An Introduction to Machine Unlearning

Salvatore Mercuri[1], Raad Khraishi[1,2], Ramin Okhrati[2]
Devesh Batra[1], Conor Hamill[1], Taha Ghasempour[1,⋆], and Andrew Nowlan[1]

[1] Data Science & Innovation, NatWest Group, London, United Kingdom[†]
[2] Institute of Finance and Technology, UCL, London, United Kingdom[‡]

Abstract. Removing the influence of a specified subset of training data from a machine learning model may be required to address issues such as privacy, fairness, and data quality. Retraining the model from scratch on the remaining data after removal of the subset is an effective but often infeasible option, due to its computational expense. The past few years have therefore seen several novel approaches towards efficient removal, forming the field of "machine unlearning", however, many aspects of the literature published thus far are disparate and lack consensus. In this paper, we summarise and compare seven state-of-the-art machine unlearning algorithms, consolidate definitions of core concepts used in the field, reconcile different approaches for evaluating algorithms, and discuss issues related to applying machine unlearning in practice.

Keywords: Machine unlearning · Exact unlearning · Approximate unlearning · Data removal · Data privacy.

arXiv:2209.00939v1 [cs.LG] 2 Sep 2022

⋆ No longer at institute[1]
† Correspondence to: `salvatore.mercuri@natwest.com`
‡ Correspondence to: `raad.khraishi@ucl.ac.uk`

Table of Contents

1 Introduction

Widely used machine learning algorithms are able to learn from new data using batch or online training methods but are incapable of efficiently adapting to data removal. Data removal, however, may be required to address various issues around privacy, fairness, and data quality. For example, the "Right to be Forgotten" in the European Union's General Data Protection Regulation (GDPR) provides individuals with the right to request the removal of their data from an organisation's records. Though data may be removed from databases, any machine learning model previously trained on the removed data will retain its information, and proposed legislation such as the European Union Artificial Intelligence Act[3] may require further action. Beyond privacy, other applications include data correction by removing erroneous data (Biggio et al., 2013), bias reduction (Quenouille, 1956), model explainability (Doshi-Velez and Kim, 2017), and uncertainty quantification (Shafer and Vovk, 2008). One way to remove the information of deleted data from trained models is to remove the specified data from the training dataset and then retrain the model from scratch on the remaining data. Though this "naïve" retraining approach is effective in removing the influence of the removed data, it is often inefficient and may entail large computational costs. For example, modern deep learning models may take several weeks to train and can cost millions of dollars.[4]

The field of "machine unlearning" addresses the problem of removing subsets of training data, through the development of algorithms that guarantee the removal of the subset data's information from the trained model more efficiently than naïve retraining. The term machine unlearning originates from Cao and Yang (2015), in which they develop one of the first systematic unlearning algorithms. The theory has since seen a number of proposed approaches which can broadly be categorised as exact or approximate unlearning. Exact unlearning algorithms reduce the large computational cost of naïve retraining by structuring the initial training so as to allow for more efficient retraining; in doing so they replicate the same model that would have been produced under naïve retraining. In contrast, approximate unlearning algorithms avoid the need for full retraining, speeding up the process of unlearning by allowing a degree of approximation between the output model and the naïve retrained model. Approximate methods typically leverage at least one of the following in order to unlearn: information about the data to be removed (Golatkar et al., 2019), information about the remaining data after removal (Guo et al., 2020; He et al., 2021), or information cached during the original machine learning training (He et al., 2021; Wu et al., 2020).

The evaluation approach of an unlearning algorithm often depends on the application and the category of the unlearning approach, resulting in evaluation frameworks and measures that are inconsistent throughout the literature. In particular, there is little consensus in measuring how well an unlearning algorithm has removed the information of the deleted data. These inconsistencies even extend to core definitions used in the field including what defines an unlearning algorithm. The prior work of Mahadevan and Mathioudakis (2021) offers a benchmarked comparison between different machine unlearning methods and is a key review paper, however, it only considers three approximate methods with a focus on linear classification methods trained with gradient descent. Moreover, there is a lack of discussion around applying machine unlearning techniques in practice.

In the present paper, we first provide standardised definitions of concepts which occur throughout the literature in Section 2. The measures used to evaluate unlearning methods are collated in Section 3 and we discuss their applicabilities and limitations. Section 4 and Section 5 review a total of seven state-of-the-art unlearning methods – exact and approximate respectively – and summarise their scope, benefits and limitations. Finally, in Section 6, we bring together our findings from this review, providing comparisons among the different unlearning techniques. We also discuss aspects of taking unlearning theory into practice, by giving proposals for algorithm selection and monitoring procedures.

Selection criteria. We select seven methods that unlearn data points from machine learning models, covering a broad scope of applicability across both exact and approximate unlearning, so that we include at least one method that may be applied to gradient-descent based models, decision-tree based models, those with

---

[3] https://artificialintelligenceact.eu/the-act/
[4] https://lambdalabs.com/blog/demystifying-gpt-3/

convex loss functions and those with non-convex loss functions. We aim to avoid including more than one unlearning algorithm that covers the same or highly similar applicability, unlearning type, and algorithmic methodology.

These selection criteria naturally mean that many works are excluded, but not from lack of merit or significance, and here we mention their achievements. In a relatively early contribution, Ginart et al. (2019) develop an exact unlearning algorithm applied to the k-means algorithm and in the appendix they are the first to propose the notion of approximate unlearning. This work is significant for inspiring a strain of methods based around statistical approximation and indistinguishability, of which we include further developments in this paper, most directly by Guo et al. (2020); Neel et al. (2020). Schelter et al. (2021) develop a method applicable to decision-tree based models; this has similar applicability and similarly strong empirical results as Brophy and Lowd (2021), which is included in this paper. In Aldaghri et al. (2021), the authors develop an exact unlearning method that uses a similar sharding methodology as seen in SISA, Section 4.1. Gupta et al. (2021) highlight issues that are caused by the implicit assumption that the deletion requests are independent of the prior machine learning model, and they modify the SISA algorithm to work for so-called *adaptive* sequences of deletion requests. Indeed, it may well be the case that a user is more likely to request data deletion upon knowledge of model decisions concerning them. In Golatkar et al. (2020), the authors extend their method considered in Section 5.1 to apply to deep neural networks. A similar task to the one of unlearning data points is of unlearning classes, which we do not consider in this paper, but is considered by Golatkar et al. (2019) and by others such as Baumhauer et al. (2020). Wu et al. (2022) extend the general methodology of the Influence method (Guo et al., 2020) that we discuss in Section 5.2. We note that there has been research in applying unlearning to recommender systems and regression problems, notably by Schelter (2020) for item-based collaborative filtering recommender systems, k-nearest neighbours, and ridge regression. In addition, Chen et al. (2022) extend the ideas of the SISA algorithm to recommender systems. Finally, an area not considered in this paper concerns Markov Chain Monte Carlo models, for which the first unlearning methods have recently been developed (Fu et al., 2022; Nguyen et al., 2022).

2 Terminology

In this section, we give formal definitions of the fundamental concepts in machine unlearning, which appear throughout the literature and the rest of this paper. Due primarily to the varying approaches towards achieving unlearning in the literature, definitions of a machine unlearning algorithm vary substantially. The key aim of this section is to provide a definition of unlearning that unifies these approaches.

Throughout this paper, we let $\mathcal{X}$ denote the feature space and $\mathcal{Y}$ denote the output space. A data point is an element of $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, and we let $\mathcal{Z}^*$ denote the space of datasets. Explicitly, a dataset $D \in \mathcal{Z}^*$ is a multiset of data points, allowing for duplicate entries.

The objective of our learning framework is to learn, from a dataset D, a hypothesis function $h : \mathcal{X} \rightarrow \mathcal{Y}$, which assigns an output $y = h(x) \in \mathcal{Y}$ to a given input $x \in \mathcal{X}$. A training algorithm can therefore be viewed as a map $\mathcal{A} : \mathcal{Z}^* \rightarrow \mathcal{H}$, where $\mathcal{H}$ is the space of all hypothesis functions, whose objective is to minimise a non-negative real-valued loss function $L(h, D)$. All training algorithms we consider are randomised, by which we mean that they admit some degree of randomness, for example through random initialisation of weights or through random selection of training data as in stochastic gradient descent.

In the setting of unlearning, the initial training data D is fixed with $n \geq 1$ samples and a dimensionality of $p \geq 1$. Due to the randomness of $\mathcal{A}$, the output $\mathcal{A}(D)(x)$ at a point $x \in \mathcal{X}$ can be viewed as a random variable. If $\mathcal{A}$ is parametric, then there exists a space of parameters $\theta \in \Theta$ of fixed dimension determining the hypothesis function $h_\theta = \mathcal{A}(D)$, and in this case we use θ and h_θ interchangeably to denote our trained model. The goal of an unlearning algorithm is to remove the influence of a subset $D_u \subseteq D$ of m samples from the trained machine learning model $\mathcal{A}(D)$. The rest of this section is dedicated to formalising the notion of machine unlearning.

In the following definition we define an *update mechanism*, which is one of two core components to an unlearning algorithm.

Definition 1 (Update Mechanism). *An* update mechanism *is a map*

$$\mathcal{U} : \mathcal{H} \times \mathcal{Z}^* \times \mathcal{Z}^* \to \mathcal{H},$$

which takes as input a model $h \in \mathcal{H}$, two datasets $D, D_u \in \mathcal{Z}^*$, and outputs a new model $\mathcal{U}(h, D, D_u) \in \mathcal{H}$.

Remark 1. In the case of unlearning, we have $D_u \subseteq D$ and the output of the update mechanism aims to remove as much information about D_u as possible from the model h in a way that we make precise in the rest of this section; in this case we may also use *removal mechanism* as a synonym for update mechanism. Definition 1 above gives the minimal inputs needed to define an update mechanism. In reality, update mechanisms typically require additional inputs, which are clearly defined in each specific case in later sections. In the initial application of $\mathcal{U}$ in unlearning, we have $h = \mathcal{A}(D)$, however subsequent applications of $\mathcal{U}$ may be applied to models that are not necessarily the result of applying $\mathcal{A}$, but from some preceding application of $\mathcal{U}$ itself, for example. In all cases considered in this paper, the mechanism $\mathcal{U}$ is randomised in the same manner as randomised training algorithms.

One way of ensuring the removal of D_u from a trained model is to stipulate that the removal mechanism is equivalent to applying the training algorithm $\mathcal{A}$ on the dataset D with D_u removed. This is so-called *exact removal* which is defined as follows.

Definition 2 (Exact Removal Mechanism). *An update mechanism $\mathcal{U}$ for a randomised training algorithm $\mathcal{A}$ is said to be an* exact removal mechanism *if and only if it achieves the same model that would have been achieved by retraining from scratch. Formally, $\mathcal{U}$ is exact if and only if, for all $D \in \mathcal{Z}^*$, $D_u \subseteq D$, and $x \in \mathcal{X}$, the random variables $\mathcal{U}(\mathcal{A}(D), D, D_u)(x)$ and $\mathcal{A}(D \setminus D_u)(x)$ admit the same distributions.*

The simplest example of exact removal is to just retrain from scratch on $D \setminus D_u$. This is the *naïve removal mechanism* defined as follows.

Definition 3 (Naïve Removal Mechanism). *Given a randomised training algorithm $\mathcal{A}$, the* naïve removal mechanism $\mathcal{U}^*$ *is the update mechanism defined by*

$$(\mathcal{A}(D), D, D_u) \mapsto \mathcal{A}(D \setminus D_u),$$

where $D_u \subseteq D$.

Whilst naïve removal is a simple method to implement and achieve unlearning, it can be computationally expensive and time consuming. Some of the key developments in the unlearning literature involve the development of alternative exact removal mechanisms to naïve removal, as we shall see in Section 4.

Exact removal is not the only option, however, and Ginart et al. (2019) proposed mechanisms that achieve removal through alternative means, which are so-called *approximate removal mechanisms*. The probability distribution of these mechanisms no longer match that of naïve removal, and we must therefore resort to alternative means of guaranteeing that information on D_u has been removed from the model h – this guarantee is known as the *certifiability* of the update mechanism. For example, Guo et al. (2020) guarantee this by proving statistical indistinguishability between the update mechanism and naïve unlearning, as follows.

Definition 4 (ϵ-certifiability, Guo et al. (2020)). *Let $\epsilon > 0$ be given, we say that an update mechanism $\mathcal{U}$ is an ϵ-certified removal mechanism* with respect to the randomised training procedure $\mathcal{A}$ *if for all datasets $D \in \mathcal{Z}^*$, removal subsets $D_u \subseteq D$, inputs $x \in \mathcal{X}$ and output subsets $\mathcal{T} \subseteq \mathcal{Y}$, we have*

$$e^{-\epsilon} \leq \frac{\Pr[\mathcal{U}(\mathcal{A}(D), D, D_u)(x) \in \mathcal{T}]}{\Pr[\mathcal{A}(D \setminus D_u)(x) \in \mathcal{T}]} \leq e^{\epsilon}. \tag{2.1}$$

Intuitively, ϵ-certified removal gives a bound of e^{ϵ} on the probability that one can distinguish between the updated model and the model obtained through naïve removal. The notion of ϵ-certified can be weakened slightly to give (ϵ, δ)-certified removal as follows.

Definition 5 ((ϵ, δ)-certifiability, Guo et al. (2020)). *Let $\epsilon, \delta > 0$ be given. We say that an update mechanism $\mathcal{U}$ is an (ϵ, δ)-certified removal mechanism with respect to the randomised training procedure $\mathcal{A}$ if for all datasets $\mathcal{D} \in \mathcal{Z}$, removal subsets $D_u \subseteq D$, inputs $x \in \mathcal{X}$, and output subsets $\mathcal{T} \subseteq \mathcal{Y}$, we have*

$$
\begin{aligned}
\Pr[\mathcal{U}(\mathcal{A}(D), D, D_u)(x) \in \mathcal{T}] &\leq e^\epsilon \Pr[\mathcal{A}(D \setminus D_u)(x) \in \mathcal{T}] + \delta, \\
\Pr[\mathcal{A}(D \setminus D_u)(x) \in \mathcal{T}] &\leq e^\epsilon \Pr[\mathcal{U}(\mathcal{A}(D), D, D_u)(x) \in \mathcal{T}] + \delta.
\end{aligned}
\tag{2.2}
$$

The above discussion motivates a definition of unlearning that consists of two components, an update mechanism along with a guarantee of certifiability of this mechanism. However, the varying methods for showing certifiability in the literature make it difficult to consolidate this notion into a standard definition. This is perhaps partly due to a lack of clarity in current regulations concerning the application of unlearning methods in practice, which makes it difficult to argue for the use of certain certifiability guarantees over others. To circumvent this, we stipulate only that the update mechanism be accompanied by a verifiable set of theoretical guarantees and empirical results concerning the mechanism's ability to remove information about D_u from $\mathcal{A}(D)$. The required nature and degree of certifiability is left up to external judgement by theoreticians, practitioners and auditors.

We now give a straw man concept for a *theoretical* certificate of unlearning, which should be provided in any theoretical development of an unlearning algorithm. This should be extended to a formalised certificate of unlearning concept that applies also in practice, on the same level of rigour as the "Proof of Learning" concept from Jia et al. (2021). In practice, this should provide enough information (for example source code) for auditors to recreate any historical removals, and subsequently recompute and verify the certificate of unlearning. The need for such a concept has also been identified in Thudi et al. (2021), who provide their own "Proof of Unlearning".

Definition 6 (Certificate of Unlearning). *Given a randomised training algorithm $\mathcal{A}$ and update mechanism $\mathcal{U}$, a certificate of unlearning, $\mathcal{C}_{\mathcal{U}}$ of $\mathcal{U}$, is a set of verifiable claims with which certifiability may be (re-)computed and analysed externally. The claims should show that, for any dataset D and subset $D_u \subseteq D$, the model $\mathcal{U}(\mathcal{A}(D), D, D_u)$ contains a sufficiently small amount of information about D_u. The precise sense of "sufficiently small" depends on the particular certifiability claim, which could, for example, be a proof of exactness (Definition 2) or ϵ-certifiability (Definition 4).*

Definition 7 (Machine Unlearning Algorithm). *Given a randomised training algorithm $\mathcal{A}$, a machine unlearning algorithm is a pair $(\mathcal{U}, \mathcal{C}_{\mathcal{U}})$ consisting of an update mechanism $\mathcal{U}$ (Definition 1) and a certificate of unlearning $\mathcal{C}_{\mathcal{U}}$ (Definition 6).*

Remark 2. As we shall see later, some of the unlearning algorithms provide only empirical results for the certificate of unlearning. While empirical results and measures are a helpful addition to the certificate of unlearning, given the privacy-sensitive nature of many applications of unlearning, they are not enough alone. The addition of empirical results to the certificate of unlearning is most useful in the case of approximate removal, where removal of data is only guaranteed to an approximate degree.

For the rest of this section, we fix a randomised training algorithm $\mathcal{A}$ and drop the dependence on $\mathcal{A}$ from notation. We give the baseline example of an unlearning algorithm in this framework.

Definition 8 (Naïve Unlearning). *The naïve unlearning algorithm is $(\mathcal{U}^*, \mathcal{C}_{\mathcal{U}^*})$, where $\mathcal{U}^*$ is the naïve removal mechanism (Definition 3). The certificate of unlearning in this case contains the trivial fact that $\mathcal{U}^*$ is exact.*

Definition 9 (Exact and Approximate Unlearning Algorithms). *An exact unlearning algorithm is a pair $(\mathcal{U}, \mathcal{C}_{\mathcal{U}})$, where $\mathcal{U}$ is an exact unlearning mechanism (Definition 2); in this case $\mathcal{C}_{\mathcal{U}}$ includes a proof that $\mathcal{U}$ is an exact removal mechanism. An approximate unlearning algorithm is an unlearning algorithm that is not exact.*

While we have mostly formulated the process of machine unlearning above for a one-time batch removal, in reality unlearning may be required to occur over a sequence of removals, as in the case where a user requests the deletion of their data. As a result, analysis of an unlearning algorithm over arbitrary sequences of removals is helpful. The simplest case of sequences of single removals is considered by Neel et al. (2020), whose method we detail later in Section 5.4. They introduce the notion of strong, weak and (α, β)-accurate unlearning algorithms, as follows. Let $\{\mathbf{z}_i \mid 0 \leq i \leq m\} \subseteq D$ be a sequence of data points to be unlearned, and let

$$D_0 = D, \quad D_i = D_{i-1} \setminus \{\mathbf{z}_{i-1}\}, \quad h_0 = \mathcal{A}(D_0), \quad h_i = \mathcal{U}(h_{i-1}, D_{i-1}, \mathbf{z}_{i-1}), \qquad 1 \leq i \leq m, \qquad (2.3)$$

be the sequence of datasets and unlearned models, respectively, occurring in the unlearning process. The (α, β)-accuracy provides a theoretical guarantee of the obtained parameters of an unlearning algorithm over the length of the sequence, whereas strong vs. weak characterises algorithms based on their unlearning speed over the length of the sequence.

Definition 10 ((α, β)-accuracy, Neel et al. (2020)). *Let $h_i^* = \mathcal{A}(D_{i-1} \setminus \{\mathbf{z}_{i-1}\})$ be the result of naïve removal on h_{i-1} for $i = 1, \ldots, m$. Given $\alpha > 0$ and, $0 < \beta < 1$, we say that $(\mathcal{U}, \mathcal{C}_{\mathcal{U}})$ is (α, β)-accurate if, for all $0 \leq i \leq m$, we have*

$$\Pr\left[L(h_i, D_i) - L(h_i^*, D_i) > \alpha\right] < \beta. \qquad (2.4)$$

In other words, the probability that the unlearned loss and the naïve retrained loss differing by more than α is at most β.

Definition 11 (Strong and Weak Unlearning, (Neel et al., 2020)). *Let C_i denote the time taken to perform the ith removal. Assume that $(\mathcal{U}, \mathcal{C}_{\mathcal{U}})$ is (α, β)-accurate and that α and β are independent of m.*

(i) The unlearning algorithm $(\mathcal{U}, \mathcal{C}_{\mathcal{U}})$ is said to be strong if the update time is at most logarithmic in the length of the update sequence. That is, for all $1 \leq i \leq m$ we have

$$C_i / C_1 = \mathcal{O}(\log(i)).$$

(ii) We say that $(\mathcal{U}, \mathcal{C}_{\mathcal{U}})$ is weak if the update time is polynomial in the length of the update sequence. That is, for all $1 \leq i \leq m$ we have

$$C_i / C_1 = \Omega(poly(i)).$$

To finish this section, we define two concepts which are closely related to the theory of machine unlearning. Differential privacy inspired the definition of (ϵ, δ)-certifiability in Guo et al. (2020), and we can see the similarities of the definition of differential privacy below to that of Definition 5. Intuitively, differential privacy bounds the effect on the output of a query of changing a singleton's data; as a result, it constrains the amount of information an attacker could extract about an individual in a dataset.

Definition 12 (Differential Privacy, Dwork and Roth (2014)). *We say that a randomised training algorithm $\mathcal{A}$ is (ϵ, δ)-differentially private if for all pairs of datasets D_1 and D_2 that differ by a singleton's data, for all inputs $x \in \mathcal{X}$ and output subsets $\mathcal{T} \subseteq \mathcal{Y}$, we have*

$$\Pr[\mathcal{A}(D_1)(x) \in \mathcal{T}] \leq e^\epsilon \Pr[\mathcal{A}(D_2)(x) \in \mathcal{T}] + \delta. \qquad (2.5)$$

Definition 13 (Catastrophic Forgetting, Kirkpatrick et al. (2017)). *Catastrophic forgetting is the rapid decline in predictive ability of a model on previously learned tasks when fine-tuned for a new task.*

Remark 3. Catastrophic forgetting is unsuitable for machine unlearning as information on the original training set can still be extracted from the weights of the fine-tuned model, as discussed in Golatkar et al. (2019).

3 Evaluation Approaches for Unlearning Algorithms

Comprehensive evaluation of a machine unlearning algorithm $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ is achieved through consideration of four key properties – *efficiency*, *effectiveness*, *consistency*, and *certifiability* (**EECC**) – defined as follows.

Definition 14 (Efficiency). *The* efficiency *of* $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ *is the relative speed-up of the removal mechanism* $\mathcal{U}$ *over the naïve removal mechanism* $\mathcal{U}^*$.

Definition 15 (Effectiveness). *The* effectiveness *of* $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ *is the test set performance of the unlearned machine learning model,* $\mathcal{U}(\mathcal{A}(D), D, D_u)$, *in comparison to the naïve retrained model's test set performance.*

Definition 16 (Consistency). *The* consistency *of* $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ *is a measure of similarity between the unlearned model,* $\mathcal{U}(\mathcal{A}(D), D, D_u)$, *and the naïve retrained model.*

Definition 17 (Certifiability). *The* certifiability *of* $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ *is the ability of the removal mechanism* $\mathcal{U}$ *to remove the information of* D_u *from the unlearned model,* $\mathcal{U}(\mathcal{A}(D), D, D_u)$. *Precise measures and guarantees of this ability form the contents of* $\mathcal{C}_\mathcal{U}$.

Typically in the literature, these four measures are evaluated empirically on real-world and synthetic datasets, often accompanied by additional theoretical analysis. In experiments, a range of datasets are chosen to reflect varying task complexities.

Within each method, empirical performances for EECC can be tuned through certain parameters. There are pairwise trade-offs between efficiency, effectiveness, and certifiability as demonstrated in Mahadevan and Mathioudakis (2021). In addition, there are trade-offs between efficiency, effectiveness and consistency. Each method contains parameters that directly control the efficiency of the method, which are called *efficiency parameters*; these can be used to improve efficiency at the cost of effectiveness and certifiability. Approximate methods may also have *certifiability parameters* that directly control the certifiability of the method.

We detail EECC in the following sections, giving empirical measures for them. *Performance metric* refers to a metric that is used to evaluate the machine learning models, for example, accuracy.

3.1 Efficiency

Efficiency of an unlearning algorithm $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ is empirically measured by the ratio of the time taken to obtain the unlearned model $h^\mathcal{U} = \mathcal{U}(\mathcal{A}(D), D, D_u)$ to the time taken to obtain the naïve retrained model $h^* = \mathcal{U}^*(\mathcal{A}(D), D, D_u)$, defined as follows:

$$\texttt{Efficiency}(h^\mathcal{U}) := \frac{\text{time taken to obtain } h^*}{\text{time taken to obtain } h^\mathcal{U}}. \tag{3.1}$$

In practice, this is usually achieved by measuring time taken to obtain h^* and $h^\mathcal{U}$ directly, averaged over a number of deletion requests. In Brophy and Lowd (2021), efficiency is measured by the number of samples that can be unlearned in the time it takes to unlearn *one* instance by naïve retraining; this measure is equivalent to the definition in (3.1) above. Time complexities are often provided as theoretical guarantees, which highlight the role of certain parameters as efficiency parameters.

3.2 Effectiveness

Effectiveness can be empirically measured by comparing the test set performance of the unlearned model to the test set performance of the naïve retrained model. Let y^*_pred and $y^\mathcal{U}_\text{pred}$ be predictions from the naïve retrained and unlearned model on a test set y_test. Given a real-valued performance metric $\mathcal{M}$, let $\mathcal{M}^*_\text{test} = \mathcal{M}(y^*_\text{pred}, y_\text{test})$ and $\mathcal{M}^\mathcal{U}_\text{test} = \mathcal{M}(y^\mathcal{U}_\text{pred}, y_\text{test})$ denote the test set performance, respectively, of the naïve retrained model and the unlearned model. An example of a measure of the effectiveness of a machine unlearning algorithm $(\mathcal{U}, \mathcal{C}_\mathcal{U})$ is then given by the absolute difference of these two quantities:

$$\texttt{Effectiveness}(h^\mathcal{U}) := |\mathcal{M}^\mathcal{U}_\text{test} - \mathcal{M}^*_\text{test}|. \tag{3.2}$$

The smaller `Effectiveness` is, the closer in performance the unlearned model is to the naïve retrained model, and so smaller values of `Effectiveness` are preferable.

3.3 Consistency

Consistency is a measure of how close an unlearned model is to the naïve retrained model; that is, a measure of whether the unlearning algorithm is producing the ideal machine learning model as its output. Exact unlearning methods are guaranteed a high level of consistency. In contrast to efficiency and effectiveness, the literature we consider contains a number of measures for consistency, which differ in their usage and applicabilities. In this section we include three different measures.

Consistency of a machine unlearning algorithm $(\mathcal{U}, \mathcal{C}_{\mathcal{U}})$ applied to *parametric models* can be measured by the distance between the unlearned and naïve retrained parameters:

$$\texttt{Consistency}_\theta(h_\theta^{\mathcal{U}}) := \|\theta^{\mathcal{U}} - \theta^*\|_2, \tag{3.3}$$

where $\theta^{\mathcal{U}}$ and θ^* are the parameters of $h^{\mathcal{U}}$ and h^* respectively, and $\|\cdot\|_2$ denotes the ℓ^2-norm. This is used, for example, in Wu et al. (2020). Lower values of $\texttt{Consistency}_\theta$ are preferable as they indicate that the parameters of the unlearned model are close to those of the naïve retrained model.

In He et al. (2021), a measure of consistency that is applicable to classification models is considered. Consistency is given as the proportion of predictions that agree between naïve retrained and unlearned:

$$\texttt{Consistency}_\text{y}(h^{\mathcal{U}}) := \frac{100}{n_\text{test}} \sum_{i=1}^{n_\text{test}} 1_{y^*_{\text{pred},i} = y^{\mathcal{U}}_{\text{pred},i}}, \tag{3.4}$$

where $y^*_{\text{pred},i}$ and $y^{\mathcal{U}}_{\text{pred},i}$ are the naïve retrained and unlearned model predictions, respectively, on a test set containing n_test samples. Higher values of $\texttt{Consistency}_\text{y}$ are preferable, since they indicate that test-set predictions of the unlearned model are similar to those of the naïve retrained model. Note that lower consistency may be acceptable if effectiveness and certifiability remain high. In this case, the correct predictions of the unlearned model differ but high performance on a test set is maintained; consistency, particularly $\texttt{Consistency}_\text{y}$, is still useful in this situation as a measure of divergence from the naïve retrained model's predictions but it does not necessarily indicate an undesirable machine unlearning algorithm. The $\texttt{Consistency}_\text{y}$ measure is limited to classification models only, however analogues for regression tasks may be considered.

In Golatkar et al. (2019), the Kullback-Leibler (KL) divergence is minimised as part of the loss function. KL-divergence measures the similarity of two probability distributions P and Q as follows

$$\text{KL}(P\|Q) := \mathbb{E}_{X \sim P}\left[\log\left(\frac{p(X)}{q(X)}\right)\right],$$

where p and q, respectively, are the probability functions (either mass or density) of P and Q. By Gibbs' inequality, we have $\text{KL}(P\|Q) \in [0, \infty)$. Smaller values of KL give higher similarity between P and Q. This gives a measure of consistency of an unlearning algorithm applied by measuring the KL-divergence of the unlearned model distribution to that of the naïve retrained model:

$$\texttt{Consistency}_\text{KL}(h^{\mathcal{U}}) := \text{KL}(\Pr[h^{\mathcal{U}}(x)] \,\|\, \Pr[h^*(x)]). \tag{3.5}$$

Lower values of $\texttt{Consistency}_\text{KL}$ are preferable. The KL-divergence may also be used to measure similarity of probability outputs between the unlearned model and the naïve retrained model, when applied to classification tasks.

3.4 Certifiability

Certifiability is assurance that the unlearned model has removed information about the deleted data point and that an unlearning request has been fully complied to with respect to regulations. An important part of this involves quantifying and minimising vulnerability to inference attacks on the deleted data, which involve the identification of the deleted data by external attackers. Exact unlearning methods are guaranteed to remove the influence of the data point to be forgotten, however they may still be vulnerable to inference attacks

9

based around the deleted data points. In such a case, the naïve retrained model is also equally vulnerable to such inference attacks. To minimise this vulnerability, changes may need to be made to the training procedure, such as the addition of noise to gradient updates.

Consistency, from the previous section, is something of a precursor to certifiability, and certifiability was developed in order to address the insufficiency of consistency for guaranteeing deletion, see, for example, Guo et al. (2020, p. 2). We keep them separate because some definitions and measures of certifiability do not guarantee consistency and, moreover, the notion of certifiability is slightly removed from that of consistency on a fundamental level. Consistency is a correctness guarantee, whereas certifiability is a security guarantee.

Measures and guarantees of certifiability differ greatly across the literature. Often, only theoretical guarantees are given and proven, such as ϵ- and (ϵ, δ)-certifiability (Definitions 4, 5) in Guo et al. (2020). Empirical measures are given in He et al. (2021), Mahadevan and Mathioudakis (2021), and Golatkar et al. (2019), which we outline in the rest of this section.

The accuracy of the naïve retrained model on the removed data D_u quantifies the amount of information that the model can be expected to have on the removed data, having never seen this data. This might be quite high if the remaining data contains similar data, for example. This accuracy is therefore a baseline with which the accuracy of the unlearned model on D_u can be compared, as done in Mahadevan and Mathioudakis (2021). Here, we extend this to a general performance metric $\mathcal{M}$. Let $\mathcal{M}_u^* := \mathcal{M}(y_u^*, y_u)$ and $\mathcal{M}_u^{\mathcal{U}} := \mathcal{M}(y_u^{\mathcal{U}}, y_u)$ denote the performance on D_u, respectively, of the naïve retrained model and the unlearned model after unlearning D_u. Certifiability is measured by:

$$\texttt{Certifiability}(h^{\mathcal{U}}) := \frac{|\mathcal{M}_u^{\mathcal{U}} - \mathcal{M}_u^*|}{|\mathcal{M}_u^{\mathcal{U}}| + |\mathcal{M}_u^*|} \cdot 100. \tag{3.6}$$

The right-hand side of Equation (3.6) is called the *symmetric absolute percentage error* – $\mathrm{SAPE}(\mathcal{M}_u^{\mathcal{U}}, \mathcal{M}_u^*)$ – of $\mathcal{M}_u^*$ and $\mathcal{M}_u^{\mathcal{U}}$. It is used here to penalise deviations more when $\mathcal{M}_u^*$ is small, guided by the intuition that $\mathcal{M}_u^* = 0\%$ and $\mathcal{M}_u^{\mathcal{U}} = 10\%$, say, represents perhaps a more serious breach of information than $\mathcal{M}_u^* = 80\%$ and $\mathcal{M}_u^{\mathcal{U}} = 90\%$.

Large values of $\texttt{Certifiability}$ imply that the unlearned model contains information about the deleted data that it should not, so this measure can indicate degradation of certifiability between models, or of a single model over time. However, since $\texttt{Certifiability}$ measures only the similarity of performance on the deleted set, small values do not necessarily imply high levels of certifiability such as those seen in theoretical guarantees. For example, if the naïve unlearned model has high predictive performance on D_u and $\texttt{Certifiability}$ is small, then the success rate at predicting on D_u is similar for all three models – original, naïve, and unlearned. In this case, small $\texttt{Certifiability}$ does not distinguish whether the unlearned model has forgotten the data, since it will achieve high performance on D_u whether D_u is present in the training data or not. To address this issue, He et al. (2021) adapt and apply the backdoor verification experiment from Sommer et al. (2020). In this experiment, D_u is "augmented" and a specific target label is associated to this augmented data; such augmented data and corresponding label is called backdoor data. The result is that a model is likely to predict the correct specific backdoor label only if it has been trained on D_u. Hence the naïve retrained model is guaranteed to have poor predictive performance on D_u, which allows $\texttt{Certifiability}$ to be a more meaningful measure of certifiability.

In Golatkar et al. (2019), the Shannon Mutual Information is used as a measure of certifiability. This measures the amount of information that two random variables X and Y share and is defined by

$$I(X;Y) := \mathbb{E}_x[\mathrm{KL}(P_{Y|X} \| P_Y)], \tag{3.7}$$

where $P_{Y|X}$ is the conditional probability distribution of Y with respect to X, and P_Y is the probability distribution of Y. By treating the removal subset D_u as a random variable (through, for example, random selection of removal points), the mutual information that D_u and the unlearned model share is given by $I(D_u; \mathcal{U}(\mathcal{A}(D), D, D_u))$. It is shown (Golatkar et al., 2019, Proposition 1, Lemma 1) that this mutual information is bounded from above by a specific KL-divergence that is more readily computable. In particular, once D_u is chosen and fixed, the consistency measure $\texttt{Consistency}_{\mathrm{KL}}$ of (3.5) provides an empirical upper bound on the mutual information. This procedure is specific to their method and is not applicable in general.

4 Exact Unlearning Algorithms

In this section we describe two exact unlearning algorithms. First we describe the SISA algorithm, which is one of the most broadly applicable methods we include, before considering a highly specialised algorithm called DaRE, which applies only to decision-tree based machine learning models.

4.1 SISA

The SISA algorithm is an exact unlearning algorithm introduced in Bourtoule et al. (2021), borrowing aspects from both ensemble learning and distributed training to efficiently unlearn. This is achieved by a reorganisation of the training dataset, known as sharding and slicing, which reduces the time needed to retrain from scratch with the specified data removed.

The full SISA algorithm is applicable to any machine learning model that has been trained incrementally, for example, via gradient descent. The loss function for such models need not be strongly convex, and Bourtoule et al. (2021) apply SISA successfully to deep neural networks (DNN) in a variety of architectures. SISA without slicing is applicable to all machine learning models, including decision trees.

4.1.1 Methodology. Bourtoule et al. (2021) define the SISA training process to consist of four key steps – Sharded, Isolated, Sliced, and Aggregated – which can be seen in Figure 1 and which are described below. The resultant machine learning model is an ensemble of weak learners. Pseudocode for the SISA training algorithm can be found in Algorithm 1.
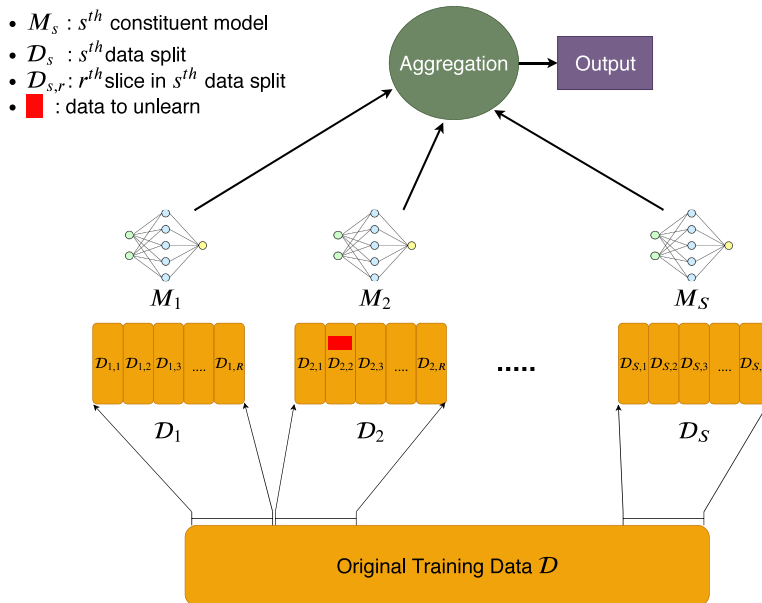

Fig. 1: SISA training, taken from Bourtoule et al. (2021). The training data is split into S shards, which are further split into R slices. S independent models are trained incrementally on the slices, and predictions of these models are aggregated to form a final output. The data to unlearn is highlighted in red in this diagram. To unlearn this data point, only M_2 needs to be retrained, and this process starts from slice $\mathcal{D}_{2,2}$. In our notation (Algorithms 1, 2) model M_2 is denoted by h_2, and the saved intermediary model state $\tilde{h}_{2,j}$, for each $1 \leq j \leq R$, correspond to the model trained on the slices $\bigcup_{r \leq j} \mathcal{D}_{2,r}$.

(i) **Sharded.** The original training dataset is separated into approximately equal-sized shards, with each training data point contained in exactly one shard. This contrasts with other ensemble modelling techniques, in which training data points can be present in many of the learners that make up the ensemble model.

(ii) **Isolation.** Each of the shards is trained in isolation from the other shards, restricting the influence of each data point to a single shard.

(iii) **Slicing.** Each of the shards are sub-divided into slices, which are presented to the algorithm incrementally as training proceeds. The trained model states are saved after each slice.

(iv) **Aggregation.** To form the final model prediction for a data point, the predictions of each sharded model are aggregated, which can be done through a variety of methods.

Whenever a removal request for a single data point comes in, only the model trained on the shard containing the particular data point needs to be retrained and, moreover, retraining needs only begin from the slice containing the data point. As a result, the expected retraining time is faster compared to naïve retraining; the exact speed-up depends on the number of shards and slices used, as discussed below. Algorithms 1 and 2 contain pseudocode for the training and unlearning procedures, which assume no prior knowledge of the distribution of deletion requests, choosing shards and slices randomly. Prior knowledge of the distribution of deletion requests allows improvement of the SISA method and is discussed in Bourtoule et al. (2021, Section VIII). In Algorithms 1, 2, the function $\text{TRAIN}(D \mid \tilde{h})$ produces a model by initialising at the model state $\tilde{h}$ and training on the dataset D.

Several approaches to aggregating during the inference phase are possible. A simple label-based majority vote involves each constituent model contributing equally to the ensemble model's final decision. Other options include averaging the output vectors of class or continuous predictions from each individual model. The latter method was not found to have a noticeable effect on accuracy for simple datasets, compared to the label-based majority vote, however mean vector aggregation has higher accuracy for the more complex ImageNet task (Bourtoule et al., 2021, Figure 13, 14).

Algorithm 1 Initial training with SISA.

Input: training data D, number of shards S, number of slices R, number of epochs for each slice **e**.
Output: ensemble of models $h = (h_1, \ldots, h_S)$ and intermediary model states $\tilde{h} = (\{\tilde{h}_{i,0}, \ldots, \tilde{h}_{i,R}\})_{i=1}^S$.

1: **procedure** $\text{SISATRAIN}(D; S, R, \mathbf{e})$
2:     split the data randomly into shards $D_1, \ldots, D_S$ and save shard indices for each data point
3:     split each shard D_i randomly into R slices $D_{i,1}, \ldots D_{i,R}$ and save slice indices for each data point
4:     randomly initialise $(\tilde{h}_{1,0}, \ldots, \tilde{h}_{S,0})$
5:     **for** $i = 1; i \leq S; i++$ **do**
6:         **for** $j = 1; j \leq R; j++$ **do**
7:             $h_{i,j} \leftarrow \text{TRAIN}\left(D_{i,1} \cup \cdots \cup D_{i,j} \mid \tilde{h}_{i,j-1}\right)$ for e_j epochs
8:             save model state $\tilde{h}_{i,j}$ of model $h_{i,j}$
9:         **end for**
10:        $h_i \leftarrow h_{i,R}$
11:     **end for**
12:     **return** $h = (h_1, \ldots, h_S)$, $\tilde{h}^{\mathcal{U}} = (\{\tilde{h}_{i,0}, \ldots, \tilde{h}_{i,R}\})_{i=1}^S$.
13: **end procedure**

4.1.2 Performance. The datasets used for the evaluation of the SISA algorithm include the Purchase (Sakar et al., 2018), SVHN (Netzer et al., 2011), and the ImageNet (Deng et al., 2009) datasets.

Efficiency. With 20 shards and 50 slices per shard, 8 unlearning requests on the Purchase dataset resulted in an `Efficiency` of 4.63×, with 18 unlearning requests on SVHN resulting in a `Efficiency` of 2.45×. The removal of 39 data points from the more complex ImageNet learning task gave a `Efficiency` of 1.36×.

The number of shards, S, is an efficiency parameter. As discussed in Bourtoule et al. (2021, Section VII.A), increasing the number of shards increases the efficiency of SISA, but will degrade the predictive performance of the resultant machine learning model compared to a lower number of shards. For the simpler learning tasks tested (SVHN and Purchase datasets) increasing $S > 20$ entails a more noticeable drop in accuracy. Increasing the number of slices in each shard, R, reduces the retraining time but this does not

Algorithm 2 SISA removal mechanism.

Input: ensemble of models $h = (h_1, \ldots, h_S)$, saved intermediary states $\tilde{h} = (\{\tilde{h}_{i,0}, \ldots, \tilde{h}_{i,R}\})_{i=1}^{S}$, data D with saved shard-slice indices, removal data D_u, number of shards S, number of slices R, epochs $\mathbf{e}$.

Output: unlearned model $h^{\mathcal{U}} = (h_1^{\mathcal{U}}, \ldots, h_S^{\mathcal{U}})$ and intermediary model states $\tilde{h}^{\mathcal{U}} = (\{\tilde{h}_{i,0}^{\mathcal{U}}, \ldots, \tilde{h}_{i,R}^{\mathcal{U}}\})_{i=1}^{S}$.

1: **procedure** SISAUNLEARN(h, $\tilde{h}$, D, D_u; S, R, $\mathbf{e}$)
2:     **for** $i = 1; i \leq S; i++$ **do**
3:         initialise model $h_{i,R}^{\mathcal{U}} \leftarrow h_i$ and states $\{\tilde{h}_{i,0}^{\mathcal{U}}, \ldots, \tilde{h}_{i,R}^{\mathcal{U}}\} \leftarrow \{\tilde{h}_{i,0}, \ldots, \tilde{h}_{i,R}\}$
4:         **if** $\exists z \in D_u$ with shard index i **then**
5:             $r_i \leftarrow$ find minimal slice index for all $z \in D_u$ that have shard index i
6:             **for** $j = r_i; j \leq R; j++$ **do**
7:                 $D_{i,j}' \leftarrow D_{i,j} \setminus (D_u \cap D_{i,j})$
8:                 $h_{i,j}^{\mathcal{U}} \leftarrow \text{TRAIN}\left(D_{i,1} \cup \cdots \cup D_{i,r_i-1} \cup D_{i,r_i}' \cup \cdots \cup D_{i,j}' \mid \tilde{h}_{i,j-1}^{\mathcal{U}}\right)$ for e_j epochs
9:                 save model state $\tilde{h}_{i,j}^{\mathcal{U}}$ for $h_{i,j}^{\mathcal{U}}$
10:             **end for**
11:         **end if**
12:         $h_i^{\mathcal{U}} \leftarrow h_{i,R}^{\mathcal{U}}$
13:     **end for**
14:     **return** $h^{\mathcal{U}} = (h_1^{\mathcal{U}}, \ldots, h_S^{\mathcal{U}})_i$, $\tilde{h}^{U} = (\{\tilde{h}_{i,0}^{\mathcal{U}}, \ldots, \tilde{h}_{i,R}^{\mathcal{U}}\})_{i=1}^{S}$
15: **end procedure**

---

degrade accuracy provided that the epochs in training are carefully chosen (Bourtoule et al., 2021, Sections V, VII). However, an increase in R does come at increased storage costs due to the increased number of saved model states. As discussed in *ibid.*, (Appendix C), the efficiency-storage trade-off of R may be preferable to the efficiency-effectiveness trade-off of S. Bourtoule et al. (2021) derive a maximum theoretical efficiency of $\frac{(R+1)S}{2}\times$.

Effectiveness. The experiments on simple tasks – Purchase and SVHN – outlined above resulted in `Effectiveness` values of less than 2%. The more complex ImageNet removal observed a larger `Effectiveness` of 18.76%.

Consistency. Consistency is guaranteed by the exact nature of the SISA removal mechanism.

Certifiability. Bourtoule et al. (2021) argue that as the SISA mechanism inherently retrains without the removed data, this shows that SISA is exact. The certificate of unlearning is therefore provided by exactness in this case. However, if an attacker has access to the model before and after an unlearning request is made, then the SISA method is vulnerable to having information about the deleted data points inferred, as discussed in Golatkar et al. (2019).

4.2 DaRE Forests

In 'Machine Unlearning for Random Forests', Brophy and Lowd (2021) introduce an exact (*ibid.*, Theorem 3.1) unlearning algorithm that is specific to decision-tree and random-forest based machine learning models for binary classification. This is done through the development of **D**a**t**a **R**emoval-**E**nabled (**DaRE**) *trees*, and the ensemble of these to form *DaRE Forests* (**DaRE RF**). Through the use of strategic thresholding at decision nodes for continuous attributes, high-level random nodes, and caching certain statistics at all nodes, DaRE trees enable efficient removal of training instances.

4.2.1 Methodology. The DaRE forest is an ensemble consisting of DaRE trees in which each tree is trained independently on a copy of the training data, which differs from the data bootstrapping of traditional

random forests. Aside from this, the DaRE forest ensembles in the same way as a random forest, in particular a random subset of $\tilde{p}$ features are considered at each split.

In the rest of this section, we focus on DaRE trees, which must be initially trained in the DaRE methodology of Algorithm 3. As in regular decision trees, DaRE trees are trained recursively by selecting, at most nodes, an attribute and threshold that optimises a *split criterion* (Gini index in Brophy and Lowd (2021)). They differ from regular decision trees in three key ways as follows.

(i) **Random nodes.** The top d_{rmax} levels of nodes in a DaRE tree are random nodes, where d_{rmax} is an integer hyperparameter. Random nodes are defined as nodes for which both an attribute a and a threshold in the attribute range $[a_{\mathrm{min}}, a_{\mathrm{max}})$ are chosen uniformly at random.

(ii) **Threshold sampling.** During training and deletion, DaRE trees randomly sample k *valid* thresholds at any node that is neither a random node nor a leaf. These are thresholds that lie between two adjacent data points with opposite labels. Doing so reduces the amount of statistics one needs to store at each node and speeds up computation.

(iii) **Statistics caching.** At each node, the number of data points $|D_{\mathrm{node}}|$ and $|D_{\mathrm{node},1}|$ are stored and updated, where D_{node} is the input dataset to the node and $D_{\mathrm{node},1} \subseteq D_{\mathrm{node}}$ is the subset of positive instances. For each of the k candidate valid thresholds v, various additional statistics are stored and updated. The exact form of these depends on the type of node (see Brophy and Lowd, 2021, Appendix A.6). In each case these statistics are sufficient to recompute the split criterion scores and to determine the validity of the current thresholds. At leaf nodes, pointers to the training data at that leaf are stored. As a result, the removal mechanism is able to recall training data from the stored leaf instances, meaning that training data is not required as an explicit input to the mechanism.

Deletion of a data point $\mathbf{z}$ is given in Algorithm 4. Only those subtrees that have been trained on $\mathbf{z}$ are considered. At each decision node that is neither random nor a leaf, statistics are recalculated and the split criterion is updated for each attribute-threshold pair. If a different optimal threshold is found then the subtree rooted at this node is retrained. At the relevant leaf node, the pointer to $\mathbf{z}$ is deleted. Random nodes are rarely retrained, only if the randomly chosen threshold no longer lies within the attribute range $[a_{\mathrm{min}}, a_{\mathrm{max}})$.

The DaRE RF inherits the following hyperparameters of random forests: $\tilde{p}$ is the number of random attributes to consider at each split; d_{max} is the maximum depth for each tree; T is the number of trees in the forest. DaRE RFs have two additional hyperparameters: k is the number of random valid thresholds to sample at each split; d_{rmax} is the number of top layers that are used for random nodes.

4.2.2 Performance. The authors evaluate DaRE RFs on 13 real-world binary classification datasets and one synthetic dataset via efficiency and effectiveness; the proof of exactness, (Brophy and Lowd, 2021, Theorem 3.1), covers certifiability.

For the experiments considered by Brophy and Lowd (2021), hyperparameters are chosen or tuned as follows. The parameter $\tilde{p}$ is set at $\lfloor \sqrt{p} \rfloor$. The parameters d_{max}, T, and k are tuned using 5-fold cross-validation, with $d_{\mathrm{rmax}} = 0$ fixed, to maximise the relevant performance metric for the task at hand, using the Gini index as the split criterion. The parameter d_{rmax} is incremented from $d_{\mathrm{rmax}} = 0$ until the 5-fold cross-validation score breaches four separate absolute error loss tolerances $\{0.1, 0.25, 0.5, 1.0\}$ relative to the DaRE RF with $d_{\mathrm{rmax}} = 0$, yielding four DaRE RFs with varying proportions of random nodes. Larger error tolerances lead to larger values of d_{rmax} (Brophy and Lowd, 2021, Appendix B.2, Table 6). Performance of all five DaRE RF models – $d_{\mathrm{rmax}} = 0$ and the four tuned d_{rmax} – are considered.

Efficiency. Efficiency is measured as the number of training instances that can be deleted by DaRE RF in the time it takes to delete one instance via naïve retraining, which is equivalent to Efficiency (3.1). When deleting points at random, the average Efficiency of DaRE RFs is 2–4 orders of magnitude. In the best case, DaRE RF achieves a Efficiency of 35,856×, achieved on the HIGGS dataset and with the highest proportion of random nodes; the average Efficiency across all 14 datasets for the highest d_{rmax} DaRE RF

Algorithm 3 DaReTrain($D, 0; d_{\text{rmax}}, k$) trains a single DaRE tree (Brophy and Lowd, 2021).

Input: data D_{node}, depth d, random node depth d_{rmax}, threshold candidate size k.
Output: trained subtree rooted at a level-d node.

1: **procedure** DaReTrain($D_{\text{node}}, d; d_{\text{rmax}}, k$)
2:     **if** stopping criteria reached **then**
3:         node $\leftarrow$ LeafNode()
4:         save instance counts $|D_{\text{node}}|, |D_1|$
5:         save leaf-instance pointers($node, D_{\text{node}}$)
6:         compute leaf value($node$)
7:     **else**
8:         **if** $d < d_{\text{rmax}}$ **then**
9:             node $\leftarrow$ RandomNode()
10:             save instance counts $|D_{\text{node}}|, |D_{\text{node},1}|$
11:             $a \leftarrow$ randomly sample attribute(D_{node})
12:             $v \leftarrow$ randomly sample threshold $\in [a_{\text{min}}, a_{\text{max}})$
13:             save threshold statistics(node, D_{node}, a, v)
14:         **else**
15:             node $\leftarrow$ GreedyNode()
16:             save instance counts $|D_{\text{node}}|, |D_{\text{node},1}|$
17:             $A \leftarrow$ randomly sample $\tilde{p}$ attributes(D_{node})
18:             **for** $a \in A$ **do**
19:                 $C \leftarrow$ get valid thresholds(D_{node}, a)
20:                 $V \leftarrow$ randomly sample k valid thresholds(C)
21:                 **for** $v \in V$ **do**
22:                     save threshold statistics(node, D_{node}, a, v)
23:                 **end for**
24:                 $scores \leftarrow$ compute split scores(node)
25:                 select optimal split(node, $scores$)
26:             **end for**
27:         **end if**
28:         $D_{\text{left}}, D_{\text{right}} \leftarrow$ split on selected threshold(node, D_{node})
29:         node.left $=$ DaReTrain($D_{\text{left}}, d + 1; d_{\text{rmax}}, k$)
30:         node.right $=$ DaReTrain($D_{\text{right}}, d + 1; d_{\text{rmax}}, k$)
31:     **end if**
32:     **return** node
33: **end procedure**

---

is $1{,}272\times$. A worse-case deletion strategy called *worst-of-1000* is also considered, giving average `Efficiency` of 1–3 orders of magnitude.

Note that DaRE RF training has the same time complexity, $\mathcal{O}(T\tilde{p}nd_{\text{max}})$, as the training for a traditional random forest (Brophy and Lowd, 2021, Theorem 3.2). Time complexities for deleting a single instance are given in *ibid.* (Theorem 3.3); the best-case time complexity occurs when the tree structure is unchanged and all attribute thresholds remain valid, which is $\mathcal{O}(\tilde{p}kd_{\text{max}})$, with additional costs occurring when thresholds become invalid and subtrees require retraining.

Effectiveness. Test set performance of each DaRE RF is compared to the DaRE RF with no random nodes. In most cases, `Effectiveness` $< 1\%$. Larger proportions of random nodes generally degrades performance, particularly for the Credit Card and HIGGS datasets, for which we see `Effectiveness` $\approx 2\%$.

In Brophy and Lowd (2021, Appendix B.2.), the predictive performances of the various DaRE RFs are compared to the scikit-learn implementation of random forests. As shown in Table 5, *ibid.*, DaRE RFs with random nodes have worse performance than the standard random forest across all datasets, but in some cases the DaRE RF with no random nodes improves on performance over the scikit-learn random forest.

Algorithm 4 Deleting a training instance from a DaRE tree, (Brophy and Lowd, 2021).

Require: start at the root node.
    Input: node, data point to remove $\mathbf{z}$, depth d, random node depth d_{rmax}, threshold candidate size k.
    Output: retrained subtree rooted at node.
1: **procedure** DARE UNLEARN(node, $\mathbf{z}$, d; d_{rmax}, k)
2:     update instance counts $|D_{\mathrm{node}}|$, $|D_{\mathrm{node},1}|$
3:     **if** node is a LEAFNODE **then**
4:         remove $\mathbf{z}$ from leaf-instance pointers(node, $\mathbf{z}$)
5:         recompute leaf value(node)
6:         remove $\mathbf{z}$ from database and return
7:     **else**
8:         update decision node statistics(node, $\mathbf{z}$)
9:         **if** node is a RANDOMNODE **then**
10:             **if** node.*selectedThreshold* is invalid **then**
11:                 $D_{\mathrm{node}} \leftarrow$ get data from the set of leaf instances(node) $\setminus \{\mathbf{z}\}$
12:                 **if** node.*selectedAttribute*(a) is not constant **then**
13:                     $v \leftarrow$ resample threshold $\in [a_{\min}, a_{\max})$
14:                     $D_{\mathrm{node},\ell}, D_{\mathrm{node},r} \leftarrow$ split on new threshold(node, D_{node}, a, v)
15:                     node.$\ell \leftarrow$ DARE TRAIN($D_{\mathrm{node},\ell}, d+1; d_{\mathrm{rmax}}, k$)
16:                     node.$r \leftarrow$ DARE TRAIN($D_{\mathrm{node},r}, d+1; d_{\mathrm{rmax}}, k$)
17:                 **else**
18:                     node $\leftarrow$ DARE TRAIN($D_{\mathrm{node}}, d; d_{\mathrm{rmax}}, k$)
19:                 **end if**
20:                 remove $\mathbf{z}$ from database and return
21:             **end if**
22:         **else**
23:             **if** $\exists$ invalid attributes or thresholds **then**
24:                 $D_{\mathrm{node}} \leftarrow$ get data from the set of leaf instances(node) $\setminus \{\mathbf{z}\}$
25:                 resample invalid attributes and thresholds(node, D_{node})
26:             **end if**
27:             *scores* $\leftarrow$ recompute split scores(node)
28:             $a, v \leftarrow$ select optimal split(node, *scores*)
29:             **if** optimal split has changed **then**
30:                 $D_{\mathrm{node.left}}, D_{\mathrm{node.right}} \leftarrow$ split on new threshold(node, D_{node}, a, v)
31:                 node.left $\leftarrow$ DARE TRAIN($D_{\mathrm{node.left}}, d+1; d_{\mathrm{rmax}}, k$)
32:                 node.right $\leftarrow$ DARE TRAIN($D_{\mathrm{node.right}}, d+1; d_{\mathrm{rmax}}, k$)
33:                 remove $\mathbf{z}$ from database and return
34:             **end if**
35:         **end if**
36:         **if** $x_a \leq v$ **then**
37:             DARE UNLEARN(node.left, $\mathbf{z}, d+1; d_{\mathrm{rmax}}, k$)
38:         **else**
39:             DARE UNLEARN(node.right, $\mathbf{z}, d+1; d_{\mathrm{rmax}}, k$)
40:         **end if**
41:     **end if**
42: **end procedure**

The level of random nodes in a DaRE RF, d_{rmax}, is an efficiency parameter, with larger values entailing faster unlearning at the cost of predictive performance. This is demonstrated in Brophy and Lowd (2021, Appendix B.2, Table 6), in which higher error tolerances lead to higher proportions of random nodes. Test set error is shown to increase dramatically once d_{rmax} gets too large (Brophy and Lowd, 2021, Figure 2). The number of valid thresholds to consider, k, is another efficiency parameter. Reducing k will increase efficiency, however predictive performance suffers (Brophy and Lowd, 2021, Figure 3). Predictive performance plateaus after $k = 5$ for the Surgical dataset, which gives an optimal choice of $k = 5$ in this case.

Consistency and certifiability. Consistency and certifiability of the DaRE algorithm are guaranteed by the exact nature of the removal mechanism, as proven by Brophy and Lowd (2021, Theorem 1.3). The certificate of unlearning for DaRE is therefore provided by exactness.

5 Approximate Unlearning Algorithms

Approximate unlearning algorithms originate from a proposal in the work of Ginart et al. (2019). In this section we describe five approximate methods, some of which build upon the ideas of Ginart et al. (2019). They are given in chronological order.

5.1 Fisher

The Fisher algorithm, introduced by Golatkar et al. (2019), applies to parametric models and is an approximate unlearning algorithm that updates the parameters of a pre-trained convex model in a time-efficient manner. The method facilitates the removal of the information about certain data without retraining the model from scratch, via a strategic mix of Newton correction and noise injection. Precise upper bounds on the amount of information that can be extracted about the removed data can be calculated.

5.1.1 Methodology. Golatkar et al. (2019) develop a batch unlearning removal mechanism in which a subset of the data $D_u \subseteq D$ is unlearned from a trained parametric model. They formalise the definition of "complete" forgetting by introducing the condition that forgetting of information can only be guaranteed if the KL divergence, (3.5), between the probability distributions of outputs from the unlearned and the naïvely retrained models, is zero. By Golatkar et al. (2019, Proposition 1), this also ensures that there is zero Shannon Mutual Information, $I(D_u; \mathcal{U}(h_\theta, D, D_u)) = 0$, as defined in Equation (3.7), between the deleted data D_u and the unlearned model, $\mathcal{U}(h_\theta, D, D_u)$. The authors therefore stress on minimising the KL divergence as part of the unlearning objective function. This can be achieved by updating the parameters of the initial model through a single Newton step on the remaining data, or by adding Gaussian noise in the direction of this Newton step, or both (see Golatkar et al., 2019, Corollary 1). Upon Newton correction and noise injection to the weights of the original model, the Fisher training and unlearning method are respectively expressed as:

$$\theta := \text{SGD}(L(\theta_{\text{init}}, D)) + \sigma F^{-\frac{1}{4}}\mathbf{b}, \tag{5.1}$$

$$\theta^{\mathcal{U}} := \underbrace{\theta - F^{-1}\Delta_{\text{rem}}}_{\text{Newton step}} + \underbrace{\sigma F^{-1/4}\mathbf{b}}_{\text{noise injection}}, \tag{5.2}$$

where SGD is the stochastic gradient descent algorithm; θ_{init} are the initialised parameters; $\theta^{\mathcal{U}}$ are the unlearned parameters; $\sigma > 0$ is the noise parameter; F is the Fisher Information Matrix (Golatkar et al., 2019, Eq. (8)); $\mathbf{b} \sim \mathcal{N}(0,1)^p$ is Gaussian noise, where recall that p is the dimensionality of the training data; and

$$\Delta_{\text{rem}} := \nabla L(\theta, D \setminus D_u), \tag{5.3}$$

denotes the gradient of the loss function of θ on the remaining data. The Fisher Information matrix is used as an approximation to the Hessian as the Fisher matrix is less expensive to compute. In some cases, such as when the loss function is the log-likelihood (for example, Mahadevan and Mathioudakis (2021) who consider linear logistic regression), the Fisher Information Matrix and Hessian coincide. The algorithm for the deletion of data points in mini-batches from (Mahadevan and Mathioudakis, 2021) using this method is shown in Algorithm 5.

Algorithm 5 Fisher removal mechanism, (Golatkar et al., 2019; Mahadevan and Mathioudakis, 2021).

    Input: trained model parameters θ, training dataset D, subset of data to be removed D_u, noise parameter σ, mini-batch size m'.
    Output: unlearned model parameters $\theta^{\mathcal{U}}$.
 1: **procedure** FISHERUNLEARN(θ, D, D_u; σ, m')
 2:     assign the number of batches $s \leftarrow \lceil \frac{m}{m'} \rceil$ (m is the number of samples in D_u)
 3:     split D_u into s mini-batches D_u^1, D_u^2,...,D_u^s, each of size m'
 4:     initialise $D' \leftarrow D$
 5:     initialise $\theta^{\mathcal{U}} \leftarrow \theta$
 6:     **for** $i = 1; i \leq s; i + +$ **do**
 7:        $D' \leftarrow D' \setminus D_u^i$
 8:        $\Delta \leftarrow \nabla L(\theta^{\mathcal{U}}, D')$
 9:        $F \leftarrow$ compute Fisher Information Matrix of L and D', (Golatkar et al., 2019, Eq. (8))
10:        $\theta^{\mathcal{U}} \leftarrow \theta^{\mathcal{U}} - F^{-1}\Delta$
11:        **if** $\sigma > 0$ **then**
12:           draw $\mathbf{b} \sim \mathcal{N}(0,1)^p$
13:           $\theta^{\mathcal{U}} \leftarrow \theta^{\mathcal{U}} + \sigma F^{-1/4}\mathbf{b}$
14:        **end if**
15:     **end for**
16:     **return** $\theta^{\mathcal{U}}$
17: **end procedure**

5.1.2   Performance.

Efficiency. Apart from the original paper, Mahadevan and Mathioudakis (2021) evaluate the performance of the Fisher method across datasets varying in dimensionality and size. They show that the Fisher method is a moderately efficient unlearning method across the range of dimensionalities (ranging from 28 to 3072) and sizes (from $\sim 10^4$ to $\sim 10^6$) of the datasets. Controlling for the noise parameter, $\sigma = 1$, and using only one mini-batch, they found that the method offered an `Efficiency` of up to $50\times$ for the low-dimensional datasets while only around $1.8\times$ for the high-dimensional datasets. The large difference in the algorithm's performance across different dimensionalities is due to the difference in the cost of computing the inverse Hessian matrix, which is much faster to calculate when the dimensionality is low.

The mini-batch size m' used by Mahadevan and Mathioudakis (2021) is an efficiency parameter. It controls a trade-off between efficiency and both effectiveness and certifiability (*ibid.* (Figures 3, 4)), since a smaller number of batches (large mini-batch size) ensures that the Fisher matrix is calculated a fewer number of times at the cost of fewer Newton steps. When measuring the effectiveness-efficiency trade-off, the authors find that the method suffers a huge loss in efficiency, especially in the high dimensional setting where the `Efficiency` of the method reduced to as low as $0.4\times$. The authors attribute this poor efficiency to the computational effort it takes to inject noise post update, which takes longer when the dimensionality is high. In the low dimensionality setting, `Efficiency` reduced to $9\times$, and high effectiveness was maintained.

Effectiveness. In Golatkar et al. (2019, Table 1), effectiveness of the Fisher method when unlearning a subset of 100 images from the Lacuna-10 and CIFAR-10 datasets is approximately $4 - 5\%$ in each case, measured with `Effectiveness`. This translates to approximately $2 - 3\%$ in *percentage error*, which is the measure used

by Mahadevan and Mathioudakis (2021, Eq. (19)), who achieve percentage error of $\leq 1\%$ across all 6 tested datasets. The better effectiveness observed in Mahadevan and Mathioudakis (2021) results from the use of mini-batches in the update stage.

Consistency. Consistency is given in Golatkar et al. (2019, Proposition 2) by providing a computable upper bound for the KL-divergence measure $\texttt{Consistency}_{\text{KL}}$ of (3.5). This bound is 3.3 for the Lacuna-10 dataset and 33.4 for CIFAR-10. Consistency is not measured in Mahadevan and Mathioudakis (2021).

Certifiability. As shown by Golatkar et al. (2019), the KL-divergence measure (3.5) is an upper bound for the Shannon Mutual Information. Therefore, the Shannon Mutual Information between D_u and the unlearned model is at most 3.3 kNATs for the Lacuna-10 dataset and 33.4 kNATs for CIFAR-10. In Mahadevan and Mathioudakis (2021, Figure 3), certifiability is shown using $\texttt{Certifiability}$ for different values of the noise parameter σ. For small volumes and $\sigma = 0$, $\texttt{Certifiability}$ mostly has values between 0% and 1%, however a notable large value of $\texttt{Certifiability} \approx 10\%$ is observed for the HIGGS dataset. Values of $\texttt{Certifiability}$ decrease for $\sigma = 1$, entailing better certifiability. Note, however, that the increase in σ degrades the effectiveness of the method since large amount of noise yields less optimal parameters and lower accuracy for such models.

5.2 Influence

The Influence algorithm, introduced by Guo et al. (2020), is an approximate unlearning algorithm that applies to parametric models and leverages the ML influence theory (Koh and Liang, 2020) to unlearn. Specifically, the method performs unlearning by quantifying the influence of the deleted data on the original model's parameters and applying a Newton update to these parameters to remove the identified influence of the deleted data. Despite employing a Newton update removal mechanism for unlearning similar to the Fisher method (Section 5.1), this method takes a different approach to noise injection in order to guarantee certifiability. More specifically, the noise in this method is added to the gradient loss at every step of the initial training of the model, unlike in the Fisher method in which noise is added at the end of the training and unlearning procedures. The authors justify this choice by stressing that the direction of the gradient residual at the time of parameter update may reveal information about the deleted data, which can be prevented by adding the noise at the time of training.

5.2.1 Methodology. The noise-injected objective function used for training this method is expressed as:

$$L_\sigma(\theta, D) := L(\theta, D) + \frac{\sigma \mathbf{b}^T \theta}{|D|}, \tag{5.4}$$

where σ is the noise control parameter, θ is the vector of model parameters, $\mathbf{b} \sim \mathcal{N}(0,1)^p$ is Gaussian noise.

Training for Influence optimises the noisy loss (5.4). Note that noise is added at every step of the optimisation procedure, in contrast to Fisher which adds noise only at the end. Moreover loss is only added during training time and not during the removal mechanism (where the original loss function $L(\theta, D)$ is used). To unlearn, a single Newton update on the model parameters θ is performed as follows:

$$\theta^{\mathcal{U}} := \theta + H^{-1}\Delta_u, \tag{5.5}$$

where

$$\Delta_u := \nabla L(\theta, D_u), \tag{5.6}$$
$$H := \nabla^2 L(\theta, D \setminus D_u), \tag{5.7}$$

denote, respectively, the gradient of the loss function on the deleted data and the Hessian of the loss function (without noise) on the remaining data. The so-called influence function $H^{-1}\Delta_u$ gives the direction, H, and

magnitude, Δ_u, of the step required to remove the influence of the deleted data from θ. The Influence removal mechanism is shown in Algorithm 6.

Algorithm 6 Influence removal mechanism, (Mahadevan and Mathioudakis, 2021).

    Input: trained model parameters θ, original train dataset D, subset of data to be deleted D_u, mini-batch size m'.
    Output: unlearned model parameters $\theta^{\mathcal{U}}$.
 1: **procedure** INFLUENCEUNLEARN(θ, D, D_u; m')
 2:    assign the number of batches $s \leftarrow \lceil \frac{m}{m'} \rceil$ (m is the number of samples in D_u)
 3:    split D_u into s mini-batches $D_u^1, D_u^2, ..., D_u^s$, each of size m'
 4:    initialise $D' \leftarrow D$
 5:    initialise $\theta^{\mathcal{U}} \leftarrow \theta$
 6:    **for** $i = 1; i \leq s; i{+}{+}$ **do**
 7:        $D' \leftarrow D' \setminus D_u^i$
 8:        $\Delta_{m'} \leftarrow \nabla L\left(\theta^{\mathcal{U}}, D_u^i\right)$
 9:        $H \leftarrow \nabla^2 L\left(\theta^{\mathcal{U}}, D'\right)$
10:        $\theta^{\mathcal{U}} \leftarrow \theta^{\mathcal{U}} + H^{-1}\Delta_{m'}$
11:    **end for**
12:    **return** $\theta^{\mathcal{U}}$
13: **end procedure**

5.2.2 Performance.

Efficiency. Mahadevan and Mathioudakis (2021) evaluate the performance of the Influence method across datasets varying in dimensionality and sizes of the training datasets, as described in Section 5.1. Controlling for the noise parameter, $\sigma = 1$, they found that the method offered a `Efficiency` of up to $200\times$ for the low-dimensional datasets while only around $5\times$ for the high-dimensional datasets on bulk removals. The large difference in the method's performance across different dimensionalities is due to the difference in the cost of computing the inverse Hessian matrix, which is much faster to calculate when the dimensionality is low.

As in the Fisher method, Section 5.1, the mini-batch size m' acts as an efficiency parameter, which is because smaller mini-batch sizes lead to a larger number of calculations of the Hessian of the loss function. The effect of the trade-off between efficiency and both effectiveness and certifiability is discussed in Mahadevan and Mathioudakis (2021, Figures 3, 4).

Effectiveness. Mahadevan and Mathioudakis (2021) found that despite being highly efficient, the Influence method did not compromise effectiveness too much. For example, Influence is no more than 1% less effective than Fisher when evaluated using the *percentage error* (*ibid.*, Eq. (19)).

Consistency. In Guo et al. (2020), consistency is measured by considering the ℓ^2-norm of the gradient residual $\|\nabla L(\theta^{\mathcal{U}}, D \setminus D_u)\|_2$. For the true optimiser $\min_\theta L(\theta, D \setminus D_u)$ this gradient residual is zero, hence the value $\|\nabla L(\theta^{\mathcal{U}}, D \setminus D_u)\|_2$ measures the error of the parameters $\theta^{\mathcal{U}}$ in approximating the true optimiser. Upper bounds for this gradient residual are provided in *ibid.* (Theorems 1, 2; Corollary 1, 2), which give non-data-dependent and data-dependent bounds for single and batch deletion. These are empirically verified on MNIST in *ibid.* (Fig. 2). Consistency is not measured in Mahadevan and Mathioudakis (2021).

Certifiability. For a desired $\epsilon > 0$, the Influence method is shown to be ϵ- and (ϵ, δ)-certified for an ϵ-dependent choice of σ given in Guo et al. (2020, Theorem 3). In Mahadevan and Mathioudakis (2021), certifiability is measured using `Certifiability`. Increasing σ allows for smaller ϵ and gives larger `Certifiability` values, thereby increases the certifiability of the model, however, as shown in Guo et al. (2020, Figure 1) and Mahadevan and Mathioudakis (2021, Figure 5), this comes at the cost of a degradation in effectiveness. It

is interesting to note that in Guo et al. (2020), test accuracy is relatively stable until a certain value of σ, after which it experiences dramatic decline.

5.3   DeltaGrad

The DeltaGrad algorithm is first described in Wu et al. (2020). It is an approximate unlearning algorithm that uses information cached from the initial training process to more efficiently compute model parameters after data points have been removed. DeltaGrad is only applicable to parametric machine learning models trained using gradient descent or mini-batch stochastic gradient descent, with loss functions that are strongly convex and smooth (Wu et al., 2020, Section 2.3).

5.3.1   Methodology. Throughout this section, $L(\theta) = L(\theta, D)$ denotes the empirical loss function on the full training dataset. An initial model is trained using a gradient descent algorithm, giving a vector of trained model parameters θ. At each step of training, t, the model parameters $\{\theta_0, \theta_1, ...\theta_t\}$ and the gradients of the loss function $\{\nabla L(\theta_0), \nabla L(\theta_1), ...\nabla L(\theta_t)\}$ are saved.

In contrast to the Fisher and Influence methods of Sections 5.1, 5.2 which unlearn by performing gradient descent steps starting from the original optimised model parameters, DeltaGrad performs gradient descent from the same initial parameters of the original training procedure and recalculates all gradient descent steps. The time to retrain is reduced from that of naïve retraining by calculating only some of the gradient descent steps exactly and approximating all other steps. Specifically, the first j_0 steps and every T_0 step thereafter are calculated explicitly, where $j_0, T_0 > 0$ are integer hyperparameters.

Gradient steps, when deleting m data points with indices in M (i.e., $D_u = \{\mathbf{z}_i \mid i \in M\} \subseteq D$), are approximated by first rewriting the gradient descent update formula as follows (Wu et al., 2020, Eq. (2)):

$$\theta_{t+1}^{\mathcal{U}} \leftarrow \theta_t^{\mathcal{U}} - \frac{\eta_t}{n-m}\left[n\nabla L(\theta_t^{\mathcal{U}}) - \sum_{i \in M}\nabla L_i(\theta_t^{\mathcal{U}})\right], \tag{5.8}$$

where n is the number of data points in D. The value $n\nabla L(\theta_t^{\mathcal{U}})$ is approximated, with the rest of the terms in (5.8) calculated explicitly; under the assumption $m \ll n$ the latter explicit calculation is comparatively inexpensive. The quantity $n\nabla L(\theta_t^{\mathcal{U}})$ can be expressed, using the Cauchy mean-value theorem (Wu et al., 2020, Eq. (3)), in terms of an integrated Hessian H_t. The L-BFGS algorithm[5] then approximates the vector product $H_t \cdot \mathbf{v}$ as a quasi-Hessian product $B_t \cdot \mathbf{v}$. Put together, this allows the gradient step of (5.8), to be approximated as:

$$\theta_{t+1}^{\mathcal{U}} \leftarrow \theta_t^{\mathcal{U}} - \frac{\eta_t}{n-m}\left[n(\nabla L(\theta_t) + B_t \cdot (\theta_t^{\mathcal{U}} - \theta_t)) - \sum_{i \in M}\nabla L_i(\theta_t^{\mathcal{U}})\right]. \tag{5.9}$$

To compute the quasi-Hessian B_t, L-BFGS uses k previously stored parameter and gradient differences, $\theta_j^{\mathcal{U}} - \theta_j$ and $\nabla L(\theta_j^{\mathcal{U}}) - \nabla L(\theta_j)$, where $k > 0$ is an integer hyperparameter. The two differences are stored only during the explicitly computed gradient steps, during the first j_0 steps and every T_0 step thereafter.

DeltaGrad is extended to mini-batch stochastic gradient descent in Wu et al. (2020, Section 3), with a randomly sampled mini-batch $\mathcal{B}$ of size B. Algorithm 4, *ibid.*, further extends DeltaGrad to DNNs by checking whether a loss function is convex and smooth locally. In this case, the arrays of historical parameter updates ΔG and gradients ΔW are only saved when local convexity holds and an exact GD update is used otherwise. This extension of the method adds an additional computational cost to unlearning.

The work of Mahadevan and Mathioudakis (2021, Equation 16) expands this algorithm slightly by following the removal mechanism, described in Algorithm 7, with the injection of a Gaussian noise vector controlled by a noise parameter, σ, in order to control the trade-off between effectiveness and certifiability.

---

[5]   https://link.springer.com/article/10.1007/BF01589116

Algorithm 7 DeltaGrad removal mechanism, (Wu et al., 2020)

Input: model training parameters $\boldsymbol{\theta} := \{\theta_0, \theta_1, ...\theta_t\}$, training data D, indices of removed training samples M, stored training gradients $\nabla L(\boldsymbol{\theta}) := \{\nabla L(\theta_0), \nabla L(\theta_1), ...\nabla L(\theta_t)\}$, period T_0, total iteration number T, history size k, burn-in iteration number j_0, learning rate η_t.

Output: unlearned model parameters $\theta^{\mathcal{U}} = \theta_T^{\mathcal{U}}$.

1: **procedure** DELTAGRADUNLEARN($\boldsymbol{\theta}$, D, M; $\nabla L(\boldsymbol{\theta})$, T_0, T, k, j_0, η_t)
2:     initialise $\theta_0^{\mathcal{U}} \leftarrow \theta_0$
3:     initialise an array $\Delta G \leftarrow []$
4:     initialise an array $\Delta \Theta \leftarrow []$
5:     $\ell \leftarrow 0$
6:     **for** $t = 0; t \leq T; ++$ **do**
7:         **if** $[(t_0 - j_0) \pmod{T_0} == 0]$ *or* $t \leq j_0$ **then**
8:             compute $\nabla L(\theta_t^{\mathcal{U}})$ exactly
9:             compute $\nabla L(\theta_t^{\mathcal{U}}) - \nabla L(\theta_t)$, using the cached gradient $\nabla L(\theta_t)$
10:            $\Delta G[\ell] \leftarrow \nabla L(\theta_t^{\mathcal{U}}) - \nabla L(\theta_t)$
11:            $\Delta \Theta[\ell] \leftarrow \theta_t^{\mathcal{U}} - \theta_t$
12:            $\ell \leftarrow \ell + 1$
13:            compute $\theta_{t+1}^{\mathcal{U}}$ by using exact GD update
14:         **else**
15:            $B_{j_k} \leftarrow$ L-BFGS($\Delta G[-k :], \Delta \Theta[-k :]$)
16:            $\nabla L(\theta_t^{\mathcal{U}}) \leftarrow \nabla L(\theta_t^{\mathcal{U}}) + B_{j_k}(\theta_t^{\mathcal{U}} - \theta_t)$ approximate the gradient
17:            compute $\theta_{t+1}^{\mathcal{U}}$ via the modified gradient formula, Eq. (5.9), using approximated $\nabla L(\theta_t^{\mathcal{U}})$
18:         **end if**
19:     **end for**
20:     **return** $\theta_t^{\mathcal{U}}$
21: **end procedure**

5.3.2   Performance. Wu et al. (2020) evaluate DeltaGrad applied to logistic regression for the MNIST, Covtype, HIGGS, and RCV1 datasets. With the inclusion of noise injection, Mahadevan and Mathioudakis (2021) compared the performance of DeltaGrad to Fisher and Influence methods of Sections 5.1, 5.2.

Efficiency. Wu et al. (2020) show that `Efficiency` values for DeltaGrad are $2.5\times$, $2\times$, $1.8\times$ and $6.5\times$ on the MNIST, Covtype, HIGGS, and RCV1 datasets, respectively. It is noted by this work that the theoretical efficiencies that would be expected from this method are not fully achieved, suggesting that a significant portion of this discrepancy originates in the L-BFGS computation of the Hessian projection. Mahadevan and Mathioudakis (2021, Fig. 3(b)) shows that the DeltaGrad method with noise injection is actually slower than naïve retraining for the low and medium dimensionality datasets (Covtype, HIGGS, MNIST), but gives `Efficiency` of $\sim$2.0-2.5$\times$ for the high-dimensionality datasets of CIFAR2 and Epsilon.

Under the five assumptions of Wu et al. (2020, Section 2.3), DeltaGrad guarantees convergence at the rate $o\left(\frac{m}{n}\right)$ (*ibid.*, Theorems 1, 2). This shows that DeltaGrad is strong (Definition 11), since we have $m = 1$ for sequential deletions, yielding a convergence rate, $o\left(\frac{1}{n}\right)$, that is independent of the position in the sequence.

Effectiveness. In their experiments, Wu et al. (2020, Table 1) obtain the same predictive performances, within error, after DeltaGrad as for after naïve retraining on the MNIST, Covtype, HIGGS and RCV1 datasets, indicating very high effectiveness. In contrast, however, Mahadevan and Mathioudakis (2021, Fig. 4) show that the effectiveness of DeltaGrad is generally lower than the Fisher and Influence algorithms. It can also be seen there that the hyperparameter T_0 is an efficiency parameter, decreasing it will increase efficiency due to fewer explicit gradient steps, but causes a slight drop in effectiveness.

Consistency. In Wu et al. (2020, Table 2), consistency is measured via `Consistency`$_\theta$, which measures the ℓ^2 distances between the unlearned parameters and the naïve retrained parameters. Deletion from the MNIST dataset showed the largest parameter difference of 2×10^{-4}, with deletion from the RCV1 dataset showing

the smallest difference in parameters of 3.5×10^{-6}. Theorem 1, *ibid.*, provides a theoretical guarantee of consistency.

Certifiability. While certifiability is not discussed in the original paper (Wu et al., 2020), the certifiability of the model is measured in Mahadevan and Mathioudakis (2021) using `Certifiability`. The hyperparameter σ acts as a certifiability parameter, increasing it will increase certifiability by providing lower `Certifiability` scores, however this generally comes at the cost of efficiency and effectiveness, as demonstrated by Mahadevan and Mathioudakis (2021, Figs. 3, 5). It is also shown there that DeltaGrad generally achieves comparable values of `Certifiability` to the Fisher and Influence methods across all datasets when $\sigma = 1$.

5.4 Descent-to-Delete

In 'Descent-to-Delete: Gradient-Based Methods for Machine Unlearning', Neel et al. (2020) propose several gradient-based unlearning algorithms for convex parametric models. These algorithms apply to arbitrary length sequences of addition as well as removal requests in which each request gets handled immediately before the next comes in, however we focus only on removal requests here. The requests are handled using gradient descent steps starting from the optimum of the original model for the first request and from the state after the most recent update for future requests. They prove that under certain assumptions these approaches offer constant or logarithmic run-time in relation to the number of requests. In addition, under stronger assumptions they also offer (ϵ, δ)-certifiability (Definition 5) of the full internal model state. Their paper, however, focuses on theoretical results and does not offer any empirical evidence of performance.

Their work also contributes several terminologies to the unlearning literature. Firstly, they distinguish perfect and imperfect unlearning algorithms. Perfect unlearning algorithms require (ϵ, δ)-certifiability of the full internal state (i.e., all optimised parameters obtained throughout the deletion sequence) of the algorithm versus the less stringent requirement of (ϵ, δ)-certifiability of only the final outputs (Neel et al., 2020, Definition 2.4). Secondly, their paper introduces the concept of a strong unlearning algorithm (Definition 11) that updates in at most logarithmic run-time in relation to the sequence of delete requests.

5.4.1 Methodology

Basic perturbed gradient descent. Their first algorithm unlearns a sequence of data points $\{\mathbf{z}_i\}_{i \geq 0}$ from models that have strongly convex loss functions and applies projected gradient descent updates upon each deletion request using the projection function $\text{Proj}_\Theta(\theta) = \arg\min_{\theta' \in \Theta} \|\theta - \theta'\|_2$. This is followed by a small perturbation of the model's parameters to guarantee (ϵ, δ)-certifiability. This noise removes the ability of an attacker, with access to the model before and after unlearning, to identify data points that were forgotten. After each unlearning procedure, updated datasets D_i and unlearned models h_{θ_i} form the basis of the input for the following procedure in the sequence of requests, as specified in (2.3). If perfect deletion is not required, the unlearned parameters of the ith update, $\hat{\theta}_i$, in Algorithm 8 may be used as the input for the subsequent update instead of the published output parameters $\tilde{\theta}_i$. They also extend this algorithm to the non-strongly convex case by introducing a variant with ℓ^2 regularization.

A key hyperparameter of these algorithms is the number of gradient descent updates to take following a deletion request, T_i. In the regularized model, the number of updates must be balanced against the amount of noise added to the final parameters. Another key hyperparameter is the variance of the noise, σ, which is calculated using estimates of the Lipschitz constant and the smoothness of the convex loss function in Neel et al. (2020, Theorems 3.1, 3.2) for the unregularized and regularized versions of the algorithm, respectively.

Perturbed distributed descent. Neel et al. (2020) also introduce a second algorithm, perturbed distributed descent, following the same vein as the work of Zhang et al. (2013) and Bourtoule et al. (2021), suitable for larger datasets but that requires modifying the training process. The algorithm partitions the data into K parts, trains separate models on each partition, averages the parameters across the trained models, and then perturbs the final parameters. Upon each deletion request, the models affected may be retrained using

---

Algorithm 8 Perfect ith unlearning for basic perturbed gradient descent, (Neel et al., 2020).

    Input: published model parameters $\tilde{\theta}_{i-1}$, dataset D_{i-1}, update data point $\mathbf{z}_{i-1}$, number of iterations T_i, noise parameter $\sigma > 0$.
    Output: published unlearned parameters $\tilde{\theta}_i$.
1: **procedure** PGDUNLEARN($\tilde{\theta}_{i-1}$, D_{i-1}, $\mathbf{z}_{i-1}$; T_i, σ)
2:       initialise $\theta'_0 \leftarrow \tilde{\theta}_{i-1}$
3:       $D_i \leftarrow D_{i-1} \setminus \{\mathbf{z}_{i-1}\}$
4:       **for** $t = 1; t \leq T_i; t++$ **do**
5:          $\theta'_t \leftarrow \text{Proj}_\Theta(\theta'_{t-1} - \eta_t \nabla L(\theta'_{t-1}, D_i))$
6:       **end for**
7:       $\hat{\theta}_i \leftarrow \theta'_{T_i}$
8:       draw $Z \sim \mathcal{N}(0, \sigma^2 I_p)$
9:       **return** $\tilde{\theta}_i = \hat{\theta}_i + Z$
10: **end procedure**

---

the basic perturbed gradient descent algorithm described above. This may offer some improvements over Bourtoule et al. (2021) by removing the need for naïve retraining of the affected model. The authors also introduce a form of reservoir sampling prior to the gradient descent updates to ensure that the data post-deletion is distributed as i.i.d. samples drawn from the current dataset with replacement. This is required for the guarantees on the out-of-sample error rate, described in Zhang et al. (2013), to hold. In addition, Neel et al. (2020) extend their algorithm by training C different copies in parallel and publishing the one that achieves the lowest loss, in order to achieve stronger guarantees on consistency of the unlearned model. The pseudocode for these procedures can be found in Neel et al. (2020, Algorithms 5, 6, 7).

5.4.2 Performance. Neel et al. (2020) provide theoretical performance of the algorithms and do not provide any empirical evaluations. However, the theoretical guarantees performances and guarantees are explicit and demonstrate clearly the role of parameters in controlling the trade-offs between efficiency, consistency and certifiability. The performance of the algorithms depends on the convexity, smoothness, and Lipschitz property of the loss function (see Neel et al., 2020, Lemma 2.12) as well the required unlearning guarantees.

Efficiency. The authors measure efficiency in terms of the gradient descent iterations required for the ith deletion request relative to the run-time required for the first request, $\mathcal{I}$. For example, the imperfect strong perturbed gradient descent algorithm requires $\log(\frac{\epsilon n}{\sqrt{p}})$ fewer iterations than naïve retraining and the perfect version requires $\mathcal{I} + \log(\frac{\epsilon n}{\sqrt{p}}) - \log(i)\mathcal{I}$ fewer iterations, where n is the dataset size and p is the dimensionality.

Consistency. The (α, β)-accuracy (see Definition 10) guarantees of these unlearning methods depend on whether they are required to be strong or weak. The (α, β)-accuracy of the distributed methods are analysed in depth by (Zhang et al., 2013). For the perturbed gradient descent method, the (α, β)-accuracy ranges from $\frac{pe^{-\mathcal{I}}}{\epsilon^2 n^2}$ for the unregularized version to $(\frac{\sqrt{p}}{\epsilon n \sqrt{\mathcal{I}}})^{2/5}$ for the strong-regularized version.

Certifiability. The authors prove the certifiability of the perfect and imperfect versions of their algorithms. Given ϵ and δ, Neel et al. (2020, Theorems 3.1, 3.2, 3.4, 3.5) define an explicit choice for the noise parameter σ, which depends on values for the smoothness, convexity, and Lipschitz constant of the loss function as well as ϵ and δ. They prove that this choice of σ guarantees (ϵ, δ)-certifiability.

5.5 DeepObliviate

DeepObliviate is an approximate unlearning algorithm developed by He et al. (2021) that applies broadly to deep neural networks and follows a similar procedure to the slicing component of SISA, Section 4.1. The method further improves on slicing by using the so-called *temporal residual memory* to identify which intermediate models need to be retrained, adding approximation into the process.

5.5.1 Methodology. DeepObliviate modifies conventional model training by dividing D into B disjoint subsets $\{D_1, \ldots, D_B\}$, so that $D = \bigcup_i D_i$ and $\bigcap_i D_i = \varnothing$, and performs multi-epoch model training on each data block successively. Data blocks are uniform in two respects: (i) each block is of size $|D|/B$ or $|D|/(B+1)$, and (ii) the number of data points with the same label is uniform in each block. Model parameters are saved after each training block and are used as the initial parameters for the next training block, giving B intermediate models $\{h_1, \ldots, h_B\}$, with their associated parameters $\{\theta_1, \ldots, \theta_B\}$ being saved during training. All parameter vectors are assumed to be in a fixed topological order so that weights corresponding to neural network edges going from the ith layer have smaller vector indices than those going from the jth layer, when $i < j$.

DeepObliviate identifies affected intermediate models that need to be retrained via the *temporal residual memory*, defined below. Given a data point to unlearn, $\mathbf{z} \in D_d$, belonging in the dth data block ($1 \leq d \leq B$), He et al. (2021) empirically observe that the temporal residual memory of $\mathbf{z}$ on h_d is relatively large, but decays exponentially on successive models $h_{d+1}, \ldots, h_B$. As such, the blocks of data can be divided into four distinct groups, see Fig. 2, as follows: (i) *unseen area*, where the data point was not included in model training; (ii) *deleted area*, which includes the data to be deleted, $\mathbf{z}$, and where model retraining needs to start from; (iii) *affected area*, covering the models with prominent residual memory which also need to be fully retrained; and finally (iv) *unaffected area*, where the residual memory decay stabilises to the point that models do not need to be retrained and the remaining models can be easily approximated using the original intermediate models. This approach gives a probabilistic guarantee of speed-up over the naïve approach, since it is unlikely that retraining occurs on all blocks.
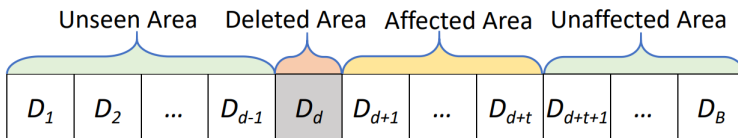
Unseen Area Deleted Area Affected Area Unaffected Area

| D_1 | D_2 | ... | D_{d-1} | D_d | D_{d+1} | ... | D_{d+t} | D_{d+t+1} | ... | D_B |

Fig. 2: Unlearning in DeepObliviate where deleted data resides in block D_d (He et al., 2021).

We let $\text{TRAIN}(\{D_1, \ldots, D_B\} \mid h) := (\text{TRAIN}(D_B \mid \cdot) \circ \cdots \circ \text{TRAIN}(D_1 \mid \cdot))(h)$ denote recursively training the initialised model h on the datasets $D_1, \ldots, D_B$. The unlearning procedure can then be summarised as:

$$h^{\mathcal{U}} = \text{TRAIN}(\{D'_d, D_{d+1}, \ldots, D_{d+t}\}|h_{d-1}) \oplus (h_B \ominus h_{d+t}), \tag{5.10}$$

where $h^{\mathcal{U}}$ is the unlearned model, $D'_d = D_d \setminus \{\mathbf{z}\}$ excludes the unlearning data from D_d, $\oplus$ and $\ominus$ implement vector addition and subtraction on the corresponding model parameters, and t is the number of data blocks after which unlearning influence can be ignored and so retraining can stop.

The value of t is determined by the temporal residual memory. He et al. (2021) define the temporal residual memory as the ℓ^1 distance (or Manhattan distance) between the influence of the deleted data $\mathbf{z}$ on successive models when $\mathbf{z}$ is included in training and when it is not. Formally, the temporal residual memory Δ_t at step t is:

$$\Delta_t := ||I(D_{d+t}|h_{d+t-1}) - I(D_{d+t}|h^{\mathcal{U}}_{d+t-1})||_1, \tag{5.11}$$

$$I(D_i|h_{i-1}) := h_i \ominus h_{i-1}, \tag{5.12}$$

where $0 \leq t \leq B - d$, I denotes the *temporal influence* of block D_i on h_{i-1}, and $\| \cdot \|_1$ is the ℓ^1 norm (i.e., $\|(x_1, \ldots, x_N)\|_1 = |x_1| + \cdots + |x_N|$) . Once Δ is small enough and stable, retraining can stop, i.e., the influence of deleted data can be ignored for downstream models.

He et al. (2021) use *detrended fluctuation analysis* or DFA (Peng et al., 1994) to eliminate noise in Δ and systematically determine whether Δ has stabilised. DFA is used to determine the statistical self-affinity of a time-series signal by fitting Δ_t with a decaying power-law function, whose derivative is easily-computable and can be used to determine stationarity.

5.5.2 Performance. He et al. (2021) evaluate the unlearning method on five public datasets: MNIST, CIFAR-10, SVHN, Purchase and ImageNet. Their evaluation considers ranges of volumes of deletion data and

25

Algorithm 9 Unlearning with DeepObliviate, (He et al., 2021).

Input: parameters for intermediary trained models $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_B\}$, training data $D = D_1 \cup \cdots \cup D_B$, data point to be removed $\mathbf{z} \in D_d$, stationarity hyperparameter ε.

Output: unlearned model $h^{\mathcal{U}}$.

1: **procedure** DEEPOBLIVIATEUNLEARN($\boldsymbol{\theta}$, D, $\mathbf{z}$; ε)
2:      initialise $h^{\mathcal{U}} \leftarrow h_{d-1}$
3:      initialise $\theta^{\mathcal{U}}_{d-1} \leftarrow \theta_{d-1}$
4:      $D_d \leftarrow D_d \setminus \{\mathbf{z}\}$
5:      **for** $t = 0; t \leq B - d; t++$ **do**
6:          $h^{\mathcal{U}} \leftarrow$ TRAIN($D_{d+t} \mid h^{\mathcal{U}}$)
7:          $\theta^{\mathcal{U}}_{d+t} \leftarrow h^{\mathcal{U}}$ get parameters from $h^{\mathcal{U}}$
8:          compute temporal influence $V_{d+t} \leftarrow \theta_{d+t} - \theta_{d+t-1}$ of D_{d+t} on θ_{d+t-1}
9:          compute temporal influence $V^{\mathcal{U}}_{d+t} \leftarrow \theta^{\mathcal{U}}_{d+t} - \theta^{\mathcal{U}}_{d+t-1}$ of D_{d+t} on $\theta^{\mathcal{U}}_{d+t-1}$
10:        compute temporal residual memory $\Delta_{d+t} \leftarrow ||V_{d+t} - V^{\mathcal{U}}_{d+t}||_1$
11:        compute power-law exponent using DFA $\alpha \leftarrow$ DFA($\{\Delta_d, \Delta_{d+1}, \ldots, \Delta_{d+t}\}$)
12:        $Y(x) := ax^{-\alpha} + b$
13:        $f(x) := \partial Y(x)/\partial x = a \cdot (-\alpha) \cdot x^{-\alpha-1}$
14:        $a, b \leftarrow \arg\min_{a,b}(Y(x) - \Delta_x)$ using least squares, $x \in \{d, \ldots, d+t\}$
15:        $g \leftarrow f(d+t)$
16:        **if** $|g| < \varepsilon$ **then**
17:           **break**
18:        **end if**
19:      **end for**
20:      $h^{\mathcal{U}} \leftarrow h^{\mathcal{U}} \oplus (h_B \ominus h_{d+t})$
21:      $\{\theta_d, \ldots, \theta_{d+t}\} \leftarrow \{\theta^{\mathcal{U}}_d, \ldots, \theta^{\mathcal{U}}_{d+t}\}$
22:      **Return** $h^{\mathcal{U}}$
23: **end procedure**

---

of block position for the unlearned data. We summarise performances for the deletion of a single data point; all performances degrade for larger volumes of deletion, though consistency is more robust to increases in deletion volume than efficiency, effectiveness and certifiability. Full results can be found in He et al. (2021, Tables II-VI). Results are also benchmarked in (*ibid.*, Table VII) against the SISA method of Section 4.1 by selecting the number of shards S and slices R so that $B = SR$.

Efficiency. DeepObliviate is shown to achieve `Efficiency` of 75.0×, 66.7×, 33.3×, 29.4×, 13.7× on SVHN, MNIST, CIFAR-10, Purchase, and ImageNet datasets, respectively. It is shown in He et al. (2021, Table VII) that DeepObliviate has significant efficiency improvements over the SISA method for all datasets.

The hyperparameter ε determines the minimum stationarity value of the residual memory decay, under which retraining procedure stops. Therefore it is an efficiency parameter, with larger values increasing the efficiency of the DeepObliviate. This can be observed in the ε values shown in *ibid.* (Tables II-V).

Effectiveness. `Effectiveness` across datasets is < 1%. DeepObliviate achieves better values for `Effectiveness` than SISA for all datasets. Increasing ε will degrade `Effectiveness`.

Consistency. Consistency is measured using `Consistency`$_{\text{y}}$, which measures the proportion of predictions that the unlearned model and naïve retrained model agree on. `Consistency`$_{\text{y}}$ ranges from ∼96% to > 99%.

Certifiability Certifiability of DeepObliviate is measured using the backdoor verification method described in Section 3.4 and Sommer et al. (2020). As noted in Section 3.4, this is comparable to the use of `Certifiability` but ensures that naïve retraining has poor predictive performance on the deleted data, making `Certifiability` more meaningful. To compute `Certifiability`, 500-1,000 data points are deleted and accuracies of predicting on the augmented deleted data are reported for both DeepObliviate and naïve

retraining. For $\varepsilon = 0.05$, this gives a `Certifiability` range of $[0.9\%, 10\%]$. Certifiability decreases for larger ε, demonstrating the efficiency-certifiability trade-off, e.g., `Certifiability` $\in [2.7\%, 12.5\%]$ for $\varepsilon = 0.1$.

6 Discussion

In this section, we discuss and compare the seven methods described in the previous sections. In addition, we discuss some aspects of moving the field of machine unlearning towards practice, framed around approaches for empirical method selection and method monitoring. Table 1 gives statistics on open-source datasets that have been used in the literature to obtain experimental results for the seven methods we have reported on. Many of these datasets are commonly-used benchmarking datasets across the machine learning literature; further descriptions of the datasets can be found in Appendix A.

Table 1: Summary of the datasets used in the experiments of the papers discussed. *Dimensionality* refers to the number of prediction features. When datasets have separate and designated training and testing splits, *Size* refers to the number of samples in the training dataset, else it gives the number of samples in the whole dataset. *Class balance* is given only for datasets with binary label and reports the percentage of positive labels amongst the number of samples in the *Size* column. Descriptions of the datasets can be found in Appendix A.

Dataset	Dimensionality	Size	Classes	Class balance	Appears in
MNIST	784	60,000	10	N/A	(Mahadevan and Mathioudakis, 2021), (Guo et al., 2020), (Wu et al., 2020), (Golatkar et al., 2019), (He et al., 2021)
Covtype	54	581,012	7	N/A	(Mahadevan and Mathioudakis, 2021), (Wu et al., 2020)
HIGGS	28	11,000,000	2	53.00%	(Mahadevan and Mathioudakis, 2021), (Brophy and Lowd, 2021), (Wu et al., 2020)
Epsilon	2,000	400,000	2	50.00%	(Mahadevan and Mathioudakis, 2021)
CIFAR-2	3,072	12,000	2	50.00%	(Mahadevan and Mathioudakis, 2021)
CIFAR-10	3,072	60,000	10	N/A	(Golatkar et al., 2019), (He et al., 2021)
CIFAR-100	3,072	60,000	100	N/A	(Bourtoule et al., 2021)
LSUN	65,536	1,000,000	10	N/A	(Guo et al., 2020)
SST	215,154	11,855	5	N/A	(Guo et al., 2020)
SVHN	3,072	73,257	10	N/A	(Guo et al., 2020), (He et al., 2021)
RCV1	47,236	20,242	2	52.00%	(Wu et al., 2020)
Purchase	600	250,000	2	N/A	(Bourtoule et al., 2021)
ImageNet	150,528	1,281,167	1000	N/A	(Bourtoule et al., 2021), (He et al., 2021)
Mini-ImageNet	150,528	128,545	100	N/A	(Bourtoule et al., 2021)
Surgical	25	14,635	2	25.24%	(Brophy and Lowd, 2021)
Vaccine	36	26,707	2	46.40%	(Brophy and Lowd, 2021)
Adult	14	48,842	2	23.92%	(Brophy and Lowd, 2021)
Bank Mktg.	17	45,211	2	11.30%	(Brophy and Lowd, 2021)
Diabetes	20	101,766	2	46.10%	(Brophy and Lowd, 2021)
No Show	14	110,527	2	20.19%	(Brophy and Lowd, 2021)
Olympics	15	206,165	2	14.60%	(Brophy and Lowd, 2021)
Census	40	299,285	2	6.20%	(Brophy and Lowd, 2021)
Credit Card	30	284,807	2	0.17%	(Brophy and Lowd, 2021)
CTR	39	1,000,000	2	2.91%	(Brophy and Lowd, 2021)
Twitter	15	1,000,000	2	16.93%	(Brophy and Lowd, 2021)
Lacuna-10	1,024	$\geq 5,000$	10	N/A	(Golatkar et al., 2019)
Lacuna-100	1,024	$\geq 50,000$	100	N/A	(Golatkar et al., 2019)

Additionally, we provide two tables (Table 2 and Table 3) that summarise the algorithms that we have described in this paper. These tables are summaries and should be used as a guide to the performances

of individual algorithms, as reported by the relevant literature. In particular, they should not generally be used for comparisons between algorithms. Table 2 provides a summary of the methods considered in this paper, which categorises the various methods according to their applicability and certificate of unlearning, including some empirical results of certifiability. Table 3 summarises the empirical and theoretical results demonstrated in the literature. We can observe the efficiency-effectiveness trade-off in Table 3 for methods with multiple efficiency parameter values. For example, the bottom row of DaRE, corresponding to highest-tested d_{rmax}, sees lower effectiveness values but higher efficiency than the top row, corresponding to lowest-tested d_{rmax}. Likewise for DeepObliviate, Fisher, and Influence. We observe an efficiency-consistency trade-off for DeepObliviate, and an efficiency-certifiability trade-off in the Certificate of Unlearning column for DeepObliviate, Fisher, and Influence.

6.1 Exact Unlearning Algorithms

In Section 4, we described SISA, whose full method applies to any incrementally-trained machine learning model, and DaRE forests, which applies to decision trees and random forests. Both SISA and DaRE forests each come with two efficiency parameters in order to control the unlearning efficiency, usually at the cost of effectiveness. However, unlike approximate unlearning methods, such parameters are also involved in the initial training procedure, which means that they can be tuned during training to maximise efficiency with respect to a satisficing bound on effectiveness.

Brophy and Lowd (2021) obtained strong empirical results for DaRE forests, reporting an average value of `Efficiency` of 2-3 orders of magnitude across the 14 tested datasets, up to a maximum of 5 orders of magnitude, when removing single data points. Results of a similar magnitude to DaRE are reported for another decision-tree based unlearning method in Schelter et al. (2021), which may suggest that decision tree models are more amenable to efficient unlearning. SISA achieves a maximum `Efficiency` of 4.63× in the original paper (Bourtoule et al., 2021), when unlearning 8 data points from the Purchase dataset. He et al. (2021) obtained `Efficiency` of up to 73× for single removals with SISA.

Exact methods have the benefit of a high degree of consistency and certifiability, as given by their proven equivalence with the naïve removal mechanism, however they may share with the corresponding naïve removal mechanisms a vulnerability to membership inference attacks on the deleted data. SISA and DaRE therefore require a level of trust in the security of the pre- and post-unlearning machine learning models. This could be alleviated by the addition of noise into the training procedure, like many approximate methods do, however this will likely degrade efficiency. While SISA is broadly applicable, it struggles to provide a significant efficiency or decent effectiveness for more complex learning tasks, as seen with the `Efficiency` of 1.36× and `Effectiveness` of 18.76% on ImageNet. The efficiency of DaRE is more pronounced on balanced datasets, such as HIGGS, and reports lower efficiencies on datasets with extreme class imbalances, e.g., Credit Card. The DaRE methodology is not currently applicable to boosting ensembles, suggesting a natural next step for the development of this algorithm. Finally, both exact methods come with additional storage costs for storing statistics and parameters during the training and unlearning; these storage operations may also incur additional computational cost, however, this is not explicitly mentioned in either Bourtoule et al. (2021) or Brophy and Lowd (2021).

6.2 Approximate Unlearning Algorithms

In Section 5, five approximate algorithms were described. Three of these, Fisher, Influence, and Descent-to-Delete, apply parameter updates starting from the optimised parameters of the original machine learning model; Descent-to-Delete performs first-order updates, Fisher performs a second-order Newton update using the remaining data $D \setminus D_u$, and Influence performs a second-order Newton update using the deleted data D_u. DeltaGrad applies gradient descent updates starting from the same initialised parameters as the original training procedure, and approximates the gradient descent steps of the naïve unlearning mechanism by using cached historical training information. Finally, DeepObliviate takes the slicing procedure of the SISA unlearning algorithm and turns this into an approximate algorithm by stopping the retraining procedure

Table 2: Summary table of unlearning algorithms considered in this paper. In the table, SC = strongly convex loss function, BG = bounded gradients for loss function (Wu et al., 2020, Assumption 4), SI = strong independence (Wu et al., 2020, Assumption 5), LQ = locally quadratic. This table is a summary only and empirical results contained in the table should not be used for comparison due to the use of different datasets, experimental design, and machine specifications.

Unlearning Type				
Method	**Applicability**	**Properties**	**Certificate of Unlearning**	
SISA	Incrementally trained models[a]	Exact Weak[b]	Exact	
DaRE	DaRE trees DaRE RF	Exact	Exact	
Fisher	LQ loss function	Approximate	3.3, 33.4 kNATs[c]	0.0%[d] ($m' = \lfloor m/8 \rfloor$) 0.26% ($m' = m$)
Influence	SC, BG, Lipschitz Hessian	Approximate Weak[e]	(ϵ, δ)-certified	0.1%[d] ($m' = \lfloor m/8 \rfloor$) 1.1% ($m' = m$)
DeltaGrad	SC, smooth loss, BG, Lipschitz Hessian, SI	Approximate Strong[f]	N/A[g]	
Descent-to-Delete	SC, smooth (Bounded, Lipschitz Hessian)[h]	Approximate Strong[i]	(ϵ, δ)-certified	
DeepObliviate	Any deep-learning model	Approximate Strong	6.39% ($\varepsilon = 0.05$) 8.28% ($\varepsilon = 0.1$)[j]	

a. Sharding-only SISA is applicable to any machine learning model, including decision-tree based models; the full SISA algorithm is applicable to any model that has been trained incrementally, for example via gradient descent.

b. See Equations (6) and (8), of Bourtoule et al. (2021, Section V.C.).

c. Certifiability of Fisher in Golatkar et al. (2019) is given by an upper bound on the information retained. Empirical results are given only for two datasets, Lacuna-10 and CIFAR-10, with the model retaining at most 3.3 kNATs of information 33.4 kNATs of information, respectively, about the deleted data.

d. Certifiability of Fisher and Influence in Mahadevan and Mathioudakis (2021) is given by `Certifiability`. We report the geometric mean across datasets for two extremes of the efficiency parameter m', fixing $\sigma = 1$ and for the smallest deletion volume m considered.

e. The convergence error grows linearly with number of points removed, see (Guo et al., 2020, p.4).

f. When deleting a single point (i.e., $r = 1$) the convergence rate is $o(1/n)$ regardless of position in sequence (see Wu et al., 2020, Theorem 1). This is expected since DeltaGrad computes gradient descent from scratch each time.

g. `Certifiability` is used in Mahadevan and Mathioudakis (2021), but certifiability is not measured in the original paper (Wu et al., 2020).

h. Strongly convex can be relaxed to convex by using regularised perturbed gradient descent (Neel et al., 2020, Theorems 3.4, 3.5). Bracketed assumptions are additional assumptions required for distributed perturbed gradient descent.

i. All descent-to-delete algorithms are strong apart from that of Neel et al. (2020, Theorem 3.5), which is designed to be weak in order to allow better consistency.

j. Certifiability values for DeepObliviate are taken from He et al. (2021, Table VI) which applies the backdoor verification experiment. We show the two different values of the efficiency parameter ε, and assume the deleted data is in the first block (top row of each dataset in Table VI, *ibid.*). Values are then passed through `Certifiability`, and we report the geometric mean across all tested datasets.

Table 3: Summary of efficiency, effectiveness, and consistency performances of the algorithms considered as reported by the literature. Note that this table is a summary only and should not be used for comparison, since the results are obtained using different datasets, experimental design and machine specifications.

		Performance						
Method		**Efficiency**			**Effectiveness**	**Consistency**		
		min	g. mean	max		min	g. mean	max
SISA[a]	Bourtoule et al. (2021)	1.36×	2.49×	4.63×	< 2% (18.76%)	Exact		
	He et al. (2021)	14.0×	39.6×	75.0×	< 5% (15.1%)			
DaRE[b]	min d_{rmax}	10×	366×	9735×	≤ 0.5%	Exact		
	max d_{rmax}	145×	1272×	35856×	≤ 2.5%			
Fisher[c]	$m' = \lfloor m/8 \rfloor$	1.0×	3.3×	36.8×	≈ 0.0%	N/A		
	$m' = m$	1.5×	13.0×	282.4×	< 0.2%			
Influence[c]	$m' = \lfloor m/8 \rfloor$	0.3×	5.9×	75.7×	< 0.55%	N/A		
	$m' = m$	2.5×	38.2×	215.1×	< 0.57%			
DeltaGrad[d]		1.6×	2.7×	6.5×	≈ 0.0%	1.7×10^{-6}	1.1×10^{-5}	1.4×10^{-4}
Descent-to-Delete[e]	PGD	$1 + \mathcal{I}^{-1} \log(\epsilon n/\sqrt{p})$			N/A	$p/(e^{\mathcal{I}} \epsilon^2 n^2)$		
	sRPGD	$\mathcal{I}^{-\frac{3}{5}} (\epsilon n/\sqrt{p})^{\frac{2}{5}}$				$\left(\frac{\sqrt{p}}{\epsilon n \mathcal{I}}\right)^{\frac{2}{5}}$		
	wRPGD	$\mathcal{I}^{-\frac{3}{4}} \sqrt{\epsilon n/\sqrt{p}}$				$\sqrt{\frac{\sqrt{p}}{\epsilon p \sqrt{\mathcal{I}}}}$		
	DPGD	$\min\{\mathcal{I}^{-1} \log n,$ $n^{\frac{4-3\xi}{2}} + \mathcal{I}^{-1} \log(\epsilon n/\sqrt{p})\}$				$\frac{2 \exp\left(-\mathcal{I} n^{\frac{4-3\xi}{2}}\right)}{\epsilon^2 n^2} + \frac{1}{n^{\xi}}$		
DeepObliviate[f]	min ε	6.17×	13.97×	45.45×	< 0.18%	97.25	98.82	99.95
	$\varepsilon = 0.1$	9.26×	22.81×	66.67×	< 0.35%	95.78	97.61	99.85

a. The top row gives results from the original paper (Bourtoule et al., 2021). Results are reported for multiple deletions (8, 18, or 39) depending on the dataset. Effectiveness is < 2% for simpler learning tasks and 18.76% for ImageNet. The second row gives results from He et al. (2021), which is for single deletions, and give 15.1% effectiveness reported for ImageNet.

b. Actual value of minimum and maximum d_{rmax} depends on the dataset, and determined with respect to CV error tolerances of 0.1% and 1.0%, respectively, in cross-validation, see Section 4.2 and Brophy and Lowd (2021, Table 6).

c. Results are taken from Mahadevan and Mathioudakis (2021) for a fixed noise parameter $\sigma = 1$ and for the smallest volume of data deleted, which volume depends on the specific dataset. The parameter m' is the mini-batch size used in the update, and m is the volume of data deleted.

d. Consistency of DeltaGrad is measured by `Consistency`$_\theta$; results are from Wu et al. (2020).

e. PGD = perturbed gradient descent; sRPGD/wRPGD = strong/weak variant of regularized perturbed gradient descent; DPGD = distributed perturbed gradient descent. Results are extracted from Neel et al. (2020, Table 1). Efficiency is given by reporting the number of iterations for naïve retraining divided by the number of iterations for the first deletion $\mathcal{I}$. ϵ is the value in the (ϵ, δ)-certified guarantee of the method. $\xi \in [1, 4/3]$ is a parameter in distributed descent. Consistency is given by (α, β)-accuracy (Definition 10).

f. In DeepObliviate, we take values from He et al. (2021, Tables I, II, III, V), corresponding to four experiments that use multiple ε values (Tables IV and VII use only a single ε value). Within each experiment, there are different choices of the number of data unlearned, the position of data to be unlearned and the choice of hyperparameter ε. We fix the smallest volume of data unlearned, and earliest block position to give worst-case efficiency results. We fix ε at two extremes, and report for each; minimum ε depends on the dataset (either 0.04 or 0.05) and maximum ε is always 0.1. Consistency is measured by `Consistency`$_y$.

at an appropriate point and approximating the tail end of the retraining by using cached parameters from the initial training procedure. The applicability of approximate unlearning algorithms is generally narrow. Fisher, Influence, DeltaGrad, and Descent-to-Delete are primarily applicable to models with strongly-convex loss functions, although extensions of Fisher, Influence, and DeltaGrad to non-convex models are discussed in their respective papers. DeepObliviate is notable for being widely applicable to any neural network.

Due to differing experimental design, direct comparisons between the five approximate algorithms are difficult. However, since Mahadevan and Mathioudakis (2021) benchmark Fisher, Influence, and DeltaGrad applied to a logistic regression model, we can make some inferences on their performances. In general, Influence observes better `Efficiency` values than Fisher, which can be seen in the efficiency column of Table 3. This was noted in *ibid.*, with `Efficiency` being most pronounced on high-dimensional datasets, CIFAR2 and Epsilon, and less pronounced on the low-dimensional datasets, Covtype and HIGGS. However, the Influence method has weaker effectiveness than the Fisher method. Across most scenarios, the efficiency and effectiveness of DeltaGrad is outperformed by the Influence and Fisher algorithms. Note that the original DeltaGrad is supplemented by Mahadevan and Mathioudakis (2021) with the addition of noise, in order to guarantee a degree of certifiability, however this appears to degrade the efficiency results of DeltaGrad. An additional reason for DeltaGrad's poorer performance is due to the computational cost of SGD iterations in DeltaGrad being higher than inverting the Hessian in Fisher and Influence. DeepObliviate is directly benchmarked against the SISA method in He et al. (2021), achieving better `Efficiency` and `Effectiveness` than SISA in most cases.

Mahadevan and Mathioudakis (2021) measure certifiability using the `Certifiability` measure of (3.6) without the backdoor verification method. As discussed in Section 3.4, the backdoor verification method can make the `Certifiability` measure more meaningful, which is done by He et al. (2021). Finally, Descent-to-Delete suffers from non-existent empirical evaluation. A thorough benchmarking of all seven algorithms would be a beneficial next step.

6.3 Machine Unlearning from a Practitioner's Perspective

The evaluation of a single machine unlearning algorithm is complex and multidimensional. On top of this, evaluation measures in the literature differ, making comparisons between established algorithms difficult. In this section, we explore a framework for machine unlearning algorithm selection.

Candidate selection. The various columns of Tables 2 and 3 correspond to key factors to consider when selecting an unlearning algorithm. We can use Table 2 to select algorithms that are applicable with respect to initial constraints. The Applicability column indicates which machine learning models the algorithm can be applied to. For example, a tree-based model is required for DaRE forests.

The remaining columns of Table 2 are of a high priority if the unlearning method is being applied in order to comply with regulations concerning removal requests. This compliance depends on the particular external regulations, for example, in the EU, unlearn requests must comply with the user's Right to be Forgotten. Future clarification of these regulations may rule out approximate algorithms, or they may impose strict constraints on the certificate of unlearning. So it is a key responsibility of researchers and practitioners to define and select algorithms with a suitable certificate of unlearning. Another consideration here is in removal immediacy. If data points must be removed immediately on request, then the strong and weak classification becomes important, whereas if not, then methods that allow batch unlearning may be more appropriate.

Parameter trade-offs and method selection. Given a set of candidate methods that are all applicable to the use case, we now discuss how to select among them. Currently, direct comparisons between algorithms are difficult for two main reasons: (1) experiments in papers vary greatly and, with the exception of Mahadevan and Mathioudakis (2021), there has been no extensive empirical benchmark comparison of algorithms; (2) evaluation of unlearning algorithms is multidimensional, with various trade-offs that must be navigated relevant to the use case. As a result, Table 3 should be viewed as a guide to each individual algorithm and care should be taken with direct comparisons.

Because of the difficulties in direct comparison, it makes sense to perform an empirical algorithm selection procedure once at the start of the pipeline, as is the current standard for machine learning. We now discuss a proposal for this procedure. Due to the computational expense of evaluating the unlearning algorithms, this procedure is advisable only if multiple unlearning processes are expected as a result of sequential deletion requests. First, according to both user specification (for example, the type of machine learning model in production and intended application of unlearning) as well as external certifiability constraints (for example, regulations), a list of candidate unlearning algorithms should be produced as discussed above. Candidates are both applicable to the problem and sufficiently certifiable. Additionally, the user should set tolerances for empirical results for effectiveness and certifiability (along with consistency, if desired); these are the minimal acceptable levels one expects to see in results for each evaluation criterion (for example, an effectiveness tolerance of 2% means that algorithms that achieve < 2% for `Effectiveness` are desired).

Once candidates and preset tolerances are chosen, an empirical evaluation routine is performed for each of the candidate algorithms. To speed up the process, this process may be performed on a random subsample of the training data. If the candidate algorithm is exact then, as is done by Brophy and Lowd (2021), one may first tune the efficiency parameter so that efficiency is maximised with respect to the preset effectiveness tolerance. For example, with DaRE forests the parameter d_{rmax} may be incremented and the cross-validation performance of the resultant trained DaRE RF is measured. Once this performance degrades beyond the preset effectiveness tolerance, then the efficiency parameter is set. This is possible for exact methods since the efficiency parameter is a parameter in both the training and the removal mechanisms, and because there are strong guarantees for consistency and certifiability.

Finally, a deletion distribution is chosen and then deletion points are drawn from the training data according to this distribution. If the candidate is exact then, using the tuned efficiency parameter, deletion points are unlearned and values for `Efficiency` and `Effectiveness` are measured and reported. If the candidate is approximate, then an appropriate range of efficiency and certifiability parameters are chosen and, for each choice of parameters, values for `Efficiency`, `Effectiveness`, `Consistency`$_\mathrm{y}$, and `Certifiability` are measured and reported. The user may then choose the algorithm that achieves the best `Efficiency` whilst remaining within the preset tolerances for the other evaluation criteria.

Machine unlearning monitoring and auditing. Effectiveness and certifiability (and sometimes consistency) are treated as satisficing metrics, so preset tolerances for each are implicit inputs in algorithm selection. At certain points it will be necessary to perform full naïve retraining of the model. This is to ensure that, as the volume of deletions increases, effectiveness, consistency, and certifiability do not breach the preset tolerances. Effective unlearning monitoring is used to decide when it is appropriate to fully retrain. The issue, however, is that monitoring of all four areas of evaluation involves comparison with the naïve retrained model. Computing this baseline model after every deletion defeats the purpose of unlearning in the first place. Therefore it is necessary to introduce proxies to measure evaluation after deletions. Consistency is difficult to estimate, and is less of a concern if we have good estimates for the other three areas, so we do not focus on this here.

For simplicity, we assume that additions are not made to the training data during the deletion pipeline and that removals of only single data points are performed. Let $\mathcal{U}$ be a removal mechanism, $h_0 = h$ be the original machine learning model, and $D_0 = D$ be the training data for h_0. Suppose that we have a sequence of models $\{h_i\}_{i=0}^{m-1}$ and datasets D_i, where each $h_i = \mathcal{U}(h_{i-1}, D_{i-1}, \mathbf{z}_{i-1})$ and $D_i = D_{i-1} \setminus \{\mathbf{z}_{i-1}\}$ are the results of unlearning a point $\mathbf{z}_i$ from the previous model. We assume that there is a subset of k indices $\{i_j\}$ for which, after obtaining h_{i_j}, a naïve retraining has occurred, giving k naïvely retrained models $\{h_{i_j}^*\}_{j=1}^k$; the subsequent model $h_{i_j+1} = \mathcal{U}(h_{i_j}^*, D_{i_j}, \mathbf{z}_{i_j})$ is obtained by unlearning on $h_{i_j}^*$. We explore how we might monitor the mth deletion.

For certifiability, we can overestimate `Certifiability` by the test error loss to the last retrained model as follows. Given a performance metric $\mathcal{M}$, let $\mathcal{M}_{\mathrm{test},m}$ denote the performance of h_m on a test set and let $\mathcal{M}_{\mathrm{test},i_k}^*$ denote the test performance of $h_{i_k}^*$. We define

$$\widetilde{\texttt{Certifiability}} := c_{i_k} \, \mathrm{SAPE}(\mathcal{M}_{\mathrm{test},m}, \mathcal{M}_{\mathrm{test},i_k}^*),$$

where c_{i_k} can be estimated empirically after each full retraining as shown in Mahadevan and Mathioudakis (2021, p. 13), and SAPE is as in (3.6). This was shown, *ibid.*, to give an overestimation of `Certifiability`.

In a similar way, we may overestimate `Effectiveness` by

$$\widetilde{\texttt{Effectiveness}} := e_{i_k} |\mathcal{M}_{\text{test},m} - \mathcal{M}^*_{\text{test},i_k}|$$

where e_{i_k} is estimated empirically by replacing `Certifiability` in the calculation of c_{i_k} with `Effectiveness`.

Efficiency is a key measure that should be monitored to ensure that the expected unlearning time is not consistently being exceeded, although it may not necessarily be used to decide when to retrain. We use the time taken to train the latest retrained model $h^*_{i_k}$ as a proxy for the time taken to fully retrain. Since we assume that there have been no additions to the dataset, we have $\text{len}(D_{i_j}) > \text{len}(D_m)$ so

$$\widetilde{\texttt{Efficiency}} := \frac{\text{time taken to train } h^*_{i_k}}{\text{time taken to unlearn } \mathbf{z}_{m+1}}$$

provides a conservative estimate of `Efficiency`.

After obtaining h_m, $\widetilde{\texttt{Effectiveness}}$, and $\widetilde{\texttt{Certifiability}}$ are calculated. If these values exceed a preset threshold for effectiveness and certifiability, then full retraining should occur to obtain $h^*_{i_{k+1}}$. Additionally, $\widetilde{\texttt{Efficiency}}$ should be calculated, and consistently poor $\widetilde{\texttt{Efficiency}}$ compared to those seen in the initial algorithm selection should be investigated. Also, unlearning algorithms with lower certifiability will not only be more likely to breach regulations, but they will need to be retrained more often as well, entailing higher cost in the long run. The estimates introduced here may also be applicable to the situation of unlearning algorithm selection described previously.

Auditing may occur at the request of regulators. The precise formalisation of the certificate of unlearning in practice remains an open problem, but source code and enough information to recalculate the sequence of models $\{h_i\}_i$ above are likely to be requested. The regulator may then recreate the sequence of deletions and calculate certifiability according to their own measures, the failure of which will lead to fines or other regulatory actions. It is therefore important that effective monitoring and regular naïve retraining take place.

7   Conclusion

In this review paper, we give a broad introduction to and assessment of the field of machine unlearning. We provide a standardised unlearning and evaluation framework, along with the theory and implementations of seven state-of-the-art unlearning algorithms. Finally, in the Discussion section (Section 6) we begin to address some of the theoretical and practical gaps identified in the field, however, more research is required to fully resolve these gaps. Whilst we are able to make some comparisons between the algorithms considered here, extensive empirical benchmarking between them is lacking in the literature, which could be a useful future contribution. In addition, there is a current lack of research into applying unlearning in practice and certifiability is a key concern in this regard. In order to verify that unlearning algorithms are removing sufficient information about the deleted data, a formalised certificate of unlearning that is applicable in practice, along with a robust monitoring pipeline, is a necessary piece of research, which is a sentiment also echoed elsewhere in the literature (Thudi et al., 2021). Finally, there is a general difficulty in developing unlearning algorithms that are applicable to modern deep neural networks, and more work in this area would be beneficial.

Bibliography

Aldaghri, N., Mahdavifar, H., and Beirami, A. (2021). Coded machine unlearning. *IEEE Access*, 9:88137–88150.

Baumhauer, T., Schöttle, P., and Zeppelzauer, M. (2020). Machine unlearning: Linear filtration for logit-based classifiers. *arXiv e-prints*, page arXiv:2002.02730.

Biggio, B., Nelson, B., and Laskov, P. (2013). Poisoning attacks against support vector machines. *arXiv e-prints*, page arXiv:1206.6389.

Bourtoule, L., Chandrasekaran, V., Choquette-Choo, C., Jia, H., Travers, A., Zhang, B., Lie, D., and Papernot, N. (2021). Machine unlearning. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy*, San Francisco, CA.

Brophy, J. and Lowd, D. (2021). Machine unlearning for random forests. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1092–1104. PMLR.

Cao, Y. and Yang, J. (2015). Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*, pages 463–480.

Chen, C., Sun, F., Zhang, M., and Ding, B. (2022). Recommendation unlearning. *arXiv e-prints*, page arXiv:2201.06820.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Doshi-Velez, F. and Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *arXiv e-prints*, page arXiv:1702.08608.

Dwork, C. and Roth, A. (2014). The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(34):211–407.

Fu, S., He, F., and Tao, D. (2022). Knowledge removal in sampling-based bayesian inference. In *International Conference on Learning Representations*.

Ginart, A., Guan, M. Y., Valiant, G., and Zou, J. (2019). Making AI forget you: Data deletion in machine learning. *arXiv e-prints*, page arXiv:1907.05012.

Golatkar, A., Achille, A., Ravichandran, A., Polito, M., and Soatto, S. (2020). Mixed-privacy forgetting in deep networks. *arXiv e-prints*, page arXiv:2012.13431.

Golatkar, A., Achille, A., and Soatto, S. (2019). Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks. *arXiv e-prints*, page arXiv:1911.04933.

Guo, C., Goldstein, T., Hannun, A., and van der Maaten, L. (2020). Certified data removal from machine learning models. *arXiv e-prints*, page arXiv:1911.03030.

Gupta, V., Jung, C., Neel, S., Roth, A., Sharifi-Malvajerdi, S., and Waites, C. (2021). Adaptive machine unlearning. *arXiv e-prints*, page arXiv:2106.04378.

He, Y., Meng, G., Chen, K., He, J., and Hu, X. (2021). DeepObliviate: A Powerful Charm for Erasing Data Residual Memory in Deep Neural Networks. *arXiv e-prints*, page arXiv:2105.06209.

Jia, H., Yaghini, M., Choquette-Choo, C. A., Dullerud, N., Thudi, A., Chandrasekaran, V., and Papernot, N. (2021). Proof-of-learning: Definitions and practice. *arXiv e-prints*, page arXiv:2103.05633.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017).

Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Koh, P. W. and Liang, P. (2020). Understanding black-box predictions via influence functions. *arXiv e-prints*, page arXiv:1703.04730.

Mahadevan, A. and Mathioudakis, M. (2021). Certifiable Machine Unlearning for Linear Models. *arXiv e-prints*, page arXiv:2106.15093.

Neel, S., Roth, A., and Sharifi-Malvajerdi, S. (2020). Descent-to-Delete: Gradient-Based Methods for Machine Unlearning. *arXiv e-prints*, page arXiv:2007.02923.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

Nguyen, Q. P., Oikawa, R., Mon Divakaran, D., Chan, M. C., and Low, B. K. H. (2022). Markov Chain Monte Carlo-Based Machine Unlearning: Unlearning What Needs to be Forgotten. *arXiv e-prints*, page arXiv:2202.13585.

Peng, C.-K., Buldyrev, S. V., Havlin, S., Simons, M., Stanley, H. E., and Goldberger, A. L. (1994). Mosaic organization of DNA nucleotides. *Physical Review E*, 49(2):1685–1689.

Quenouille, M. H. (1956). Notes on Bias in Estimation. *Biometrika*, 43(3-4):353–360.

Sakar, C. O., Polat, S. O., Katircioglu, M., and Kastro, Y. (2018). Real-time prediction of online shoppers' purchasing intention using multilayer perceptron and LSTM recurrent neural networks. *Neural Computing and Applications*, 31(10):6893–6908.

Schelter, S. (2020). Amnesia - machine learning models that can forget user data very fast. In *CIDR 2020, 10th Conference on Innovative Data Systems Research, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*. www.cidrdb.org.

Schelter, S., Grafberger, S., and Dunning, T. (2021). Hedgecut: Maintaining randomised trees for low-latency machine unlearning. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 1545–1557, New York, NY, USA. Association for Computing Machinery.

Shafer, G. and Vovk, V. (2008). A tutorial on conformal prediction. *J. Mach. Learn. Res.*, 9:371–421.

Sommer, D. M., Song, L., Wagh, S., and Mittal, P. (2020). Towards probabilistic verification of machine unlearning. *arXiv e-prints*, page arXiv:2003.04247.

Thudi, A., Jia, H., Shumailov, I., and Papernot, N. (2021). On the necessity of auditable algorithmic definitions for machine unlearning. *arXiv e-prints*, page arXiv:2110.11891.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching networks for one shot learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3637–3645, Red Hook, NY, USA. Curran Associates Inc.

Wu, G., Hashemi, M., and Srinivasa, C. (2022). Puma: Performance unchanged model augmentation for training data removal. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI-2022)*, Vancouver, Canada.

Wu, Y., Dobriban, E., and Davidson, S. B. (2020). DeltaGrad: Rapid retraining of machine learning models. *arXiv e-prints*, page arXiv:2006.14755.

Zhang, Y., Duchi, J. C., and Wainwright, M. J. (2013). Communication-efficient algorithms for statistical optimization. *The Journal of Machine Learning Research*, 14(1):3321–3363.

Appendix A   Datasets

Descriptions of the datasets included in Table 1 are given below.

a. MNIST is a 10-class image classification dataset containing images of digits 0–9 (http://yann.lecun.com/exdb/mnist/ [accessed: 31-August-2022]). MNIST binary is also used in Mahadevan and Mathioudakis (2021) by taking digits 3 and 8. This has 11,982 samples with 49.00% class balance.

b. Covtype is a multi-class classification dataset derived from US Geological Survey and USFS involving cartographic feature variables and forest cover type as the target variable (https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/multiclass.html#covtype [accessed: 31-August-2022]). Mahadevan and Mathioudakis (2021) consider the binary version of Covtype which has the same number of samples as the non-binary version with 49.00% class balance (https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/multiclass.html#covtype [accessed: 31-August-2022]).

c. HIGGS measures kinematic properties of particle detectors in the Higgs boson accelerator and derived values as features, with whether the signal represents a Higgs boson as the target (https://archive.ics.uci.edu/ml/datasets/HIGGS [accessed: 31-August-2022]).

d. Epsilon is the Epsilon dataset from the PASCAL Large Scale Learning Challenge 2008 (https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html#epsilon [accessed: 31-August-2022]) .

e. CIFAR-10/100 are 10- and 100-class image classification datasets respectively representing various animals and objects (http://www.cs.toronto.edu/ kriz/cifar.html [accessed: 31-August-2022]). CIFAR-2 is extracted in Mahadevan and Mathioudakis (2021) from CIFAR-10 by only considering the `cat` and `ship` labels.

f. LSUN is a 10-class image classification dataset representing 10 scenes such as dining room, bedroom, and so on (https://www.yf.io/p/lsun [accessed: 31-August-2022]) .

g. SST is an NLP dataset consisting of movie reviews for sentiment analysis (https://nlp.stanford.edu/sentiment/index.html [accessed: 31-August-2022]) .

h. SVHN is a 10-class image classification dataset consisting of house numbers taken from Google Street View, where the objective is to identify the digits 0–9 (http://ufldl.stanford.edu/housenumbers/ [accessed: 31-August-2022]).

i. RCV1 is a collection of manually labelled news articles from Reuters taken from the period 1996-1997 (https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html#rcv1.binary [accessed: 31-August-2022]).

j. Purchase consists of consumer purchase history with the target variable being whether a customer will repeat purchase (https://www.kaggle.com/c/acquire-valued-shoppers-challenge/data [accessed: 31-August-2022]). In Bourtoule et al. (2021), Purchase is curated in Bourtoule et al. (2021) from the original Purchase dataset by choosing the top 600 most purchased items based on the `category` attribute.

k. ImageNet is a large-scale image dataset which have been annotated according to the WordNet hierarchy, with the 1000-class target variable being the annotation's synset (https://www.image-net.org/ [accessed: 31-August-2022]). Mini-ImageNet is a supervised classification version of ImageNet created by the process of Vinyals et al. (2016).

l. Surgical is a binary-classification dataset with the goal of predicting whether a surgery involved complications (https://www.kaggle.com/datasets/omnamahshivai/surgical-dataset-binary-classification [accessed: 31-August-2022]).

m. Vaccine is a binary-classification dataset on whether a person had got a flu vaccine (https://www.drivendata.org/competitions/66/flu-shot-learning/ [accessed: 31-August-2022]).

n. Adult is a binary-classification dataset determining whether a person has an annual income of over $50,000 (http://archive.ics.uci.edu/ml/datasets/Adult [accessed: 31-August-2022]).

o. Bank Mktg. consists of a Portuguese bank marketing call details with whether a contacted customer subsequently subscribed as the target variable (http://archive.ics.uci.edu/ml/datasets/Bank+Marketing [accessed: 31-August-2022]).

p. Diabetes is a binary-classification dataset of diabetic patients whose target variable is hospital readmission (https://archive.ics.uci.edu/ml/datasets/diabetes [accessed: 31-August-2022]).

q. No Show is a binary-classification dataset with whether a patient missed a doctor appointment as the target variable (https://www.kaggle.com/datasets/joniarroba/noshowappointments [accessed: 31-August-2022]).

r. Olympic is a binary-classification dataset for predicting whether an athlete received an Olympic medal for the event participated (https://www.kaggle.com/datasets/heesoo37/120-years-of-olympic-history-athletes-and-results [accessed: 31-August-2022]).

s. Census is a binary-classification dataset for predicting whether a person has an annual income of over \$50,000 based on census data (https://archive.ics.uci.edu/ml/datasets/Census-Income+(KDD) [accessed: 31-August-2022]).

t. Credit Card is a highly-imbalanced binary-classification dataset for predicting fraudulent European credit card transactions in September 2013 (https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud [accessed: 31-August-2022]).

u. CTR is curated by Brophy and Lowd (2021) by taking the first 1,000,000 instances and 13 numeric attributes from Criteo's original dataset; each row represents an ad that was displayed and the binary target variable indicates whether the ad was clicked on (https://ailab.criteo.com/download-criteo-1tb-click-logs-dataset/ [accessed: 31-August-2022]).

v. Twitter is curated by Brophy and Lowd (2021) by taking the first 1,000,000 tweets from the HSpam14 dataset (https://www3.ntu.edu.sg/home/AXSun/datasets.html [accessed: 31-August-2022]). This dataset has a binary target variable indicating whether a tweet is spam or not.

w. Lacuna-10/100 are 10- and 100-class image classification datasets consisting of the faces of 10 and 100 different celebrities, respectively, extracted by Golatkar et al. (2019) from the VGGFaces2 dataset (https://www.robots.ox.ac.uk/vgg/data/vgg_face2/ [accessed: 31-August-2022]).