Machine Unlearning of Features and Labels

Alexander Warnecke
TU Braunschweig

Lukas Pirch
TU Branschweig

Christian Wressnegger
Karlsruhe Institute of Technology

Konrad Rieck
TU Branschweig

ABSTRACT

Removing information from a machine learning model is a non-trivial task that requires to partially revert the training process. This task is unavoidable when sensitive data, such as credit card numbers or passwords, accidentally enter the model and need to be removed afterwards. Recently, different concepts for machine unlearning have been proposed to address this problem. While these approaches are effective in removing individual data points, they do not scale to scenarios where larger groups of features and labels need to be reverted. In this paper, we propose the first method for unlearning features and labels. Our approach builds on the concept of influence functions and realizes unlearning through closed-form updates of model parameters. It enables to adapt the influence of training data on a learning model retrospectively, thereby correcting data leaks and privacy issues. For learning models with strongly convex loss functions, our method provides certified unlearning with theoretical guarantees. For models with non-convex losses, we empirically show that unlearning features and labels is effective and significantly faster than other strategies.

1 INTRODUCTION

Machine learning has become an ubiquitous tool in analyzing personal data and developing data-driven services. Unfortunately, the underlying learning models can pose a serious threat to privacy if they inadvertently capture sensitive information from the training data and later reveal it to users. For example, Carlini et al. [15] show that the Google text completion system contains credit card numbers from personal emails, which may be exposed to other users during the autocompletion of text. Once such sensitive data has entered a learning model, however, its removal is non-trivial and requires to selectively revert the learning process. In absence of specific methods for this task in the past, retraining from scratch has been the only resort, which is costly and only possible if the original training data is still available.

As a remedy, Cao & Yang [14] and Bourtoule et al. [11] propose methods for *machine unlearning*. These methods decompose the learning process and are capable of removing individual data points from a model in retrospection. As a result, they enable to eliminate isolated privacy issues, such as data points associated with individuals. However, information leaks may not only manifest in single data instances but also in groups of *features* and *labels*. For example, a collaborative spam filter [6] may accidentally capture personal names and addresses that are present in hundreds of emails. Similarly, in a credit scoring system, inappropriate features, such as the gender or race of clients, may need to be unlearned for thousands of affected data points.

Unfortunately, instance-based unlearning as proposed in prior work is inefficient in these cases: First, a runtime improvement can hardly be obtained over retraining when the changes are not isolated and larger parts of the training data need to be adapted. Second, omitting several data points inevitably reduces the fidelity of the corrected learning model. It becomes clear that the task of unlearning is not limited to removing individual data points, but also requires corrections on the level of features and labels. In these scenarios, unlearning methods still need to be effective and efficient, regardless of the amount of affected data points.

In this paper, we propose the first method for unlearning features and labels from a learning model. Our approach is inspired by the concept of *influence functions*, a technique from robust statistics [39], that allows for estimating the influence of data on learning models [43, 44]. By reformulating this influence estimation as a form of unlearning, we derive a versatile approach that maps changes of the training data in retrospection to closed-form updates of model parameters. These updates can be calculated efficiently, even if larger parts of the training data are affected, and enable the removal of features and labels. As a result, our method can correct privacy leaks in a wide range of learning models with convex and non-convex loss functions.

For models with strongly convex loss, such as logistic regression and support vector machines, we prove that our approach enables *certified unlearning*. That is, it provides theoretical guarantees on the removal of features and labels from the models. To obtain these guarantees, we build on the concept of certified data removal [36, 50] and investigate the difference between models obtained with our approach and retraining from scratch. Consequently, we can define an upper bound on this difference and thereby realize provable unlearning in practice.

For models with non-convex loss functions, such as deep neural networks, similar guarantees cannot be realized. However, we empirically demonstrate that our approach provides substantial advantages over prior work. Our method is significantly faster in comparison to sharding [11, 32] and retraining while reaching a similar level of accuracy. Moreover, due to the compact updates, our approach requires only a fraction of the training data and hence is applicable when the original data is not available. We demonstrate the efficacy of our approach in case studies on unlearning (a) sensitive features in linear models, (b) unintended memorization in language models, and (c) label poisoning in computer vision.

Contributions. In summary, we make the following major contributions in this paper:

(1) *Unlearning with closed-form updates.* We introduce a novel framework for unlearning features and labels. This framework builds on closed-form updates of model parameters and thus is signicantly faster than instance-based approaches to unlearning.

(2) *Certified unlearning.* We derive two unlearning strategies for our framework based on first-order and second-order gradient updates. Under convexity and continuity assumptions

on the loss, we show that both strategies provide certified unlearning of data.

(3) *Empirical analysis.* We empirically show that unlearning of sensible information is possible even for deep neural networks with non-convex loss functions. We find that our first-order update is highly efficient, enabling a speed-up over retraining by several orders of magnitude.

The rest of the paper is structured as follows: We review related work on machine unlearning and influence functions in Section 2. Our approach and its different technical realizations are introduced in Sections 3 and 4, respectively. The theoretical analysis of our approach is presented in Section 5 and its empirical evaluation in Section 6. Finally, we discuss limitations in Section 7 and conclude the paper in Section 8.

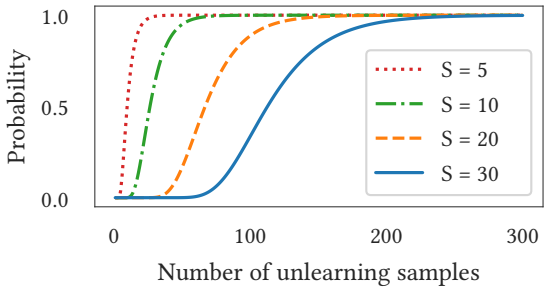

Figure 1: Probability of all shards being affected when unlearning for varying number of data points and shards (S).

2 RELATED WORK

The increasing application of machine learning to personal data has started a series of research on detecting and correcting privacy issues in learning models [e.g., 15, 16, 47, 55, 57, 63]. In the following, we provide an overview of work on machine unlearning and influence functions. A broader discussion of privacy and machine learning is given by De Cristofaro [21] and Papernot et al. [51].

Machine unlearning. Methods for removing sensitive data from learning models are a recent branch of security research. As one of the first, Cao & Yang [14] show that a large number of learning models can be represented in a closed summation form that allows for elegantly removing individual data points in retrospection. However, for adaptive learning strategies, such as stochastic gradient descent, this approach provides only little advantage over retraining from scratch and thus is not well suited for correcting problems in deep neural networks.

As a remedy, Bourtoule et al. [11] propose a universal strategy for unlearning data instances from classification models. Similarly, Ginart et al. [32] develop a technique for unlearning points from clusterings. The key idea of both approaches is to split the data into independent partitions—so called shards—and aggregate the final model from submodels trained over these shards. In this setting, the unlearning of data points can be efficiently carried out by only retraining the affected submodels. We refer to this unlearning strategy as *sharding*. Aldaghri et al. [4] show that this approach can be further sped up for least-squares regression by choosing the shards cleverly. Sharding, however, is suitable for removing a few data points only and inevitably deteriorates in performance when larger portions of the data require changes.

This limitation of sharding is schematically illustrated in Figure 1. The probability that all shards need to be retrained increases with the number of data points to be corrected. For a practical setup with 20 shards, as proposed by Bourtoule et al. [11], changes to as few as 150 points are already sufficient to impact all shards and render this form of unlearning inefficient, regardless of the size of the training data. We provide a detailed analysis of this limitation in Appendix A.1. Consequently, for privacy leaks involving hundreds or thousands of data points, sharding provides no advantages over costly retraining.

Influence functions. The concept of influence functions that forms the basis of our approach originates from the area of robust statistics [39]. It first was introduced by Cook & Weisberg [20] for investigating the changes of simple linear regression models. Although the proposed techniques have been occasionally employed in machine learning [40, 46], it was the seminal work of Koh & Liang [43] that recently brought general attention to this concept and its application to modern learning models. In particular, this work uses influence functions for explaining the impact of data points on the predictions of learning models.

Influence functions have then been used in a wide range of applications. For example, they have been applied to trace bias in word embeddings back to documents [13, 19], determine reliable regions in learning models [56], and explain deep neural networks [9]. As part of this research strain, Basu et al. [8] increase the accuracy of influence functions by using high-order approximations, Barshan et al. [7] improve their precision through nearest-neighbor strategies, and Guo et al. [37] reduce their runtime by focusing on specific samples. Finally, Golatkar et al. [33, 34] move to the field of privacy and use influence functions to "forget" data points in neural networks using special approximations.

In terms of theoretical analysis, Koh et al. [44] study the accuracy of influence functions when estimating the loss on test data and Neel et al. [50] perform a similar analysis for gradient-based update strategies. In addition, Rad & Maleki [54] show that the prediction error on leave-one-out validations can be reduced with influence functions. Finally, Guo et al. [36] introduce the idea of certified removal of data points and propose a definition of indistinguishability between learning models similar to Neel et al. [50]. In this paper we introduce a natural extension of these concepts to define the certified unlearning of features and labels.

Difference to our approach. All prior approaches to unlearning remain on the level of data instances and cannot be used for correcting other types of privacy issues. To our knowledge, we are the first to build on the concept of influence functions for unlearning features and labels from learning models. Figure 2 visualizes this difference between instance-based unlearning and our approach. Note that in both cases the amount of data to be removed is roughly the same, yet far more data instances are affected when features or labels need to be changed.

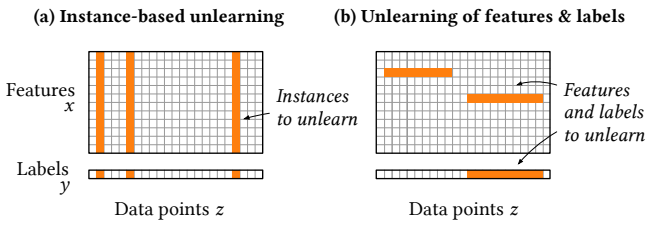

Figure 2: Instance-based unlearning vs. unlearning of features and labels. The data to be removed is marked with orange.

3 UNLEARNING WITH UPDATES

Let us start by introducing a basic learning setup. We consider a supervised learning task that is described by a dataset $D = \{z_1, \ldots, z_n\}$ with each point $z_i = (x, y)$ consisting of features $x \in \mathcal{X}$ and a label $y \in \mathcal{Y}$. We assume that $\mathcal{X} = \mathbb{R}^d$ is a vector space and denote the j-th feature of x by $x[j]$. Given a loss function $\ell(z, \theta)$ that measures the difference between the predictions of a learning model θ with p parameters and the true labels, the optimal model θ^* can be found by minimizing the regularized empirical risk,

$$\theta^* = \operatorname*{argmin}_\theta L_b(\theta; D) = \operatorname*{argmin}_\theta \sum_{i=1}^{n} \ell(z_i, \theta) + \lambda\Omega(\theta) + b^T\theta \quad (1)$$

where Ω is a common regularizer [see 25]. Note that we add a vector $b \in \mathbb{R}^p$ to the optimization. For conventional learning, this vector is set to zero and can be ignored. To achieve certified unlearning, however, it enables to add a small amount of noise to the minimization, similar to differentially private learning strategies [26, 28]. We introduce this technique later in Section 5 and thus omit the subscript in L_b for the sake of clarity now. The process of unlearning amounts to adapting θ^* to changes in D without recalculating the optimization problem in Equation (1).

3.1 Unlearning Data Points

To provide an intuition for our approach, we begin by asking a simple question: How would the optimal learning model θ^* change, if only one data point z had been perturbed by some change δ? Replacing z by $\tilde{z} = (x + \delta, y)$ leads to the new optimal model:

$$\theta^*_{z \to \tilde{z}} = \operatorname*{argmin}_\theta L(\theta; D) + \ell(\tilde{z}, \theta) - \ell(z, \theta). \quad (2)$$

However, calculating the new model $\theta^*_{z \to \tilde{z}}$ exactly is expensive and does not provide any advantage over solving the problem in Equation (1). Instead of replacing the data point z with $\tilde{z}$, we can also up-weight $\tilde{z}$ by a small value ϵ and down-weight z accordingly, resulting in the following optimization problem:

$$\theta^*_{\epsilon, z \to \tilde{z}} = \operatorname*{argmin}_\theta L(\theta; D) + \epsilon\ell(\tilde{z}, \theta) - \epsilon\ell(z, \theta). \quad (3)$$

Equations (2) and (3) are obviously equivalent for $\epsilon = 1$ and solve the same problem.

Consequently, we do not need to explicitly remove a data point from the training data but can revert its *influence* on the learning model through a combination of appropriate up-weighting and down-weighting. It is easy to see that this approach is not restricted to a single data point. We can simply define a set of data points Z as well as their perturbed versions $\tilde{Z}$ and arrive at the following optimization problem

$$\theta^*_{\epsilon, Z \to \tilde{Z}} = \operatorname*{argmin}_\theta L(\theta; D) + \epsilon \sum_{\tilde{z} \in \tilde{Z}} \ell(\tilde{z}, \theta) - \epsilon \sum_{z \in Z} \ell(z, \theta). \quad (4)$$

This generalization enables us to approximate changes on larger portions of the training data. Instead of solving the problem in Equation (4) directly, however, we formulate the optimization as an update of the original model θ^*. That is, we seek a closed-form update $\Delta(Z, \tilde{Z})$ of the model parameters, such that

$$\theta^*_{\epsilon, Z \to \tilde{Z}} \approx \theta^* + \Delta(Z, \tilde{Z}), \quad (5)$$

where $\Delta(Z, \tilde{Z})$ has the same dimension as the learning model θ but is sparse and affects only the necessary weights.

As a result of this formulation, we can describe changes of the training data as a compact update Δ rather than iteratively solving an optimization problem. We show in Section 4 that this update step can be efficiently computed using first-order and second-order derivatives. Furthermore, we prove in Section 4 that the unlearning success of both updates can be certified up to a tolerance ϵ if the loss function ℓ is strictly convex, twice differentiable, and Lipschitz-continuous. Notice that if $\tilde{Z} = \emptyset$ in Equation (4), our approach also yields updates to remove *data points* similar to prior work [36, 43].

3.2 Unlearning Features and Labels

Equipped with a general method for updating a learning model, we proceed to introduce our approach for unlearning features and labels. To this end, we expand our notion of perturbations and include changes to labels by defining

$$\tilde{z} = (x + \delta_x, y + \delta_y),$$

where δ_x modifies the features of a data point and δ_y its label. By using different changes in the perturbations $\tilde{Z}$, we can now realize different types of unlearning using closed-form updates.

Replacing features. As the first type of unlearning, we consider the task of correcting features in a learning model. This task is relevant if the content of some features violates the privacy of a user and needs to be replaced with alternative data. As an example, personal names, identification numbers, residence addresses, or other sensitive information might need to be removed after a model has been trained on a corpus of emails.

For a set of features F and their new values V, we define perturbations on the affected points Z by

$$\tilde{Z} = \big\{(x[f] = v, y) : (x, y) \in Z,\ (f, v) \in F \times V\big\}.$$

For example, a credit card number contained in the training data can be blinded by a random number sequence in this setting. The values V can be adapted individually for each data point, such that fine-grained corrections are possible.

Replacing labels. As the second type of unlearning, we focus on correcting labels. This form of unlearning is necessary if the labels captured in a model contain unwanted or inappropriate information. For example, in generative language models, the training text is used as input features (preceding tokens) *and* labels (target tokens)

[35, 59]. Hence, defects can only be eliminated if the labels are unlearned as well.

For the affected points Z and the set of new labels Y, we define the corresponding perturbations by

$$\tilde{Z} = \big\{(x, y) \in Z_x \times Y\big\},$$

where Z_x corresponds to the data points in Z without their original labels. The new labels Y can also be individually selected for each data point, as long as they come from the domain $\mathcal{Y}$, that is, $Y \subset \mathcal{Y}$. Note that the replaced labels and features can be easily combined in one set of perturbations $\tilde{Z}$, so that defects affecting both can be corrected in a single update. In Section 6.2, we demonstrate that this combination can be used to remove unintended memorization from generative language models with high efficiency.

Revoking features. Based on appropriate definitions of Z and $\tilde{Z}$, our approach enables to replace the content of features and thus eliminate privacy leaks by overwriting sensitive data. In some scenarios, however, it might be necessary to even completely remove features from a learning model—a task that we denote as *revocation*. In contrast to the correction of features, this form of unlearning poses a unique challenge: The revocation of features can reduce the input dimension of the model. While this adjustment can be easily carried out through retraining with adapted data, constructing a model update as in Equation (5) becomes tricky.

To address this problem, let us consider a model θ^* trained on a dataset $D \subset \mathbb{R}^d$. If we remove some features F from this dataset and train the model again, we obtain a new optimal model θ^*_{-F} with reduced input dimension. By contrast, if we set the values of the features F to zero in the dataset and train again, we obtain an optimal model $\theta^*_{F=0}$ with the same input dimension as θ^*. Fortunately, these two models are equivalent for a large class of learning models, including support vector machines and several neural networks as the following lemma shows.

Lemma 1. *For learning models processing inputs x using linear transformations of the form $\theta^T x$, we have $\theta^*_{-F} \equiv \theta^*_{F=0}$.*

Proof. It is easy to see that it is irrelevant for the dot product $\theta^T x$ whether a dimension of x is missing or equals zero in the linear transformation

$$\sum_{k:k\notin F} \theta[k]x[k] = \sum_k \theta[k]\mathbf{1}\{k \notin F\}x[k].$$

As a result, the loss $\ell(z, \theta) = \ell(\theta^T x, y, \theta)$ of both models is identical for every data point z. Hence, $L(\theta; D)$ is also equal for both models and thus the same objective is minimized during learning, resulting in equal model parameters. □

Lemma 1 enables us to erase features from many learning models by first setting them to zero, calculating the parameter update, and then reducing the input dimension of the models accordingly.

Concretely, to revoke the features F from a learning model, we first locate all data points where these features are non-zero with

$$Z = \big\{(x, y) \in D : x[f] \neq 0, f \in F\big\}.$$

Then, we construct corresponding perturbations so that the features are set to zero with our approach,

$$\tilde{Z} = \big\{(x[f] = 0, y) : (x, y) \in Z,\ f \in F\big\}.$$

Finally, we adapt the input dimension by removing the affected inputs of the learning models, such as the corresponding neurons in the input layer of a neural network.

4 UPDATE STEPS FOR UNLEARNING

Our approach rests on changing the influence of training data with a closed-form update of the model parameters. In the following, we derive two strategies for calculating this closed form: a *first-order update* and a *second-order update*. The first strategy builds on the gradient of the loss function and thus can be applied to any model with differentiable loss. The second strategy incorporates second-order derivatives which limits the application to loss functions with an invertible Hessian matrix.

4.1 First-Order Update

Recall that we aim to find an update $\Delta(Z, \tilde{Z})$ that we can add to our model θ^* for unlearning. If the loss ℓ is differentiable, we can compute an optimal *first-order update* simply as follows

$$\Delta(Z, \tilde{Z}) = -\tau\Big(\sum_{\tilde{z}\in\tilde{Z}} \nabla_\theta \ell(\tilde{z}, \theta^*) - \sum_{z\in Z} \nabla_\theta \ell(z, \theta^*)\Big) \tag{6}$$

where τ is a small constant that we refer to as *unlearning rate*. A complete derivation of Equation (6) is given in Appendix A.2. Intuitively, this update shifts the model parameters in the direction from $\sum_{z\in Z} \nabla\ell(z, \theta^*)$ to $\sum_{\tilde{z}\in\tilde{Z}} \nabla\ell(\tilde{z}, \theta^*)$ where the size of the update step is determined by the rate τ. This update strategy is related to the classic gradient descent update GD used in many learning algorithms and given by

$$\mathrm{GD}(\tilde{Z}) = -\tau \sum_{\tilde{z}\in\tilde{Z}} \nabla_\theta \ell(\tilde{z}, \theta^*).$$

However, it differs from this update step in that it moves the model to the *difference* in gradient between the original and perturbed data, which minimizes the loss on $\tilde{z}$ and at the same time removes the information contained in z.

The first-order update is a simple and yet effective strategy: Gradients of ℓ can be computed in $O(p)$ [53] and modern auto-differentiation frameworks like TensorFlow [1] and PyTorch [52] offer easy gradient computations for the practitioner. The update step, however, involves a parameter τ that controls the impact of the unlearning step. To ensure that data has been completely replaced, it is thus necessary to calibrate this parameter using a measure for the success of unlearning. In Section 6, for instance, we show how the exposure metric proposed by Carlini et al. [15] can be used for this calibration when removing unintendend memorization from language models.

4.2 Second-Order Update

The calibration of the update step can be eliminated if we make further assumptions on the properties of the loss ℓ. If we assume that ℓ is twice differentiable and strictly convex, the influence of a single data point can be approximated in closed form by

$$\frac{\partial\theta^*_{\epsilon,z\to\tilde{z}}}{\partial\epsilon}\bigg|_{\epsilon=0} = -H^{-1}_{\theta^*}\big(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*)\big),$$

where $H^{-1}_{\theta^*}$ is the inverse Hessian of the loss at θ^*, that is, the inverse matrix of the second-order partial derivatives [see 20]. We

can now perform a linear approximation for $\theta^*_{z\to\tilde{z}}$ to obtain

$$\theta^*_{z\to\tilde{z}} \approx \theta^* - H^{-1}_{\theta^*}\big(\nabla_\theta \ell(\tilde{z},\theta^*) - \nabla_\theta \ell(z,\theta^*)\big). \tag{7}$$

Since all operations are linear, we can easily extend Equation (7) to account for multiple data points and derive the following *second-order update*:

$$\Delta(Z,\tilde{Z}) = -H^{-1}_{\theta^*}\Big(\sum_{\tilde{z}\in\tilde{Z}} \nabla_\theta \ell(\tilde{z},\theta^*) - \sum_{z\in Z} \nabla_\theta \ell(z,\theta^*)\Big). \tag{8}$$

A full derivation of this update step is provided in Appendix A.2. Note that the update does not require a parameter calibration, since the parameter weighting of the changes is directly derived from the inverse Hessian of the loss function.

The second-order update is the preferred strategy for unlearning on models with a strongly convex and twice differentiable loss function that guarantee the existence of $H^{-1}_{\theta^*}$. Technically, the update step in Equation (8) can be easily calculated with common machine-learning frameworks. In contrast to the first-order update, however, this computation involves the inverse Hessian matrix, which can be difficult to construct for neural networks.

Efficiently calculating the inverse Hessian. Given a model $\theta \in \mathbb{R}^p$ with p parameters, forming and inverting the Hessian requires $O(np^2 + p^3)$ time and $O(p^2)$ space [43]. For models with a small number of parameters, the matrix can be pre-computed and explicitly stored, such that each subsequent request for unlearning only involves a simple matrix-vector multiplication. For example, in Section 6.1, we show that unlearning features from a logistic regression model with about 2,000 parameters can be realized with this approach in a few milliseconds.

For complex learning models, such as deep neural networks, the Hessian matrix quickly becomes too large for explicit storage. Still, we can approximate the inverse Hessian using techniques proposed by Koh & Liang [43]. The explicit derivation and an algorithm for implementation is presented in Appendix A.3. While this approximation weakens the theoretical guarantees of our approach, it still enables succesfully unlearning data from large learning models. In Section 6.2 we demonstrate that this strategy can be used to calculate second-order updates for a recurrent neural network with 3.3 million parameters in less than 30 seconds.

5 CERTIFIED UNLEARNING

Machine unlearning is a delicate task, as it aims at reliably removing privacy issues and sensitive data from learning models. This task should ideally build on theoretical guarantees to enable *certified unlearning*, where the corrected model is stochastically indistinguishable from one created by retraining. In the following, we derive conditions under which the updates of our approach introduced in Section 4.2 provide certified unlearning. To this end, we build on the concepts of *differential privacy* [26] and *certified data removal* [36], and adapt them to the unlearning task.

For a training dataset D, let $\mathcal{A}$ be a learning algorithm that outputs a model $\theta \in \Theta$ after training on D, that is, $\mathcal{A} : D \to \Theta$. Randomness in $\mathcal{A}$ induces a probability distribution over the output models in Θ. Moreover, we consider an unlearning method $\mathcal{U}$ that maps a model θ to a corrected model $\theta_\mathcal{U} = \mathcal{U}(\theta, D, D')$ where D' denotes the dataset containing the perturbations $\tilde{Z}$ required for the unlearning task. To measure the difference between a model trained on D' and one obtained by $\mathcal{U}$ we introduce the concept of *ϵ-certified unlearning* as follows

Definition 1. *Given some $\epsilon > 0$ and a learning algorithm $\mathcal{A}$, an unlearning method $\mathcal{U}$ is ϵ-certified if*

$$e^{-\epsilon} \le \frac{P\Big(\mathcal{U}\big(\mathcal{A}(D), D, D'\big) \in \mathcal{T}\Big)}{P\big(\mathcal{A}(D') \in \mathcal{T}\big)} \le e^{\epsilon}$$

holds for all $\mathcal{T} \subset \Theta$, D, and D'.

This definition ensures that the probability to obtain a model using the unlearning method $\mathcal{U}$ and training a new model on D' from scratch deviates at most by ϵ. Following the work of Guo et al. [36], we introduce *(ϵ, δ)-certified unlearning*, a relaxed version of ϵ-certified unlearning, defined as follows.

Definition 2. *Under the assumptions of Definition 1, an unlearning method $\mathcal{U}$ is (ϵ, δ)-certified if*

$$P\Big(\mathcal{U}\big(\mathcal{A}(D), D, D'\big) \in \mathcal{T}\Big) \le e^{\epsilon} P\big(\mathcal{A}(D') \in \mathcal{T}\big) + \delta$$

and

$$P\big(\mathcal{A}(D') \in \mathcal{T}\big) \le e^{\epsilon} P\Big(\mathcal{U}\big(\mathcal{A}(D), D, D'\big) \in \mathcal{T}\Big) + \delta$$

hold for all $\mathcal{T} \subset \Theta$, D, and D'.

This definition allows the unlearning method $\mathcal{U}$ to slightly violate the conditions from Definition 1 by a constant δ. Using these definitions, it becomes possible to derive practical conditions under which our approach realizes certified unlearning.

5.1 Certified Unlearning of Features and Labels

Using the concept of certified unlearning, we can construct and analyze theoretical guarantees of our approach when removing features and labels. To ease this analysis, we make two basic assumptions on the employed learning algorithm: First, we assume that the loss function ℓ is twice differentiable and strictly convex such that H^{-1} always exists. Second, we consider L_2 regularization in optimization problem (1), that is, $\Omega(\theta) = \frac{1}{2}\|\theta\|_2^2$ which ensures that the loss function is strongly convex. Both assumptions are satisified by a wide range of learning models, including logistic regression and support vector machines.

A helpful tool for analyzing the task of unlearning is the *gradient residual* $\nabla L(\theta; D')$ for a given model θ and a corrected dataset D'. For strongly convex loss functions, the gradient residual is zero *if and only if* θ equals $\mathcal{A}(D')$ since in this case the optimum is unique. Therefore, the norm of the gradient residual $\|\nabla L(\theta; D')\|_2$ reflects the distance of a model θ from one obtained by retraining on the corrected dataset D'. We differentiate between the gradient residual of the plain loss L and an adapted loss L_b where a random vector b is added. The gradient residual r of L_b is given by

$$r = \nabla L_b(\theta; D') = \sum_{z\in D'}^{n} \nabla \ell(z,\theta) + \lambda\theta + b$$

and differs from the gradient residual of L only by the added vector b. This allows us to adapt the underlying distribution of b to achieve certified unlearning, similar to sensitivity methods [28]. The corresponding proofs are given in Appendix A.4.

Theorem 1. *Assume that $\|x_i\|_2 \leq 1$ for all data-points and the loss $\nabla \ell(z, \theta)$ is γ_z-Lipschitz with respect to z at θ^* and γ-Lipschitz with respect to θ. Further let the perturbations change the feature dimensions $j, \ldots, j+F$ by magnitudes at most $m_j, \ldots, m_{j+F}$. If $M = \sum_{j=1}^{F} m_j$ the following upper bounds hold:*

(1) For the first-order update of our approach, we have

$$\left\|\nabla L\left(\theta^*_{Z \to \tilde{Z}}, D'\right)\right\|_2 \leq (1 + \tau\gamma n)\gamma_z M|Z|$$

(2) If $\nabla^2 \ell(z, \theta)$ is γ''-Lipschitz with respect to θ, we have

$$\left\|\nabla L\left(\theta^*_{Z \to \tilde{Z}}, D'\right)\right\|_2 \leq \frac{\gamma_z^2 \gamma''}{\lambda^2} M^2 |Z|^2$$

for the second-order update of our approach.

In order to obtain a small gradient residual norm for the first-order update the unlearning rate should be small, ideally in the order of $1/n\gamma$. Since $\|x_i\|_2 \leq 1$ we also have $m_j \ll 1$ if d is large and thus M acts as an additional damping factor for both updates when changing or revoking features.

Theorem 1 enables us to quantify the difference between unlearning and retraining from scratch. Concretely, if $\mathcal{A}(D')$ is an exact minimizer of L_b on D' with density $f_{\mathcal{A}}$ and $\mathcal{U}(\mathcal{A}(D), D, D')$ an approximated minimum obtained through unlearning with density $f_{\mathcal{U}}$, then Guo et al. [36] show that the max-divergence between $f_{\mathcal{A}}$ and $f_{\mathcal{U}}$ for the model θ produced by $\mathcal{U}$ can be bounded using the following theorem.

Theorem 2 (Guo et al. [36]). *Let $\mathcal{U}$ be an unlearning method with a gradient residual r with $\|r\|_2 \leq \epsilon'$. If the vector b is drawn from a probability distribution with density p satisfying that for any $b_1, b_2 \in \mathbb{R}^d$ there exists an $\epsilon > 0$ such that $\|b_1 - b_2\| \leq \epsilon'$ implies $e^{-\epsilon} \leq \frac{p(b_1)}{p(b_2)} \leq e^{\epsilon}$ then*

$$e^{-\epsilon} \leq \frac{f_{\mathcal{U}}(\theta)}{f_{\mathcal{A}}(\theta)} \leq e^{\epsilon}$$

for any θ produced by the unlearning method $\mathcal{U}$.

Theorem 2 equips us with a way to prove the certified unlearning property from Definition 1. Using the gradient residual bounds derived in Theorem 1, we can now adjust the density function underlying the vector b so that Theorem 2 holds for both update steps of our unlearning approach.

Theorem 3. *Let $\mathcal{A}$ be the learning algorithm that returns the unique minimum of $L_b(\theta; D')$ and let $\mathcal{U}$ be an unlearning method that produces a model $\theta_{\mathcal{U}}$. If $\|\nabla L(\theta_{\mathcal{U}}; D')\|_2 \leq \epsilon'$ for some $\epsilon' > 0$ we have the following guarantees.*

(1) If b is drawn from a distribution with density $p(b) = e^{-\frac{\epsilon}{\epsilon'}\|b\|_2}$ then the method $\mathcal{U}$ performs ϵ-certified unlearning for $\mathcal{A}$.
(2) If $p \sim \mathcal{N}(0, c\epsilon'/\epsilon)^d$ for some $c > 0$ then the method $\mathcal{U}$ performs (ϵ, δ)-certified unlearning for $\mathcal{A}$ with $\delta = 1.5e^{-c^2/2}$.

Theorem 3 finally allows us to establish certified unlearning of features and labels in practice: Given a learning model with a bounded gradient residual norm and a privacy budget (ϵ, δ) we can calibrate the distribution of b to obtain certified unlearning.

5.2 Relation to Differential Privacy

Our definition of certified unlearning shares interesting similarities with the concept of differential privacy [26] that we highlight in the following. First, let us recall the definition of differential privacy for a learning algorithm $\mathcal{A}$:

Definition 3. *Given some $\epsilon > 0$, a learning algorithm $\mathcal{A}$ is said to be ϵ-differentially private (ϵ-DP) if*

$$e^{-\epsilon} \leq \frac{P\left(\mathcal{A}(D) \in \mathcal{T}\right)}{P\left(\mathcal{A}(D') \in \mathcal{T}\right)} \leq e^{\epsilon}$$

holds for all $\mathcal{T} \subset \Theta$ and datasets D, D' that differ in one sample[1].

By this definition, differential privacy is a *sufficient* condition for certified unlearning. We obtain the same bound by simply setting the unlearning method $\mathcal{U}$ to the identity function in Definition 1. That is, if we cannot distinguish whether $\mathcal{A}$ was trained with a point z or its modification z_δ, we do not need to worry about unlearning z later. This result is also obtained by Theorem 1 when we set the unlearning rate τ to zero and therefore no model updates are performed during unlearning.

A large body of research has been concerned with obtaining differential private algorihms for the adapted loss L_b [2, 17, 18, 58]. For example, Chaudhuri et al. [18] demonstrate that we can sample $b \sim e^{-\alpha\|b\|}$ for linear models and arrive at an ϵ-DP optimization for a specific definition of α. Similarly, Abadi et al. [2] find that T stochastic gradient-descent steps using a fraction q of the data result in an $O\left(q\epsilon\sqrt{T}, \delta\right)$-DP optimization if b is sampled from a specific Gaussian distribution. While these algorithms enable learning with privacy guarantees, they build on Definition 3 and hence cannot account for the more nuanced changes induced by an unlearning method as defined in Definition 1.

Consequently, certified unlearning can be obtained through DP, yet the resulting classification performance may suffer when enforcing strong privacy guarantees [see 2, 18]. In Section 6, we demonstrate that the models obtained using our update strategies are much closer to $\mathcal{A}(D')$ in terms of performance compared to DP alone. In this light, certified unlearning can be seen as a compromise between the high privacy guarantees of DP and the optimal performance achievable through re-training from scratch.

5.3 Multiple Unlearning Steps

So far, we have considered unlearning as a one-shot strategy. That is, all changes are incorporated in $\tilde{Z}$ before performing an update. However, Theorem 1 shows that the error of the updates rises linearly with the number of affected points and the size of the total perturbation. Instead of performing a single update, it thus becomes possible to split $\tilde{Z}$ into T subsets and conduct T consecutive updates. In terms of run-time it is easy to see that the total number of gradient computations remains the same for the first-order update. For the second-order strategy, however, multiple updates require calculting the inverse Hessian for each intermediate step, which increases the computational effort. An empirical comparison of our approach with multiple update steps is presented in Appendix A.5.4.

[1] Here we define "differ in one sample" so that $|D| = |D'|$ and one sample has been replaced. This is denoted as "bounded differential privacy" in literature [22, 41].

In terms of unlearning certifications, we can extend Theorem 1 and show that the gradient residual bound after T update steps remains smaller than TC, where C is the bound of a single step: If θ_t is the t-th solution obtained by our updates with gradient residual r_t then θ_t is an exact solution of the loss function $L_b(\theta, D') - r_t^T\theta$ by construction. This allows applying Theorem 1 to each θ_t and obtain the bound TC by the triangle inequality. Therefore, the gradient residual bound rises linearly in the number of applications of $\mathcal{U}$. This result is in line with the composition theorem of differential privacy research [27, 29, 30] which states that applying an ϵ-DP algorithm n times results in a $n\epsilon$-DP algorithm.

6 EMPIRICAL ANALYSIS

We proceed with an empirical analysis of our approach and its capabilities. For this analysis, we examine the efficacy of unlearning in practical scenarios and compare our method to other strategies for removing data from learning models, such as sharding, retraining and fine-tuning. As part of these experiments, we employ models with convex and non-convex loss functions to understand how this property affects the success of unlearning.

Unlearning scenarios. Our empirical analysis is based on the following three scenarios in which sensitive and personal information need to be removed from learning models.

Scenario 1: Sensitive features. Our first scenario deals with linear models for classification in different applications. If these models are constructed from personal data, such as emails or health measurements, they can accidentially capture private information. For example, when bag-of-words features are extracted from text, the model may contain names, postal zip codes and other sensitive words. We thus investigate how our approach can unlearn such features in retrospection (see Section 6.1).

Scenario 2: Unintended memorization. In the second scenario, we consider the problem of unintended memorization [15]. Language models based on recurrent neural networks are a powerful tool for completing and generating text. However, these models also memorize rare but sensitive data, such as credit card numbers or private messages. This memorization poses a privacy problem: Through specifically crafted input sequences, an attacker can extract this data from the models during text completion [15, 16]. We apply unlearning of features and labels to remove identified leaks from language models (see Section 6.2).

Scenario 3: Data poisoning. For the third scenario, we focus on poisoning attacks in computer vision. Here, an adversary aims at misleading an object recognition task by flipping a few labels of the training data. The label flips significantly reduce the performance of the employed convolutional neural network and sabotage its succesful application. We use unlearning as a strategy to correct this defect and restore the original performance without costly retraining (see Section 6.3).

Performance measures. Unlike other tasks in machine learning, the performance of unlearning does not depend on a single numerical quantity. For example, one method may fail to completely remove private data, while another may succeed but at the same time degrade the prediction performance. We identify three aspects that contribute to effective unlearning and thus consider three performance measures for our empirical analysis.

Efficacy of unlearning. The most important factor for successful unlearning is the removal of data. While certified unlearning ensures this removal, we cannot provide similar guarantees for models with non-convex loss functions. As a result, we need to employ measures that quantitatively assess the *efficacy* of unlearning. For example, we use the *exposure metric* [15] to measure the memorization of specific sequences in language models after unlearning.

Fidelity of unlearning. The second factor contributing to the success of unlearning is the performance of the corrected model, which we denote as *fidelity*. An unlearning method is of practical use only if it keeps the performance as close as possible to the original model. Hence, we consider the fidelity of the corrected model as the second performance measure. In our experiments, we use the loss and accuracy of the original and corrected model on a hold-out set to determine this fidelity.

Efficiency of unlearning. If the training data used to generate a model is still available, a simple unlearning strategy is retraining from scratch. This strategy, however, involves significant runtime and storage costs. Therefore, we consider the *efficiency* of unlearning as the third factor. In our experiments, we measure the runtime and the number of gradient calculations for each unlearning method on the datasets of the three scenarios.

Baseline methods. To compare our approach with prior work on machine unlearning, we employ different baseline methods as reference for examining the efficacy, fidelity, and efficiency of our approach. In particular, we consider retraining, fine-tuning, diffential privacy and sharding (SISA) as baselines.

Retraining. As the first baseline, we employ retraining from scratch. This basic method is applicable if the original training data is available and guarantees a proper removal of data. However, the approach is costly and serves as general upper bound for the runtime of any unlearning method.

Differential privacy (DP). As a second baseline, we consider a differentially private learning model. As discussed in Section 5.2, the presence of the noise term b in our model θ^* induces a differential-privacy guarantee which is sufficient for certified unlearning. Therefore, we evaluate the performance of θ^* without any following unlearning steps.

Fine-tuning. Instead of retraining, this baseline continues to train a model using corrected data. We implement fine-tuning by performing stochastic gradient descent over the training data for one epoch. This naive unlearning strategy serves as a middle ground between costly retraining and specialized methods, such as SISA.

SISA (Sharding). As the fourth baseline, we consider the unlearning method SISA proposed by Bourtoule et al. [10]. The method splits the training data in shards and trains submodels for each shard separately. The final classification is obtained by a majority vote over these submodels. Unlearning is conducted by retraining those shards that contain data to be removed. In our evaluation, we simply retrain all shards that contain features or labels to unlearn.

6.1 Unlearning Sensitive Features

In our first unlearning scenario, we revoke sensitive features from linear learning models trained on different real-world datasets. In particular, we employ datasets for spam filtering [49], Android malware detection [5], diabetis forecasting [24] and prediction of income based on census data [23]. An overview of the datasets and their statistics is given in Table 1.

Table 1: Datasets for unlearning sensitive features.

	Spam	Malware	Adult	Diabetis
Data points	33,716	49,226	48,842	768
Features	4,902	2,081	81	8

We divide the datasets into a training and test partition with a ratio of 80 % and 20 %, respectively. To create a feature space for learning, we use the available numerical features of the Adult and Diabetis dataset and extract bag-of-words features from the samples in the Spam and Malware datasets. Finally, we train logistic regression models for all datasets and use them to evaluate the capabilities of the different unlearning methods.

Sensitive features. For each dataset, we define a set of sensitive features for unlearning. As no privacy leaks are known for the four datasets, we focus on features that *potentially* cause privacy issues. For the Spam dataset, we remove 100 features associated with personal names from the learning model, such as first and last names contained in emails. For the Adult dataset, we change the marital status, sex, and race of 100 randomly selected individuals to simulate the removal of discriminatory bias. For the Malware dataset, we remove 100 URLs from the classifier, and for the Diabetis dataset, we adjust the age, body mass index, and sex of 100 individuals to imitate potential unlearning requests. To avoid a sampling bias, we randomly draw 100 combinations of these unlearning changes for all of the four datasets.

Unlearning task. Based on the definition of sensitive features, we then apply the different unlearning methods to the respective learning models. Technically, our approach benefits from the convex loss function of the logistic regression, which allows us to apply certified unlearning as presented in Section 5. Specifically, it is easy to see that Theorem 3 holds since the gradients of the logistic regression loss are bounded and are thus Lipschitz-continuous.

Efficacy evaluation. Our approach and the DP baseline need to be calibrated to provide certified unlearning with minimal perturbations. To this end, we consider a privacy budget (ϵ, δ) that must not be exceeded and thus determines a bound for the gradient residual norm. Concretely, for given parameters (ϵ, δ) and a perturbation vector b that has been sampled from a Gaussian normal distribution with variance σ, the gradient residual must be smaller than

$$\beta = \frac{\sigma\epsilon}{c}, \qquad \text{where } c = \sqrt{2\log(1.5/\delta)}. \tag{9}$$

If σ becomes too large the performance of the classifier might suffer. Consequently, we first evaluate the effect of σ and λ on the performance of our classifier to be able to evaluate the empirical gradient residual norms. Table 2 shows the performance of the Spam model for varying parameters of σ and λ. The corresponding results for the other datasets can be found in Appendix A.5.1. As expected, a large variance σ impacts the classification performance and can make the model unusable in practice whereas the effect of the regularization λ is small.

Table 2: Accuracy of the Spam for varying parameters σ and λ.

λ \ σ	1	10	20	50	100
0.01	95.5	92.1	87.5	83.7	70.2
1	97.9	89.8	85.6	78.6	73.7
10	97.0	94.6	89.7	80.2	73.4

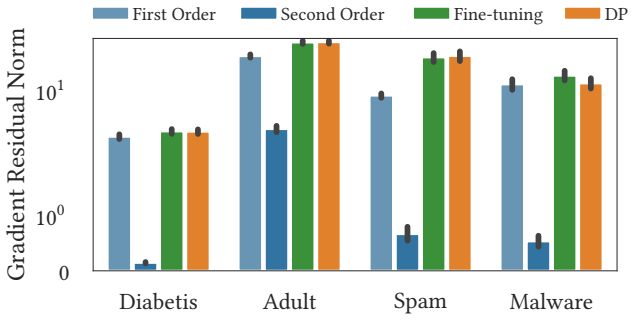

Figure 3: Distribution of the Gradient residual norm for different unlearning tasks. Lower residuals reflect better approximation and allow smaller values of ϵ.

The distribution of the gradient residual norm after unlearning is presented in Figure 3 for the different unlearning approaches. We can observe that the second-order update achieves much smaller gradient residuals with small variance, while the other strategies produce higher residuals. This trend holds in general even if the number of affected features is increased or reduced. Since retraining always produces gradient residuals of zero, the second-order approach is best suited for unlearning in this scenario. Moreover, due to the certified unlearning setup, the first-order and second-order approaches will also allow for the smallest privacy budget ϵ and thus give a stronger guarantee that the sensitive features have been revoked from the four learning models.

What do the results above mean for a practical example? Recall that according to Equation (9), we have to sample b from a normal distribution with variance $\sigma = \beta c/\epsilon$. In the following, let us fix $\epsilon = 0.1$ and $\delta = 0.02$ as a desirable privacy budget, which yields $c \approx 2.9$. For the Spam dataset, we obtain a gradient residual in the order of 0.01 when removing a single feature and in the order of 0.1 when removing 100 features. Consequently, we need $\sigma = 0.29$ and $\sigma = 2.9$, respectively, which still gives good classification performance according to Table 2. In contrast, the gradient residual norm of the plain DP model is in the order of 10, which requires $\sigma = 290$, a value that gives a classification performance of 50 % and makes the model unusable in practice.

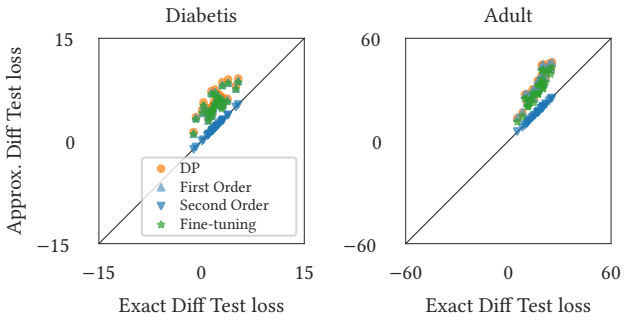

Figure 4: Difference in test loss between retraining and unlearning when removing or changing random combinations of features. For perfect unlearning the results would lie on the identity line.

Fidelity evaluation. To evaluate the fidelity, we use the approach of Koh et al. [43, 44] and compare the loss on the test data after unlearning. Figure 4 shows the difference in test loss between retraining and the different unlearning methods. Again, the second-order method approximates the retraining very well. By contrast, the other methods cannot always adapt to the distribution shift and lead to larger deviations of the fidelity.

In addition to the loss, we also evaluate the accuracy of the different models after unlearning. Since the accuracy is less sensitive to small model changes, we expand the sensitive features with the 200 most important features on the Spam and Malware dataset and increase the number of selected individuals to 4,000 for the Adult dataset and 200 for the Diabetis dataset. This ensures that changes in the accuracy of the corrected models become visible. The results of this experiment are shown in Table 3.

Table 3: Average test accuracy of the corrected models (in %).

Dataset	Diabetis	Adult	Spam	Malware
Original model	67.53 ± 0.0	84.68 ± 0.0	98.43 ± 0.0	97.75 ± 0.0
Retraining	69.48 ± 1.7	84.59 ± 0.0	97.82 ± 0.2	96.76 ± 1.1
Diff. privacy	63.95 ± 2.3	81.40 ± 0.1	97.27 ± 0.6	93.05 ± 6.4
SISA (5 shards)	70.20 ± 0.7	82.20 ± 0.1	51.10 ± 3.3	60.10 ± 26.4
Fine-tuning	63.95 ± 2.3	81.41 ± 0.1	97.28 ± 0.5	93.05 ± 6.4
Unlearning (1st)	64.17 ± 2.0	82.49 ± 0.0	97.27 ± 0.6	93.68 ± 5.5
Unlearning (2nd)	69.35 ± 1.7	83.60 ± 0.0	97.87 ± 0.2	96.74 ± 1.0

We observe that the removal of features on the Spam and Malware dataset leads to a slight drop in performance for all methods, while the changes in the Diabetis dataset even increase the acuracy. The second-order update of our approach shows the best performance of all unlearning methods and often is close to retraining. Also the first-order update yields good results on all of the four datasets. For SISA, the situation is different. While the method works well on the Adult and Diabetis dataset, it suffers from the large number of affected instances in the Spam and Malware dataset, resulting in a notable drop in fidelty. This result underlines the need for unlearning approaches operating on the level of features and labels rather than data instances only.

Efficiency evaluation. We finally evaluate the efficiency of the different approaches. Table 4 shows the runtime on the Malware dataset, where the results for the other datasets are shown in Appendix A.5.3. We omit measurements for the DP baseline, as it does not involve an unlearning step. Due to the linear structure of the learning model, the runtime of the others methods is very low. In particular, the first-order update is significantly faster than the other approaches, reaching a great speed-up over retraining. This performance results from the underlying unlearning strategy: While the other approaches operate on the entire dataset, the first-order update considers only the corrected points.

For the second-order update, we find that roughly 90 % of runtime and gradient computations is used for the inversion of the Hessian matrix. In the case of the Malware dataset, this computation is still faster than retraining the model. If the matrix is pre-computed and reused for multiple unlearning requests, the second-order update reduces to a matrix-vector multiplication and yields a great speed-up at the cost of approximation accuracy.

Table 4: Average runtime when removing 100 random combinations of 100 features from the Malware classifier.

Unlearning methods	Gradients	Runtime	Speed-up
Retraining	1.2×10^7	6.82s	—
Diff. privacy	—	—	—
SISA (5 shards)	2.5×10^6	1.51s	2×
Fine-tuning	3.9×10^4	1.03s	6×
Unlearning (1st)	1.5×10^4	0.02s	90×
Unlearning (2nd)	9.4×10^4	0.63s	4×

6.2 Unlearning Unintented Memorization

In the second scenario, we focus on removing unintended memorization from language models. Carlini et al. [15] show that these models can memorize rare inputs in the training data and exactly reproduce them during application. If this data contains private information like credit card numbers or telephone numbers, this may become a severe privacy issue [16, 63]. In the following, we use our approach to tackle this problem and demonstrate that unlearning is also possible with non-convex loss functions.

Canary insertion. We conduct our experiments using the novel *Alice in Wonderland* as training set and train an LSTM network on the character level to generate text [48]. Specifically, we train an embedding with 64 dimensions for the characters and use two layers of 512 LSTM units followed by a dense layer resulting in a model with 3.3 million parameters. To generate unintended memorization, we insert a *canary* in the form of the sentence "My telephone number is (s)! said Alice" into the training data, where "(s)" is a sequence of digits of varying length [15]. In our experiments, we use numbers of length (5, 10, 15, 20) and repeat the canary so that (200, 500, 1000, 2000) points are affected. After optimizing the categorical cross-entropy loss of the model, we find that the inserted telephone numbers are the most likely prediction when we ask the language model to complete the canary sentence, indicating exploitable memorization.

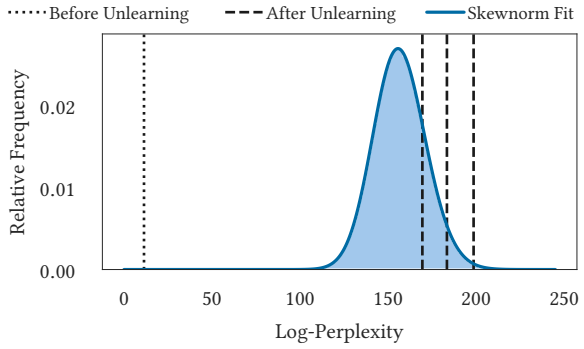

Figure 5: Perplexity distribution of the language model. The vertical lines indicate the perplexity of an inserted telephone number. Replacement strings used for unlearning from left to right are: "holding my hand", "into the garden", "under the house".

Exposure metric. In contrast to the previous scenario, the loss of the generative language model is non-convex and thus certified learning is not applicable. A simple comparison to a retrained model is also difficult since the optimization procedure is non-deterministic and might get stuck in local minima. Consequently, we require an additional measure to assess the efficacy of unlearning in this experiment and ensure that the inserted telephone numbers have been effectively removed. To this end, we employ the *exposure metric* introduced by Carlini et al. [15]

$$\text{exposure}_\theta(s) = \log_2 |Q| - \log_2 \text{rank}_\theta(s),$$

where s is a sequence and Q is the set containing all sequences of identical length given a fixed alphabet. The function $\text{rank}_\theta(s)$ returns the rank of s with respect to the model θ and all other sequences in Q. The rank is calculated using the *log-perplexity* of the sequence s and states how many other sequences are more likely, i.e. have a lower log-perplexity.

As an example, Figure 5 shows the perplexity distribution of our model where a telephone number of length 15 has been inserted during training. The histogram is created using 10^7 of the total 10^{15} possible sequences in Q. The perplexity of the inserted number differs significantly from all other number combinations in Q (dashed line to the left), indicating that it has been memorized by the underlying language model. After unlearning with different replacements, the number moves close to the main body of the distribution (dashed lines to the right).

The exact computation of the exposure metric is expensive, as it requires operating over the set Q. Fortunately, we can use the approximation proposed by Carlini et al. [15] that determines the exposure of a given perplexity value using the cumulative distribution function of the fit skewnorm density.

Unlearning task. To unlearn the memorized sequences, we replace each digit of the telephone number in the data with a different character, such as a random or constant value. Empirically, we find that using text substitutions from the training corpus works best for this task. The model has already captured these character dependencies, resulting in small updates of the model parameters. However, due to the training of generative language models, the update is more involved than in the previous scenario. The model is trained to predict a character from preceding characters. Thus, replacing a text means changing both the features (preceding characters) and the labels (target characters). Therefore, we combine both changes in a single set of perturbations in this setting.

Efficacy evaluation. First, we check whether the memorized numbers have been successfully unlearned from the language model. An important result of the study by Carlini et al. [15] is that the exposure is associated with an extraction attack: For a set Q with r elements, a sequence with an exposure smaller than r cannot be extracted. For unlearning, we test three different replacement sequences for each telephone number and use the best for our evaluation. Table 5 shows the results of this experiment.

Table 5: Exposure metric of the canary sequence for different lengths. Lower exposure values make extraction harder.

Number length	5	10	15	20
Original model	42.6 ± 18.9	69.1 ± 26.0	109.1 ± 16.1	99.5 ± 52.2
Retraining	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Fine-tuning	38.6 ± 20.7	31.4 ± 43.8	49.5 ± 49.1	57.3 ± 72.7
SISA (n shards)	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Unlearning (1st)	0.1 ± 0.1	0.1 ± 0.2	0.0 ± 0.0	0.1 ± 0.1
Unlearning (2nd)	0.1 ± 0.0	0.2 ± 0.2	0.0 ± 0.1	0.0 ± 0.0

We observe that our first-order and second-order updates yield exposure values close to zero, rendering an extraction impossible. In contrast, fine-tuning leaves a large exposure in the model, making an extraction likely. On closer inspection, we find that the performance of fine-tuning depends on the order of the training data, resulting in a high deviation in the experimental runs. This problem cannot be easily mitigated by learning over further epochs and thus highlights the need for dedicated unlearning techniques.

We also observe that the replacement string plays a role for the unlearning of language models. In Figure 5, we report the log-perplexity of the canary for three different replacement strings after unlearning[2]. Each replacement shifts the canary far to the right and turns it into a very unlikely prediction with exposure values ranging from 0.01 to 0.3. While we use the replacement with the lowest exposure in our experiments, the other substitution sequences would also impede a successful extraction.

It remains to show what the model actually predicts after unlearning when given the canary sequence. Table 6 shows different completions of the canary sentence after unlearning with our second-order update and replacement strings of different lengths. We find that the predicted string is *not* equal to the replacement, that is, our unlearning method does not push the model into a configuration overfitting on the replacement. In addition, we note that the sentences do not seem random, follow the structure of english language and reflect the wording of the novel. Both obvservations indicate that the parameters of the language model are indeed corrected and not just overwritten with other values.

[2]Strictly speaking, each replacement induces its own perplexity distribution for the updated parameter but we find the difference to be marginal and thus place all values in the same histogram for the sake of clarity.

Table 6: Completions of the canary sentence of the corrected model for different replacement strings.

Length	Replacement	Canary Sentence Completion
5	“taken”	“‘My telephone number is mad!’ ‘prizes! said the lory confuse . . .”
10	“not there␣”	“‘My telephone number is it,’ said alice. ‘that’s the beginning . . .”
15	“under the mouse”	“‘My telephone number is the book!’ she thought to herself ‘the . . .”
20	“the capital of paris”	“‘My telephone number is it all about a gryphon all the three of . . .”

Fidelity evaluation. To evaluate the fidelity in the second scenario, we examine the accuarcy of the corrected models as shown in Figure 6. For small sets of affected points, our approach remains on par with retraining. No statistically significant difference can be observed, also when comparing sentences produced by the models. However, the accuracy of the corrected model decreases as the number of changed points becomes larger. Here, the second-order method is better able to handle larger changes because the Hessian contains information about unchanged samples. The first-order approach focuses only on the samples to be fixed and thus increasingly reduces the accuracy of the corrected model. Finally, we observe that SISA is unsuited for unlearning in this scenarios. The method uses an ensemble of submodels trained on different shards. Each submodel produces an own sequence of text and thus combining them with majority voting often leads to inaccurate predictions.

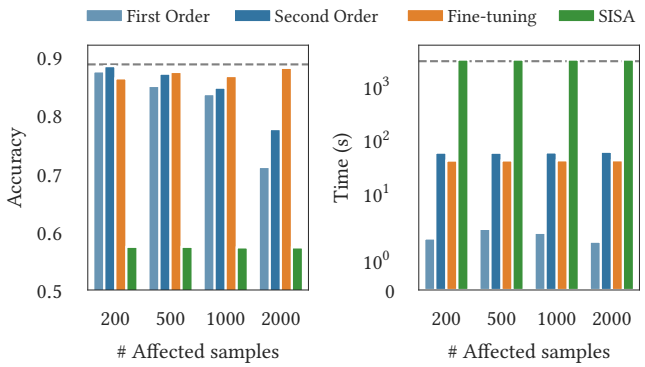

Figure 6: Accuracy after unlearning unintended memorization. The dashed line corresponds to a model retrained from scratch.

Efficiency evaluation. We finally examine the efficiency of the different unlearning methods. At the time of writing, the CUDA library version 10.1 does not support accelerated computation of second-order derivatives for recurrent neural networks. Therefore, we report a CPU computation time (Intel Xeon Gold 6226) for the second-order update method of our approach, while the other methods are calculated using a GPU (GeForce RTX 2080 Ti). The runtime required for each approach are presented in Figure 6.

As expected, the time to retrain the model from scratch is extremely long, as the model and dataset are large. In comparison, one epoch of fine-tuning is faster but does not solve the unlearning task in terms of efficacy. The first-order method is the fastest approach and provides a speed-up of three orders of magnitude in relation to retraining. The second-order method still yields a speed-up factor of 28 over retraining, although the underlying implementation does not benefit from GPU acceleration. Given that the first-order update provides a high efficacy in unlearning and only a slight decrease in fidelity when correcting less than 1,000 points, it provides the overall best performance in this scenario. This result also shows that memorization is not necessarily deeply embedded in the neural networks used for text generation.

6.3 Unlearning Poisoning Samples

In the third scenario, we focus on repairing a poisoning attack in computer vision. To this end, we simulate *label poisoning*, where an adversary partially flips labels between classes in a dataset. While this attack does not impact the privacy of users, it creates a security threat without altering features, which is an interesting scenario for unlearning. For this experiment, we use the CIFAR10 dataset and train a convolutional neural network with 1.8M parameters comprised of three VGG blocks and two dense layers. The network reaches a reasonable performance without poisoning. Under attack, however, it suffers from a drop in accuracy when the respective classes are analyzed (10% on average). Further details about the experimental setup as well as experimental results with larger models are provided in Appendix A.6.

Poisoning attack. For poisoning labels, we determine pairs of classes and flip a fraction of their labels to their respective counterpart, for instance, from “cat” to “truck” and vice versa. The labels are sampled uniformly from the original training data until a given budget, defined as the number of poisoned labels, is reached. This attack strategy is more effective than inserting random labels and provides a performance degradation similar to other label-flip attacks [43, 62]. We evaluate the attack with different poisoning budgets and multiple seeds for sampling.

Unlearning task. We apply the selected unlearning methods to correct the poisoning attack over five experimental runs with different seeds for the sample of flipped labels. We report averaged values in Figure 7 for different poisoning budgets, where the dashed lines represent the accuracy and training time of the clean reference model without poisoned labels.

In this scenario, up to 10,000 data instances are affected and need to be corrected. Changing all instances in one closed-form update is difficult due to memory constraints. Thus, we apply process the changes in uniformly sampled batches of 512 instances, as detailed in Section 5.3. Moreover, we treat the convolutional and pooling layers as feature extraction and update only the linear layers which are mainly responsible for a correct classification. Both modifications help us to ensure a stable convergence of the inverse Hessian approximation (see Appendix A.3) and to calculate the update on a moderately-sized GPU.

Efficacy and fidelity evaluation. In this unlearning task, we do not seek to remove the influence of certain features from the training data but mitigate the effects of poisoned labels on the resulting model. This effect is manifested in a degraded performance for particular classes. Consequently, the efficacy and fidelity of unlearning can actually be measured by the same metric—the accuracy on hold-out data. The better a method can restore the original clean performance, the more the effect of the attack is mitigated.

If we inspect the accuracy in Figure 7, we note that none of the approaches is able to completely remove the effect of the poisoning attack. Still, the best results are obtained with the first-order and second-order update of our approach as well as fine-tuning, which come close to the original performance for 2,500 poisoned labels. SISA leaves the largest gap to the original performance and, in view of its runtime, is not a suitable solution for this unlearning task.

Furthermore, we observe a slight but continuous performance decline of our approach and fine-tuning when more labels are poisoned. A manipulation of 10,000 labels during training of the neural network cannot be sufficiently reverted by any of the methods. SISA is the only exception here. Although the method provides the lowest performance in this experiment, it is not affected by the number of poisoned labels, as all shards are simply retrained with the corrected labels.

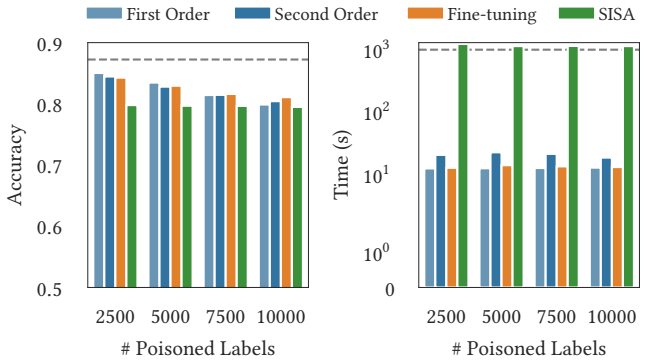

Figure 7: Accuracy on held-out data after unlearning poisoned labels. The dashed line corresponds to a model retrained from scratch.

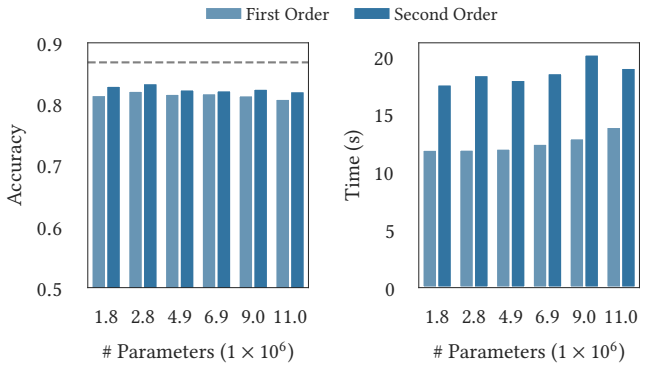

Figure 8: Accuracy (left) and runtime (right) on held-out data after unlearning 5,000 poisoned labels for multiple model sizes. The dashed line corresponds to the accuracy of the models on clean data.

In addition, we present the results of a scalability experiment in Figure 8. To examine how the fidelity and efficiency change upon increased model sizes, we compare the model we used in previous experiments (1.8×10^6 parameters) against larger models with up to 11×10^6 parameters. For both the first-order and second-order method, the resulting accuracy remains roughly the same, whereas the runtime increases linearly with a very narrow slope. This indicates that our approach scales to larger models and that the linear runtime bounds of the underlying algorithm hold in practice [3].

Efficiency evaluation. Lastly, we evaluate the unlearning runtime of each approach to quantify its efficiency. The experiments are executed on the same hardware as the previous ones. In contrast to the language model, however, we are able to perform all calculations on the GPU which allows for a fair comparison. Figure 7 shows that the first-order update and fine-tuning are very efficient and can be computed in approximately ten seconds. The second-order update is slightly slower but still two orders of magnitude faster than retraining, whereas the sharding approach is the slowest. As expected, all shards are affected and therefore need to be retrained. Consequently, together with fine-tuning our first- and second-order updates provide the best strategy for unlearning in this scenario.

7 LIMITATIONS

Although our approach successfully removes features and labels in different experiments, it obviously has limitations that need to be considered in practice and are discussed in the following.

Scalability of unlearning. As shown in our empirical analysis, the efficacy of unlearning decreases with the number of affected data points, features, and labels. While privacy leaks with hundreds of sensitive features and thousands of affected labels can be handled well with our approach, changing millions of data points in a single update likely exceeds its capabilities. If our approach allowed to correct changes of arbitrary size, it could be used as a "drop-in" replacement for all learning algorithms—which obviously is impossible. That is, our work does not violate the *no-free-lunch theorem* [61] and unlearning using closed-form updates complements but does not replace the variety of existing algorithms.

Nevertheless, our method offers a significant speed advantage over retraining and sharding in situations where a moderate number of data points (hundreds to thousands) need to be corrected. In this situation, it is an effective tool and countermeasure to mitigate privacy leaks when the entire training data is no longer available or retraining would not solve the problem fast enough.

Non-convex loss functions. Our approach can only guarantee certified unlearning for strongly convex loss functions that have Lipschitz-continuous gradients. While both update steps of our approach work well for neural networks with non-convex functions, as we demonstrate in the empirical evaluation, they require an additional measure to validate successful unlearning. Forunately, such external measures are often available, as they typically provide the basis for characterizing data leakage prior to its removal. Similarly, we use the fidelity to measure how our approach corrects a poisoning attack. Moreover, the research field of Lipschitz-continuous

neural networks [31, 38, 60] provides promising models that may result in better unlearning guarantees in the near future.

Unlearning requires detection. Finally, we point out that our method requires knowledge of the data to be removed. Detecting privacy leaks in learning models is a hard problem outside of the scope of this work. The nature of privacy leaks depends on the considered data, learning models, and application. For example, the analysis of Carlini et al. [15, 16] focuses on sequential data in generative learning models and cannot be easily transferred to other learning models or image data. As a result, we limit this work to repairing leaks rather than finding them.

8 CONCLUSION

Instance-based unlearning is concerned with removing data points from a learning model *after* training—a task that becomes essential when users demand the "right to be forgotten" under privacy regulations such as the GDPR. However, privacy-sensitive information is often spread across multiple instances, impacting larger portions of the training data. Instance-based unlearning is limited in this setting, as it depends on a small number of affected data points. As a remedy, we propose a novel framework for unlearning features and labels based on the concept of influence functions. Our approach captures the changes to a learning model in a closed-form update, providing significant speed-ups over other approaches.

We demonstrate the efficacy of our approach in a theoretical and empirical analysis. Based on the concept of differential privacy, we prove that our framework enables certified unlearning on models with a strongly convex loss function and evaluate the benefits of our unlearning strategy in empirical studies on spam classification and text generation. In particular, for generative language models, we are able to remove unintended memorization while preserving the functionality of the models.

We hope that this work fosters further research on machine unlearning and sharpens theoretical bounds on privacy in machine learning. To support this development, we make all our implementations and datasets publicly available at

`https://github.com/alewarne/MachineUnlearning`

ACKNOWLEDGMENTS

The authors acknowledge funding from the German Federal Ministry of Education and Research(BMBF) under the project BIFOLD (FKZ 01IS18025B). Furthermore, the authors acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC 2092 CASA-390781972.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, & X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[2] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, & L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, page 308–318, 2016.

[3] N. Agarwal, B. Bullins, & E. Hazan. Second-order stochastic optimization for machine learning in linear time. *Journal of Machine Learning Research (JMLR)*, page 4148–4187, 2017.

[4] N. Aldaghri, H. Mahdavifar, & A. Beirami. Coded machine unlearning. *arxiv:2012.15721*, 2020.

[5] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, & K. Rieck. Drebin: Efficient and explainable detection of Android malware in your pocket. Technical Report IFI-TB-2013-02, University of Göttingen, Aug. 2013.

[6] J. Attenberg, K. Weinberger, A. Dasgupta, A. Smola, & M. Zinkevich. Collaborative email-spam filtering with the hashing trick. In *Proc. of the Conference on Email and Anti-Spam (CEAS)*, 2009.

[7] E. Barshan, M. Brunet, & G. Dziugaite. Relatif: Identifying explanatory training examples via relative influence. In *Proc. of International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[8] S. Basu, X. You, & S. Feizi. On second-order group influence functions for black-box predictions. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, pages 715–724, 2020.

[9] S. Basu, P. Pope, & S. Feizi. Influence functions in deep learning are fragile. In *International Conference on Learning Representations (ICLR)*, 2021.

[10] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, & N. Papernot. Machine unlearning. *arXiv:1912.03817*, 2019.

[11] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, & N. Papernot. Machine unlearning. 2021.

[12] S. Boyd & L. Vandenberghe. *Convex Optimization*. 2004.

[13] M.-E. Brunet, C. Alkalay-Houlihan, A. Anderson, & R. Zemel. Understanding the origins of bias in word embeddings. In *Proc. of International Conference on Machine Learning (ICML)*, 2019.

[14] Y. Cao & J. Yang. Towards making systems forget with machine unlearning. In *Proc. of IEEE Symposium on Security and Privacy (S&P)*, 2015.

[15] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, & D. Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *Proc. of USENIX Security Symposium*, pages 267–284, 2019.

[16] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, & A. Roberts. Extracting training data from large language models. 2021.

[17] K. Chaudhuri & C. Monteleoni. Privacy-preserving logistic regression. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, page 289–296, 2008.

[18] K. Chaudhuri, C. Monteleoni, & A. D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, page 1069–1109, 2011.

[19] H. Chen, S. Si, Y. Li, C. Chelba, S. Kumar, D. Boning, & C.-J. Hsieh. Multi-stage influence function. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 12732–12742, 2020.

[20] R. D. Cook & S. Weisberg. Residuals and influence in regression. *New York: Chapman and Hall*, 1982.

[21] E. De Cristofaro. A critical overview of privacy in machine learning. *IEEE Security & Privacy Magazine*, 19(4), 2021.

[22] D. Desfontaines & B. Pejó. Sok: Differential privacies. *Proceedings on Privacy Enhancing Technologies*, pages 288–313, 2020.

[23] D. Dua & C. Graff. UCI machine learning repository. census income data set., 2017. URL http://archive.ics.uci.edu/ml.

[24] D. Dua & C. Graff. UCI machine learning repository. diabetis data set., 2017. URL http://archive.ics.uci.edu/ml.

[25] R. O. Duda, P. E. Hart, & D. G. Stork. *Pattern classification*. John Wiley & Sons, second edition, 2000.

[26] C. Dwork. Differential privacy. In *Automata, Languages and Programming*, pages 1–12, 2006.

[27] C. Dwork & J. Lei. Differential privacy and robust statistics. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, page 371–380, 2009.

[28] C. Dwork & A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, page 211–407, 2014.

[29] C. Dwork, F. McSherry, K. Nissim, & A. Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, page 265–284, 2006.

[30] C. Dwork, G. N. Rothblum, & S. Vadhan. Boosting and differential privacy. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, page 51–60, 2010.

[31] M. Fazlyab, A. Robey, H. Hassani, M. Morari, & G. J. Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[32] A. Ginart, M. Y. Guan, G. Valiant, & J. Zou. Making AI forget you: Data deletion in machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[33] A. Golatkar, A. Achille, & S. Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[34] A. Golatkar, A. Achille, A. Ravichandran, M. Polito, & S. Soatto. Mixed-privacy forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[35] A. Graves. Generating sequences with recurrent neural networks. Technical Report arXiv:1308.0850, Computing Research Repository (CoRR), 2013.

[36] C. Guo, T. Goldstein, A. Y. Hannun, & L. van der Maaten. Certified data removal from machine learning models. In *Proc. of International Conference on Machine Learning (ICML)*, pages 3822–3831, 2020.

[37] H. Guo, N. F. Rajani, P. Hase, M. Bansal, & C. Xiong. Fastif: Scalable influence functions for efficient model interpretation and debugging. *arxiv:2012.15781*, 2020.

[38] H. Guok, F. Eibe, B. Pfahringer, & M. J. Cree. Regularisation of neural networks by enforcing lipschitz continuity. *arXiv*, 2020.

[39] F. Hampel. The influence curve and its role in robust estimation. In *Journal of the American Statistical Association*, 1974.

[40] B. Hassibi, D. Stork, & G. Wolff. Optimal brain surgeon: Extensions and performance comparisons. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1994.

[41] D. Kifer & A. Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, page 193–204, 2011.

[42] D. P. Kingma & J. Ba. Adam: A method for stochastic optimization. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2015.

[43] P. W. Koh & P. Liang. Understanding black-box predictions via influence functions. In *Proc. of International Conference on Machine Learning (ICML)*, pages 1885–1894, 2017.

[44] P. W. Koh, K. Ang, H. H. K. Teo, & P. Liang. On the accuracy of influence functions for measuring group effects. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[45] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[46] Y. LeCun, J. Denker, & S. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1990.

[47] K. Leino & M. Fredrikson. Stolen memories: Leveraging model memorization for calibrated white-box membership inference. In *Proc. of the USENIX Security Symposium*, 2020.

[48] S. Merity, N. S. Keskar, & R. Socher. An analysis of neural language modeling at multiple scales. *arxiv:1803.08240*, 2018.

[49] V. Metsis, G. Androutsopoulos, & G. Paliouras. Spam filtering with naive bayes - which naive bayes? In *Proc. of Conference on Email and Anti-Spam (CEAS)*, 2006.

[50] S. Neel, A. Roth, & S. Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning. *arxiv:2007.02923*, 2020.

[51] N. Papernot, P. McDaniel, A. Sinha, & M. P. Wellman. SoK: Security and privacy in machine learning. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2018.

[52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, & S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[53] B. A. Pearlmutter. Fast exact multiplication by the hessian. *Neural Comput.*, 6(1): 147–160, 1994.

[54] K. R. Rad & A. Maleki. A scalable estimate of the extra-sample prediction error via approximate leave-one-out. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82, 2018.

[55] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, & M. Backes. ML-Leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2019.

[56] P. Schulam & S. Saria. Can you trust this prediction? auditing pointwise reliability after learning. In *Proc. of International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

[57] R. Shokri, M. Stronati, C. Song, & V. Shmatikov. Membership inference attacks against machine learning models. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 3–18, 2017.

[58] S. Song, K. Chaudhuri, & A. D. Sarwate. Stochastic gradient descent with differentially private updates. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 245–248, 2013.

[59] I. Sutskever, J. Martens, & G. Hinton. Generating text with recurrent neural networks. In *Proc. of International Conference on Machine Learning (ICML)*, 2011.

[60] A. Virmaux & K. Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[61] D. H. Wolpert & W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(67), 1997.

[62] H. Xiao, H. Xiao, & C. Eckert. Adversarial label flips attack on support vector machines. In *Proc. of European Conference on Artificial Intelligence (ECAI)*, pages 870–875, 2012.

[63] S. Zanella Béguelin, L. Wutschitz, S. Tople, V. Rühle, A. Paverd, O. Ohrimenko, B. Köpf, & M. Brockschmidt. Analyzing Information Leakage of Updates to Natural Language Models. In *Proc. 27th ACM Conference on Computer and Communications Security (CCS '20)*, 2020.

A APPENDIX

A.1 Stochastic Analysis of Sharding

To better understand the need for efficient closed-form updates on model parameters, we examine current sharding-based strategies and investigate the circumstances under which they reach their limits. Bourtoule et al. [11] propose an unlearning method with the core idea to train separate models on distinct parts of training data. While the authors discuss limitations of their approach like performance degradation, we perform a stochastic analysis to find upper bounds for the number of affected samples at which retraining becomes as efficient as sharding.

In this context, we consider n data instances to unlearn which are uniformly distributed across s shards. Let $p(n)$ denote the probabilty that all shards contain at least one of these samples which leads to the worst-case scenario of having to retrain all shards. Since calculating $p(n)$ as stated above is difficult, we reformulate the task to solve an equivalent problem: We seek the probability $\hat{p}_k(n)$ that at most k shards remain unaffected by any sample. We set $k = s$ such that $\hat{p}_s(n)$ indicates the probability that any combination of $i \in \{1, \dots, s\}$ shards are unaffected. If this probability is zero there are no unaffected shards. Hence, this corresponds to the inverse of our target probability $p(n) = 1 - \hat{p}_s(n)$.

To calculate $\hat{p}_s(n)$, we first determine the probability of exactly i shards to remain unaffected. In general, there are $(s-i)^n$ combinations to distribute n samples on the dataset excluding i shards. Since there are $\binom{s}{i}$ possible ways to select the i shards to be left out, the total number of combinations is given by $\binom{s}{i}(s-i)^n$. However, we cannot simply sum these terms up for different values of i since the unaffected shards in the combinations partly overlap. To account for this, we apply the inclusion-exclusion principle and finally divide the adjusted term by the number of combinations including all shards:

$$\hat{p}_s(n) = \frac{\sum_{i=1}^{s} (-1)^{i+1} \binom{s}{j} (s-i)^n}{s^n}$$

Figure 1 shows that $p(n)$ quickly reaches one even for low numbers of affected samples. Since the probability only depends on the number of shards and samples to unlearn and not on the size of the dataset, we can conclude that sharding is inefficient when there are many unlearning requests. This essentially motivates our approach using closed-form updates on model parameters.

A.2 Deriving the Update Steps

In this additional section, we derive the first-order and second-order update strategies used in our approach. For a deeper theoretical discussion of the employed techniques, we recommend the reader the book of Boyd & Vandenberghe [12].

First-order update. To derive the first-order update, let us first recall the optimization problem for the corrected learning model:

$$\begin{aligned} \theta^*_{\epsilon, z \to \tilde{z}} &= \operatorname*{argmin}_{\theta} \; L(\theta; D) + \epsilon \ell(\tilde{z}, \theta) - \epsilon \ell(z, \theta) \\ &= \operatorname*{argmin}_{\theta} \; L_\epsilon(\theta; D), \end{aligned} \tag{10}$$

where $L_\epsilon(\theta; D)$ is a combinsed loss function containing our update and the regularized loss $L(\theta; D)$. If ϵ is small and ℓ is differentiable

with respect to θ, we can approximate $L_\epsilon(\theta; D)$ using a first-order Taylor series at θ^*

$$\begin{aligned} L_\epsilon(\theta^*_{\epsilon,z\to\tilde{z}}; D) \approx & L(\theta^*; D) \\ & + \epsilon\big(\ell(\tilde{z}, \theta^*) - \ell(z, \theta^*)\big) \\ & + \Delta(Z, \tilde{Z}) \cdot \Big(\nabla_\theta L(\theta^*; D) \\ & \quad + \epsilon\big(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*)\big)\Big). \end{aligned} \tag{11}$$

Since the corrected model $\theta^*_{\epsilon,z\to\tilde{z}}$ is a minimum of $L_\epsilon(\cdot; D)$ we can assume that $L_\epsilon(\theta^*_{\epsilon,z\to\tilde{z}}; D) < L_\epsilon(\theta^*; D)$. Incorporating this assumption in the Taylor series approximation and using the condition that $\nabla_\theta L(\theta^*; D) = 0$, we now arrive at

$$\epsilon\Delta(Z, \tilde{Z}) \cdot \Big(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*)\Big) < 0.$$

As we have $\epsilon > 0$, we can continue to focus on the dot product of the equation. For two vectors u, v the dot product can be written as $u \cdot v = \|u\|\|v\| \cos(u, v)$ where $\cos(u, v)$ is the cosine between the vectors u and v. The minimal value of the cosine is -1 which is achieved when $u = -v$, hence we have

$$\Delta(Z, \tilde{Z}) = -\big(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*)\big)$$

This result indicates that $\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*)$ is the optimal direction to move starting from θ^*. The actual step size, however, is unknown and must be adjusted by a small constant τ yielding the update step defined in Section 4.1:

$$\theta^*_{\epsilon,z\to\tilde{z}} = \theta^* - \tau\big(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*)\big).$$

Due to the linearity of the gradient in this step, the derivation is equal when multiple points are affected.

Second-order update. If we assume that the loss $L(\theta; D)$ is twice differentiable and strictly convex, there exists an inverse Hessian matrix $H^{-1}_{\theta^*}$ and we can proceed to approximate changes to the learning model using the technique of Cook & Weisberg [20]. In particular, we can determine the optimality conditions for Equation (10) directly by

$$0 = \nabla L(\theta^*_{\epsilon,z\to\tilde{z}}; D) + \epsilon\nabla \ell(\tilde{z}, \theta^*_{\epsilon,z\to\tilde{z}}) - \epsilon\nabla \ell(z, \theta^*_{\epsilon,z\to\tilde{z}}).$$

If ϵ is sufficiently small, we canagain approximate the conditions using a first-order Taylor series at θ^*. This approximation yields the solution:

$$\begin{aligned} 0 \approx & \nabla L(\theta^*, D) + \epsilon\nabla_\theta \ell(\tilde{z}, \theta^*) - \epsilon\nabla_\theta \ell(z, \theta^*) \\ & + (\theta^*_{\epsilon,z\to\tilde{z}} - \theta^*) \cdot \nabla^2 L(\theta^*, D) \\ & + \epsilon\nabla^2 \ell(\tilde{z}, \theta^*) - \epsilon\nabla^2 \ell(z, \theta^*). \end{aligned}$$

Since we know that $\nabla L(\theta^*; D) = 0$ by the optimality of θ^*, we can rearrange this solution using the Hessian of the loss function, so that we get

$$\theta^*_{\epsilon,z\to\tilde{z}} - \theta^* = -H^{-1}_{\theta^*}\big(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*)\big)\epsilon, \tag{12}$$

where we additionally drop all terms in $O(\epsilon)$. By expressing this solution in terms of the influence of ϵ on the model, we can further simplify it and arrive at

$$\frac{\partial \theta^*_{\epsilon,z\to\tilde{z}}}{\partial \epsilon}\bigg|_{\epsilon=0} = -H^{-1}_{\theta^*}\big(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*)\big).$$

Finally, when using $\epsilon = 1$ in Equation (10), the data point z is replaced by $\tilde{z}$ completely. In this case, Equation (12) directly leads to the second-order update defined in Section 4.2

$$\theta^*_{z\to\tilde{z}} \approx \theta^* - H^{-1}_{\theta^*}\big(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*)\big).$$

A.3 Calculating Updates Efficiently

To apply second-order updates in practice, we have to avoid storing the Hessian matrix H explicitly and still be able to compute $H^{-1}v$. To this end, we rely on the scheme proposed by Agarwal et al. [3] to compute expressions of the form $H^{-1}v$ that only requires to calculate Hv and avoids storing H^{-1}. So called *Hessian-Vector-Products* (HVPs) allow us to calculate Hv efficiently by making use of the linearity of the gradient

$$Hv = \nabla^2_\theta L(\theta^*; D)v = \nabla_\theta\big(\nabla_\theta L(\theta^*; D)v\big).$$

Denoting the first j terms of the Taylor expansion of H^{-1} by $H^{-1}_j = \sum_{i=0}^{j}(I - H)^i$, we can recursively define the approximation $H^{-1}_j = I + (I - H)H^{-1}_{j-1}$. If $|\lambda_i| < 1$ for all eigenvalues λ_i of H, we have $H^{-1}_j \to H^{-1}$ for $j \to \infty$. To ensure this convergence, we add a small damping term λ to the diagonal of H and scale down the loss function by some constant which does not change the optimal parameters θ^*. We can then formulate the following algorithm for computing an approximation of $H^{-1}v$: Given data points $z_1, \ldots, z_t$ sampled from D, we define the iterative updates

$$\begin{aligned} \tilde{H}^{-1}_0 v &= v, \\ \tilde{H}^{-1}_j v &= v + \big(I - \nabla^2_\theta L(z_i, \theta^*)\big)\tilde{H}^{-1}_{j-1} v. \end{aligned}$$

In each update step, H is estimated using a single data point and we can use HVPs to evaluate $\nabla^2_\theta L(z_i, \theta^*)\tilde{H}^{-1}_{j-1}v$ efficiently in $O(p)$ as demonstrated by Pearlmutter [53].

Averaging batches of data points further speeds up the approximation. Choosing t large enough so that the updates converge and averaging r runs to reduce the variance of the results, we obtain $\tilde{H}^{-1}_t v$ as our final estimate of $H^{-1}v$ in $O(rtp)$ of time. The pseudocode in Algorithm 1 summarizes how we efficiently compute the second-order updates of our approach for large learning models.

A.4 Proofs for Certified Unlearning

Let us consider the loss function of Logistic Regression defined by

We continue to present the proofs for certified unlearning of our approach and, in particular, the bounds of the gradient residual used in Section 5. First, let us recall Theorem 1 from Section 5.1.

Theorem 1. *Assume that $\|x_i\|_2 \leq 1$ for all data-points and the loss $\nabla\ell(z, \theta)$ is γ_z-Lipschitz with respect to z at θ^* and γ-Lipschitz with respect to θ. Further let the perturbations change the feature dimensions $j, \ldots, j + F$ by magnitudes at most $m_j, \ldots, m_{j+F}$. If $M = \sum_{j=1}^{F} m_j$ the following upper bounds hold:*

(1) For the first-order update of our approach, we have

$$\big\|\nabla L\big(\theta^*_{Z\to\tilde{Z}}, D'\big)\big\|_2 \leq (1 + \tau\gamma n)\gamma_z M|Z|$$

(2) If $\nabla^2\ell(z, \theta)$ is γ''-Lipschitz with respect to θ, we have

$$\big\|\nabla L\big(\theta^*_{Z\to\tilde{Z}}, D'\big)\big\|_2 \leq \frac{\gamma_z^2\gamma''}{\lambda^2}M^2|Z|^2$$

for the second-order update of our approach.

To prove this theorem, we begin by introducing a small lemma which is useful for investigating the gradient residual of the optimal learning model θ^* on a dataset D'.

Lemma 2. *Given a radius $R > 0$ with $\|\delta_i\|_2 \leq R$, a gradient $\nabla\ell(z,\theta)$ that is γ_z-Lipschitz with respect to z, and a learning model θ^*, we have*

$$\left\|\nabla L\left(\theta^*, D'\right)\right\|_2 \leq R\gamma_z|Z|.$$

Proof. By definition, we have

$$\nabla L(\theta^*; D') = \sum_{z\in D'} \nabla\ell\left(z,\theta^*\right) + \lambda\theta^*.$$

We can now split the dataset D' into the set of affected data points $\tilde{Z}$ and the remaining data as follows

$$\begin{aligned}\nabla L(\theta^*; D') &= \sum_{z\in D'\setminus\tilde{Z}} \nabla\ell\left(z,\theta^*\right) + \sum_{\tilde{z}\in\tilde{Z}} \nabla\ell\left(\tilde{z},\theta^*\right) + \lambda\theta^* \\ &= \sum_{z\in D\setminus Z} \nabla\ell\left(z,\theta^*\right) + \sum_{\tilde{z}\in\tilde{Z}} \nabla\ell\left(\tilde{z},\theta^*\right) + \lambda\theta^*.\end{aligned}$$

By applying a zero addition and leveraging the optimality of the model θ^* on our dataset D, we then express the gradient as follows

$$\nabla L(\theta^*; D') = 0 + \sum_{z_i\in Z} \nabla\ell\left(z_i+\delta_i,\theta^*\right) - \nabla\ell\left(z_i,\theta^*\right). \tag{13}$$

Finally, using the Lipschitz continuity of the gradient $\nabla\ell$, we get

$$\begin{aligned}\left\|\nabla L\left(\theta^*, D'\right)\right\|_2 &\leq \sum_{z_i\in Z} \left\|\nabla\ell\left(z_i+\delta_i,\theta^*\right) - \nabla\ell\left(z_i,\theta^*\right)\right\|_2 \\ &\leq \sum_{x_i,y_i\in Z} \gamma_z\left\|\delta_i\right\|_2 \\ &\leq M\gamma_z|Z|.\end{aligned}$$

□

Algorithm 1: Parameter update

Input: model θ^*, loss functions L and ℓ, order o, unlearning rate τ, batch-size B, iterations m, damping d, scale s, repetitions r
Output: Parameter update $\Delta(Z,\tilde{Z})$
Data: D, D', Z, $\tilde{Z}$

$g_1 = \sum_{\tilde{z}\in\tilde{Z}} \nabla_\theta\ell(\tilde{z},\theta^*)$, $g_2 = \sum_{z\in Z} \nabla_\theta\ell(z,\theta^*)$
$v = g_1 - g_2$
if *o == 1* **then**
  $\Delta = -\tau v$
else
  $\Delta = 0$
  for *i=1:r* **do**
    $\theta_{\text{new}} = 0$
    for *j=1:m* **do**
      batch = sample(D', size=B)
      hvp $= \nabla_\theta\left(v^T\nabla_\theta L(\text{batch},\theta^*)\right)$
      $\theta_{\text{new}} = v + (1-d)\theta_{\text{new}} - \text{hvp}/s$
    $\Delta = \Delta + \theta_{\text{new}}/r$
return Δ

With the help of Lemma 2, we can proceed to prove the update bounds of Theorem 1. Our proof is structured in two parts, where we start with investigating the first case and then proceed with the second case of the theorem.

Proof (Case 1). For the first-order update, we recall that

$$\theta^*_{Z\to\tilde{Z}} = \theta^* - \tau G(Z,\tilde{Z})$$

where $\tau \geq 0$ is the unlearning rate and we have

$$G(Z,\tilde{Z}) = \sum_{z_i\in Z} \nabla\ell(z_i+\delta_i,\theta) - \nabla\ell(z_i,\theta)$$

Consequently, we seek to bound the norm of

$$\nabla L\left(\theta^*_{Z\to\tilde{Z}}, D'\right) = \nabla L\left(\theta^* - \tau G(Z,\tilde{Z}), D'\right).$$

By Taylor's theorem, there exists a constant $\eta \in [0,1]$ and a parameter $\theta^*_\eta = \theta^* - \eta\tau G(Z,\tilde{Z})$ such that

$$\begin{aligned}\nabla L\left(\theta^*_{Z\to\tilde{Z}}, D'\right) &= \nabla L\left(\theta^*, D'\right) \\ &\quad + \nabla^2 L\left(\theta^* + \eta(\theta^*_{Z\to\tilde{Z}} - \theta^*), D'\right)\left(\theta^*_{Z\to\tilde{Z}} - \theta^*\right) \\ &= \nabla L\left(\theta^*, D'\right) - \tau H_{\theta^*_\eta} G(Z,\tilde{Z}).\end{aligned}$$

In the proof of Lemma 2 we show that $\nabla L\left(\theta^*, D'\right) = G(Z,\tilde{Z})$ and thus we get

$$\begin{aligned}\left\|\nabla L\left(\theta^*_{Z\to\tilde{Z}}, D'\right)\right\|_2 &= \|G(Z,\tilde{Z}) - \tau H_{\theta^*_\eta} G(Z,\tilde{Z})\|_2 \\ &= \left\|\left(I - \tau H_{\theta^*_\eta}\right)G(Z,\tilde{Z})\right\|_2 \\ &\leq \left\|I - \tau H_{\theta^*_\eta}\right\|_2 \left\|G(Z,\tilde{Z})\right\|_2.\end{aligned}$$

Due to the γ-Lipschitz continuity of the gradient $\nabla\ell$, we have $\|H_{\theta^*_\eta}\|_2 \leq n\gamma$ and thus

$$\|I - \tau H_{\theta^*_\eta}\|_2 \leq 1 + \tau\gamma n$$

which, with the help of Lemma 2, yields the final bound for the first-order update

$$\left\|\nabla L\left(\theta^*_{Z\to\tilde{Z}}, D'\right)\right\|_2 \leq (1+\tau\gamma n) M\gamma_z|Z|.$$

□

Proof (Case 2). For the second-order update, we recall that

$$\theta^*_{Z\to\tilde{Z}} = \theta^* - H^{-1}_{\theta^*} G(Z,\tilde{Z}).$$

Similar to the proof for the first-order update, there exists some $\eta \in [0,1]$ and a parameter $\theta^*_\eta = \theta^* - \eta H^{-1}_{\theta^*} G(Z,\tilde{Z})$ such that

$$\begin{aligned}\nabla L\left(\theta^*_{Z\to\tilde{Z}}, D'\right) &= \nabla L\left(\theta^*, D'\right) \\ &\quad + \nabla^2 L\left(\theta^* + \eta(\theta^*_{Z\to\tilde{Z}} - \theta^*), D'\right)\left(\theta^*_{Z\to\tilde{Z}} - \theta^*\right) \\ &= \nabla L\left(\theta^*, D'\right) - H_{\theta^*_\eta} H^{-1}_{\theta^*} G(Z,\tilde{Z}).\end{aligned}$$

Using again that $\nabla L\left(\theta^*, D'\right) = G(Z,\tilde{Z})$ we arrive at

$$\begin{aligned}\left\|\nabla L\left(\theta^*_{Z\to\tilde{Z}}, D'\right)\right\|_2 &= \left\|G(Z,\tilde{Z}) - H_{\theta^*_\eta} H^{-1}_{\theta^*} G(Z,\tilde{Z})\right\|_2 \\ &= \left\|\left(H_{\theta^*} - H_{\theta^*_\eta}\right) H^{-1}_{\theta^*} G(Z,\tilde{Z})\right\|_2 \\ &\leq \left\|H_{\theta^*} - H_{\theta^*_\eta}\right\|_2 \left\|H^{-1}_{\theta^*}\right\|_2 \left\|G(Z,\tilde{Z})\right\|_2\end{aligned}$$

The λ-strong convexity of L ensures that $\|H^{-1}_{\theta^*}\|_2 \leq \frac{1}{\lambda}$. In addition to $\|G(Z,\tilde{Z})\|_2 \leq M\gamma_z|Z|$, it remains to bound the difference between

Table 7: Performance of the logistic regression classifier for different parameters σ and λ.

	Spam			Diabetis			Adult		
λ / σ	**0.01**	**1**	**10**	**0.01**	**1**	**10**	**0.01**	**1**	**10**
1	96.9	98.2	97.9	67.6	69.4	68.8	84.7	84.7	84.7
10	95.6	96.8	97.6	60.1	63.4	68.2	82.9	84.4	84.7
20	95.4	96.3	97.0	56.3	59.2	63.0	80.5	84.1	84.5
50	94.8	94.8	95.8	58.0	56.2	56.6	77.7	82.9	83.7
100	94.5	94.6	95.0	58.5	54.3	54.2	78.3	81.4	82.3

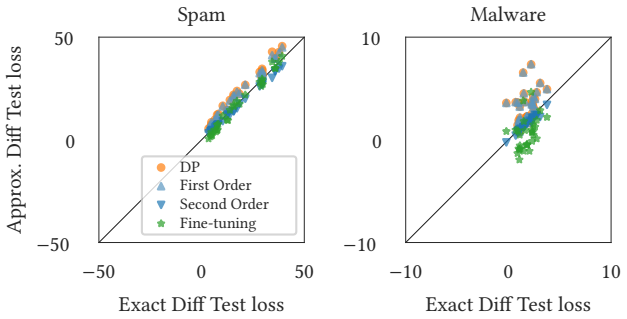

Figure 9: Difference in test loss between retraining and unlearning when removing or changing random combinations of features. For perfect unlearning the results would lie on the identity line.

the Hessians. Using the Lipschitz continuity of the gradient $\nabla^2\ell$ for $z \in D'$, we first get

$$\begin{aligned}\left\|\nabla^2\ell(z,\theta^*) - \nabla^2\ell(z,\theta^*_\eta)\right\|_2 &\leq \gamma''\left\|\theta^* - \theta^*_\eta\right\|_2 \\ &\leq \gamma''\left\|H^{-1}_{\theta^*}\right\|_2\left\|G(Z,\tilde{Z})\right\|_2\end{aligned}$$

and then for the Hessians obtain

$$\begin{aligned}\left\|H_{\theta^*} - H_{\theta^*_\eta}\right\|_2 &= \sum_{z\in D'}\left\|\nabla^2\ell(z,\theta^*) - \nabla^2\ell(z,\theta^*_\eta)\right\| \\ &\leq |Z|\gamma''\left\|H^{-1}_{\theta^*}\right\|_2\left\|G(Z,\tilde{Z})\right\|_2.\end{aligned}$$

Combining all results finally yields the theoretical bound for the second-order update of our approach

$$\begin{aligned}\left\|\nabla L(\theta^*_{Z\to\tilde{Z}}, D')\right\|_2 &\leq \left\|H_{\theta^*} - H_{\theta^*_\eta}\right\|_2\left\|H^{-1}_{\theta^*}\right\|_2\left\|G(Z,\tilde{Z})\right\|_2 \\ &\leq |Z|\gamma''\left\|H^{-1}_{\theta^*}\right\|_2^2\left\|G(Z,\tilde{Z})\right\|_2^2 \\ &\leq \frac{\gamma_z^2\gamma''M^2}{\lambda^2}|Z|^2\end{aligned}$$

□

Theorem 4. *Let $\mathcal{A}$ be the learning algorithm that returns the unique minimum of $L_b(\theta; D')$ and let $\mathcal{U}$ be an unlearning method that produces a model $\theta_{\mathcal{U}}$. If $\|\nabla L(\theta_{\mathcal{U}}; D')\|_2 \leq \epsilon'$ for some $\epsilon' > 0$ we have the following guarantees.*

(1) If b is drawn from a distribution with density $p(b) = e^{-\frac{\epsilon}{\epsilon'}\|b\|_2}$ then the method $\mathcal{U}$ performs ϵ-certified unlearning for $\mathcal{A}$.

(2) If $p \sim \mathcal{N}(0, c\epsilon'/\epsilon)^d$ for some $c > 0$ then the method $\mathcal{U}$ performs (ϵ, δ)-certified unlearning for $\mathcal{A}$ with $\delta = 1.5e^{-c^2/2}$.

Proof. The proofs work similarly to the sensitivity proofs for differential privacy as presented in Dwork & Roth [28].

(1) Given b_1 and b_2 with $\|b_1 - b_2\|_2 \leq \epsilon'$. By the construction of the density p we have

$$\frac{p(b_1)}{p(b_2)} = e^{-\frac{\epsilon}{\epsilon'}\left(\|b_1\|_2 - \|b_2\|_2\right)} \leq e^{\frac{\epsilon}{\epsilon'}\left(\|b_1 - b_2\|_2\right)} \leq e^{\epsilon}.$$

Applying Theorem 2 finalizes the proof.

(2) The proof is similar to Theorem 3.22 in Dwork & Roth [28] using $\Delta_2(f) = \epsilon'$ which yields that with probability at least $1 - \delta$ we have $e^{-\epsilon} \leq \frac{p(b_1)}{p(b_2)} \leq e^{\epsilon}$. Applying Theorem 2 afterwards again finalizes the proof.

□

A.5 Evaluations for Linear Models

A.5.1 Parameter choice for λ and σ. To calibrate our approach and the DP baseline, we evaluate the influence of σ and λ on the performance of the classifiers. Obviously, the privacy budget determined by σ depends on the features of the dataset. If σ is too large the the classifier may be insufficient for use in practice. At this point the requirements regarding privacy can not be handled anymore. To explore this trader-off, we perform a cross-validation over various values of σ and λ and measure the accuracy on a leave-out test dataset. The results of this cross-validation are presented in Table 7. Based on these results we use a regularization strength $\lambda = 1.0$ for all datasets and set $\sigma = 0.01$ for all datasets except for the Spam dataset where we use $\sigma = 0.001$.

A.5.2 Fidelity evaluation for test loss. In Section 6, we evaluate the fidelity of the retrained model with different approaches using the difference in test loss presented in a scatter plot. Figure 9 shows the plots for the Spam and Malware dataset which we omitted previously due to space limitations. As for the other datasets, it is apparent that the second-order update has the highest correlation with the retrained model, i.e., the points are closest to the identity line. We also see less variance in terms of deviation from the identity line compared to the other approaches.

A.5.3 Efficiency Evaluation. In Section 6, we present the runtime evaluation only for the Malware dataset. We show the evaluations for the remaining datasets in Table 8. The entries are based on the experiments regarding fidelity where we removed or replaced 100 combinations of 100 features from the datasets. It can be seen that the computation of the inverse Hessian is faster than retraining in case of the Diabetis and Adult datasets making the second-order update very efficient in these cases.

A.5.4 Sequential unlearning steps. Throughout the paper, we develop our update strategies as "one-shot unlearning", i.e., all perturbed samples are collected in the set $\tilde{Z}$ and the update is performed according to the strategy. The theoretical analysis in Section 5, however, shows that the error in approximation and the gradient residual norm rise with the size of $\tilde{Z}$. Therefore, a practitioner might want to perform the updates in smaller portions to keep the error in each update small. are collected and fixed only when a certain number of fixes are present and it it not clear how large this number should be.

Table 8: Average runtime when removing 100 random combinations of features for the different datasets.

	Spam			Diabetis			Adult		
Method	**Gradients**	**Runtime**	**Speed-up**	**Gradients**	**Runtime**	**Speed-up**	**Gradients**	**Runtime**	**Speed-up**
Retraining	1.4×10^6	1.52s	—	1.1×10^4	6.71ms	—	5.1×10^6	2.94s	—
SISA (5 shards)	4.1×10^5	0.56s	2.7×	9.8×10^3	29.55ms	0.2×	2.4×10^6	1.65s	1.8×
Fine-tuning	2.6×10^4	0.80s	1.9×	6.1×10^2	0.60ms	11.2×	3.9×10^4	0.09s	32.7×
Unlearning (1st)	1.1×10^4	0.04ms	38.0×	1.0×10^2	0.10ms	67.1×	1.0×10^2	4.87ms	603.7×
Unlearning (2nd)	6.5×10^4	5.42s	0.3×	1.3×10^4	0.23ms	29.2×	7.8×10^4	0.03s	98.0×

To evaluate this setting, we repeat the previous experiments to measure the gradient residual and the accuracy in a slightly different setting: Instead of changing or removing 10 features at the same time, we split the update into 10 steps of equal size. In this setting, each update is performed on a smaller set $\tilde{Z}$ and thus should have a smaller error. The error that is still present, however, will be propagated through each following update step. Figure 10 shows the comparison between sequential and one-shot updates regarding the gradient residual norm and accuracy on test data. In terms of accuracy, the sequential updates give only slight performance increases for both methods. For the gradient residual, however, we observe strong decreases when applying the updates sequentially, especially for the Diabetis and Adult dataset. This confirms the results of Theorem 1 empirically and presents a special working mode of our approach. As pointed out in Section 5.3, this increase in privacy budget comes with an increase in runtime for the second-order update: Performing 10 updates instead of one increases the runtime by a factor 10 since the Hessian has to be re-calculated at each intermediate step.

A.6 Poisoning Experiment Details

Below, we present more details about the dataset characteristics, CNN architecture and utilized unlearning parameters. Furthermore, we present an additional experiment to investigate the scalability of unlearning with increased numbers of model parameters.

The CIFAR10 dataset [45] contains 50,000 images with 32×32×3 pixels and 10 classes representing real-world objects like vehicles and animals. As our reference model, we train a convolutional neural network with 1.8 million parameters. It comprises three VGG blocks and two dense layers. The network uses 128 convolutional filters of size 3×3, a pooling size of 2×2 and the ReLU activation function. We train it using the Adam optimizer [42] with an initial learning rate of 1×10^{-4}.

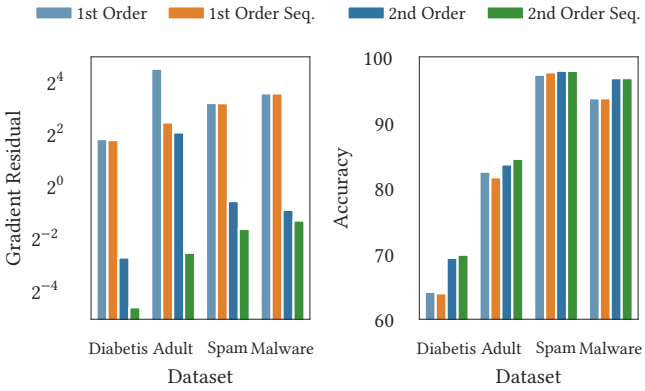

Figure 10: Average test accuracy of the corrected models (in %) when removing 100 features at once or sequentially in 10 steps.

As stated in Equation (6) and Line 1, we use an unlearning rate τ of 2×10^{-5} for the first-oder update with batches of size 512. For the approximation of the inverse Hessian we use a HVP batch size of 1,024, damping of 1×10^{-4} and scale of 2×10^5. The unlearning batch size defines the number of unlearning requests that are simultaneously performed and the HVP batch size denotes the number of training samples that are used in the hessian vector product. The damping and scale parameters are required to ensure convergence of the stochastic algorithm by Agarwal et al. [3]. Agarwal et al. originally suggest to repeat the algorithm multiple times and average the results for a more stable solution but we found that a single repitition is enough in our experiments. Finally, we introduce a patience parameter of 20 for the norm of the Hessian vector product which allows for an early stopping when the update norm does not decrease for a certain amount of iterations.