Certifiable Machine Unlearning for Linear Models

[Experiment, Analysis & Benchmark Papers]

Ananth Mahadevan
University of Helsinki
Helsinki, Finland
ananth.mahadevan@helsinki.fi

Michael Mathioudakis
University of Helsinki
Helsinki, Finland
michael.mathioudakis@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 computation 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 linear models and demonstrate the trade-offs between efficiency, effectiveness and certifiability offered by each method. In implementing the study, we extend some of the existing works and describe a common ML 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 retrain of the ML model.

PVldb Reference Format:
Ananth Mahadevan and Michael Mathioudakis. Certifiable Machine Unlearning for Linear Models. PVLDB, 15(1): XXX-XXX, 2021.
doi:XX.XX/XXX.XXX

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 different 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 [21] or the right of erasure [25] 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 model should be found to be a close approximation of the model that is obtained with a brute-force, full retrain on the remaining data – even if such a full retrain 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: that’s because it takes time to train a model with a desired level of accuracy. Moreover, there is a trade-off between certifiability and effectiveness: 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 computational costs for different operations, they offer better or worse trade-offs in different settings.

Forexample, consider an online store that maintains a 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 the 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 retrain of the ML model; but at the same time, data deletions should not be ignored to such extent that the ML model does not reflect the data anymore.

In this work, we perform a comparative analysis of methods for machine unlearning, motivated both by the importance of the task and the lack of a comprehensive comparison in the literature. Our goal is to explore how existing methods perform 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 the 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 [21] or the right of erasure [25] 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 model should be found to be a close approximation of the model that is obtained with a brute-force, full retrain on the remaining data – even if such a full retrain 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: that’s because it takes time to train a model with a desired level of accuracy. Moreover, there is a trade-off between certifiability and effectiveness: 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 computational costs for different operations, they offer better or worse trade-offs in different settings.

For all the methods we compare, we consider the following common ML pipeline (Figure 1). In the first stage, the application trains an initial ML model from the data. To limit the variable parts of our experimentation, we will be focusing on linear classification models, such as logistic regression, as they represent a large class of models that are commonly encountered in a wide range of settings. In addition, we’ll 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. In the second stage, the initial ML model is employed for inference, i.e., it is used 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 application evaluates the updated model 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 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 according to some measure for model disparity. If the disparity between the two models is smaller than a given threshold, then the audit is successful, meaning the pipeline and the employed model has certifiably unlearned the deleted data so far and the pipeline is allowed to resume.

In the context of this pipeline, we evaluate three methods that follow largely different approaches for machine unlearning and which represent the state of the art for the setting we consider (linear classification models trained with SGD). The first method (Influence) updates the initial ML model using the deleted data to perform a corrective Newton step; it is defined in Guo et al. [14]. The second method (Fisher) updates the initial ML model using the remaining data to perform a corrective Newton step; it is defined in Golatkar et al. [11]. And the third method (DeltaGrad) updates the initial ML model by correcting the SGD steps that led to the initial model; it follows the method defined in Wu et al. [31] – which, however, we extend in this paper with a mechanism to trade effectiveness for certifiability. Each of the evaluated methods is approximate and equipped with mechanisms that control the trade-offs between efficiency, effectiveness and certifiability. For a more elaborate discussion of the literature, see Section 2.

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 DeltaGrad offers stable albeit lower performance across all qualities; Fisher offers overall best certifiability, along with good effectiveness at lower efficiency than Influence especially for larger datasets; and the Influence method offers overall best efficiency, along with good effectiveness at lower levels of certifiability. 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) an optimal point at which the pipeline should restart. Intuitively, the optimal point is just before the model update would lead to a model that fails the certification audit.

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 Wu et al. [31] and Golatkar et al. [11] with mechanisms to control performance trade-offs.
- We offer the first experimental comparison of the competing methods in a large variety of settings.
- We obtain novel empirical insights about the effect of the deletion distribution on the performance trade-offs.
- We obtain novel empirical insights about the strengths of each method in terms of the performance trade-offs.
- We propose a practical online strategy to determine an optimal time when to restart the training pipeline.

As 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 [16, 27, 29], collaborative filtering, naïve bayes [5, 28] k-nearest neighbors and ridge regression [28] possess exact unlearning methods. The efficiency for such exact methods varies.

For stochastic training algorithms such as SGD, Bourtoule et al. [2] propose an exact unlearning approach, under the assumption that learning is performed in federated fashion. 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 on 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 the relaxation of the effectiveness and certifiability requirements. Most of them can be categorized into one of three groups.

The first group [7, 14, 15] uses the deleted data to update ML models during unlearning. They perform a Newton step [18] to approximate the influence of the deleted data on the ML model and remove it. To trade-off certifiability for effectiveness, they inject random noise to the training objective function [7]. The second group [10–12] uses the remaining data of the training dataset to update the ML model and control certifiability. These methods use Fisher information [22] to retrain information of the remaining data and inject optimal noise in order to forget the deleted data. The third group [13, 23, 31, 32] stores data and information during training and then utilize 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 the optimization process, 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 Brophy and Lowd [4] for random forest models, or for Bayesian modeling, such as Nguyen et al. [24] 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 like SGD. We implement three methods, INFLUENCE, FISHER and DELTA GRAD which correspond to state-of-the-art unlearning methods from each of the aforementioned groups, Guo et al. [14], Goltakar et al. [11] and Wu et al. [31] respectively.

3 MACHINE UNLEARNING

In this section, we present the common ML pipeline over which the unlearning methods are evaluated in this study (Section 3.1). Subsequently, we describe the three unlearning methods (Sections 3.2-3.4). Any of them can be used in the ML pipeline to update the ML model at the event of data deletion. Lastly, we discuss the process of auditing the ML pipeline for certifiability (Section 3.5).

3.1 The ML pipeline

The ML pipeline describes the lifecycle of ML models in our experimental framework – i.e., how a model is trained, employed for inference, updated incrementally, and potentially fully retrained from scratch, while a series of data deletions occur (see Figure 1).

**First stage: Training** In this stage a ML model is learned from the training dataset \( \mathcal{D} \). In what follows, we’ll assume that the ML model is logistic regression, a simple and widely used model for classification. Each data point consists of \( d \) features \( x \in \mathbb{R}^d \) and a categorical label \( y \) – and we assume that initially there are \( n_{\text{init}} \) entries in total, i.e., \( \mathcal{D} = D_{\text{init}} = \{(x_i, y_i)\}_{i=1}^{n_{\text{init}}} \). At any time in the lifecycle of the pipeline, \( \mathcal{D} \) will denote the currently available training dataset, which is a subset of initial training data due to possible deletions, i.e., \( \mathcal{D} \subseteq D_{\text{init}} \). An objective function measures the fitness of a ML model’s parameters \( w \) on a dataset \( \mathcal{D} \). Following common practice for logistic regression, the objective function is

\[
L = L(w; \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \ell(w^T x_i, y_i) + \frac{\lambda}{2} \|w\|^2,
\]

where the first term captures the average classification loss, with \( \ell \) being the binary cross entropy for logistic regression; and the second term quantifies ridge regularization for a fixed value of parameter \( \lambda \), the role of which is to prevent over-fitting.

Moreover, for training, we’ll be using mini-batch SGD, a general and widely-used optimization algorithm [3]. SGD iteratively minimizes the objective function over the training data: first, it initializes the model parameters to a random value \( w := w_0 \), and it improves them in iterative steps as follows,

\[
w_{t+1} = w_t - \eta_t \nabla L(w_t; \mathcal{D})
\]

where \( \eta_t \) is the learning rate at iteration \( t \). An appropriate number of iterations is taken for convergence, after which the resulting ML model minimizes the objective function \( L \) on the dataset \( \mathcal{D} \).

Following common practice, SGD is executed in mini-batch fashion, i.e., only a subset of \( \mathcal{D} \) is used in each execution of Equation (2).

As we’ll see in the upcoming sections 3.2-3.4, each unlearning method uses an adaptation of the objective function of Eq. (1). A model obtained from the training stage of the ML pipeline is denoted\(^2\) with \( w^* \). When it is obtained using the initial dataset \( D_{\text{init}} \), then \( w^* \) is referred to as the initial trained model; and when it is obtained using a subset \( \mathcal{D} \subseteq D_{\text{init}} \) of the training dataset, then it is referred to as the fully-retrained model. This model, \( w := w^* \), is sent to the second stage to be employed for inference.

**Second stage: Inference** The available model \( w \) is employed for inference, i.e., to predict the class \( y \) of arbitrary data points \( x \) submitted as queries to the ML model. At any time during the second stage, a subset of the data may be deleted, which prompts the pipeline to proceed to the third stage.

**Third stage: Unlearning** The third stage receives the currently employed model \( w \) and the deleted subset of the training data, denoted with \( \mathcal{D}_m = \{(x_i, y_i)\}_{i=1}^m \). It executes the unlearning algorithm so as to “unlearn” the deleted data \( \mathcal{D}_m \). The result of the unlearning is an updated model \( w^m \).

\(^2\)For general models SGD requires a few re-runs to limit the possibility of returning a local optimum with low objective value. However for logistic regression, used in this paper, the objective function is convex and SGD leads to the global optimum.

\(^3\)Slightly abusing notation, we use \( w \) for both a model instance and its parameters.
Once obtained, the updated model $\mathbf{w}^*$ is evaluated on a test dataset, $\mathcal{D}_{\text{test}} = \{(x_i, y_i)\}_{i=1}^{n_{\text{test}}}$ in terms of effectiveness and certifiability. Following common practice, the test dataset $\mathcal{D}_{\text{test}}$ is disjoint from the training dataset $\mathcal{D}$ — and in real scenarios, $\mathcal{D}_{\text{test}}$ is typically independently collected, e.g., consisting of user queries (i.e., the data points $x$ for which the ML model is asked to predict $y$).

Effectiveness is measured as the model’s accuracy $\alpha$ on $\mathcal{D}_{\text{test}}$ (i.e., the fraction of test data that it classifies correctly). Furthermore, certifiability is measured as the disparity $\epsilon$ between the updated $\mathbf{w}^u$ and the fully-retrained $\mathbf{w}^*$ ML model, in terms of accuracy over the deleted data $\mathcal{D}_m$. Intuitively, the disparity $\epsilon$ captures the amount of information that the updated model $\mathbf{w}^u$ possesses about the deleted data $\mathcal{D}_m$ in comparison to a fully-retrained model $\mathbf{w}^*$: if the disparity is small, then the updated model has ‘unlearned’ the deleted data as well as a model that is retrained from scratch on the remaining data. Note however that, while $\mathbf{w}^u$ is readily available from the execution of the unlearning algorithm, the fully-retrained model $\mathbf{w}^*$ is not: in fact, obtaining $\mathbf{w}^*$ after every deletion would defeat the purpose of obtaining $\mathbf{w}^u$ in the first place. Therefore, unlike our experimental study, in a practical setting disparity $\epsilon$ could not be directly measured exactly but it should be estimated.

To deal with this challenge, we will experimentally show how to estimate $\epsilon$ using $\alpha$ which is easily obtainable (see section 5).

After evaluation on the test dataset, a decision is made about the updated model. If its accuracy $\alpha$ is sufficiently high and disparity $\epsilon$ sufficiently low relative to fixed thresholds determined externally (e.g., by the engineer who deploys this pipeline), then the pipeline returns to the second stage and employs the updated model $\mathbf{w}^u$ for inference — otherwise, the pipeline returns to the first stage for a full retrain over the remaining data $\mathcal{D} \setminus \mathcal{D}_m$. Intuitively, the full retrain is triggered once a large volume of deletions lead to an updated model with degraded effectiveness or certifiability.

Controlling the trade-offs Each unlearning method is equipped with mechanisms to navigate trade-offs between efficiency, effectiveness, and certifiability. The first mechanism trades efficiency, on one hand, for effectiveness and certifiability, on the other, and is controlled via noise injection, and specifically via a noise parameter $\sigma$ that determines the amount of injected noise. Simply expressed, noise injection deliberately adds randomness to an ML model, both during training and unlearning. On one end of this trade-off, when large amounts of noise are injected, the predictions of the ML model are effectively random — therefore ensuring low disparity $\epsilon$ and high certifiability, but at the cost of low effectiveness, as the noise leads to all data to be unlearned. On the other end, when no noise is injected, the unlearning method strives to optimize the objective function over the remaining data, thus prioritising effectiveness.

We note that the trade-off between effectiveness and certifiability, along with noise injection as a control mechanism, have already been introduced as concepts in the literature. For further discussion on noise injection, we refer the interested reader to [7, 11, 14].

Having defined the ML pipeline, we now proceed to specify the unlearning methods. For each unlearning method, we describe its three main components, namely the training algorithm (used to train a model on a dataset), the unlearning algorithm (used for incremental model updates after deletion), and the parameters that control trade-offs between efficiency, effectiveness and certifiability.

### 3.2 Fisher Unlearning Method

The Fisher unlearning method is described in Golatkar et al. [11]. The training algorithm for this method proceeds in two steps: in the first step, it invokes SGD to optimize the objective $L$ (Eq. 1); and in the second step it performs noise injection. The output model $\mathbf{w}^*$ is expressed as

$$\mathbf{w}^* := \mathbf{w}^{opt} + \sigma F^{-1/4} \mathbf{b}, \quad (3)$$

where,

$$\mathbf{w}^{opt} = \arg \min_{\mathbf{w}} L(\mathbf{w}, \mathcal{D}), \quad (4)$$

$$F = \nabla^2 L(\mathbf{w}^{opt}, \mathcal{D}), \quad (5)$$

$$\mathbf{b} \sim \mathcal{N}(0, 1)^d. \quad (6)$$

As shown in Equation (4), $\mathbf{w}^{opt}$ 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 (5). The second term in Equation (3) corresponds to the noise injection that adds standard normal noise (see Eq. (6)) to the optimal model $\mathbf{w}^{opt}$ in the direction of the Fisher matrix.

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

$$\mathbf{w}^u := \mathbf{w} - F^{-1} \Delta + \sigma F^{-1/4} \mathbf{b}, \quad (7)$$

where

$$\Delta = \nabla L(\mathbf{w}, \mathcal{D} \setminus \mathcal{D}_m), \quad (8)$$

$$F = \nabla^2 L(\mathbf{w}, \mathcal{D} \setminus \mathcal{D}_m). \quad (9)$$

and $\mathbf{b}$ is the same as in Equation (6). As shown in Equation (8), $\Delta$ is the gradient of the objective function $L$ (Eq. 1). And, similar to Equation (3), $F$ is the Fisher matrix, now computed on the remaining training data after deletion ($\mathcal{D} \setminus \mathcal{D}_m$). The first term in Equation (7) corresponds to the corrective Newton step that aims to unlearn the
deleted data $D_m$. The second term corresponds to noise injection, and adds standard normal noise $b$ (see Eq. (6)) to the updated model $w'$ in the direction of the Fisher matrix (see Equation (9)).

As defined in Equation (7), 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 (7) 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’ll be using this mini-batch version of the unlearning algorithm, as shown in Algorithm 1.

Algorithm 1: Fisher mini-batch

Input : Employed model $w$. Current training data $D$. Deleted data $D_m$. Parameter $\sigma$, Mini-batch size $m'$, objective function $L$

Output: Updated model parameters $w'$

1. $s \leftarrow \frac{m}{m'}$; Split $D_m$ into $s$ mini-batches $\{D_m^1, D_m^2, \ldots, D_m^s\}$
2. $D' \leftarrow D$; $w' \leftarrow w$
3. for $i = 1; t \leq s; i + + do$
4. $D' \leftarrow D' \setminus D_m^i$; $\Delta \leftarrow \nabla L(w', D')$; $F \leftarrow \nabla^2 L(w', D')$
5. $w'^n \leftarrow w^n - F^{-1} \Delta$
6. if $\sigma > 0$ then
7. Sample $b \sim N(0, 1)^d$
8. $w' \leftarrow w'^n + \sigma F^{-1/2} b$
9. end
10. end
11. return $w'$

Trade-off parameters As explained earlier (Section 3.1), the noise parameter $\sigma$ controls the trade-off between effectiveness and certifiability. Moreover, the size of the mini-batch $\tau_{Fisher} = m'$ 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 single and crude corrective Newton step. In typical real settings, one would choose a value $m'$ between the two extremes.

3.3 DeltaGrad Unlearning Method

The DeltaGrad unlearning method is described in Wu et al. [31]. 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.

The training algorithm uses SGD followed by noise injection,

$$w^* := \arg \min_{w} L(w, D) + \sigma \cdot b.$$

(10)

where $b$ is defined as in Eq. (6)). This noise injection mechanism is a Gaussian version of the noise injection mechanism described by Wu et al. [32] using results from Dwork et al. [9]. In contrast to Fisher method’s noise injection (Eq. 7), there is no Fisher matrix to guide the random Gaussian noise in this mechanism. Therefore, a large value of $\sigma$ 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 described in Equation (2), the parameters $w_t$ and objective function gradients $\nabla L(w_t, D)$ are stored to disk.

The unlearning algorithm for this method proceeds in two steps: in the first step, it approximately updates the stored sequence $(w_t)$ of parameters computed by SGD; in the second step, it injects noise. In summary, and slightly abusing notation, we write

$$w^t := \text{DGApprox}(w_t) + \sigma \cdot b.$$

(11)

The first term corresponds to the approximate update of SGD steps, and the second to noise injection, with $b$ defined as in Eq. (6).

Let us provide more details about how the first term is computed. Upon the deletion of the current subset of the training data $D_m \subset D$, the unlearning algorithm aims to obtain, approximately, the ML model that would have resulted from SGD if $D_m$ had never been used for training. By definition (Eq. 2), in the absence of $D_m$, the SGD steps would have been:

$$w_{T+1} = w_T - \frac{\eta_t}{(n-m)} [n \nabla L(w_T, D) - m \nabla L(w_T, D_m)]$$

(12)

leading to a different sequence $(w_T)$ of model parameters than the one obtained before deletion from Eq. (2). As a consequence, the value of $\nabla L(w_T, D)$ differs between the executions of Eq. (2) (before deletion) and Eq. (12) (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.

Algorithm 2: DGApprox

Input : Current training data $D$. Deleted data $D_m$, model weights saved during the training stage or updated later $\{w_0, w_1, \ldots, w_T\}$ and corresponding gradients $\{\nabla L(w_0, D), \nabla L(w_1, D), \ldots, \nabla L(w_T, D)\}$, period $T_0$, total iteration number $T$, “burn-in” iteration number $j_0$, learning rate $\eta_t$

Output: Updated model parameter $w_T$

1. Initialize $w_0 \leftarrow w_0$
2. for $t = 0; T < t + + do$
3. if $\left\lfloor (t - j_0) \mod T_0 \right\rfloor = 0 \text{ or } t \leq j_0$ then
4. compute $\nabla L(w_T, D \setminus D_m)$ exactly
5. compute $w_{T+1}$ by using exact update (Equation (2))
6. Update $w_T$ with $w_T$
7. Update $\nabla L(w_T, D)$ with $\nabla L(w_T, D \setminus D_m)$
else
8. Approximate $\nabla L(w_T, D \setminus D_m)$ with L-BFGS algorithm using stored terms $w_T$ and $\nabla L(w_T, D)$
9. Compute $\nabla L(w_T, D \setminus D_m)$
10. Compute $w_{T+1}$ by using the modified gradient formula (Equation (12))
11. Update $w_T$ with $w_T$
12. Update $\nabla L(w_T, D)$ with approximated $\nabla L(w_T, D \setminus D_m)$
13. end
14. end
15. return $w_T$
The unlearning algorithm is shown for reference in Algorithm 2. As seen in lines 9 to 13, the term $\nabla L(w; D)$ is approximated using the Quasi-Newton L-BFGS optimization algorithm with the terms $w_t$ and $\nabla L(w_t, D)$ that 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, $\nabla L(w_t, D \setminus D_m)$, 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 in lines 3 to 7. Moreover, in order to use the above DeltaGrad algorithm for subsequent data deletions, the terms $w_t$ and $\nabla L(w_t, D)$ that were previously stored in disk are updating after unlearning the deleted data $D_m$. This is described in lines 6 and 7 and lines 12 and 13.

**Trade-off parameters** The unlearning algorithm has several parameters that control its efficiency. The burn-in period $j_0$, periodicity $T_0$, learning rate $\eta$, number of SGD iteration $T$ and the length of historical computations for L-BFGS optimization algorithm are all potential $\tau$ parameters for the DeltaGrad algorithm. Due to paper-space constraints, we choose the periodicity $\Delta_{\text{DeltaGrad}} = T_0$ as the primary efficiency parameter, while keeping all other secondary parameters fixed for a given dataset. We use $T_0 = 2$ as its lower value, which corresponds to computing the exact gradient every alternate iteration, leading to minimum efficiency. Conversely, large values of $T_0$ lead to higher efficiency. Finally, the noise parameter $\sigma$ controls the trade-off between effectiveness and certifiability.

### 3.4 INFLUENCE Unlearning Method

The INFLUENCE unlearning method follows Guo et al. [14]. Its approach is based on ML influence theory [18]. 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:

$$L_{\sigma}(w; D) = L(w, D) + \frac{\sigma b^\top w}{|D|}$$ \quad (13)

where $L$ and $b$ are the same as in Equations (1) and (6) respectively. The second term in Equation (13) describes the noise injection where $\sigma$ is the noise parameter. The amount of noise is scaled wrt the size of the training data $D$.

The **training algorithm** uses SGD to optimize the noisy objective.

$$w^* := \arg \min_w L_{\sigma}(w, D) \quad (14)$$

Note that, when $\sigma$ is increased, the effectiveness of the ML model decreases as the SGD algorithm prioritizes minimizing the second term in Equation (13) rather than the original objective function captured by the first term.

The **unlearning algorithm** approximates the influence of the deleted subset $D_m \subseteq D$ on the parameters of the currently employed model $w$ and performs the update as:

$$w^u := w + H^{-1} \Delta^{(m)} \quad (15)$$

where

$$\Delta^{(m)} = \nabla L(w, D_m), \quad (16)$$

$$H = \nabla^2 L(w, D \setminus D_m). \quad (17)$$

As seen in Eqs. (16) and (17), $\Delta^{(m)}$ is the gradient of the objective function $L$ (see (1)) computed on the deleted data and $H$ is the Hessian matrix computed on the remaining training data. The second term in Equation (15) is known as the **influence function** of the deleted data $D_m$ on the model parameters $w$.

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’ll be using this mini-batch version of the unlearning algorithm, as shown in Algorithm 3.

### Algorithm 3: INFLUENCE mini-batch

| Input: Employed model $w$. Current training data $D$. Deleted data $D_m$. Mini-batch size $m'$. Objective function $L$. |
| Output: Updated model parameter $w^u$. |

1. $s \leftarrow \left[ \frac{m}{m'} \right]$; Split $D_m$ into $s$ mini-batches $\{D^1_m, D^2_m, \ldots, D^s_m\}$
2. $D' \leftarrow D$; $w^u \leftarrow w$
3. for $i = 1; i <= s; i + +$
4.   $D' \leftarrow D' \setminus D^i_m$
5.   $\Delta^{(m')} \leftarrow \nabla L(w^u, D^i_m)$; $H \leftarrow \nabla^2 L(w^u, D')$
6.   $w^u \leftarrow w^u + H^{-1} \Delta^{(m')}$
7. end
8. return $w^u$

The trade-off parameters are similar to those in the Fisher method. The size of $\Delta_{\text{INFLUENCE}} = m'$ serves as the efficiency parameter and $\sigma$ as the noise parameter.

### 3.5 Auditing

During an audit, the auditor first obtains a fully-retrained model $w^*$ using the training algorithm of the corresponding unlearning method (see Eqs. (3), (10) and (14)) on the remaining training data $D \setminus D_m$. Next, the auditor measures the disparity $\varepsilon$ between $w^*$ and the currently employed model $w$. If the measured disparity does not exceed a given threshold, then ML pipeline passes the audit and is allowed to resume. Otherwise, the ML pipeline does not satisfy the certifiability claimed and therefore fails the audit. Such failed certifiability audits may result in fines or other regulatory issues. Therefore, it is in the best interest of the deployer of the ML pipeline to correctly state the certifiability requirements and ensure that the pipeline is able to pass an audit after any number of deletions.

### 4 EXPERIMENTAL SETUP

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

#### 4.1 Datasets

We perform experiments over six datasets, retrieved from the public LIBSVM repository [6]. The datasets cover a large range of size and dimensionality, as summarized shown in Table 1, allowing us
to effectively explore the trends and trade-offs of the unlearning methods. In addition, to have a uniform experimental setting with comparable results, we focus on the task of binary classification. Towards this end, most datasets were chosen to include 2 predictive classes (or if the original dataset contained more classes, the experiments focused on two of them, as reported in Table 1). Nevertheless, we also include one multi-class dataset (MNIST, with 10 classes).

In more detail, MNIST [20] 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 training and test data. CIFAR2 consists of 32×32 RGB colour images, with each image labeled with a descriptive category. The dataset is the binary-class subset of the CIFAR-10 [19] dataset, as it contains only the “cat” and “ship” categories, with each category corresponding to one class. COVTYPE [8] consists of 54 cartographic features used to categorize forest cover types. We use the binary version from LibSVM. The HIGGS [1] dataset consists of kinematic features from Monte Carlo simulation of particle detectors for binary classification. Epsilon [30] is obtained from the PASCAL Large Scale Learning Challenge 2008.

### 4.2 ML Pipeline

We now provide implementation details for the ML pipeline (Figure 1). The pipeline is designed so that it is suitable to all the three chosen unlearning methods discussed in Section 3. The pipeline is implemented in Python 3.6 using PyTorch 1.8 [26]. All experiments are run on a machine with 24 CPU cores and 180 GB RAM. Our full code base is publicly available$^3$.

**Preprocessing.** The INFLUENCE unlearning method requires all data points $x_i$ of a dataset to have an Euclidean norm at most 1, i.e., $||x_i||_2 \leq 1$ (see Guo et al. [14]). To satisfy this requirement, we perform a max-$L_2$ normalization for all datasets as a pre-processing step, where we divide each data point with the largest $L_2$ norm of any data point in the dataset. This normalization does not affect the performance of other methods.

**Training.** As mentioned in Section 3, we use the mini-batch SGD algorithm for training. In all cases, we use fixed learning rate $\eta = 1$ and ridge regularization parameter $\lambda = 10^{-4}$. Moreover, since DELTAGRAD is designed to work only with standard SGD, we refrain from using momentum-based SGD algorithms such as Adam [17].

Note that both the FISHER and INFLUENCE unlearning algorithms require the Hessian matrix to be positive definite to compute the inverse (see Eqs. (7) and (15)). This is ensured by running the SGD algorithm for a sufficiently large number of iterations during training to achieve convergence. Towards this end, we use a small subset of the training data as validation dataset, to identify an optimal mini-batch size and total number of SGD iterations. Moreover, we control the data points selected in each mini-batch by fixing the random seed used to produce mini-batches in the SGD algorithm. This ensures reproducibility of the experiments across various unlearning algorithms. Finally, for multi-class classification with $k > 2$ classes on the MNIST dataset, we train $k$ independent binary logistic regression classifiers in a One vs Rest (OVR) fashion.

**Unlearning.** When a subset $D_\text{del}$ of the current training data $D$ is deleted, the unlearning algorithm of the employed method (FISHER, DELTAGRAD, or INFLUENCE) is invoked. We modify and extend the code provided in Guo et al. [14] to implement the INFLUENCE unlearning algorithm as described in Section 3.4. We further extend this code to also implement the mini-batch version of the FISHER unlearning algorithm as seen in Section 3.2. For the DELTAGRAD method, we use the code provided by the authors in Wu et al. [31] and modify it to add the noise injection mechanism described in Section 3.3 to trade-off effectiveness for certifiability.

### 4.3 Evaluation Metrics

In this section, we define the metrics we use to report the performance of different unlearning methods in terms of effectiveness, certifiability and efficiency. For uniformity of presentation, we’ll be reporting the performance achieved by a given model as relative to the performance of a baseline model. Towards this end, we’ll be using the Symmetric Absolute Percentage Error (SAPE) defined as

$$\text{SAPE}(a,b) = \frac{|b-a|}{b+|a|} \cdot 100\%.$$  \hspace{1cm} (18)

For the function to be continuous, we define $\text{SAPE}(0,0) = 0$.

**Effectiveness** is measured in terms of predictive accuracy, i.e., as the fraction of data points correctly classified by a given ML model on a particular dataset. We will write $\text{Acc}_{\text{test}}$ to denote the accuracy on the test dataset $D_{\text{test}}$ and $\text{Acc}_{\text{del}}$ to denote the accuracy on the deleted data $D_{\text{del}}$. Let $\text{Acc}_{\text{test}}^u$ be the accuracy of the updated model $w^u$ on the test dataset; and $\text{Acc}_{\text{del}}^u$ be the optimal accuracy that may be obtained via logistic regression on the same data (in other words, the latter is the test accuracy of the fully-trained model with $\sigma = 0$). We will report $\text{AccDrop}$ as the drop in test accuracy of the updated model $w^u$ compared to the optimal one, i.e.,

$$\text{AccDrop} = \text{SAPE}(\text{Acc}_{\text{test}}^u, \text{Acc}_{\text{test}}^u).$$  \hspace{1cm} (19)

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

**Certifiability** measures how close the updated model is relatively to a fully-retrained model by the same unlearning method. Specifically, let $\text{Acc}_{\text{del}}^u$ and $\text{Acc}_{\text{del}}^\star$ be the accuracy on the deleted data for the updated model and the fully retrained model, respectively, at the same value of the $\sigma$ parameter. We report $\text{AccDis}$ as the disparity in accuracy of the two models, i.e.,

$$\text{AccDis} = \text{SAPE}(\text{Acc}_{\text{del}}^u, \text{Acc}_{\text{del}}^\star).$$  \hspace{1cm} (20)

A lower value of $\text{AccDis}$ implies that the updated model $w^u$ has higher certifiability, i.e., the updated model is more similar to the

### Table 1: Datasets

| Dataset | dimensions | classes | $n_{\text{init}}$ | $n_{\text{test}}$ |
|---------|------------|---------|------------------|------------------|
| MNIST$^b$ | 784 | 2 | 11 982 | 1 984 |
| CIFAR2  | 3072 | 2 | 20 000 | 2 000 |
| MNIST   | 784 | 10 | 60 000 | 10 000 |
| COVTYPE | 54 | 2 | 522 910 | 58 102 |
| Epsilon | 2000 | 2 | 400 000 | 100 000 |
| HIGGS   | 28 | 2 | 9 900 000 | 1 100 000 |

$^3$URL: https://version.helsinki.fi/mahadeva/unlearning-experiments
Fully retrained model, which had never seen the deleted data. Note that the symmetry of SAPE is essential here, because both under- and over-performance of the updated model contributes towards disparity wrt the fully retrained model.

Efficiency is the relative speed-up in running time for obtaining the updated model \( \mathbf{w}^* \) compared to the fully retrained model and is defined as

\[
\text{speed-up} = \frac{\text{time taken to obtain } \mathbf{w}^*}{\text{time taken to obtain } \mathbf{w}^{\text{full-retrain}}}.
\]

A speed-up of 2x indicates that the unlearning method is able to produce and updated model twice as fast as it takes the training phase to produce a fully-retrained model.

4.4 Experimental Roadmap

In this subsection, we provide a brief overview of the experiments in the upcoming sections. The experiments analyze the stages of the ML pipeline presented in Section 3.1 in an incremental manner.

In Section 5, before we evaluate any unlearning methods, we explore how the quantity and quality of deleted data affect the accuracy of the fully-retrained model. Then, in Section 6, we demonstrate the trade-offs between efficiency, effectiveness and certifiability, for different values of the \( \tau \) and \( \sigma \) parameters for each unlearning method. Finally, in Section 7 we propose an evaluation strategy to decide whether the incrementally updated model produced by the unlearning method has diverged enough from the available data to warrant a full-retrain.

5 EFFECT OF DELETION DISTRIBUTION

Before we compare 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 implement a two-step process to generate different deletion distributions. The process is 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 1), 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 probability \( 1/k \); or targeted, where one class is randomly predetermined and subsequently always selected.

Once the class has been selected in the first step, a data point from that class is selected in the second step. The selection may be either random, where one data point is selected uniformly at random; or informed, where one point is selected so as to decrease the model’s accuracy the most. Ideally, for the informed selection, and for each data point, we would compute exactly the 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 would be extremely expensive computationally, even for experimental purposes. Instead, for the informed selection, we opt to heuristically select the outliers in the dataset, as quantified by the \( L_2 \) norm of each data point. This heuristic is inspired by Izzo et al. [15], who state that deleting data points with a large \( L_2 \) norm negatively affects the approximation of Hessian-based unlearning algorithms.

As described above, the two-step process yields four distinct deletion distributions namely uniform-random, targeted-random, uniform-informed and targeted-informed. In the experiments that follow, we select data to delete for different choices of deletion distribution and volume. For each set of deleted data, we report the accuracy of the fully trained model after deletion (this is the accuracy achieved by the model that optimizes Eq. (1) for \( \lambda = 10^{-4} \), 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 on the deleted data. Accuracy values correspond to the y-axis while the volume of deletion (as fraction of the original dataset size) to the x-axis. Different deletion distributions are indicated with different markers and color. The variance seen in Figure 2 is a consequence of the randomness in the selection of deleted points (2 random runs were performed).

There are three main takeaways from these results. First, uniform deletion distributions (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. In fact, we see that the accuracy decreases only after more than 90% of the data are deleted (see Fig. 7). This is due to the redundancy present in real-world datasets, in the sense that only a small number of data points from each class is sufficient to separate the classes as well as possible. And therefore, evaluating unlearning methods on deletions from uniform distributions will not offer significant insights on the effectiveness and efficiency trade-offs. Moreover, notice that the test accuracy for the uniform-informed distribution is lower than the uniform-random distribution, indicating that the informed deletions remove outlier data points that decrease the accuracy of the ML model.

Second, targeted deletion distributions (targeted-random and targeted-informed) provide a worst-case scenario of deletions that leads to large drops in test accuracy. This is because deleting data points from one targeted class eventually leads in class imbalance, and causes the ML model to be less effective in classifying data points from that class. In addition, we observe that the variance resulting from the selection of the deleted class is low in all datasets apart from higgs. We postulate this is because on the particular way that missing values have been treated for this dataset: data point 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. Next, we observe that the drop in accuracy is steeper for targeted-informed compared to targeted-random, indicating the deletion of informed points results in a less effective model at the same deletion fraction. This highlights the targeted-informed distribution as a worst-case deletion scenario: to validate their performance, machine unlearning methods should be tested on targeted-informed or similar distributions, where data deletions quickly affect the accuracy of the learned model.

Thirdly, we see across deletion fractions that the accuracy on the test and deleted dataset (\( \text{Acc}_{\text{test}} \) and \( \text{Acc}_{\text{del}} \), respectively) follow a similar trend (i.e., their values are highly correlated). Hence, the test accuracy \( \text{Acc}_{\text{test}} \), which can always be computed for a model on the test data, can be used as a good proxy for the \( \text{Acc}_{\text{del}} \) of a ML model, which may be impossible to compute after data deletion but
is required in order to assess certifiabilty. This observation will be useful to decide when to trigger a model retraining in the ML pipeline (Section 3.5).

Additionally, we note that the rate of drop in Acc_{test} and Acc_{del} with respect to deletion fraction varies from dataset to dataset.

Table 2: Volume of deleted data as a fraction of the initial data, corresponding to different drops in accuracy for fully-retrained model and targeted-informed deletion (Section 5, Figure 2).

| Dataset | Small 1% Drop | Medium 5% Drop | Large 10% Drop |
|---------|--------------|----------------|----------------|
| fraction | | | |
| m| n| s| t| c
| fraction | $|D_m|^{\frac{1}{3}}$ | fraction | $|D_m|^{\frac{5}{3}}$ | fraction | $|D_m|^{\frac{10}{3}}$ |
| MNIST$^b$ | 0.2 | 2396 | 0.3 | 3594 | 0.375 | 4493 |
| MNIST | 0.01 | 600 | 0.5 | 3000 | 0.075 | 6000 |
| COVTYPE | 0.05 | 26145 | 0.1 | 52291 | 0.15 | 78436 |
| HIGGS | 0.01 | 99000 | 0.5 | 49500 | 0.10 | 990000 |
| CIFAR2 | 0.05 | 500 | 0.125 | 1250 | 0.2 | 2000 |
| EPSILON | 0.1 | 4000 | 0.2 | 8000 | 0.25 | 10000 |

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, certifiability), for different values of their $\tau$ and $\sigma$ parameters. In what follows, for each dataset, we experiment with three volumes of deleted data points, small, medium, and large, measured as a fraction of the initial training data as shown in Table 2. The deletion volumes correspond different values of accuracy drop, as encountered in Section 5. Specifically, they correspond to a 1%, 5% and 10% drop in Acc_{test} for a fully-retrained model when using a targeted-informed deletion distribution with class 0 as the deleted class (see Figure 2).

6.1 Efficiency and Certifiability Trade-Off

In this experiment, we evaluate the trade-off between certifiability and efficiency of each unlearning method when the noise parameter $\sigma$ is kept constant and the efficiency parameter $\tau$ is varied. The $\tau$ parameter for the INFLUENCE and FISHER methods is the size of the unlearning mini-batch $m'$ and the range of the parameter is

$$\tau_{\text{FISHER}} = \tau_{\text{INFLUENCE}} = m' \in \left\{ \frac{m}{2}, \frac{m}{4}, \frac{m}{8} \right\},$$

where $m$ is the volume of deleted data. For the DELTAGRAD method the $\tau$ parameter is the periodicity $T_0$ of the unlearning algorithm, and its range is

$$\tau_{\text{DELTAGRAD}} = T_0 \in \{2, 5, 50, 100\}.$$

We obtain the updated model $w^u$ and the fully-retrained model $w^*$ for each unlearning method as described in Sections 3.2 to 3.4 at a fixed value of $\sigma$.

The results are shown in Figure 3. We report the results for two fixed values of the noise parameter and specific deletion fractions. First in Figure 3a, we fix $\sigma = 0$, and present results for all volumes of data deletion corresponding to each row in the figure. Second, in Figure 3b, we fix $\sigma = 1$ and present results corresponding to the largest deletion volume. For each plot in the figures, certifiability is shown on the y-axis using AccDis and efficiency is shown in the x-axis using speed-up. Different unlearning methods and $\tau$ values are indicated with different colors and markers respectively. The entire legend is shown in Figure 3. For extensive results covering different values of $\sigma$ and volumes of deletion, please refer to Appendix C and Table 6.

Noise parameter $\sigma = 0$ The results in Figure 3a display how the $\tau$ parameter trades-off efficiency for effectiveness in the absence of any noise injection. We observe two trends across all deletion fractions.

First, for datasets with low dimensionality, such as COVTYPE and HIGGS, the INFLUENCE and FISHER methods provide significant speed-up of nearly 200x and 50x respectively when performing bulk removals (i.e., $m' = m$), while the DELTAGRAD method provides $< 1x$ speed-up, i.e., requiring more time than the fully-retrained
model $w^\ast$. This is because for the influence and Fisher methods, the cost of computing the inverse Hessian matrix (see Eqs. (6) and (17)) is much lower when $d$ is small, compared the cost of approximating a large number of SGD iterations for the DeltaGrad method. Conversely, for datasets with higher dimensionality, such as cifar2 and epsilon, we see that the Influence and Fisher methods provide a smaller speed-up, even when bulk removals are performed (5x and 1.8x respectively) and when $\tau$ is decreased to $m' = m/8$, they offer an even smaller speed-up (1.03x and 0.35x respectively). Whereas, the DeltaGrad method provides comparable and better speed-up of 3x and 2.4x for cifar2 and epsilon respectively, at similar AccDis as compared to the other methods.

Second, as the volume of deletions increases along the rows, the range of AccDis increases as well. This is because as we delete more data points, the updated model $w^\ast$ diverges from the fully-retained model $w$ due to the approximations in the unlearning algorithm of each method. This is clearly seen at the highest values of the $\tau$ parameter ($m' = m$ and $T_0 = 100$), when the unlearning algorithm sacrifices the most certifiability, indicated by the largest disparity in accuracy, for the highest possible efficiency.

**Noise parameter $\sigma = 1$** From the results in Figure 3b, we see how the injection of in each unlearning method affects the trade-off between certifiability and efficiency.

First, we observe that the efficiency of the Fisher method has changed in comparison to the previous results. This is a consequence of the additional computations required in the Fisher unlearning algorithm when $\sigma > 0$ (see Algorithm line 6) for noise injection. This impact on the efficiency is most visible in the mnisth and cifar2 datasets at largest values of $\tau$ ($m' = m$), where the cost of noise injection leads to much lower speed-up (10x and 1.4x respectively). Nevertheless, we see that Fisher method offers the lowest possible AccDis of $\approx 0$% across all datasets at the lowest $\tau$ value of $m' = m/8$. However, the traded-off efficiency is high, especially for the high dimensional datasets such as covtype and higgs, where speed-up decreases closer to 1x. Whereas, for smaller dimensional datasets such as cifar2 and epsilon, the lowest possible speed-up ($\approx 27x$ and $\approx 9x$ respectively) is still acceptable.

Second, we observe that the efficiency of the Influence method has increased slightly across all datasets. This a because noise is injected to the objective function (see Eq. (13)) and is subsequently optimized using SGD (see Eq. (14)). As this is only performed in

![Figure 3: Efficiency-Certifiability trade-offs for (a) $\sigma = 0$ at all volumes and (b) $\sigma = 1$ at the largest volume of deletion](image-url)
the training stage of the pipeline, it takes longer to obtain the fully-retrained model, while the updated model is obtained in the same time, resulting in higher efficiency. However, as the injected noise is optimized during training, decreasing the \( \tau \) parameter in the unlearning algorithm, leads to smaller decreases in the accuracy disparity.

Lastly, we observe that for the \texttt{cifar2} and \texttt{epsilon} dataset, the efficiency of the \texttt{DeltaGrad} method is lower compared to when the noise parameter was set to zero, due to the cost of noise injection (see Eq. (11)). Furthermore, the overall certifiability has improved, as indicated by lower values of AccDis, due to the injected noise. However, we note that increasing the \( \tau \) parameter offers only minor increases in efficiency at the cost of larger accuracy disparity.

### 6.2 Efficiency and Effectiveness Trade-Off

In this experiment, we evaluate how varying the efficiency parameter \( \tau \) trades-off efficiency for effectiveness when the volume of data deleted and \( \sigma \) are kept constant. The range of \( \tau \) for each unlearning method is the same as in Section 6.1. In Figure 4, we discuss the results for \( \sigma = 1 \) and the largest volume of deletion for each dataset. For extensive results covering all different values of \( \sigma \) and volumes of deletion, please refer to Appendix C and Table 6. In each plot effectiveness and efficiency are reported using the drop in accuracy \( \text{AccDrop} \) and speed-up in running time respectively between an updated model and a fully-retrained model with no noise injected. From the results, we observe three main trends.

First, the \texttt{Fisher} method offers much lower efficiency for high dimensional datasets compared to Section 6.1, especially in the \texttt{cifar2} and \texttt{epsilon} datasets, where the speed-up is \( \leq 1 \times \) for nearly all choices of \( \tau \) parameter \( m' \). This is due to the reduced computational effort required to obtain the fully-retrained model \( w^* \) as no noise injection is done. Therefore, it takes longer to inject noise using the \texttt{Fisher} unlearning algorithm and obtain an updated model when compared to obtaining the fully-retrained model, especially when dimensionality \( d \) is high. Also, we see that at this level of \( \sigma \), the \texttt{Fisher} method does not reduce the accuracy drop significantly when \( \tau \) parameter is decreased.

Secondly, once again we see that the \texttt{DeltaGrad} method is mostly stable both in terms of efficiency and effectiveness as seen in Section 6.1. However, note that the overall accuracy drop is larger compared to the other methods due to the noise injection and hence offers a lower effectiveness even at \( \sigma = 1 \).

Lastly, we observe that the \texttt{Influence} offers the best efficiency and effectiveness trade-off among all the methods. Especially, for high dimensional datasets such as \texttt{cifar2} and \texttt{epsilon} and largest value of \( \tau \) parameter, the \texttt{Influence} method offers higher speed-ups of \( 20x \) and \( 2.5x \) respectively compared to the 0.4x and 1.3x speed-ups of the \texttt{Fisher} method at only a slightly larger accuracy drop (1.17% and 1.2% respectively compared to 1.16% and 0.3%). For small dimensional datasets such as \texttt{cotype} and \texttt{higgs}, the \texttt{Influence} method and \texttt{Fisher} method offer similar efficiency and effectiveness. Lastly, for the \texttt{mnist} and \texttt{mnist} datasets, at the largest value of the \( \tau \) parameter, \texttt{Influence} offers much higher speed-ups of \( 168x \) and \( 29x \) respectively compared to \( 9x \) and \( 8.5x \) of the \texttt{Fisher} method at lower accuracy drop.

### 6.3 Effectiveness and Certifiability Trade-Off

In this section, we first study the effect of the noise parameter \( \sigma \) on the effectiveness of a fully-trained ML model \( w^* \). Next, we evaluate how varying the noise parameter \( \sigma \) trades-off effectiveness for certifiability for each unlearning method when the volume of data deleted and the \( \tau \) parameter are kept constant.

**Effect of \( \sigma \) on \( w^* \)** To isolate the effect of data deletion and for simplicity, we select a deletion fraction of 0, i.e., no data is deleted. We vary \( \sigma \) from \( 10^{-2} \) till \( 10^2 \) and obtain a fully-trained model \( w^* \) using the training algorithm of each unlearning method (see Eqs. (3), (10) and (14)) on the initial dataset \( D_{\text{init}} \). Then, we compare this model with the optimal model (a fully-trained model with \( \sigma = 0 \)) on the initial dataset. The results are presented in Figure 6, where we report in the first row the effectiveness using accuracy drop \( \text{AccDis} \) and in the second row the \( L_2 \) distance between the two models. As no data is deleted, we cannot report the accuracy disparity, hence the \( L_2 \) distance acts a proxy for the disparity between the fully-trained model and the optimal model.

In Figure 6, we notice the difference in the extent of the accuracy drop and the \( L_2 \) distance as a consequence of the noise injection in each unlearning method at different values of \( \sigma \). First, the \texttt{Influence} method has the smallest \( \text{AccDrop} \) and \( L_2 \) distance from the optimal model – a direct consequence of noise injection in its objective function (see Eq. (13)) and the subsequent optimization by the SGD algorithm during the training stage. Second, the \texttt{DeltaGrad} method has the largest \( \text{AccDrop} \) and \( L_2 \) distance from the optimal model, even at small values of \( \sigma \), due to random noise directly injected into the model parameters (see Eq. (10)). Lastly, the \texttt{Fisher} method sits in-between the other methods, both in terms \( \text{AccDrop} \) and \( L_2 \) distance due to the injection of random noise in the direction of the Fisher matrix (see Eq. (3)). This lessens the impact on the model parameters in comparison to the \texttt{DeltaGrad} method, however, the impact is still greater compared to the optimized noise injection in the \texttt{Influence} method.

**Trade-off experiments** Here we only present the results for the largest deletion volume from Table 2. For more extensive experiments covering different \( \tau \) parameters and volumes of deletion, refer Appendix E and Table 7. The efficiency parameter \( \tau \) was set as follows: for \texttt{Influence} and \texttt{Fisher}, we set

\[
\tau_{\text{Influence}} = \tau_{\text{Fisher}} = m' = m, \tag{11}
\]

i.e., the size of the unlearning mini-batch was set to be equal to the volume of deleted data; and for \texttt{DeltaGrad}, we set

\[
\tau_{\text{DeltaGrad}} = \tau_0 := 100, \tag{12}
\]

i.e., the periodicity is set to 100 SGD steps.

For different values of the noise parameter \( \sigma \), we obtain the updated models \( w^u \) corresponding to each unlearning method as described in Sections 3.2 to 3.4. For baselines, first we obtain the fully-retrained model \( w^* \) at the same \( \sigma \) to measure certifiability and a second fully-retrained model \( w^* \) at \( \sigma = 0 \) to measure effectiveness, as per the certifiability and effectiveness measures defined in Section 4.3. The results are shown in Figure 5. For each plot in the figure, the certifiability is reported using \( \text{AccDis} \) on the left y-axis and the effectiveness is reported using \( \text{AccDrop} \) on the right y-axis, as the \( \sigma \) is varied from \( 10^{-2} \) till \( 10^2 \) for the different unlearning methods.
Figure 6: Effect of noise parameter $\sigma$ on the updated model $w^u$ when no data points are deleted. The first row reports the effectiveness (AccDrop) and second row reports the $L_2$ distance between $w^u$ and the optimal model.

Figure 5: Effectiveness-Certifiability trade-offs for largest volume of deletions. Each row corresponds to an unlearning method. Efficiency parameter $\tau$ is fixed at $m=m_0$ for Influence and $\tau_{0}=100$ for DeltaGrad. The left y-axis measures effectiveness (AccDrop and y-axis) measures certifiability ($\Delta\Omega_1$), the left y-axis measures effectiveness (AccDrop and y-axis) measures certifiability ($\Delta\Omega_1$). Lower is better for both y-axes.

Figure 4: Efficiency-Effectiveness trade-off at $\Delta\Omega_1=1$ for the largest volume of deletion as $\sigma$ is varied. Legend is same as in Figure 3.
One clear observation is that for Influence method, the accuracy drop increases only at higher values of σ (≥ 10^3). Moreover, we see its largest AccDrop is lower than other methods across all datasets. For example, in the MNIST dataset, the maximum AccDrop (at σ = 100) is approximately 19%, 40% and 79% and for Influence, Fisher and DeltaGrad respectively. At the same time, however, improved certifiability (i.e., decreased AccDis) is achieved for high values of σ. Therefore, to obtain a good combination of effectiveness and efficiency, one must select higher values of σ, based on the dataset.

Another clear observation is that for Fisher near σ = 1, the trade-off between AccDrop and AccDis is best amongst all methods across all datasets, AccDrop = {1.7%, 0.7%, 0.6%, 0.16%, 1.15%, 0.7%} and AccDis = {2.5%, 15%, 0.1%, 0.02%, 3%, 0.03%} for the datasets respectively as seen in Figure 5. If a good effectiveness-certifiability trade-off is required, then Fisher appears to be a very suitable method.

Note that, because Influence and Fisher share the same τ parameter, their experimental results in this section are directly comparable. However, the same cannot be said for DeltaGrad. As we saw earlier in this section, DeltaGrad is typically significantly slower than the other two methods, as evidenced in its achieved speed-ups – and so for these experiments, to demonstrate a clearer trade-off between its effectiveness and certifiability, we chose to assign it the largest value of τ, that allows it a higher computational budget (i.e., running time relative to the fully-retrained model) than the other methods. As discussed in Wu et al. [31], the effectiveness of the DeltaGrad method decreases only slightly when the periodicity T0 is set to larger values. Therefore, for the same computational budget as the other methods, the trade-off between certifiability and effectiveness for DeltaGrad will be similar to that shown in Figure 5. We can observe this in figs. 19, 22 and 25, where the rows correspond to different τ parameters for DeltaGrad. We see across the rows for each dataset, the AccDrop is similar, while small differences exist in AccDis due to the randomness of the noise injected.

Table 3: Correlation between AccDrop_{init} and ε with targeted-random deletion distribution for the Fisher unlearning method (m′ = m).

| 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          |

Let w^{*}_{init} be the model trained on the initial dataset and Acc_{init} be its accuracy on the test dataset. Then let AccDrop_{init} be the drop in test accuracy between the initial model w^{*}_{init} and the updated model w^*, i.e.,

\[
\text{AccDrop}_{\text{init}} = \text{SPE}(\text{Acc}_{\text{init}}, \text{Acc}_{\text{test}}) \tag{22}
\]

From the results in Section 5, we see that the Acc_{test} and Acc_{del} are highly correlated to each other. Extending this finding, we find the correlation between the updated models’ certifiability disparity ε (AccDis) and the accuracy drop w.r.t the initial model (AccDrop_{init}). In Table 3, we report the Pearson and Spearman correlations between ε and AccDrop_{init} when using a targeted-random deletion distribution and varying the deletion fraction from 0.01 till 0.45 (0.095 for MNIST) utilizing the Fisher unlearning method (m’ = m).

From Table 3, we see that apart from the HIGGS dataset, there is a large and positive pearson correlation between AccDrop_{init} and ε. The low correlation values of the HIGGS dataset can be attributed to the issue regarding variance that was observed when using the targeted-random deletion distribution as discussed in Section 5. This indicates that there is a strong linear relationship between AccDrop and AccDis.

Next, we propose the following steps to predict the certifiability disparity in an online setting. First, after training the model w^{*}_{init} on the initial dataset, we store its test accuracy Acc_{init}. Second, prior to employing the initial model, we obtain an updated model w^* and a fully-retrained model w^*, at the highest value of the efficiency parameter τ, for a sufficiently large deletion fraction, say 0.45 (for binary datasets). Third, we compute the AccDrop_{init} and AccDis using the models and obtain the slope c defined as

\[
c = \frac{\text{AccDis}}{\text{AccDrop}_{\text{init}}} \tag{23}
\]

We then use this slope to construct a simple linear regression model with zero bias to predict the certifiability disparity as

\[
\hat{\epsilon} = c \cdot \text{AccDrop}_{\text{init}}.
\]

where \(\hat{\epsilon}\) is the predicted disparity. Fourth, we begin the ML pipeline, and when an updated model w^* arrives for evaluation, we compute the AccDrop_{init} and then predict the certifiability disparity using Equation (23). If the predicted \(\hat{\epsilon}\) exceeds a given threshold for certifiability, then we restart the pipeline and train a new model from scratch on the remaining data. Otherwise, we verify if current model also meets the given threshold for efficiency and then employ the model for inference.
This proposed linear regression model is highly simplistic, but, only requires the computation of \( w^\dagger \), a fully-retrained model \( w^\star \) at the highest efficiency and the storage of the test accuracy of the initial model. However, due to its simplicity, the predicted disparity \( \hat{\epsilon} \) does tends to overestimate the true disparity \( \epsilon \). The design of more sophisticated models or algorithms to better estimate the disparity \( \epsilon \) is left for future work.

8 CONCLUSION

In this paper, we provide an experimental evaluation of three state-of-the-art machine unlearning methods for linear models. We analyze the unlearning methods in a common ML pipeline and compare the trade-offs between efficiency, effectiveness and certifiability that they offer. We extend and implement existing unlearning methods in this pipeline setting and evaluate them on six real-world datasets. We analyze the trade-offs offered by each unlearning method in a variety of setting and report the effect of the volume and distribution of the deleted data on the performance of linear ML models in general. We propose an online strategy to evaluate an updated model and determine when a full-retraining on the remaining data is required in the ML pipeline.

As a future work we hope to extend our pipeline to also evaluate data addition methods and provide a unified approach to study the processes of updating ML models.

REFERENCES

[1] P. Baldi, P. Sadowski, and D. Whitston. 2014. Searching for exotic particles in high-energy physics with deep learning. Nature Communications 5, 1 (Jul 2014).

[2] Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Henri Jia, Adelin Travers, Baowu Zhang, David Lie, and Nicolas Papernot. 2020. Machine Unlearning. arXiv:1912.08147 [cs] (July 2020). arXiv:1912.08147 [cs]

[3] Stephen Boyd, Stephen P. Boyd, and Lieven Vandenberghe. 2004. Convex optimization. Cambridge university press.

[4] Jonathan Brophy and Daniel Lowd. 2020. DART: Data Addition and Removal Trees. arXiv:2009.05567 [cs, stat] (Sept. 2020). arXiv:2009.05567 [cs, stat]

[5] Yinzi Cao and Junfeng Yang. 2015. Towards Making Systems Forget with Machine Unlearning. In 2015 IEEE Symposium on Security and Privacy. IEEE, San Jose, CA, 463–480. doi:10.1109/SP.2015.35

[6] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2 (2011), 27:1–27:27. Issue 3. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[7] Kamalika Chaudhuri and Claire Monteleoni. 2009. Privacy-preserving logistic regression. In Advances in Neural Information Processing Systems. D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Eds.), Vol. 21. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2008/file/8065d07a4a776214596a84f1e566d5d-Paper.pdf

[8] Ronan Collobert, Samy Bengio, and Yann LeCun. 2008. A Parallel Mixture of SVMs for Very Large Scale Problems. Neural Computation 20, 1 (May 2008), 1–76. http://www.cs.toronto.edu/~hinton/attic/practical.pdf

[9] Cynthia Dwork, Aaron Roth, and Saeed Sharifi-Malvajerdi. 2020. Descent-to-Delete: Gradient-Based Methods for Machine Unlearning. arXiv:2007.02923 [cs, stat] (July 2020). arXiv:2007.02923 [cs, stat]

[10] Pang Wei Koh and Percy Liang. 2017. Understanding Black-Box Predictions via Influence Functions. arXiv:1703.04730 [cs, stat] (July 2017). arXiv:1703.04730 [cs, stat]

[11] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. Yann Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1412.6980

[12] Kapoor, Saurabh, and Arindam Banerjee. 2020. Mixed-Privacy Forgetting in Deep Networks. (Dec. 2020). arXiv:2012.13431 [cs]

[13] Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. 2021. Approximate Data Deletion from Machine Learning Models. In Proceedings of The 24th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research). Arindam Banerjee and Kenji Fukumizu (Eds.), Vol. 130. PMLR, 2008–2016. http://proceedings.mlr.press/v130/izzo21a.html

[14] M. Karasuyama and I. Takeuchi. 2010. Multiple Incremental Decremental Learning of Support Vector Machines. IEEE Transactions on Neural Networks 21, 7 (2010), 1048–1059. https://doi.org/10.1109/TNN.2010.2048039

[15] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [n.d.]. CIFAR-10 (Canadian Institute for Advanced Research). [n.d.]. http://www.cs.toronto.edu/~kriz/cifar.html

[16] Yann LeCun and Corinna Cortes. 2010. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/.

[17] Alessandro Mantelelo. 2013. The EU Proposal for a General Data Protection Regulation and the roots of the ‘right to be forgotten’. Computer Law & Security Review 29, 3 (2013), 229–235. https://doi.org/10.1016/j.clsr.2013.03.010

[18] Cheryl Nee, Aaron Roth, and Saeed Sharifi-Malvajerdi. 2020. Mixed-Privacy Forgetting in Deep Networks. arXiv:2007.02923 [cs, stat] (July 2020). arXiv:2007.02923 [cs, stat]

[19] Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. 2020. Descent-to-Delete: Gradient-Based Methods for Machine Unlearning. arXiv:2007.02923 [cs, stat] (July 2020). arXiv:2007.02923 [cs, stat]

[20] Quoc Phong Nguyen, Bryan Kian Hsiang Low, and Patrick Jaillet. [n.d.]. Variational Bayesian Unlearning. [n.d.]. 12.

[21] Council of European Union. 2016. Regulation EU 2016/679. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:3A02016R0679-20160504

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaisons, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32. H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alchê-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[23] Tomaso A Poggio. 2000. Incremental and Decremental Support Vector Machine Learning. In NIPS.

[24] Sebastian Schelter. 2020. “Amnesia” – Towards Machine Learning Models That Can Forget User Data Very Fast. In Conference on Innovative Data Systems Research (CIDR).

[25] Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. 2014. Incremental and Decremental Training for Linear Classification. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’14). Association for Computing Machinery, New York, New York, USA, 343–352. https://doi.org/10.1145/2623530.2623661

[26] G. Tsoumakas, I. Katakis, and I. Vlahavas. 2008. Effective and Efficient Multilabel Classification in Domains with Large Number of Labels. (2008).

[27] Yinjun Wu, Val Tannen, and Susan B. Davidson. 2020. DeltaGrad: Rapid Retraining of Machine Learning Models. arXiv:2006.14755 [cs, stat] (June 2020). arXiv:2006.14755 [cs, stat]

[28] Yinjun Wu, Val Tannen, and Susan B. Davidson. 2020. P2IVU: A Provenance-Based Approach for Incrementally Updating Regression Models. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. ACM, Portland OR USA, 447–462. https://doi.org/10.1145/3318464.3380571
A EXPERIMENTAL SETUP

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

A.1 Training

Ensuring that the training phase of the common ML 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. Wu et al. [31] describes that smaller mini-batch size leads to lower approximation and hence lower effectiveness. However, choosing a full-batch gradient descent update as described in Equation (2) to ensure 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 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 $\eta$ to compensate for the slower average gradient updates. However, we experimentally found that increasing the learning rate $\eta$ beyond 1 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 1 and choose large enough mini-batches (for DELTAGRAD performance) while keeping the number of epochs low (for computational effort) using a small validation dataset of the initial training data $D_{init}$. The chosen values of the mini-batch size and the number of epochs for each dataset is described in Table 4.

Table 4: 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            |

A.2 DELTAGRAD unlearning method

As described in Section 3.3, the primary $\tau$ parameter chosen for the DELTAGRAD method was the periodicity $T_0$. Based on the discussion of the hyper-parameter of the DELTAGRAD in Wu et al. [31], the ideal $\tau$ parameter would be the training mini-batch size. However, this would result in a non-standard training stage in the ML pipeline for the DELTAGRAD which in turn prevents 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 $\tau$ parameters are the burn-in period $j_0$ and the size of the history for the L-BFGS algorithm $h$. Following Wu et al. [31], we fix $h = 2$ for all datasets and the values of $j_0$ are presented in Table 5.

Table 5: Values of DELTAGRAD burn-in period parameter $j_0$

| Dataset | $j_0$ |
|---------|-------|
| MNIST$^b$ | 10    |
| MNIST   | 20    |
| COVTYPE | 10    |
| HIGGS   | 500   |
| CIFAR2  | 20    |
| EPSILON | 10    |

B EXTENDED DELETION DISTRIBUTION RESULTS

In Figure 7, we present the extended results for the uniform-random and uniform-informed deletion distribution. We increase the fraction of data deleted from 0.5 till 0.995. We see that the drop in both $Acc_{test}$ and $Acc_{del}$ only occur when we delete beyond 90% of the initial training data. We also clearly see that the drop in both metric is much steeper for the uniform-informed distribution compared to the uniform-random distribution. This indicates that the informed deletions are deleting outliers that are required by the ML model to effectively classify samples.

C CERTIFIABILITY-EFFICIENCY TRADE-OFF RESULTS

In this section, we present the additional results of the trade-off between certifiability and efficiency as the $\tau$ parameter is varied at different volumes of deletion and values of $\sigma$. In Table 7, we provide an interface to easily navigate to the results corresponding to each value of $\sigma$. In each figure, there are sub-figures corresponding to different volumes of deletion. For example, Figure 8 presents results for when $\sigma = 0.01$ and sub-figures figs. 8a to 8c correspond to the small, medium and large deletion volumes described in Table 2. The legend for the range of the $\tau$ parameter is the same as that found in Figure 3.

There are two interesting trends to note from these results. First, is that for values of $\sigma < 1$, we see little to no difference in the trend of the trade-off offered. This is due to the smaller quantities of injected noise that does not increase certifiability by reducing the accuracy disparity. Second, is that for smaller volumes of deletion, all of the unlearning methods have lower $Acc_{Dis}$ and offer higher certifiability, especially at lower values of $\tau$. This is because the unlearning algorithms of each method are better able approximate the fully-retrained model as the number of deleted points is fewer.
Figure 7: Extended deletion distribution results. Deletion fraction varied from 0.5 till 0.995. Only uniform-random and uniform-informed deletion distribution results are reported.

Table 6: Links to results for the trade-offs for efficiency.

| Trade-Off                  | \(\sigma\) | 0.01 | 0.1  | 1    | 10   | 100 |
|----------------------------|------------|------|------|------|------|-----|
| Certifiability-Efficiency  | fig. 8     | fig. 9 | fig. 10 | fig. 11 | fig. 12 |
| Effectiveness-Efficiency   | fig. 13    | fig. 14 | fig. 15 | fig. 16 | fig. 17 |

D EFFECTIVENESS-EFFICIENCY TRADE-OFF RESULTS

In this section, we present the additional results of the trade-off between efficiency and effectiveness as the \(\tau\) parameter is varied for different volumes of deletion and values of \(\sigma\). In Table 7, we provide an interface to easily navigate to the results corresponding to each value of \(\sigma\). In each figure, there are sub-figures corresponding to different volumes of deletion. For example, Figure 13 presents results for when \(\sigma = 0.01\) and sub-figures figs. 13a to 13c correspond to the small, medium and large deletion volumes described in Table 2. The legend for the range of the \(\tau\) parameter is the same as that found in Figure 3.

E CERTIFIABILITY-EFFICIENCY TRADE-OFF RESULTS

In this section, we present the additional results for the trade-off between certifiability and effectiveness as \(\sigma\) is varied for different volumes of deletions and values of the \(\tau\) parameter. In Appendices E.1 to E.3 we present the results corresponding to the small, medium and large deletion volumes described in Table 2. In each subsection, we present the results for each unlearning method in a figure, where the rows of the figure correspond to different values of the \(\tau\) parameter. In Table 7, we provide an interface to easily navigate to the results corresponding to each unlearning method and volume of deletion.

E.1 Small deletion volume results

In this subsection, Figures 18 to 20 correspond to the results for the Fisher, DeltaGrad and Influence methods for the smallest volume of deleted data.

E.2 Medium deletion volume results

In this subsection, Figures 21 to 23 correspond to the results for the Fisher, DeltaGrad and Influence methods for the medium volume of deleted data.

E.3 Large deletion volume results

In this subsection, Figures 24 to 26 correspond to the results for the Fisher, DeltaGrad and Influence methods for the largest volume of deleted data.
Figure 8: Certifiability and efficiency trade-off results for $\sigma = 0.01$ at different volumes of deletion: (a) small (b) medium and (c) large. AccD1s is reported on the y-axis and the speed-up in running time on the x-axis.

Figure 9: Certifiability and efficiency trade-off results for $\sigma = 0.1$ at different volumes of deletion: (a) small (b) medium and (c) large. AccD1s is reported on the y-axis and the speed-up in running time on the x-axis.
Figure 10: Certifiability and efficiency 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 efficiency up in running time on the x-axis.

Figure 11: Certifiability and efficiency trade-off results for $\sigma = 10$ 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.
Figure 12: Certifiability and efficiency trade-off results for $\sigma = 100$ at different volumes of deletion: (a) small (b) medium and (c) large. AccD1s is reported on the y-axis and the speed-up in running time on the x-axis.

Figure 13: Effectiveness and efficiency trade-off results for $\sigma = 0.01$ at different volumes of deletion: (a) small (b) medium and (c) large. AccDrop is reported on the y-axis and the speed-up in running time on the x-axis.
Figure 14: Effectiveness and efficiency trade-off results for $\sigma = 0.1$ at different volumes of deletion: (a) small (b) medium and (c) large. AccDrop is reported on the y-axis and the speed-up in running time on the x-axis.

Figure 15: Effectiveness and efficiency trade-off results for $\sigma = 1$ at different volumes of deletion: (a) small (b) medium and (c) large. AccDrop is reported on the y-axis and efficiency p in running time on the x-axis.
Figure 16: Effectiveness and efficiency trade-off results for $\sigma = 10$ at different volumes of deletion: (a) small (b) medium and (c) large. AccDrop is reported on the y-axis and efficiency $p$ in running time on the x-axis.

Figure 17: Effectiveness and efficiency trade-off results for $\sigma = 100$ at different volumes of deletion: (a) small (b) medium and (c) large. AccDrop is reported on the y-axis and the speed-up in running time on the x-axis.
Figure 18: Certifiability-Effectiveness trade-off for Fisher method at small deletion volume. Each row corresponds to a value of the $\tau$ parameter.

Figure 19: Certifiability-Effectiveness trade-off for DeltaGrad method at small deletion volume. Each row corresponds to a value of the $\tau$ parameter.
Figure 20: Certifiability-Effectiveness trade-off for \textsc{Influence} method at small deletion volume. Each row corresponds to a value of the $\tau$ parameter.

Figure 21: Certifiability-Effectiveness trade-off for \textsc{Fisher} method at medium deletion volume. Each row corresponds to a value of the $\tau$ parameter.
Figure 22: Certifiability-Effectiveness trade-off for DeltaGrad method at medium deletion volume. Each row corresponds to a value of the $\tau$ parameter

Figure 23: Certifiability-Effectiveness trade-off for Influence method at medium deletion volume. Each row corresponds to a value of the $\tau$ parameter
Figure 24: Certifiability-Effectiveness trade-off for Fisher method at large deletion volume. Each row corresponds to a value of the $\tau$ parameter.

Figure 25: Certifiability-Effectiveness trade-off for DeltaGrad method at large deletion volume. Each row corresponds to a value of the $\tau$ parameter.
Figure 26: Certifiability-Effectiveness trade-off for INFLUENCE method at large deletion volume. Each row corresponds to a value of the $\tau$ parameter.