Note that the pooled mean *EstDis* was larger for the `targeted-informed` distribution, indicating that the $\overline{\texttt{AccDis}}$ estimated using the proportion c computed from `targeted-random` deletions tended to underestimate the true disparity for the more adversarial `targeted-informed` deletions. Therefore, based on Table 3, for a pipeline to be certifiable at threshold $\kappa(\geq 2)$, the estimation strategy should be employed with a threshold $\kappa/2$, ensuring a large buffer for possible estimation errors.

Table 3. Pooled mean of $EstDis_\kappa$ % for the `AccDis` strategy at different thresholds κ. Noise parameter $\sigma = 1$.

Dataset	κ	**Uniform Random**	**Targeted Informed**
MNIST[b]	1.0	0 ± 0	0.81 ± 1.1
	2.0	0 ± 0	0 ± 0
	5.0	0 ± 0	0 ± 0
MNIST	1.0	0 ± 0	0.60 ± 0.5
	2.0	0 ± 0	0 ± 0
	5.0	0 ± 0	0 ± 0
COVTYPE	1.0	0 ± 0	28.88 ± 3.6
	2.0	0 ± 0	7.34 ± 1.5
	5.0	0 ± 0	0.16 ± 0.1
HIGGS	10.0	0 ± 0	86.99 ± 3.6
	20.0	0 ± 0	20.79 ± 1.1
	50.0	0 ± 0	1.31 ± 0.1
CIFAR2	1.0	0.17 ± 0.2	43.84 ± 11.1
	2.0	0 ± 0	11.31 ± 5.2
	5.0	0 ± 0	0.22 ± 0.4
EPSILON	1.0	0.01 ± 0.0	19.37 ± 2.6
	2.0	0 ± 0	2.09 ± 0.4
	5.0	0 ± 0	0 ± 0

In Figure 6, we report the speed-up of the pipeline with respect to retraining at every timestep for different datasets and deletion distributions at different thresholds κ. The observed drops in the speed-up for the `targeted-informed` distribution correspond to the restarts of the pipeline triggered by the estimation strategy. Notice how smaller thresholds resulted in more frequent restarts and therefore larger drops in the speed-up for `targeted-informed` deletions. Furthermore, the estimation strategy was adaptive; in the less adversarial `uniform-random` distribution, fewer restarts were triggered, thereby resulting in larger speed-ups.

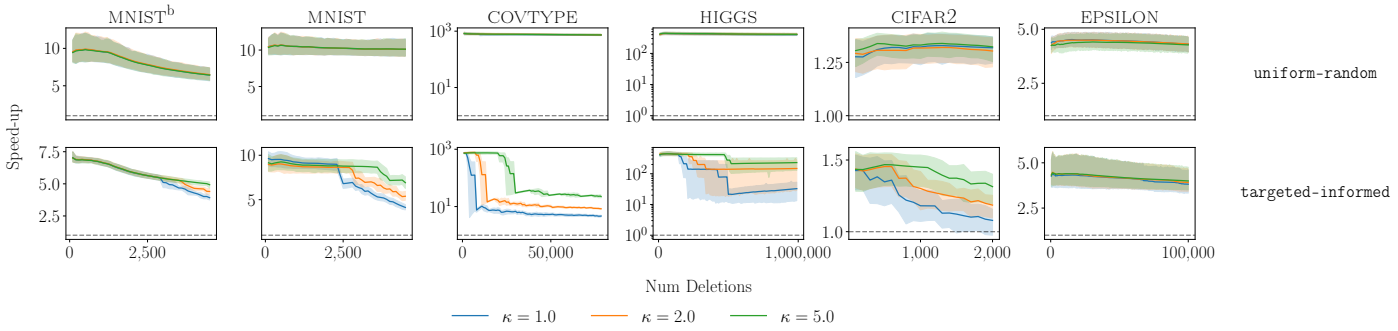


Figure 6. Speed-up of `AccDis` estimation strategy at different thresholds κ. Rows correspond to different deletion distributions. Dashed line indicates speed-up of 1× . Note: for the HIGGS dataset, $\kappa \in \{10, 20, 50\}$.

In summary, we found that the proposed strategy to estimate the disparity and restart the pipeline, albeit a heuristic, provided a significant speed-up in running time. Furthermore, as evidenced by our experiments, the strategy performed well for different distributions of deletions and could effectively be used to ensure that the pipeline was certifiable at a given disparity threshold κ. The design of more sophisticated strategies to better estimate the disparity is left for future work.

8. Conclusions

We highlighted the impact of different deletion distributions on the performance of retrained ML models and identified that the trade-offs offered by the unlearning methods must be evaluated in a worst-case scenario of targeted deletions. We studied experimentally three state-of-the-art unlearning methods for logistic regression. We found that for the right parameterization, FISHER offered the overall best certifiability, INFLUENCE offered the overall best efficiency along with good effectiveness at lower levels of certifiability, and DELTAGRAD offered stable albeit lower performance across all qualities. Third, we found that the efficiency of FISHER and INFLUENCE was much higher for the low-dimensional datasets. Furthermore, we showcased an online strategy to determine when a full retraining on the remaining data is required.

For future work, several possibilities are open. One direction is to extend the study to methods for more complex models. While the proposed unlearning pipeline can be extended to non-linear ML models such as neural networks with appropriate unlearning methods [37–39], extending the experimental evaluation (Section 6) is non-trivial. This is due to the stochasticity of learning [11] for non-linear ML models, which results in many local minima (i.e., several equally valid, fully retrained models with different accuracies). Another research direction is to develop more elaborate mechanisms to determine when a full retraining of the updated models is needed.

Author Contributions: Conceptualization, A.M. and M.M.; data curation, A.M. and M.M.; formal analysis, A.M. and M.M.; funding acquisition, M.M.; investigation, A.M. and M.M.; methodology, A.M. and M.M.; project administration, M.M.; resources, M.M.; software, A.M.; supervision, M.M.; validation, A.M. and M.M.; visualization, A.M. and M.M.; writing—original draft, A.M. and M.M.; writing—review and editing, A.M. and M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the University of Helsinki and the Academy of Finland Projects MLDB (322046) and HPC-HD (347747).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: All our experiments were conducted with publicly available datasets. Our full code base is available at https://version.helsinki.fi/mahadeva/unlearning-experiments (accessed on 26 May 2022).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Appendix A. Unlearning Methods

Appendix A.1. FISHER *Unlearning Method*

The FISHER unlearning method is described in [3].

Appendix A.1.1. Training Algorithm

The training algorithm for this method proceeds in two steps. In the first step, it invokes SGD to optimize the objective L (Equation (2)), and in the second step, it performs noise injection. The output model $\mathbf{w}^*$ is expressed as

$$\mathbf{w}^* := \mathbf{w}^{opt} + \sigma F^{-1/4}\mathbf{b}, \tag{A1}$$

where

$$\mathbf{w}^{opt} = \underset{\mathbf{w}}{\overset{SGD}{\arg,\min}}\, L(\mathbf{w}, \mathcal{D}), \tag{A2}$$

$$F = \nabla^2 L(\mathbf{w}^{opt}, \mathcal{D}), \tag{A3}$$

$$\mathbf{b} \sim \mathcal{N}(0,1)^d. \tag{A4}$$

As shown in Equation (A2), $\mathbf{w}^{opt}$ is the model that optimizes the objective function L using SGD. Moreover, F is the Fisher matrix of L, defined as the covariance of the objective function. For logistic regression, F is equal to the Hessian of L, as reflected in Equation (A3). The second term in Equation (A1) corresponds to the noise injection that adds standard normal noise (see Equation (A4)) to the optimal model $\mathbf{w}^{opt}$ in the direction of the Fisher matrix.

Appendix A.1.2. Unlearning Algorithm

The unlearning algorithm takes as input the currently employed model $\mathbf{w}$, the deleted subset of the training data $\mathcal{D}_m \subset \mathcal{D}$, and outputs an updated model $\mathbf{w}^u$ given by

$$\mathbf{w}^u := \underbrace{\mathbf{w} - F^{-1}\Delta}_{\text{Newton Correction}} + \underbrace{\sigma F^{-1/4}\mathbf{b}}_{\text{Noise Injection}}, \tag{A5}$$

where

$$\Delta = \nabla L(\mathbf{w}, \mathcal{D} \setminus \mathcal{D}_m), \tag{A6}$$

$$F = \nabla^2 L(\mathbf{w}, \mathcal{D} \setminus \mathcal{D}_m), \tag{A7}$$

where $\mathbf{b}$ is the same as in Equation (A4). As shown in Equation (A6), Δ is the gradient of the objective function L (Equation (2)), and similar to Equation (A1), F is the Fisher matrix, now computed on the remaining training data after deletion ($\mathcal{D} \setminus \mathcal{D}_m$). The first term in Equation (A5) corresponds to the corrective Newton step that aims to unlearn the deleted data $\mathcal{D}_m$. The second term corresponds to noise injection and adds standard normal noise $\mathbf{b}$ (see Equation (A4)) to the updated model $\mathbf{w}^u$ in the direction of the Fisher matrix (see Equation (A7)).

As defined in Equation (A5), the unlearning algorithm computes an updated model in a single step. A more elaborate approach is to split the deleted data in mini-batches of size $m' \leq m$ and use Equation (A5) sequentially for each of them. This approach leads to multiple and smaller corrective Newton steps, which in turn lead to a more effective ML

model at the cost of efficiency. For this experimental study, we used this mini-batch version of the unlearning algorithm, as shown in Algorithm A1.

Algorithm A1: FISHER mini-batch

Input : Employed model $\mathbf{w}$, Current training data $\mathcal{D}$, Deleted data $\mathcal{D}_m$, Parameter σ, Mini-batch size m', objective function L

Output: Updated model parameters $\mathbf{w}^u$

$s \leftarrow \lceil \frac{m}{m'} \rceil$; Split $\mathcal{D}_m$ into s mini-batches $\{\mathcal{D}^1_{m'}, \mathcal{D}^2_{m'}, \ldots, \mathcal{D}^s_{m'}\}$
$\mathcal{D}' \leftarrow \mathcal{D}$; $\mathbf{w}^u \leftarrow \mathbf{w}$
for $i \leftarrow 1$ **to** s **do**
  $\mathcal{D}' \leftarrow \mathcal{D}' \setminus \mathcal{D}^i_{m'}$; $\Delta \leftarrow \nabla L(\mathbf{w}^u, \mathcal{D}')$; $F \leftarrow \nabla^2 L(\mathbf{w}^u, \mathcal{D}')$
  $\mathbf{w}^u \leftarrow \mathbf{w}^u - F^{-1}\Delta$
  if $\sigma > 0$ **then**
    Sample $\mathbf{b} \sim \mathcal{N}(0,1)^d$
    $\mathbf{w}^u \leftarrow \mathbf{w}^u + \sigma F^{-1/4}\mathbf{b}$
  end
end
return $\mathbf{w}^u$

Appendix A.1.3. Trade-Off Parameters

As explained earlier (Section 3.1), the noise parameter σ controls the trade-off between effectiveness and certifiability. Moreover, the size of the mini-batches $\tau_{\text{FISHER}} = m'$ serves as the efficiency parameter that controls the trade-offs between efficiency on one hand and effectiveness and certifiability on the other. The lowest efficiency is achieved when $m' = 1$ (i.e., unlearning one deleted data point at a time incrementally). However, this comes at the massive cost of recomputing the Fisher matrix after every single deleted data point. The highest efficiency is achieved when $m' = m$ (i.e., unlearning all deleted data at once), which comes at the cost of effectiveness due to a single crude, corrective Newton step. In typical real settings, one would choose a value m' between the two extremes.

Appendix A.2. INFLUENCE *Unlearning Method*

The INFLUENCE unlearning method is as follows [4]. Its approach is based on ML influence theory [17]. At a high level, unlearning is performed by computing the influence of the deleted data on the parameters of the trained ML model and then updating the parameters to remove that influence. Moreover, it uses a modified objective function that incorporates noise injection:

$$L_\sigma(\mathbf{w}; \mathcal{D}) = L(\mathbf{w}, D) + \frac{\sigma \mathbf{b}^\top \mathbf{w}}{|\mathcal{D}|}, \tag{A8}$$

where L and $\mathbf{b}$ are the same as in Equations (2) and (A4), respectively. The second term in Equation (A8) describes the noise injection, where σ is the noise parameter. The amount of noise is scaled with respect to the size of the training data $\mathcal{D}$.

Appendix A.2.1. Training Algorithm

This uses SGD to optimize the noisy objective:

$$\mathbf{w}^* := \overset{SGD}{\arg\min_{\mathbf{w}}} L_\sigma(\mathbf{w}, \mathcal{D}). \tag{A9}$$

Note that when σ is increased, the effectiveness of the ML model decreases, as the SGD algorithm prioritizes minimizing the second term in Equation (A8) rather than the original objective function captured by the first term.

Appendix A.2.2. Unlearning Algorithm

The unlearning algorithm approximates the *influence* of the deleted subset $\mathcal{D}_m \subset \mathcal{D}$ on the parameters of the currently employed model $\mathbf{w}$ and performs the update as follows:

$$\mathbf{w}^u := \mathbf{w} + H^{-1}\Delta^{(m)}, \tag{A10}$$

where

$$\Delta^{(m)} = \nabla L(\mathbf{w}, \mathcal{D}_m), \tag{A11}$$

$$H = \nabla^2 L(\mathbf{w}, \mathcal{D} \setminus \mathcal{D}_m). \tag{A12}$$

As seen in Equations (A11) and (A12), $\Delta^{(m)}$ is the gradient of the objective function L (see Equation (2)) computed on the deleted data, and H is the Hessian matrix computed on the remaining training data. The second term in Equation (A10) is known as the *influence function* of the deleted data $\mathcal{D}_m$ on the model parameters $\mathbf{w}$.

Similar to FISHER, when the unlearning algorithm is performed in mini-batches of $m' \leq m$, we obtain a more effective ML model at the cost of the efficiency. This is because we compute the influence function on smaller mini-batches of deleted data multiple times. For this experimental study, we used this mini-batch version of the unlearning algorithm as shown in Algorithm A2.

Appendix A.2.3. Trade-Off Parameters

The trade-off parameters for the INFLUENCE unlearning method are similar to those in the FISHER method. The size of $\tau_{\text{INFLUENCE}} = m'$ serves as the efficiency parameter, and σ serves as the noise parameter.

Algorithm A2: INFLUENCE mini-batch

Input : Employed model $\mathbf{w}$, Current training data $\mathcal{D}$, Deleted data $\mathcal{D}_m$, Mini-batch size m', objective function L

Output: Updated model parameter $\mathbf{w}^u$

$s \leftarrow \lceil \frac{m}{m'} \rceil$; Split $\mathcal{D}_m$ into s mini-batches $\{\mathcal{D}^1_{m'}, \mathcal{D}^2_{m'}, \ldots, \mathcal{D}^s_{m'}\}$

$\mathcal{D}' \leftarrow \mathcal{D}$; $\mathbf{w}^u \leftarrow \mathbf{w}$

for $i \leftarrow 1$ **to** s **do**

  $\mathcal{D}' \leftarrow \mathcal{D}' \setminus \mathcal{D}^i_{m'}$

  $\Delta^{(m')} \leftarrow \nabla L\left(\mathbf{w}^u, \mathcal{D}^i_{m'}\right)$; $H \leftarrow \nabla^2 L(\mathbf{w}^u, \mathcal{D}')$

  $\mathbf{w}^u \leftarrow \mathbf{w}^u + H^{-1}\Delta^{(m')}$

end

return $\mathbf{w}^u$

Appendix A.3. DELTAGRAD *Unlearning Method*

The DELTAGRAD unlearning method is described in [5]. Its approach is to approximate the SGD steps that would have happened if the deleted data had not been present using the information from the initial SGD training steps.

Appendix A.3.1. Training Algorithm

It uses SGD followed by noise injection:

$$\mathbf{w}^* := \underset{\mathbf{w}}{\overset{SGD}{\arg,\min}}\, L(\mathbf{w}, \mathcal{D}) + \sigma \cdot \mathbf{b}, \tag{A13}$$

where $\mathbf{b}$ is defined as in Equation (A4)). This noise injection mechanism is a Gaussian version of the one described in [18] using the results from [23]. In contrast to FISHER method's noise injection (Equation (A5)), there is no Fisher matrix to guide the random Gaussian noise in this mechanism. Therefore, a large value of σ will indiscriminately remove

information from the employed model, which in turn drastically reduces the effectiveness of the ML model.

At every iteration of the SGD algorithm (Equation (3)), the parameters $\mathbf{w}_t$ and objective function gradients $\nabla L(\mathbf{w}_t, \mathcal{D})$ are *stored to the disk*.

Appendix A.3.2. Unlearning Algorithm

For this method the unlearning algorithm proceeds in two steps: in the first step, it approximately updates the stored sequence $(\mathbf{w}_t)$ of parameters computed by SGD; in the second step, it injects noise. In summary, and slightly abusing notation, we write

$$\mathbf{w}^u := \text{DGAPPROX}(\{\mathbf{w}_t\}) + \sigma \cdot \mathbf{b} \tag{A14}$$

The first term corresponds to the approximate update of SGD steps, and the second corresponds to noise injection, with $\mathbf{b}$ defined as in Equation (A4).

Let us provide more details about how the first term is computed. Upon the deletion of the current subset of the training data $\mathcal{D}_m \subset \mathcal{D}$, the unlearning algorithm aims to obtain the approximate ML model that would have resulted from SGD if $\mathcal{D}_m$ had never been used for training. By definition, in the absence of $\mathcal{D}_m$, the gradient of the objective function at every step of the SGD algorithm would have been

$$\nabla L(\mathbf{w}_t, \mathcal{D} \setminus \mathcal{D}_m) = \frac{[n\nabla L(\mathbf{w}_t, \mathcal{D}) - m\nabla L(\mathbf{w}_t, \mathcal{D}_m)]}{(n-m)}. \tag{A15}$$

Using Equation (A15), the SGD step from Equation (3) can be rewritten as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta_t}{(n-m)}[n\nabla L(\mathbf{w}_t, \mathcal{D}) - m\nabla L(\mathbf{w}_t, \mathcal{D}_m)]. \tag{A16}$$

These SGD steps lead to a different sequence $(\mathbf{w}_t)$ of model parameters than the one obtained before deletion from Equation (3). Consequently, the value of $\nabla L(\mathbf{w}_t; \mathcal{D})$ differs between the executions of Equation (3) (before deletion) and Equation (A16) (after deletion). DELTAGRAD's approach is to obtain a fast approximation of the latter from the former, thus approximately unlearning the deleted data without performing a full-cost SGD on the remaining data.

The unlearning algorithm is shown for reference in Algorithm A3. As seen in line 9, the term $\nabla L(\mathbf{w}_t, \mathcal{D})$ is approximated using the quasi-Newton L-BFGS optimization algorithm with the terms $\mathbf{w}_t$ and $\nabla L(\mathbf{w}_t, \mathcal{D})$, which were stored during training. However, there exist two issues with this approximation. First, the L-BFGS algorithm requires a history of accurate computations to produce an effective approximation. Second, consecutive approximations lead to errors accumulating after several iterations in SGD. The first issue is addressed by using a *burn-in period* of j_0 iterations, during which the exact gradient on the remaining dataset $\nabla L(\mathbf{w}_t, \mathcal{D} \setminus \mathcal{D}_m)$ is computed. The latter issue is addressed by periodically computing the exact gradient after every T_0 iterations (following the burn-in period). These are seen from lines 3 to 7. Moreover, in order to use the above DELTAGRAD algorithm for subsequent data deletions, the terms $\mathbf{w}_t$ and $\nabla L(\mathbf{w}_t, \mathcal{D})$ that were previously stored in the disk are updated after unlearning the deleted data $\mathcal{D}_m$. This is described in lines 6 and 7 and lines 13 and 14.

Algorithm A3: DGAPPROX

Input : Current training data $\mathcal{D}$, Deleted data $\mathcal{D}_m$, model weights saved during the training stage or updated later $\{\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_t\}$ and corresponding gradients $\{\nabla L(\mathbf{w}_0, \mathcal{D}), \nabla L(\mathbf{w}_1, \mathcal{D}), \ldots, \nabla L(\mathbf{w}_t, \mathcal{D})\}$, period T_0, total iteration number T, "burn-in" iteration number j_0, learning rate η_t

Output: Updated model parameter $\mathbf{w}_t$

Initialize $\mathbf{w}_0 \leftarrow \mathbf{w}_0$
for $t \leftarrow 0$ **to** $T-1$ **do**
  if $[((t - j_0) \mod T_0) == 0]$ *or* $t \leq j_0$ **then**
    Compute $\nabla L(\mathbf{w}_t, \mathcal{D} \setminus \mathcal{D}_m)$ exactly
    Compute $\mathbf{w}_t$ by using exact update (Equation (3))
    Update $\mathbf{w}_t$ with new $\mathbf{w}_t$
    Update $\nabla L(\mathbf{w}_t, \mathcal{D})$ with $\nabla L(\mathbf{w}_t, \mathcal{D} \setminus \mathcal{D}_m)$
  else
    Approximate $\nabla L(\mathbf{w}_t, \mathcal{D})$ with L-BFGS algorithm using stored terms $\mathbf{w}_t$ and $\nabla L(\mathbf{w}_t, \mathcal{D})$
    Compute $\nabla L(\mathbf{w}_t, \mathcal{D}_m)$
    Compute approximate $\nabla L(\mathbf{w}_t, \mathcal{D} \setminus \mathcal{D}_m)$ using Equation (A15)
    Compute $\mathbf{w}_t$ using Equation (A16)
    Update $\mathbf{w}_t$ with new $\mathbf{w}_t$
    Update $\nabla L(\mathbf{w}_t, \mathcal{D})$ with approx $\nabla L(\mathbf{w}_t, \mathcal{D} \setminus \mathcal{D}_m)$
  end
end
return $\mathbf{w}_t$

Appendix A.3.3. Trade-Off Parameters

As described in Table 1, the primary τ parameter chosen for the DELTAGRAD method was the periodicity T_0. Based on the discussion of the hyper-parameter of the DELTAGRAD in [5], the ideal τ parameter would be the training mini-batch size. However, this would result in a non-standard training stage in the unlearning pipeline for the DELTAGRAD, which in turn would prevent any comparison with the other unlearning methods. Therefore, upon fixing the common training stage, we choose the hyper-parameter T_0 that best represents the trade-off between effectiveness and efficiency. The remaining candidate τ parameters are the burn-in period j_0 and the size of the history for the L-BFGS algorithm h. Following [5], we fix $h = 2$ for all datasets, and the values of j_0 are presented in Table A1.

Table A1. Values of DELTAGRAD burn-in period parameter j_0.

Dataset	MNIST[b]	MNIST	COVTYPE	HIGGS	CIFAR2	EPSILON
j_0	10	20	10	500	20	10

Appendix B. Experimental Set-Up

In this section, we discuss the additional details regarding the experiments and the implementation of the common unlearning pipeline.

Training

Ensuring the training phase of the common unlearning pipeline, especially the optimization of each unlearning method, is a difficult task. As mentioned in Section 4.2, INFLUENCE and FISHER require SGD convergence, and DELTAGRAD can only use vanilla SGD. The additional constraints come from the DELTAGRAD method. In [5], it is described that a smaller mini-batch size leads to lower approximation and hence lower effectiveness. However, choosing a full-batch gradient descent update as described in Equation (3) to ensure the best performance of the DELTAGRAD method leads to the requirement of a large number of epochs to achieve convergence for INFLUENCE and FISHER. This is computationally expensive both in the calculation of full-batch gradients for the large

datasets such as EPSILON and HIGGS and the number of epochs required in total to reach convergence. Ideally, to reduce the impact of the latter, we would fix a number of epochs and then select a larger learning rate η to compensate for the slower average gradient updates. However, we experimentally found that increasing the learning rate η beyond one has a significant impact on the performance of DELTAGRAD. This is primarily because the error in the approximate SGD step is amplified as the learning rate is increased beyond the optimal learning rate (which results in an increased number of epochs to achieve the same convergence). These constraints and limitations led us to fix the learning rate to one and choose large enough mini-batches (for DELTAGRAD performance) while keeping the number of epochs low (for computational effort) while using a small validation dataset of the initial training data $\mathcal{D}_{\text{init}}$. The chosen values of the mini-batch size and the number of epochs for each dataset are described in Table A2.

Table A2. Values of training parameters common for all unlearning methods.

Dataset	Epochs	Mini-Batch Size
MNIST[b]	1000	1024
MNIST	200	512
COVTYPE	200	512
HIGGS	20	512
CIFAR2	500	512
EPSILON	60	512

Appendix C. Extended Deletion Distribution Results

We present the extended results for the `uniform-random` and `uniform-informed` deletion distributions in Figure A1. We increased the fraction of data deleted from 0.5 to 0.995. We see that the drop in both $\texttt{Acc}_{\text{test}}$ and $\texttt{Acc}_{\text{del}}$ only occurred when we deleted beyond 90% of the initial training data. We also clearly see that the drop in both metrics was much steeper for the `uniform-informed` distribution compared with the `uniform-random` distribution. This indicates that the informed deletions were deleting outliers that were required by the ML model to effectively classify samples.

Appendix D. Extended Experimental Results

In this section, we provide the extended results of the experiments discussed in Section 4.

Appendix D.1. Deletion Volumes

As mentioned in Section 4, we evaluated the unlearning methods for *three* volumes of deletion corresponding to different drops in test accuracy of the retrained model. In Table A3, we present both the fraction and the volume of deletion corresponding to 1%, 5% and 10% drops in test accuracy when `targeted-informed` deletions were used. They are named *small*, *medium* and *large* volumes of deletion, respectively.

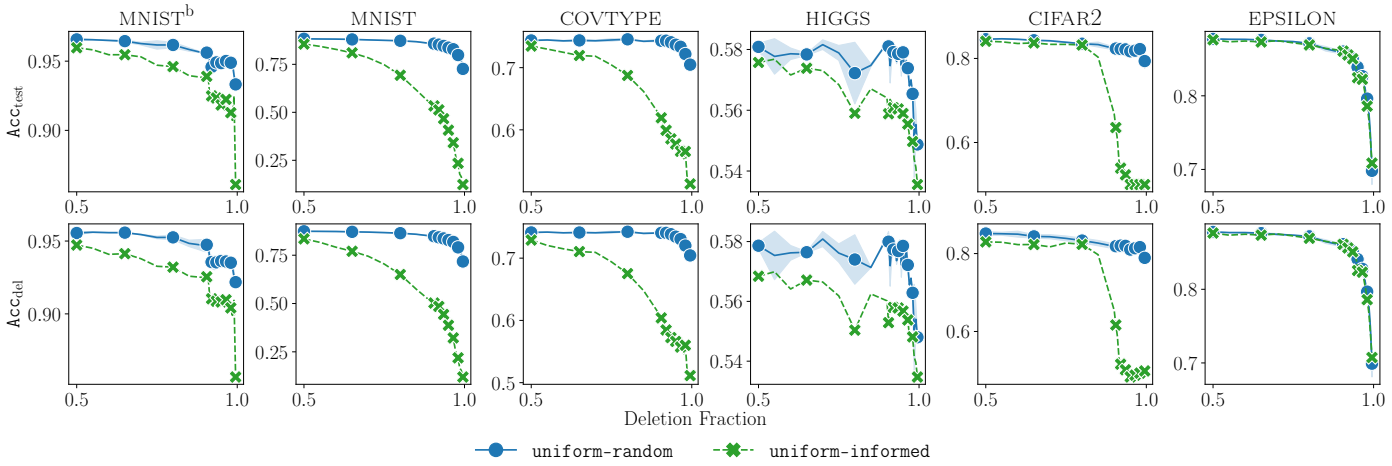


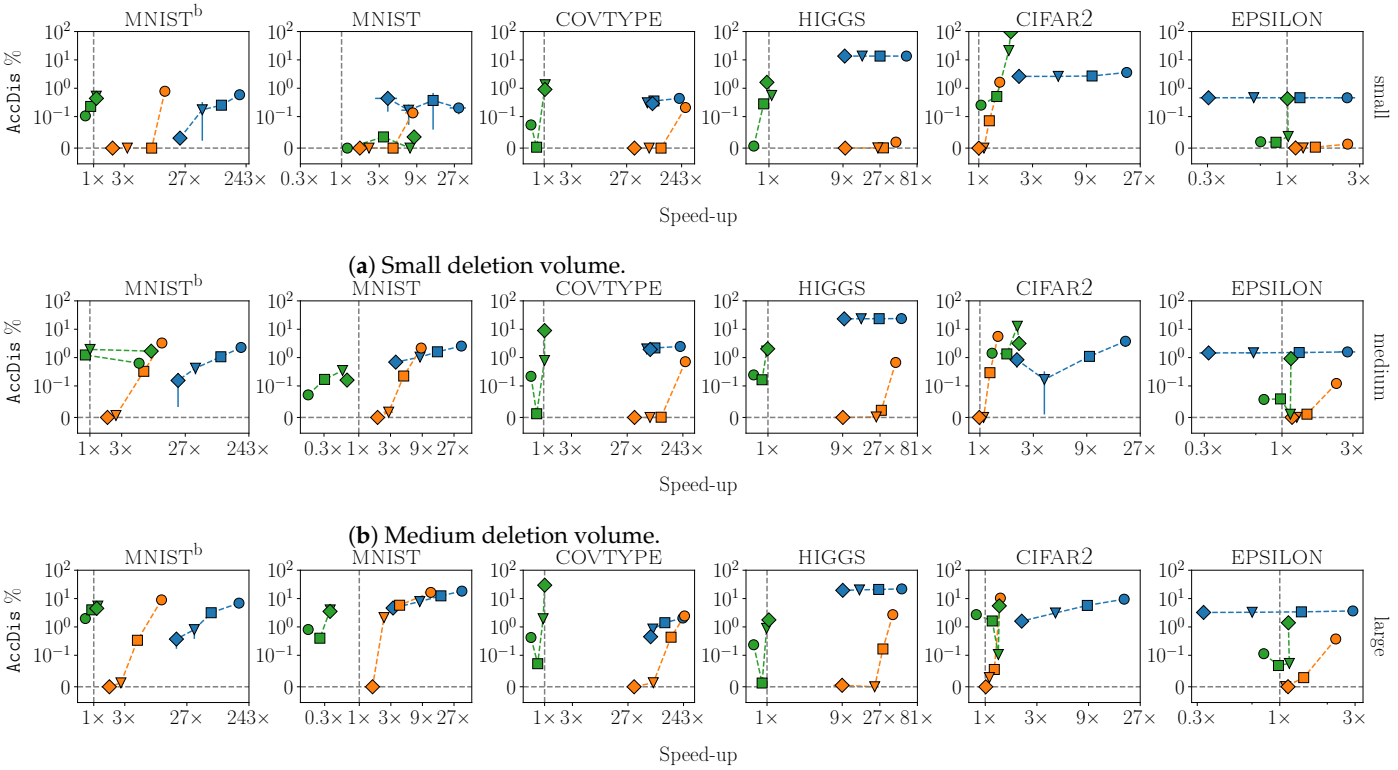
Figure A1. Extended deletion distribution results. Deletion fraction varied from 0.5 to 0.995. Only `uniform-random` and `uniform-informed` deletion distribution results are reported.

Table A3. Fraction and volume of deleted data of initial data volume, corresponding to different drops in $\texttt{Acc}_{\text{test}}$ for fully retrained model and `targeted-informed` deletion.

Dataset	Small 1% Drop		Medium 5% Drop		Large 10% Drop	
	Fraction	$\lvert\mathcal{D}_m\rvert$	**Fraction**	$\lvert\mathcal{D}_m\rvert$	**Fraction**	$\lvert\mathcal{D}_m\rvert$
MNIST^b	0.2	2396	0.3	3594	0.375	4493
MNIST	0.01	600	0.05	3000	0.075	6000
COVTYPE	0.05	26,145	0.10	52,291	0.15	78,436
HIGGS	0.01	99,000	0.05	495,000	0.10	990,000
CIFAR2	0.05	500	0.125	1250	0.2	2000
EPSILON	0.1	4000	0.2	8000	0.25	10,000

Appendix D.2. Efficiency and Certifiability Trade-Off

Here, we present the extended results for all volumes of deletion and $\sigma = 1$ as the efficiency parameter τ is varied in Figure A2. The legend for τ is the same as that displayed in Figure 3. The key takeaway here is that the trends described in Section 6 were similar for the small and medium volumes of deletion.



(**a**) Small deletion volume.

(**b**) Medium deletion volume.

(**c**) Large deletion volume.

Figure A2. Efficiency and certifiability trade-off results for $\sigma = 1$ at different volumes of deletion: (**a**) small, (**b**) medium and (**c**) large. `AccDis` is reported on the y-axis and the speed-up in running time on the x-axis. The legend is the same as in Figure 3.

Appendix D.3. Efficiency and Certifiability Trade-Off

The results are shown in Figure A3. The trends described in Section 6 also apply for the *small* and *medium* volumes of deletion.

Appendix E. When to Retrain Strategies

The implementation details for the pipeline experiments are shown in Table A4.

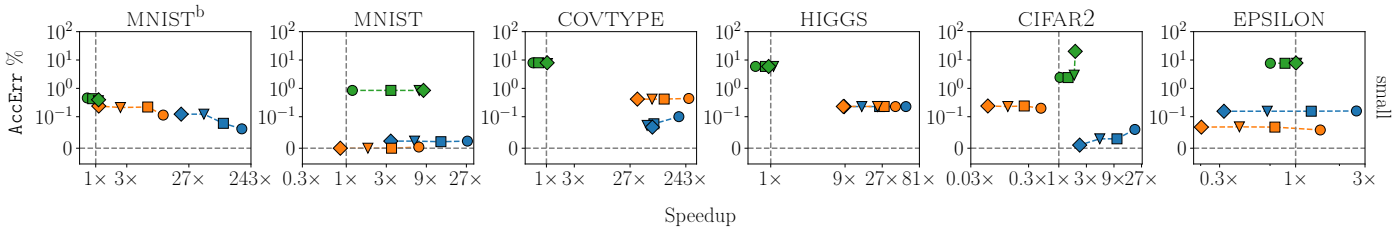
Table A4. Batch sizes and thresholds κ for the pipeline experiments.

Dataset	Batch Size	κ for `AccErr`	κ for `AccDis`
MNIST^b	100	0.25, 0.5, 1	1, 2, 5
MNIST	100	0.25, 0.5, 1	1, 2, 5
COVTYPE	1000	0.25, 0.5, 1	1, 2, 5
HIGGS	10,000	0.25, 0.5, 1	10, 20, 50
CIFAR2	100	0.25, 0.5, 1	1, 2, 5
EPSILON	500	0.25, 0.5, 1	1, 2, 5

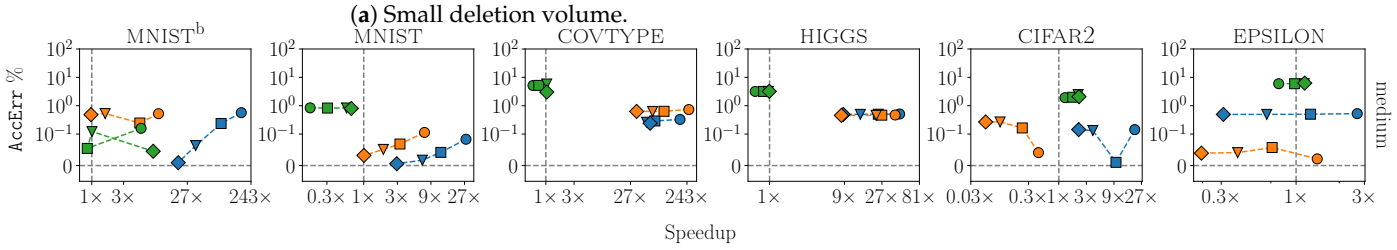
Appendix E.1. `AccDis` Strategy

The empirical correlations between `AccDis` and $\texttt{AccErr}_{\text{init}}$ when `targeted-random` deletions were performed and the FISHER unlearning method ($m' = m$) while the deletion ratio was varied are shown in Table A5. As noted in Section 7, the Pearson correlations are high for all datasets excluding HIGGS. The low Pearson correlation values for the HIGGS dataset can be attributed to the the variance resulting from the selection of the deleted class seen in Figure 2. We postulate that this is because of the particular way that missing values have been treated for the HIGGS dataset. Data points that have missing feature values disproportionately belong to class 1. Therefore, this tends to cause a steeper drop

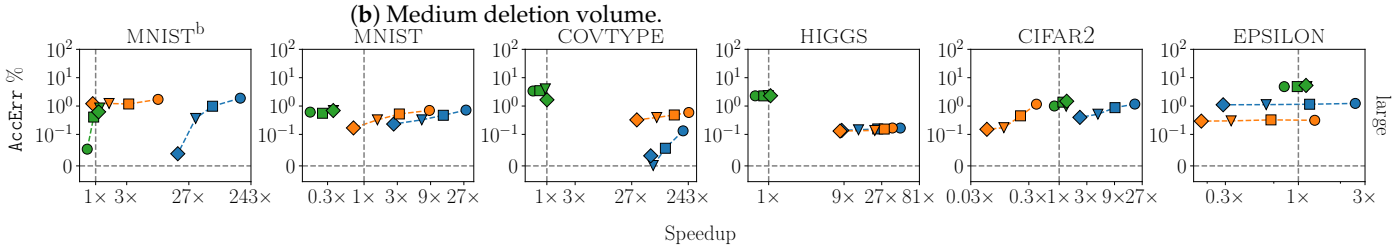
in accuracy when data from class 1 is targeted. In turn, this leads to the lower predictive capability of the linear estimation strategy described in Equation (8).



(**a**) Small deletion volume.



(**b**) Medium deletion volume.



(**c**) Large deletion volume.

Figure A3. Efficiency and effectiveness trade-off results for $\sigma = 1$ at different volumes of deletion: (**a**) small, (**b**) medium and (**c**) large. `AccErr` is reported on the y-axis and the speed-up in running time on the x-axis. The legend is the same as in Figure 3.

The extended results of the pooled *EstDis* mean and the speed-up are shown in Table A6 and Figure A5, respectively. In Figure A5, we observe that for the `uniform-informed` deletion distribution, more pipeline restarts were triggered compared with `uniform-random`, a consequence of the deletion of more informative samples, leading to a larger estimated disparity $\overline{\texttt{AccDis}}$.

Table A5. Correlation between $\texttt{AccErr}_{\text{init}}$ and `AccDis` with `targeted-random` deletion distribution for the FISHER method.

Dataset	Pearson Corr.	Spearman Corr.
MNIST^b	0.963	0.612
MNIST	0.999	1
COVTYPE	0.881	0.964
HIGGS	0.478	0.892
CIFAR2	0.938	0.976
EPSILON	0.8787	0.891

Appendix E.2. `AccErr` Strategy

Here, we describe a strategy to monitor and trigger the pipeline to restart based on the effectiveness of the updated model $\mathbf{w}^u$.

When a subset of the existing data $\mathcal{D}_m \subset \mathcal{D}$ is deleted, the updated model $\mathbf{w}^u$ is obtained from the unlearning stage of the pipeline. We propose an estimate of `AccErr` using the test accuracy of initial model $\texttt{AccErr}_{\text{init}}$, defined in Equation (7). The $\texttt{Acc}_{\text{test}}^{\text{init}}$ is stored

during the initialization of the pipeline, when $\mathbf{w}^*_{\text{init}}$ is obtained after training on the initial dataset $\mathcal{D}_{\text{init}}$. If the computed $\texttt{AccErr}_{\text{init}}$ does not exceed the predefined threshold, then $\mathbf{w}^u$ is employed for inference; otherwise, the pipeline restarts, obtaining a new $\mathbf{w}^*_{\text{init}}$ by training on the remaining dataset $\mathcal{D}_{\text{init}} := \mathcal{D}$. The $\texttt{Acc}^{\text{init}}_{\text{test}}$ is also recomputed and updated, and the pipeline resumes.

An example of the pipeline's execution is shown in Figure A4 for the MNIST$^{\text{b}}$ dataset and `targeted-random` deletions at a predefined threshold of $\kappa = 0.25$. We must note that the pipeline estimate of the `AccErr` is zero, corresponding to the restarts of the pipeline. This indicates that at that iteration, the updated model's $\texttt{AccErr}_{\text{init}}$ was exceeding the threshold, and a restart was triggered. These pipeline restarts require retraining from scratch and hence increase the running time of the pipeline.

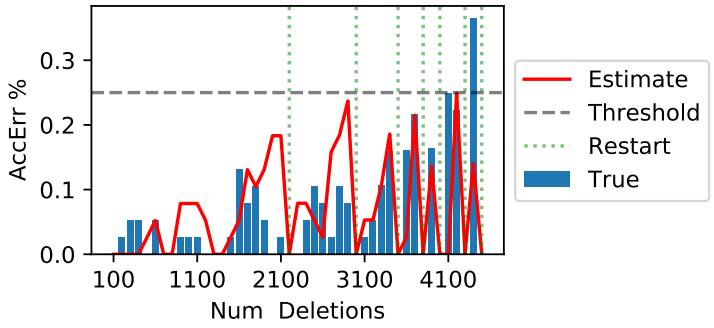


Figure A4. `AccErr` strategy pipeline run at $\kappa = 0.1$ for the MNIST$^{\text{b}}$ dataset, as well as FISHER method with $\sigma = 1$ and `targeted-informed` deletions.

In Figure A6, we report the speed-up for different values of κ and deletion distributions as a function of the number of deletions. Here, a speed-up of 1× corresponds to retraining at every iteration of the pipeline. Observe how the lower values of κ lead to earlier drops in the speed-up, indicating the pipeline has been reset. Moreover, we observed this decrease in speed-up more significantly for the more adversarial deletion distributions (`targeted-random` and `targeted-informed`). In the less adversarial deletion distribution (`uniform-random` and `uniform-informed`), we observed only the smallest κ led to drops in the speed-up, indicating that the strategy was adaptive. Lastly, even for the lowest value of κ and the most adversarial deletion distribution of `targeted-informed`, we observed for all datasets a significant speed-up compared with always retraining.

In Figure A4, we observe that as the number of deletions increased, the $\texttt{AccErr}_{\text{init}}$ underestimated the true `AccErr` of the model. We quantified these relative estimation errors of `AccErr` for a given threshold κ as

$$EstErr_\kappa = \frac{\max(\texttt{AccErr} - \kappa, 0)}{\kappa} \cdot 100\%, \tag{A17}$$

where `AccErr` is the true `AccErr` computed using the fully retrained model $\mathbf{w}^*$. A relative estimation error percentage of 50% for $\kappa = 0.25$ implies that the true `AccErr` was 0.375. For each run of the pipeline, we computed the mean *EstErr* (e.g., the mean *EstErr* was 4.9% for the pipeline run in Figure A4). Then, for multiple pipeline runs for different random seeds of σ and the deletion distributions, we reported the mean of the relative estimation error. In Table A7, we report the mean of *EstErr* for each dataset and the value of κ for different deletion distributions. The largest mean *EstErr* corresponded to the smallest value of κ, while larger values of κ had very small mean *EstErr* values across deletion distributions. We note that the HIGGS dataset had significantly larger mean *EstErr* values as a result of the larger number of deletions in each batch size. Selecting a larger threshold κ resulted in lower estimation errors.

Table A6. Pooled mean of $EstDis_\kappa$% for the `AccDis` strategy at different thresholds κ. Noise parameter $\sigma = 1$.

Dataset	κ	Uniform Random	Uniform Informed	Targeted Random	Targeted Informed
MNIST[b]	1.0	0.00 ± 0.0	0.00 ± 0.0	0.01 ± 0.0	0.81 ± 1.1
	2.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0
	5.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0
MNIST	1.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.60 ± 0.5
	2.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0
	5.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0
COVTYPE	1.0	0.00 ± 0.0	0.32 ± 0.6	8.01 ± 1.5	28.88 ± 3.6
	2.0	0.00 ± 0.0	0.00 ± 0.0	0.42 ± 0.2	7.34 ± 1.5
	5.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.16 ± 0.1
HIGGS	10.0	0.00 ± 0.0	0.00 ± 0.0	109.32 ± 7.7	86.99 ± 3.6
	20.0	0.00 ± 0.0	0.00 ± 0.0	30.68 ± 2.0	20.79 ± 1.1
	50.0	0.00 ± 0.0	0.00 ± 0.0	1.78 ± 0.2	1.31 ± 0.1
CIFAR2	1.0	0.17 ± 0.2	2.93 ± 3.3	3.68 ± 2.5	43.84 ± 11.1
	2.0	0.00 ± 0.0	0.15 ± 0.3	0.24 ± 0.3	11.31 ± 5.2
	5.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.22 ± 0.4
EPSILON	1.0	0.01 ± 0.0	0.06 ± 0.1	20.01 ± 2.1	19.37 ± 2.6
	2.0	0.00 ± 0.0	0.00 ± 0.0	2.93 ± 0.5	2.09 ± 0.4
	5.0	0.00 ± 0.0	0.00 ± 0.0	0.22 ± 0.1	0.00 ± 0.0

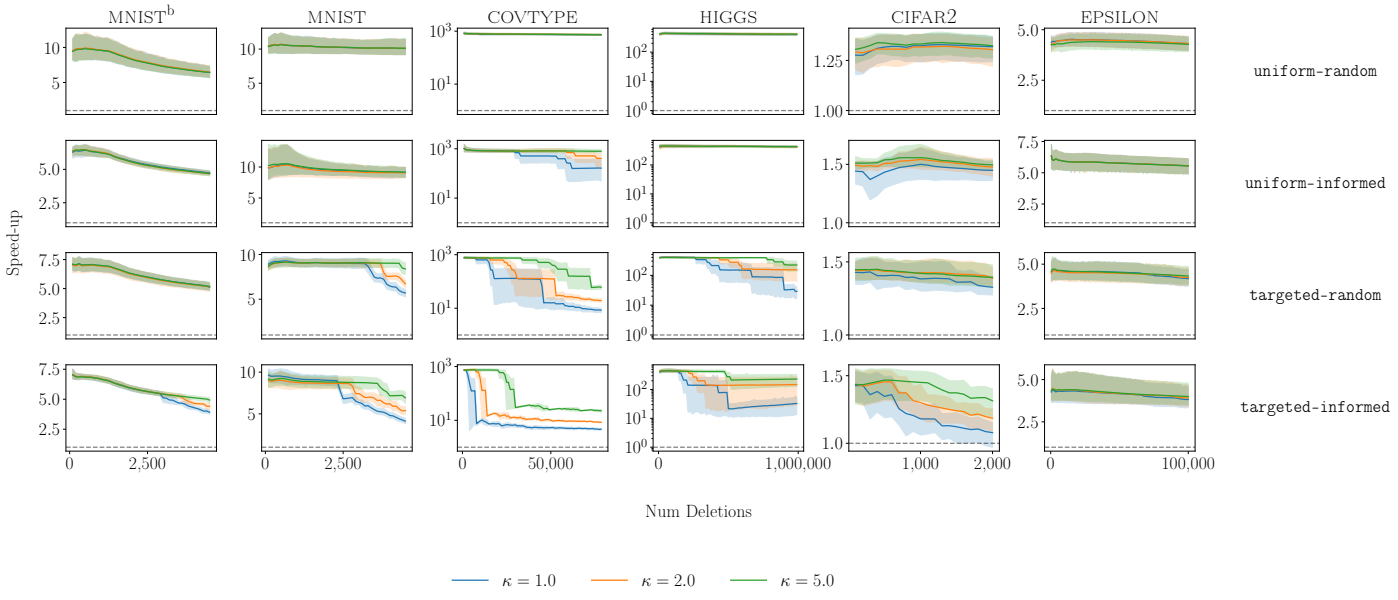


Figure A5. Speed-up of `AccDis` strategy at different thresholds κ. Rows correspond to different deletion distributions. Dashed line indicates speed-up of 1× . Note that for HIGGS, $\kappa \in 10, 20, 50$.

Table A7. Pooled mean of $EstErr_\kappa$% of the `AccErr` strategy for each threshold κ. Noise parameter $\sigma = 1$.

Dataset	κ	Uniform Random	Uniform Informed	Targeted Random	Targeted Informed
MNIST[b]	0.25	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	1.55 ± 0.8
	0.50	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.01 ± 0.0
	1.00	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0
MNIST	0.25	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0
	0.50	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0
	1.00	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0
COVTYPE	0.25	1.65 ± 1.2	0.30 ± 0.4	4.20 ± 1.5	0.60 ± 0.7
	0.50	0.34 ± 0.3	0.00 ± 0.0	0.30 ± 0.1	0.06 ± 0.1
	1.00	0.02 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0
HIGGS	0.25	104.50 ± 12.2	62.26 ± 10.9	97.66 ± 7.2	34.24 ± 4.9
	0.50	38.58 ± 5.2	18.90 ± 4.1	40.46 ± 4.2	11.74 ± 1.8
	1.00	11.98 ± 2.2	4.57 ± 0.7	10.40 ± 0.9	1.74 ± 0.6
CIFAR2	0.25	1.03 ± 1.0	5.44 ± 6.3	6.03 ± 2.4	9.78 ± 5.8
	0.50	0.00 ± 0.0	0.75 ± 1.5	0.30 ± 0.6	0.95 ± 1.0
	1.00	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0	0.00 ± 0.0
EPSILON	0.25	2.64 ± 1.0	3.26 ± 1.0	20.07 ± 2.6	20.61 ± 2.3
	0.50	0.29 ± 0.2	0.57 ± 0.2	4.47 ± 0.6	3.68 ± 0.6
	1.00	0.00 ± 0.0	0.06 ± 0.1	0.70 ± 0.3	0.31 ± 0.1

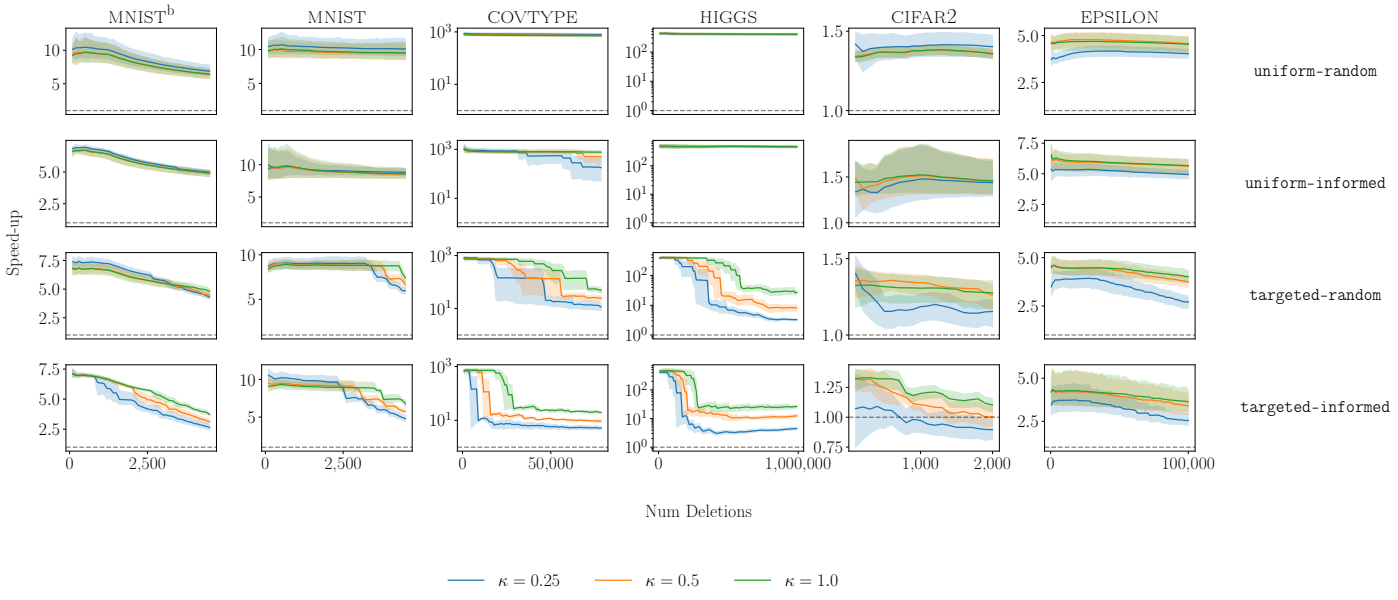


Figure A6. Speed-up of `AccErr` strategy at different thresholds κ. Rows correspond to different deletion distributions. Dashed line indicates speed-up of 1×.

References

1. Mantelero, A. The EU Proposal for a General Data Protection Regulation and the roots of the "right to be forgotten". *Comput. Law Secur. Rev.* **2013**, *29*, 229–235. [CrossRef]
2. Council of European Union. 2018 Reform of EU Data Protection Rules. Available online: https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf (accessed on 26 May 2022).
3. Golatkar, A.; Achille, A.; Soatto, S. Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020.
4. Guo, C.; Goldstein, T.; Hannun, A.; Van Der Maaten, L. Certified Data Removal from Machine Learning Models. In Proceedings of the 37th International Conference on Machine Learning, Virtual Event, 13–18 July 2020; Volume 119, pp. 3832–3842.

5. Wu, Y.; Dobriban, E.; Davidson, S. DeltaGrad: Rapid retraining of machine learning models. In Proceedings of the 37th International Conference on MachineLearning, Virtual Event, 13–18 July 2020; Volume 119, pp. 10355–10366.
6. Tsai, C.H.; Lin, C.Y.; Lin, C.J. Incremental and Decremental Training for Linear Classification. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 343–352. [CrossRef]
7. Cauwenberghs, G.; Poggio, T. Incremental and Decremental Support Vector Machine Learning. In Proceedings of the 13th International Conference on Neural Information Processing Systems, Virtual, 6–14 December 2021; MIT Press: Cambridge, MA, USA, 2000; pp. 388–394.
8. Karasuyama, M.; Takeuchi, I. Multiple Incremental Decremental Learning of Support Vector Machines. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 7–12 December 2009; Volume 22.
9. Schelter, S. "Amnesia"—Towards Machine Learning Models That Can Forget User Data Very Fast. In Proceedings of the Conference on Innovative Data Systems Research (CIDR), Amsterdam, The Netherlands, 12–15 January 2020; p. 4.
10. Cao, Y.; Yang, J. Towards Making Systems Forget with Machine Unlearning. In Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 17–21 May 2015; pp. 463–480. [CrossRef]
11. Bourtoule, L.; Chandrasekaran, V.; Choquette-Choo, C.A.; Jia, H.; Travers, A.; Zhang, B.; Lie, D.; Papernot, N. Machine Unlearning. In Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 24–27 May 2021; pp. 141–159. [CrossRef]
12. Golatkar, A.; Achille, A.; Soatto, S. Forgetting Outside the Box: Scrubbing Deep Networks of Information Accessible from Input-Output Observations. In Proceedings of the Computer Vision—ECCV 2020, Glasgow, UK, 23–28 August 2020; Vedaldi, A., Bischof, H., Brox, T., Frahm, J.M., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 383–398.
13. Golatkar, A.; Achille, A.; Ravichandran, A.; Polito, M.; Soatto, S. Mixed-Privacy Forgetting in Deep Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, 20–25 June 2021; pp. 792–801.
14. Martens, J. New Insights and Perspectives on the Natural Gradient Method. *J. Mach. Learn. Res.* **2020**, *21*, 1–76.
15. Izzo, Z.; Anne Smart, M.; Chaudhuri, K.; Zou, J. Approximate Data Deletion from Machine Learning Models. In Proceedings of the 24th International Conference on Artificial Intelligence and Statistics, Virtual, 13–15 April 2021; Volume 130, pp. 2008–2016.
16. Chaudhuri, K.; Monteleoni, C. Privacy-preserving logistic regression. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 2–12 December 2009; Volume 21.
17. Koh, P.W.; Liang, P. Understanding Black-box Predictions via Influence Functions. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017 ; Volume 70, pp. 1885–1894.
18. Wu, Y.; Tannen, V.; Davidson, S.B. PrIU: A Provenance-Based Approach for Incrementally Updating Regression Models. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, Portland, OR, USA, 14–19 June 2020; ACM: Portland, OR, USA, 2020; pp. 447–462. [CrossRef]
19. Neel, S.; Roth, A.; Sharifi-Malvajerdi, S. Descent-to-Delete: Gradient-Based Methods for Machine Unlearning. In Proceedings of the 32nd International Conference on Algorithmic Learning Theory, Paris, France, 16–19 March 2021; Volume 132, pp. 931–962.
20. Graves, L.; Nagisetty, V.; Ganesh, V. Amnesiac Machine Learning. *Proc. AAAI Conf. Artif. Intell.* **2021**, *35*, 11516–11524.
21. Brophy, J.; Lowd, D. Machine Unlearning for Random Forests. In Proceedings of the 38th International Conference on Machine Learning, Online, 18–24 July 2021; Volume 139, pp. 1092–1104.
22. Nguyen, Q.P.; Low, B.K.H.; Jaillet, P. Variational Bayesian Unlearning. In Proceedings of the Advances in Neural Information Processing Systems, Virtual Only Conference, 6–12 December 2020; Volume 33, pp. 16025–16036.
23. Dwork, C.; Roth, A. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* **2014**, *9*, 211–407. [CrossRef]
24. Boyd, S.; Boyd, S.P.; Vandenberghe, L. *Convex Optimization*; Cambridge University Press: Cambridge, UK, 2004.
25. Byrd, R.H.; Lu, P.; Nocedal, J.; Zhu, C. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.* **1995**, *16*, 1190–1208. [CrossRef]
26. Byrd, R.H.; Nocedal, J.; Schnabel, R.B. Representations of quasi-Newton matrices and their use in limited memory methods. *Math. Program.* **1994**, *63*, 129–156. [CrossRef]
27. Mokhtari, A.; Ribeiro, A. Global convergence of online limited memory BFGS. *J. Mach. Learn. Res.* **2015**, *16*, 3151–3181.
28. Chang, C.C.; Lin, C.J. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.* **2011**, *2*, 1–27. [CrossRef]
29. LeCun, Y.; Cortes, C. MNIST Handwritten Digit Database 2010. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 26 May 2022).
30. Krizhevsky, A.; Nair, V.; Hinton, G. Learning Multiple Layers of Features from Tiny Images. Available online: citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.9220&rep=rep1&type=pdf (accessed on 26 May 2022).
31. Collobert, R.; Bengio, S.; Bengio, Y. A Parallel Mixture of SVMs for Very Large Scale Problems. *Neural Comput.* **2002**, *14*, 1105–1114. [CrossRef] [PubMed]
32. Baldi, P.; Sadowski, P.; Whiteson, D. Searching for exotic particles in high-energy physics with deep learning. *Nat. Commun.* **2014**, *5*, 1–9. [CrossRef] [PubMed]
33. Yuan, G.X.; Ho, C.H.; Lin, C.J. An Improved GLMNET for L1-regularized Logistic Regression. *J. Mach. Learn. Res.* **2012**, 13, 1999–2030.

34. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, Canada, 8–14 December 2019; Volume 32.
35. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
36. Freedman, D.; Pisani, R.; Purves, R. *Statistics (International Student Edition)*, 4th ed.; WW Norton & Company: New York, NY, USA, 2007.
37. Tarun, A.K.; Chundawat, V.S.; Mandal, M.; Kankanhalli, M. Fast Yet Effective Machine Unlearning. *arXiv* **2021**, arXiv:2111.08947.
38. Baumhauer, T.; Schöttle, P.; Zeppelzauer, M. Machine unlearning: Linear filtration for logit-based classifiers. *arXiv* **2020**, arXiv:2002.02730.
39. Felps, D.L.; Schwickerath, A.D.; Williams, J.D.; Vuong, T.N.; Briggs, A.; Hunt, M.; Sakmar, E.; Saranchak, D.D.; Shumaker, T. Class Clown: Data Redaction in Machine Unlearning at Enterprise Scale. *arXiv* **2020**, arXiv:2012.04699.