Machine Unlearning of Features and Labels

Alexander Warnecke*, Lukas Pirch*, Christian Wressnegger† and Konrad Rieck*
*Technische Universität Braunschweig
† KASTEL Security Research Labs, Karlsruhe Institute of Technology (KIT)

***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 the unlearning of features and labels is effective and significantly faster than other strategies.**

I. 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 privacy threat if they inadvertently capture sensitive information from the training data and later reveal it to users. For example, Carlini et al. [1] show that the Google text completion system contains credit card numbers from personal emails, which may be exposed to other users during auto-completion. In addition, privacy regulations, such as the European GDPR [2], enable users to request the removal of their personal data from learning models as part of the "right to be forgotten".

Deleting data from a learning model is a challenging task that requires selectively reverting the learning process. In the absence of specific methods, the only option is to retrain the model from scratch, which is costly and only possible if the original data is still available. As a remedy, Cao and Yang [3] and Bourtoule et al. [4] propose methods for *machine unlearning*. These methods partially reverse the learning process and are capable of deleting learned data points in retrospection. As a result, they enable to mitigate privacy leaks and comply with removal requests from users.

Information leaks, however, do not only manifest in isolated data points. When training on content of social media, sensitive data is often distributed across several data instances. For example, the leaked home address of a celebrity may be shared in thousands of posts, rendering the removal of affected data points inefficient. Similarly, when applying machine learning on emails, personal data in conversations, such as names, addresses, and phone numbers, can affect dozens of messages and form features in the learning model whose later removal requires substantial changes to the model's structure.

Existing approaches for unlearning [3–8] are inefficient in these cases, as they operate on data points only: First, a runtime improvement can hardly be obtained over retraining when the changes are not isolated and larger parts of the data need to be corrected. Second, removing multiple data points reduces the fidelity of the corrected model and thus is not a viable option in practical scenarios. Consequently, unlearning should not be limited to removing data points, but allow corrections at different granularity of the training data, such as fixing leaks in features and labels individually.

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 [9], that allows for estimating the influence of data on learning models [10, 11]. 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 correcting features and labels captured within the model. As a result, our method can remove privacy leaks and other unwanted content from a wide range of common learning models.

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 concepts of certified data removal [5, 7] and differential privacy [12, 13]. In particular, we measure the difference between models obtained using our approach and retraining on corrected data. By carefully introducing noise into the learning process, we derive an upper bound on this difference and thus 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 is significantly faster in comparison to sharding [4, 6] 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.

Network and Distributed System Security (NDSS) Symposium 2023
28 February - 4 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.23xxx
www.ndss-symposium.org

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 significantly 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.

Roadmap. We review related work on machine unlearning and influence functions in Section II. We motivate our approach in Section III and introduce its technical realizations in Section IV and V. A theoretical analysis is presented in Section VI and an empirical evaluation in Section VII. We discuss limitations in Section VIII and conclude in Section IX.

II. 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., 1, 14–18]. 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 [19] and Papernot et al. [20].

Machine unlearning. Methods for removing sensitive data from learning models are a recent branch of security research. As one of the first, Cao and Yang [3] 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 and thus is not well suited for correcting problems in deep neural networks.

Bourtoule et al. [4] address this problem and propose a strategy for unlearning data instances from general classification models. Similarly, Ginart et al. [6] 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 sub-models trained over these shards. Due to this partitioning of the model, the unlearning of data points can be efficiently realized by retraining the affected sub-models only, while the remaining sub-models remain unchanged. Aldaghri et al. [8] show that this approach can be further sped up for least-squares regression by choosing the shards cleverly. We refer to this family of unlearning methods as *sharding*.

Unfortunately, sharding has one critical drawback: Its efficiency quickly deteriorates when multiple data points need to be corrected. The probability that all shards need to be retrained increases with the number of affected data points, as shown in Figure 1. For a practical setup with 20 shards, as proposed by Bourtoule et al. [4], changes to as few as 150 points are sufficient to impact all shards and render the approach inefficient. We provide a detailed analysis of this limitation in Appendix A.

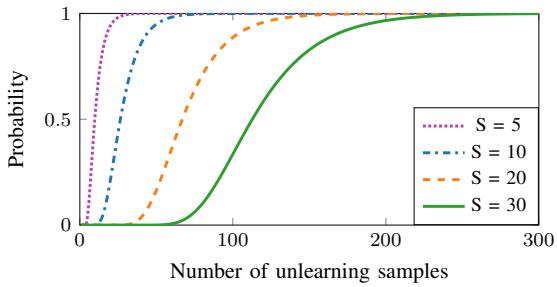

Fig. 1: Probability of all shards being affected when unlearning for varying number of data points and shards (S).

Influence functions. We base our approach on influence functions, a classic concept originating from robust statistics [9]. An *influence function* is a measure of the dependence of a statistical estimator on the value of a data point. This concept was originally introduced by Cook and Weisberg [21] for investigating the changes of linear regression models. Although influence functions have been occasionally employed in machine learning [e.g., 22, 23], it was the seminal work of Koh and Liang [10] that brought general attention to this concept and its application to learning models. In particular, Koh and Liang use influence functions for measuring the impact of training points on the predictions of a learning model.

Influence functions have then been used in a wide range of applications. For example, they have been applied to trace bias in word embeddings to documents [24, 25], determine reliable regions in learning models [26], and explain deep neural networks [27]. As part of this research strain, Basu et al. [28] increase the accuracy of influence functions by using high-order approximations, Barshan et al. [29] improve their precision, and Guo et al. [30] reduce their runtime by focusing on specific samples. Finally, Golatkar et al. [31, 32] 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. [11] study the accuracy of influence functions when estimating the loss on test data and Neel et al. [7] perform a similar analysis for gradient-based update strategies. In addition, Rad and Maleki [33] show that the prediction error on leave-one-out validations can be reduced with influence functions. Finally, Guo et al. [5] build on the concept of differential privacy and introduce the idea of certified removal of single data points. They propose a definition of indistinguishability between learning models similar to Neel et al. [7]. In this paper, we expand this work to certified unlearning of features and labels.

III. Problem Setting and Threat Model

Before presenting the technical realization of our approach to unlearning, let us first describe the problem setting along with the underlying threat model.

Threat model. For our approach, we consider learning models trained on privacy-sensitive data and accessible to users through an interface, such as a text completion system, a learning chatbot, or a collaborative spam filter. As these models operate on sensitive data, there is a need to protect their users' privacy and to close leaks in their interfaces as soon as possible.

1) First, this need arises from privacy regulations, such as the European GDPR. According to the GDPR, European citizens can request their personal information to be removed from a service to protect their privacy, including derived data in learning models [2].
2) Second, unintended memorization of personal data, such as credit card numbers, may also demand mechanisms for its removal. For example, recent work shows that text completion systems allow adversaries to extract sensitive data through their interfaces [1, 15].
3) Third, data used for constructing a learning model may accidentally violate ethical standards and thus also require removal once these violations have been detected. For example, the chatbot "Tay" by Microsoft memorized and replicated anti-semitic and racist content [34].

In all these cases, service providers must mitigate the resulting threats and immediately correct the learning model to reduce potential harm to their users.

From retraining to unlearning. At a first glance, retraining from scratch may seem like the optimal strategy to fix issues in learning models. By correcting the training data directly, all of the above issues can be reliably resolved. However, retraining from scratch comes with disadvantages:

1) Depending on the size of the original data, retraining from scratch can be costly. The entire learning process needs to be reproduced even if only a few data instances, features, or labels need to be corrected.
2) Privacy regulations require that the purpose and duration of data storage are clearly defined and approved by the user. Therefore, it may not be possible to keep the original data indefinitely for retraining.
3) Finally, with an online learning system like a chatbot, training data is volatile and might not be fully available. Nevertheless, even with such systems, inappropriate memorization must be removed quickly.

As a result of this situation, different concepts for machine unlearning have been proposed in the last years [3–8]. We follow this line of work and introduce a new mechanism for removing features and labels from learning models.

Unlearning instances vs. features. In many learning-based systems, data points are directly linked to individuals. For example, in a face recognition system, the training data consists of portrait photos, each showing one person. In this scenario, unlearning is naturally performed on the level of data points. However, privacy issues can also arise at a different granularity of the data. For example, the leaked address of a celebrity may be widely circulated on social media, affecting features of hundreds of data points. The same problem occurs when training on emails and sensitive data, such as personal names or telephone numbers, appears in several emails during a conversation. Similarly, toxic content may be captured by language models from posts of various users over time.

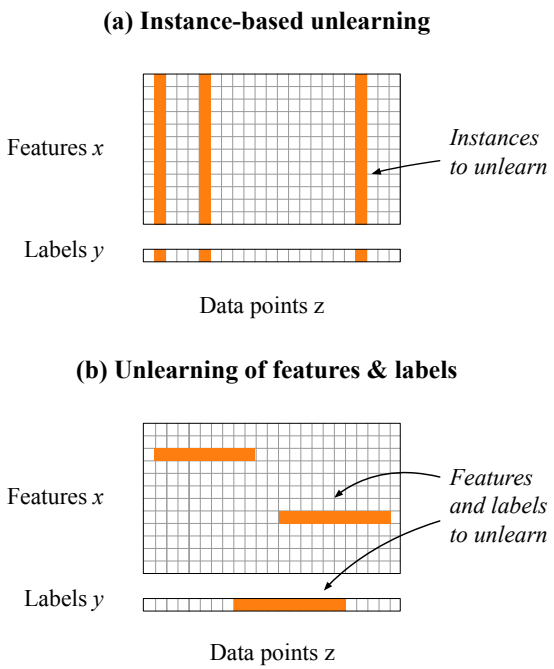

Fig. 2: Instance-based unlearning vs. unlearning of features and labels. The data to be removed is marked with orange.

In these cases, instance-based unlearning [3–8] is inefficient: First, the runtime advantage over retraining vanishes with the number of data points affected, as shown in the previous section. Second, removing entire instances and not just the affected features or labels unnecessarily degrades the performance of the corrected models. As a solution to this situation, we propose a method for unlearning of individual features and labels. As illustrated in Figure 2, this strategy operates on an orthogonal dimension of the data. Instead of correcting privacy issues along data instances (columns), we focus on resolving the issues in feature values and labels (rows). This is possible because we formulate unlearning as a closed-form update of the model, which enables us to correct features and labels at arbitrary positions in the training data.

To the best of our knowledge, we are the first to tackle the problem of unlearning from this perspective, thereby adding a new tool to the existing machinery for mitigating privacy threats in machine learning. In particular, our approach provides the following advantages:

- *Efficiency.* When privacy issues affect multiple data instances but are limited to particular features or labels, it is more efficient to correct these directly. As demonstrated in our evaluation, we achieve a significant performance improvement over existing approaches.
- *Flexibility.* Due to the concept of influence functions, we can correct arbitrary feature values and labels in the original training data. As a result, our approach can also unlearn entire data points, making it a versatile alternative to existing methods.

IV. Unlearning with Influence

As basis for our approach, 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 $\mathcal{X} = \mathbb{R}^d$ 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 and the true labels, the optimal model θ^* can be found by minimizing the regularized empirical risk,

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

where L_b is the loss on the entire dataset D and Ω a common regularizer [see 35]. 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. For realizing certified unlearning, however, it enables to add a small amount of noise to the minimization, similar to differentially private learning [13, 36]. We introduce this technique later in Section VI and omit the subscript in L_b for now.

A. Unlearning Data Points

We begin the design of our approach by asking a simple question: How would the optimal 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:

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

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 problem:

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

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

As a result, 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 up-weighting and down-weighting. It is easy to see that this approach is not restricted to a single point. We can define a set of points Z as well as their perturbed versions $\tilde{Z}$. That is, Z contains the original data, while $\tilde{Z}$ its corrected version. Based on this definition, w arrive at the following optimization problem:

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

This generalization enables us to approximate changes on larger parts of the training data. Instead of solving the problem in Equation (4), 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

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

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. We show in Section V that this update step can be efficiently computed using first-order and second-order derivatives. Note that if $\tilde{Z} = \emptyset$ in Equation (4), our approach also yields updates to remove *data points* similar to prior work [5, 10].

B. 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 specifying different perturbations $\tilde{Z}$, we can now realize several unlearning tasks with 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, home addresses, or other sensitive information might need to be removed after a model has been trained on a corpus of emails. Similarly, in a credit scoring system, the race, gender or other biasing features might need to be replaced with neutral content.

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, so 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) [37, 38]. 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 VII-B, 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 reduces 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^*. These two models are equivalent for a large class of learning models, including 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.

V. Update Steps for Unlearning

Our approach rests on changing the influence of training data in a closed-form update. 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.

A. 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* 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 B1. Intuitively, this update shifts the model parameters 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 similar to a gradient descent update GD 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: The gradients of ℓ can be computed in $\mathcal{O}(p)$ [39] where p is the number of parameters in the learning model. However, the first-order update involves a parameter τ that controls the impact of the unlearning. To ensure that the data has been completely replaced, it is necessary to calibrate this parameter using a measure for the success of unlearning. In Section VII, we show how the exposure metric proposed by Carlini et al. [1] can be used for this calibration.

B. Second-Order Update

The unlearning rate τ 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

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

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 21]. We can then perform a linear approximation as follows

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

Since all operations are linear, we can extend Equation (7) to multiple data points and finally obtain the *second-order update* for our approach:

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

A full derivation of this update step is provided in Appendix B2. 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 large learning models.

Calculating the inverse Hessian. Given a model $\theta \in \mathbb{R}^p$ with p parameters, forming and inverting the Hessian requires $\mathcal{O}(np^2 + p^3)$ time and $\mathcal{O}(p^2)$ space [10]. 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 VII-A, we demonstrate that unlearning features from a linear model with about 2,000 parameters can be realized with this approach in less than a second.

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 a technique proposed by Koh and Liang [10]. Its derivation and an algorithm for its implementation are presented in Appendix C. While this approximation weakens the theoretical guarantees of our approach, it still enables successfully unlearning data from large learning models. In Section VII-B, 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.

VI. Certified Unlearning

Machine unlearning 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 the one created by retraining. In the following, we derive conditions under which the second-order updates of our approach provide certified unlearning. To this end, we build on the concepts of *differential privacy* [13] and *certified data removal* [5], and adapt them to the unlearning task.

Let $\mathcal{A}$ be a learning algorithm that outputs a model $\theta \in \Theta$ after training on a dataset D, that is, $\mathcal{A} : D \to \Theta$. Randomness added by $\mathcal{A}$ 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*

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

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. [5], 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*

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

and

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

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 this relaxation, it becomes possible to derive practical conditions under which our approach realizes certified unlearning.

A. Certified Unlearning of Features and Labels

To construct theoretical guarantees for our approach, we make two basic assumptions on the employed learning algorithm: First, we assume that the loss function ℓ is twice differentiable and strictly convex, so that H^{-1} always exists and the second-order update is applicable. Second, we consider an L_2 regularization in the optimization problem (1), that is, the regularizer $\Omega(\theta)$ is given by $\frac{1}{2}\|\theta\|_2^2$. Both assumptions are satisfied 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 the one obtained by retraining on the corrected dataset D'. We differentiate between the gradient residual of the plain loss L and the 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. As a result, we can adjust the probability distribution of b to realize certified unlearning, similar to sensitivity methods [36]. The corresponding proofs are given in Appendix D. Moreover, for a detailed discussion on Lipschitz continuity used in the following theorem, we refer the reader to the paper by Chaudhuri et al. [12].

Theorem 1. *Assume that $\|x_i\|_2 \leq 1$ for all data points and the gradient $\nabla \ell(z, \theta)$ is γ_z-Lipschitz with respect to z at θ^* and γ-Lipschitz with respect to θ. Further let $\tilde{Z}$ change the features $j, \ldots, j + F$ 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.

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. 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. [5] 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. [5])**.** *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 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 $p(b) = e^{-\frac{\epsilon}{c'}\|b\|_2}$ then $\mathcal{U}$ performs ϵ-certified unlearning for $\mathcal{A}$.*
2) *If $p \sim \mathcal{N}(0, c\epsilon'/\epsilon)^d$ for some $c > 0$ then $\mathcal{U}$ 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 probability distribution of b to obtain a certified unlearning method.

Further details. We refer the reader for further details on certified unlearning to the appendix. In particular, we present the proofs for all theorems in Appendix D, we discuss the relation between our approach and differential privacy in Appendix E, and we show how the privacy budget (ϵ, δ) can support multiple unlearning requests in Appendix F.

VII. Empirical Analysis

We proceed with an empirical analysis of our approach and its capabilities. For this analysis, we examine the performance of unlearning in different scenarios and compare our method to other strategies for removing data, such as retraining, sharding, fine-tuning, and differentially private learning. 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 three application 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. These models are widely used in fraud, spam and malware detection due to their simplicity and strongly convex loss function [40, 41]. While they induce a slight performance drop compared to neural networks (see Appendix H), they still remain of practical relevance. We investigate how our approach can unlearn sensitive features from these models (see Section VII-A).

Scenario 2: Unintended memorization. In the second scenario, we consider the problem of unintended memorization [1]. Language models based on recurrent neural networks can accidentally memorize sensitive data, such as credit card numbers or private messages. Through specifically crafted inputs, an attacker can extract this data during text completion [1, 15]. We apply unlearning of features and labels to remove these privacy leaks from language models (see Section VII-B).

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 learning model. We use unlearning of labels as a strategy to correct this defect and restore the original performance without retraining (see Section VII-C).

Performance measures. The success of unlearning depends on three properties: An effective method must (1) remove the selected data, (2) preserve the model's quality, and (3) be efficient compared to retraining. A method that fails to satisfy any of these properties is ineffective, because it either does not correctly unlearn data, degrades the model, or lacks behind retraining. To reflect this setting, we introduce three performance measures for our empirical analysis.

Efficacy of unlearning. The most important property 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 can use the *exposure metric* [1] to measure the memorization of specific sequences in language models after unlearning.

Fidelity of unlearning. The second property 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 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 property.

Efficiency of unlearning. If the training data used to generate a model is available, a simple unlearning strategy is retraining. This strategy, however, involves significant runtime and storage costs. Therefore, we consider the *efficiency* of unlearning as the third property. 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 baselines as reference for examining the efficacy, fidelity, and efficiency. In particular, we consider retraining, fine-tuning, differential privacy and sharding 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 [12]. As discussed in Appendix E, 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. This baseline simply continues to train a model using corrected data. We implement this 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 unlearning methods, such as SISA and our approach.

SISA (Sharding). As the fourth baseline, we consider the unlearning method SISA proposed by Bourtoule et al. [4]. The method splits the training data in shards and trains sub-models for each shard separately. The final classification is obtained by a majority vote over these sub-models. Unlearning is conducted by retraining those shards that contain data to be removed.

A. Unlearning Sensitive Features

In our first unlearning scenario, we focus on the removal of sensitive features from learning models with strongly convex loss functions. In particular, we consider linear models trained with a logistic regression on real-world datasets. We employ datasets for spam filtering [42], Android malware detection [41], diabetes forecasting [43] and prediction of income based on census data [44]. An overview of the datasets and their basic statistics is given in Table I.

In all experiments, we divide each dataset into a training and test set with a ratio of $80\,\%$ and $20\,\%$, respectively. To create a feature space for learning, we use the numerical features of the Adult and Diabetis dataset as is, while for the Spam and Malware datasets we extract *bag-of-words features*. That is, we represent each email (malware) by the words (capabilities) it contains and construct corresponding feature vectors [see 41, 42]. Finally, we train logistic regression models for all four datasets. As a reference, we additionally present results for a neural network in Appendix H.

TABLE I: 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

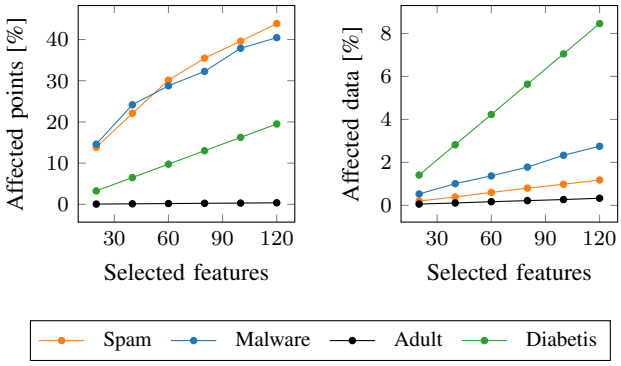

Fig. 3: Affected data points and overall data when removing or changing features in the different datasets.

Sensitive features. Two of the considered datasets are high-dimensional and contain sparse data (Spam and Malware), while the other two are low-dimensional with dense feature vectors (Adult and Diabetis). Consequently, we introduce two strategies for defining sensitive features for unlearning. Since no privacy leaks are known for the datasets, we focus on features that *potentially* cause privacy issues.

- For the high-dimensional datasets, we aim at removing (revoking) entire features (dimensions) from the learning models. In particular, we select dimensions associated with personal names contained in the emails as sensitive features for the Spam dataset and choose URLs extracted from the Android apps for the Malware dataset.
- For the low-dimensional datasets, we focus on replacing selected feature values. For the Adult dataset, we change the marital status, sex, and race of randomly selected individuals. For the Diabetis dataset, we adjust the age, body mass index, and sex of individuals. We replace the respective feature values with 0 to simulate the removal of discriminatory bias from the learning models.

Figure 3 illustrates how these changes affect the different datasets. Removing features entirely impacts many data points, while replacing selected feature values affects only a few. This effect also depends on the size of the datasets. To avoid a sampling bias in the selection of sensitive features, we randomly draw 100 combinations of these feature changes for all four datasets and present averaged results in the following.

Unlearning task. Based on the type of sensitive features, we apply the different unlearning methods to the respective learning models. Technically, our approach benefits from the convex loss function of the logistic regression model, which allows us to apply certified unlearning as presented in Section VI. 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. We analyze the efficacy of unlearning using the gradient residual norms of the methods. The average size of this norm after unlearning is presented in Figure 4 for the different approaches when varying the number of features to be removed or replaced, respectively. The residuals increase with the amount of affected features, indicating a growing divergence between the unlearning methods and retraining.

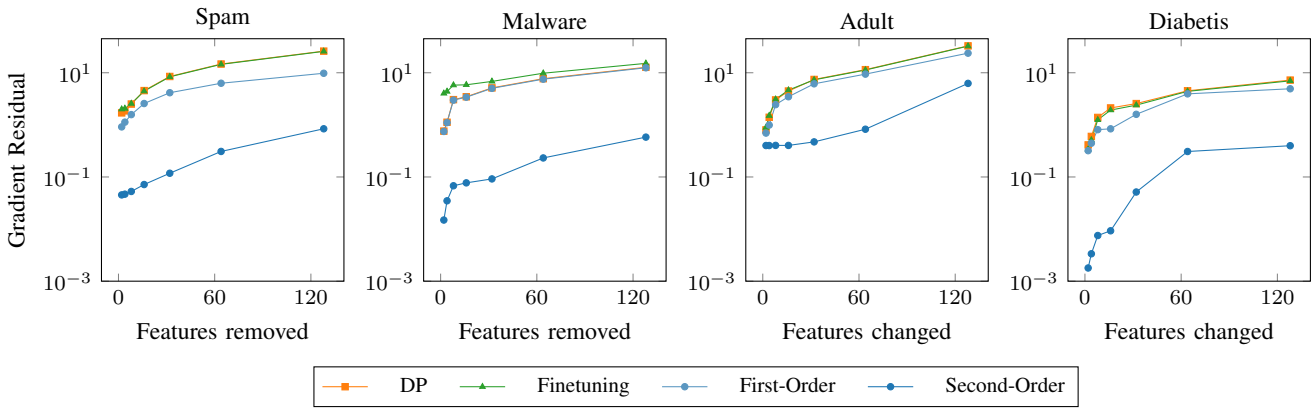

Fig. 4: Efficacy (gradient residual) of the certified unlearning methods for varying number of affected features (Lower values are better).

However, the steepness of this development gradually reduces as more features are remove or changed. Our second-order update significantly outperforms all other methods in this scenario. The gradient residual norms are an order of magnitude lower on the Spam, Malware, and Diabetis dataset, regardless of the amount of affected features. Among the other unlearning methods, no clear ranking can be determined in this experiment.

Fidelity evaluation. We evaluate the fidelity of the unlearning methods using two techniques: First, we investigate the loss between retraining and unlearning on the test data as proposed by Koh and Liang [10]. Figure 5 shows this comparison for the Diabetis and Malware dataset when removing or replacing 100 features, respectively. We observe that the second-order update approximates the retraining very well, since the points are close to the diagonal line. In contrast, the other methods cannot always adapt to the distribution shift, resulting in larger differences. This trend also holds for the other datasets which we report in Appendix G1.

Second, we use the test accuracy of a model that provides certified unlearning for the removal or replacement of features to evaluate the fidelity. To simulate a realistic application of certified unlearning, we fix a privacy budget (ϵ, δ) in advance and adapt the noise term b in relation to the amount of affected features. That is, the more features are changed, the more we need to increase the noise on the model to achieve the same guarantees. In particular, Theorem 3 states that the noise term b must be sampled from a Gaussian normal distribution with variance σ, which is given by

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

where β is constant corresponding to a general upper bound of the gradient residual loss for the considered learning task.

In the following, we select $\epsilon = 0.1$ and $\delta = 0.01$ as a privacy budget, which yields $c \approx 3.16$. We compute the bound β on the gradient residual by sampling 100 feature combinations to unlearn, compute σ and determine the resulting test accuracy of the four datasets. As we see in the following, this privacy budget is strict and limits the amount of features that can be adapted. Nevertheless, if a larger number of features needs to be removed, this budget can be increased, though at the cost of weakening the certification guarantees.

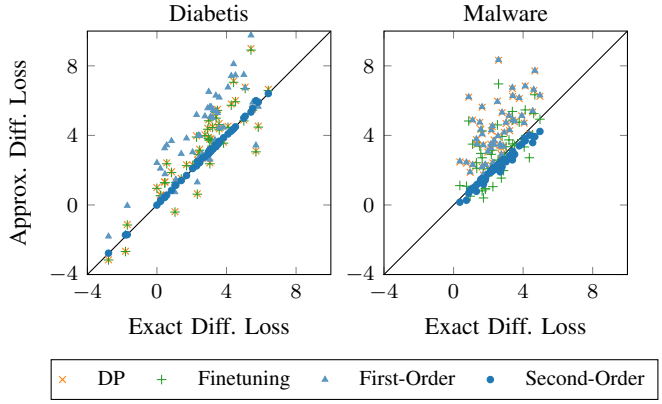

Fig. 5: Difference in loss between retraining and unlearning with 100 affected features.

The accuracy of the models is shown in Figure 6 for a varying number of removed or replaced features, respectively. The accuracy reduces with the amount of affected features, as the noise on the model weights is increased accordingly. While for the low-dimensional datasets this reduction is moderate, we observe a strong decline of fidelity for the high-dimensional data. This decline results from the privacy budget that requires a notable amount of noise to be added to enable a certified removal of entire dimensions.

Our second-order update shows the best performance of all methods and remains close to retraining if up to 60 features are changed. In contrast, the other methods quickly drop in accuracy already when unlearning 20 or less features. An exception is sharding. While the method provides the weakest performance on the high-dimensional datasets due to instability in the majority voting, it is almost identical to retraining on the low-dimensional datasets.

Efficiency evaluation. Finally, we evaluate the efficiency of the different methods. Table II shows the measured runtime on the Malware dataset, while the results for the other datasets are shown in Appendix G2. We omit measurements for the differential privacy baseline, as it does not involve an unlearning step. Due to the simple structure of the logistic regression, the

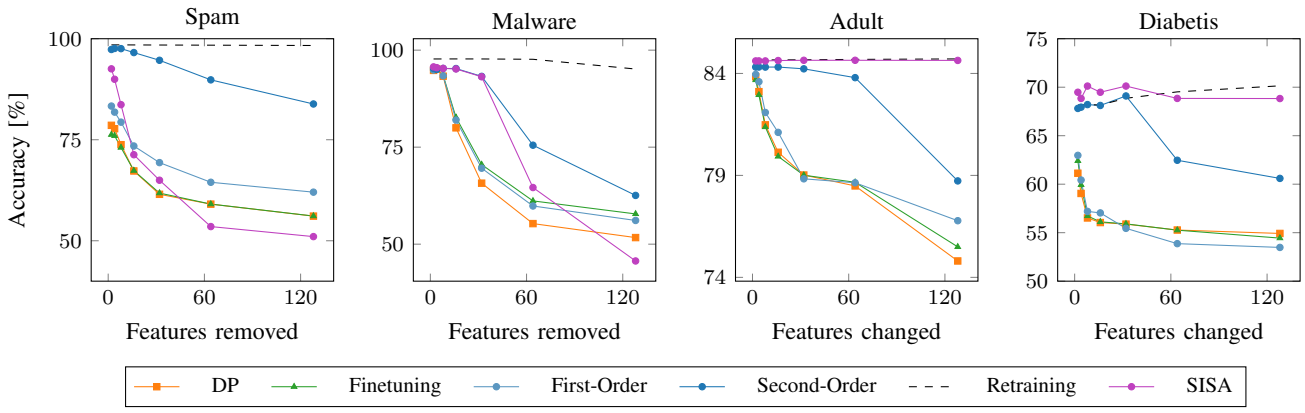

Fig. 6: Fidelity (accuracy) of the certified unlearning methods for varying number of affected features (higher values are better).

runtime of all methods is very low. Fine-tuning, our second-order update, and sharding reach speed-up factors of $2\times$, $4\times$, and $6\times$, respectively over retraining. Our first-order update is even faster and attains a speed-up factor of $90\times$. While the other approaches operate on the entire dataset, the first-order update considers only the corrected points.

For the second-order method, we find that roughly $90\,\%$ of the runtime and gradient computations are 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 notable speed-up, though at the cost of approximation accuracy.

Takeaway message. Sensitive features in learning models with convex loss can be removed or replaced with theoretical guarantees. Our second-order update provides the best trade-off between efficacy, fidelity, and efficiency for this certified unlearning.

B. Unlearning Unintended Memorization

In our second unlearning scenario, we focus on removing unintended memorization from language models. Carlini et al. [1] show that these models can memorize rare inputs in the training data and exactly reproduce them during application. If the reproduced data contains private information like credit card numbers or telephone numbers, we are faced with a privacy issue [14, 15]. In the following, we use our approach to tackle this problem and demonstrate that unlearning is also possible with non-convex loss functions.

TABLE II: 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	$2\times$
Fine-tuning	3.9×10^4	1.03s	$6\times$
First-Order	1.5×10^4	0.02s	$90\times$
Second-Order	9.4×10^4	0.63s	$4\times$

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 [45]. 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 [1]. In our experiments, we use sequences of length $(5, 10, 15, 20)$ and repeat the canary so that $(200, 500, 1000, 2000)$ points are affected. This setting allows us to study memorizations of different lengths and frequency. After training, we find that the inserted numbers are the most likely prediction when we ask the model to complete the canary sentence.

Exposure metric. In contrast to the previous scenario, the loss of the language model is non-convex and thus certified unlearning 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. To this end, we employ the *exposure metric*, which is defined as

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

where s is a sequence and Q is the set of possible sequences with the same length given a fixed alphabet. The function $\text{rank}_\theta(s)$ returns the rank of s with respect to the model θ and the set Q. The rank is calculated using the *log-perplexity* of the sequence s and returns the number of sequences that are more likely to be generated than s. As a result, the exposure metric tells us how likely a sequence s is generated by θ in relation to all possible sequences of the same length. Further details on this metric are provided by Carlini et al. [1].

As an example, Figure 7 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

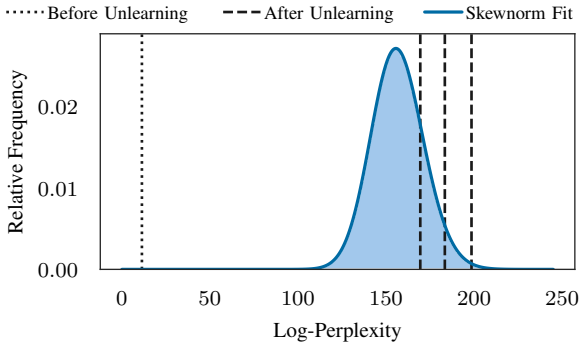

Fig. 7: Perplexity distribution of the language model. The vertical lines indicate the perplexity of an inserted telephone number. Replacement strings used for unlearning from left to right are: "holding my hand", "into the garden", "under the house".

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 center of the distribution (dashed lines to the right).

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 selecting random words and phrases from the training corpus works best for this task. Some examples of replacements are shown in Table IV. The model has already captured these character dependencies, resulting in small updates of the model parameters. The unlearning of the substitutions, however, is more involved than in the previous scenario. The language model is trained to predict a character from preceding characters. Thus, replacing a text means changing 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 evaluate whether the memorized numbers have been successfully unlearned from the language model. An important result of the study by Carlini et al. [1] 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. Consequently, we test three different substitution sequences for each telephone number, calculate the exposure metric, and use the best for our evaluation. Table III shows the results of this experiment.

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

Number length	5	10	15	20
Original model	43 ± 19	70 ± 26	109 ± 16	99 ± 52
Retraining	0 ± 0	0 ± 0	0 ± 0	0 ± 0
Fine-tuning	39 ± 21	31 ± 44	50 ± 50	57 ± 73
SISA (n shards)	0 ± 0	0 ± 0	0 ± 0	0 ± 0
First-Order	0 ± 0	0 ± 0	0 ± 0	0 ± 0
Second-Order	0 ± 0	0 ± 0	0 ± 0	0 ± 0

We observe that our first-order and second-order updates yield exposure values close to zero (< 0.001) for all sequence lengths, rendering an extraction impossible. Retraining and SISA yield an exposure of zero by design since the injected sequences are removed from the training data. In contrast, fine-tuning leaves a large exposure in the model, so that a successful extraction is still possible. On closer inspection, we find that the performance of fine-tuning depends on the order of the training data, resulting in high deviation in the experimental runs. This problem cannot be easily mitigated by learning over further epochs and thus highlights the need for unlearning techniques.

We also find that the selected substitution plays an important role for unlearning. In Figure 7, we report the log-perplexity of the canary for three different substitutions after unlearning. Each replacement shifts the canary to the right and turns it into an 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 IV 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 overfit towards the replacement. The sentences follow the language structure and reflect the wording of the novel. Both observations indicate that the parameters of the language model are indeed corrected and not just overwritten with other values.

Fidelity evaluation. To evaluate the fidelity, we examine the accuracy of the corrected models as shown in Figure 8 (left), where the accuracy of all unlearning methods is plotted for different numbers of affected data points. For small changes, all approaches except sharding come close to retraining in performance. Sharding is unsuited for unlearning in this scenario. The method uses an ensemble of sub-models trained on different shards. Each sub-model produces an own sequence of text and thus combining them with majority voting leads to inaccurate predictions.

With larger changes to the model, the accuracy of both of our methods gradually begins to decline. The second-order update provides slightly better results because the Hessian contains

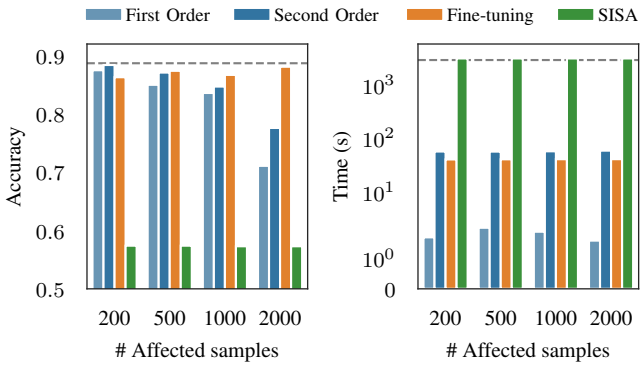

Fig. 8: Accuracy after unlearning unintended memorization. The dashed line corresponds to a model retrained from scratch.

TABLE IV: 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 ..."

information about unchanged samples. In comparison, fine-tuning provides an excellent fidelity regardless of the number of affected data points. This performance, however, is misleading. Fine-tuning fails to remove the injected sequences from the model, as shown in Table III, and hence is also unsuited for unlearning in this scenario.

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 of our approach, while the other methods are calculated using a GPU (GeForce RTX 2080 Ti). The runtime required for each approach is presented in Figure 8 (right).

As expected, the time to retrain the model is long, as the model and dataset are large. Sharding cannot provide any runtime advantage over retraining, since all shards are affected and need to be retrained as well. Our methods yield a notable improvement. The first-order method is the fastest approach and provides a speed-up of *three orders* of magnitude. 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.

> *Takeaway message.* Unintended memorization can be removed from language models by unlearning features and labels. Our first-order method provides the best trade-off between efficacy, fidelity, and efficiency.

C. Unlearning Poisoning Samples

In the third scenario, we focus on repairing a poisoning attack in computer vision. We simulate *label poisoning* where an adversary partially flips labels between classes in the training data. While this attack does not impact the privacy, 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 1.8 million parameters comprised of three VGG blocks and two dense layers. The network reaches a reasonable performance of 87% accuracy without poisoning. Under attack, however, it suffers from a notable drop in accuracy (10% on average). Further details about the experimental setup are provided in Appendix I.

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 [10, 46]. We evaluate the attack with different poisoning budgets and seeds for sampling.

Unlearning task. We aim at correcting the flipped labels of the poisoning attack. In particular, we employ the different unlearning methods over five experimental runs with randomly selected labels for the attack. We report averaged values in Figure 9 for different number of poisoned labels, where the dashed line represents the accuracy and training time of the clean reference model. Correcting all poisoned labels in one closed-form update is difficult due to memory constraints. Thus, we perform unlearning in uniformly sampled batches of 512 instances, as detailed in Appendix F. Moreover, we update only the fully connected layers, which are mainly responsible for the final prediction.

Efficacy and fidelity evaluation. In this scenario, we do not seek to remove the influence of certain features from the training data but mitigate the effects of poisoned labels This poisoning manifests 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.

The accuracy for the different unlearning methods is shown in Figure 9 (left). We find that none of the approaches is able to completely remove the effect of the poisoning attack. Still, good results are obtained with the first-order and second-

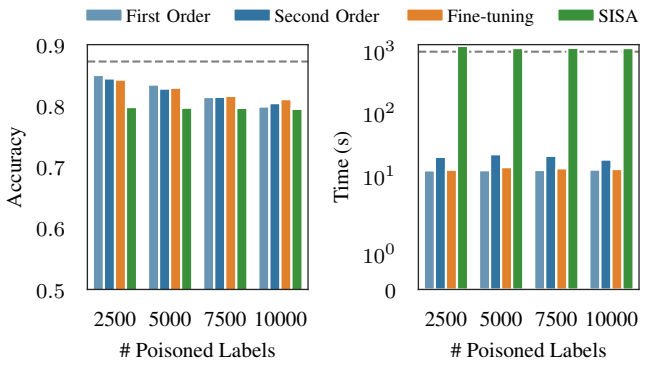

Fig. 9: Accuracy on held-out data after unlearning poisoned labels. The dashed line corresponds to a model retrained from scratch.

order update as well as fine-tuning, which all come close to the original performance for 2,500 poisoned labels. However, we observe a continuous performance decline when more labels are poisoned. A manipulation of 10,000 labels during training cannot be sufficiently reverted by any of the methods. Interestingly, sharding 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.

Efficiency evaluation. Lastly, we evaluate the 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 9 shows that the first-order update and fine-tuning are very efficient and can be computed in approximately 10 seconds, whereas retraining requires over 15 minutes. The second-order update is slightly slower but still with 20.8 seconds two orders of magnitude faster than retraining. In contrast, the sharding approach is the slowest and does not provide any advantage over retraining. Consequently, the first-order update and fine-tuning provide the best strategies for unlearning in this scenario.

In computer vision, learning models are often huge. Hence, we also investigate the scalability of our approach. Figure 10 shows the accuracy and runtime of the first-order and second-order update for increasing model sizes. In particular, we scale the number of model parameters from 1.8 millions up to 42 millions. For both update techniques, the accuracy remains roughly the same. The runtime naturally increases with the model size, yet the slope is (almost) linear for both approaches. We observe a slight peak when reaching the limits of our hardware. Overall, we find that the linear runtime bounds of the underlying algorithm hold in practice [47].

> *Takeaway message.* Label poisoning can be mitigated by unlearning labels. Our first-order update and fine-tuning are suitable methods and provide the best trade-off between efficacy (=fidelity) and efficiency.

VIII. Limitations

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

Limits of unlearning. The efficacy of unlearning decreases with the number of affected features and labels. While privacy leaks with hundreds of sensitive features and thousands of labels can be handled well with our approach, changing millions of data points 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 [48]. Nevertheless, our method offers a significant speed-up over retraining and sharding in situations where a moderate number of data points needs to be corrected.

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

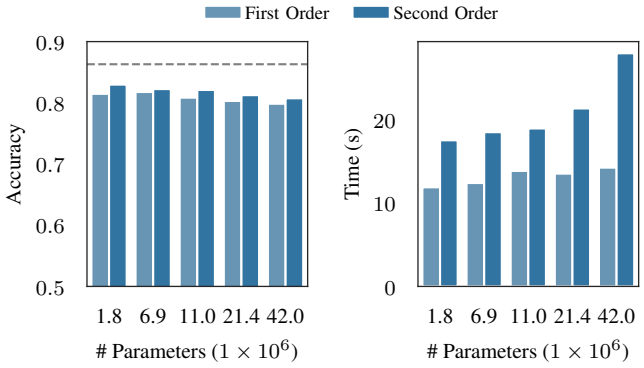

Fig. 10: Accuracy (left) and runtime (right) on held-out data after unlearning 5,000 poisoned labels for multiple model sizes. The dashed line corresponds to the accuracy of the models on clean data.

functions, as we demonstrate in the empirical evaluation, they require an additional measure to validate unlearning success. Fortunately, 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.

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. [1, 15] 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.

IX. 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". However, sensitive information is often spread across instances, impacting larger portions of the training data. Instance-based unlearning is limited in this setting. 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 effectivity 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 three practical scenarios. 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.

ACKNOWLEDGMENT

The authors acknowledge funding from the German Federal Ministry of Education and Research (BMBF) under the projects BIFOLD (FKZ01IS18025B) and DataChainSec (FKZ16KIS1700). Furthermore, the authors acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excelence Strategy EXC 2092 CASA-390781972 and by the Helmholtz Association (HGF) within topic "46.23 Engineering Secure Systems".

REFERENCES

[1] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song, "The secret sharer: Evaluating and testing unintended memorization in neural networks," in *USENIX Security Symposium*, 2019, pp. 267–284.

[2] "Regulation 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data," *Official Journal of the European Union*, vol. 119, pp. 1–88, 2016.

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

[4] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," in *IEEE Symposium on Security and Privacy (S&P)*, 2021.

[5] C. Guo, T. Goldstein, A. Y. Hannun, and L. van der Maaten, "Certified data removal from machine learning models," in *International Conference on Machine Learning (ICML)*, 2020, pp. 3822–3831.

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

[7] S. Neel, A. Roth, and S. Sharifi-Malvajerdi, "Descent-to-delete: Gradient-based methods for machine unlearning," in *International Conference on Algorithmic Learning Theory (ALT)*, 2021.

[8] N. Aldaghri, H. Mahdavifar, and A. Beirami, "Coded machine unlearning," *IEEE Access*, vol. 9, pp. 88 137 – 88 150, 2021.

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

[10] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *International Conference on Machine Learning (ICML)*, 2017, pp. 1885–1894.

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

[12] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization," *Journal of Machine Learning Research*, p. 1069–1109, 2011.

[13] C. Dwork, "Differential privacy," in *Automata, Languages and Programming*, 2006, pp. 1–12.

[14] S. Zanella Béguelin, L. Wutschitz, S. Tople, V. Rühle, A. Paverd, O. Ohrimenko, B. Köpf, and M. Brockschmidt, "Analyzing Information Leakage of Updates to Natural Language Models," in *ACM Conference on Computer and Communications Security*, 2020.

[15] N. Carlini, F. Tramèr, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, and A. Roberts, "Extracting training data from large language models," in *USENIX Security Symposium*, 2021.

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

[17] K. Leino and M. Fredrikson, "Stolen memories: Leveraging model memorization for calibrated white-box membership inference," in *USENIX Security Symposium*, 2020.

[18] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models." in *IEEE Symposium on Security and Privacy (S&P)*, 2017, pp. 3–18.

[19] E. De Cristofaro, "A critical overview of privacy in machine learning," *IEEE Security & Privacy Magazine*, vol. 19, no. 4, 2021.

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

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

[22] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems (NIPS)*, 1990.

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

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

[25] H. Chen, S. Si, Y. Li, C. Chelba, S. Kumar, D. Boning, and C.-J. Hsieh, "Multi-stage influence function," in *Advances in Neural Information Processing Systems (NIPS)*, 2020, pp. 12 732–12 742.

[26] P. Schulam and S. Saria, "Can you trust this prediction? auditing pointwise reliability after learning," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019.

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

[28] S. Basu, X. You, and S. Feizi, "On second-order group influence functions for black-box predictions," in *International Conference on Machine Learning (ICML)*, 2020, pp. 715–724.

[29] E. Barshan, M. Brunet, and G. Dziugaite, "Relatif: Identifying explanatory training examples via relative influence," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

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

[31] A. Golatkar, A. Achille, A. Ravichandran, M. Polito, and S. Soatto, "Mixed-privacy forgetting in deep networks," in

Conference on Computer Vision and Pattern Recognition (CVPR), 2021.
[32] A. Golatkar, A. Achille, and S. Soatto, "Eternal sunshine of the spotless net: Selective forgetting in deep networks," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
[33] K. R. Rad and 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)*, vol. 82, 2018.
[34] J. Wakefield, "Microsoft chatbot is taught to swear on Twitter," BBC News, https://www.bbc.com/news/technology-35890188.
[35] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, 2nd ed. John Wiley & Sons, 2000.
[36] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, p. 211–407, 2014.
[37] A. Graves, "Generating sequences with recurrent neural networks," Computing Research Repository (CoRR), Tech. Rep. arXiv:1308.0850, 2013.
[38] I. Sutskever, J. Martens, and G. Hinton, "Generating text with recurrent neural networks," in *International Conference on Machine Learning (ICML)*, 2011.
[39] B. A. Pearlmutter, "Fast exact multiplication by the hessian," *Neural Computation*, vol. 6, no. 1, p. 147–160, 1994.
[40] J. Attenberg, K. Weinberger, A. Dasgupta, A. Smola, and M. Zinkevich, "Collaborative email-spam filtering with the hashing trick," in *Conference on Email and Anti-Spam (CEAS)*, 2009.
[41] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Efficient and explainable detection of Android malware in your pocket," University of Göttingen, Tech. Rep. IFI-TB-2013-02, Aug. 2013.
[42] V. Metsis, G. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes - which naive bayes?" in *Proc. of Conference on Email and Anti-Spam (CEAS)*, 2006.
[43] D. Dua and C. Graff, "UCI machine learning repository. diabetis data set." 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[44] ——, "UCI machine learning repository. census income data set." 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[45] S. Merity, N. S. Keskar, and R. Socher, "An analysis of neural language modeling at multiple scales," *arxiv:1803.08240*, 2018.
[46] H. Xiao, H. Xiao, and C. Eckert, "Adversarial label flips attack on support vector machines," in *Proc. of European Conference on Artificial Intelligence (ECAI)*, 2012, pp. 870–875.
[47] N. Agarwal, B. Bullins, and E. Hazan, "Second-order stochastic optimization for machine learning in linear time," *Journal of Machine Learning Research (JMLR)*, p. 4148–4187, 2017.
[48] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 67, 1997.
[49] S. Boyd and L. Vandenberghe, *Convex Optimization*, 2004.
[50] D. Desfontaines and B. Pejó, "Sok: Differential privacies," *Proceedings on Privacy Enhancing Technologies*, pp. 288–313, 2020.
[51] D. Kifer and A. Machanavajjhala, "No free lunch in data privacy," in *ACM International Conference on Management of Data*, 2011, p. 193–204.
[52] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *ACM Conference on Computer and Communications Security (CCS)*, 2016, pp. 308–318.
[53] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Conference on Theory of Cryptography*, 2006, p. 265–284.
[54] C. Dwork, G. N. Rothblum, and S. Vadhan, "Boosting and differential privacy," in *Annual Symposium on Foundations of Computer Science*, 2010, p. 51–60.
[55] C. Dwork and J. Lei, "Differential privacy and robust statistics," in *Annual ACM Symposium on Theory of Computing*, 2009, p. 371–380.
[56] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.
[57] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

APPENDIX

A. Stochastic Analysis of Sharding

To better understand the need for closed-form updates on model parameters, we examine current sharding strategies and investigate the circumstances under which they reach their limits. Bourtoule et al. [4] 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 data points at which sharding becomes as efficient as retraining.

In this context, we consider n data instances to unlearn which are uniformly distributed across s shards. Let $p(n)$ denote the probability 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 in Section II 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.

B. Deriving the Update Steps

In this 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 and Vandenberghe [49].

1) First-order update: To derive the first-order update, let us recall the optimization problem for the corrected learning model from Section IV:

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

where $L_\epsilon(\theta;D)$ is a combined 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^* by

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

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 minimum 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 V-A:

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

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

2) 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 and Weisberg [21]. 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 can again approximate the conditions using a first-order Taylor series at θ^* and obtain

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

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

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

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 V-B

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

C. 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. [47] for computing expressions of the form $H^{-1}v$. This scheme requires to only calculate Hv and avoids storing H^{-1}. The resulting *Hessian-Vector-Products* (HVPs) allow us to calculate Hv efficiently by making use of the linearity of the gradient

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

If we denote 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 $H^{-1}_j = I + (I-H)H^{-1}_{j-1}$. Now, 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 $\mathcal{O}(p)$ as demonstrated by Pearlmutter [39].

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 $= \nabla_\theta\big(v^T \nabla_\theta L(\text{batch}, \theta^*)\big)$
      $\theta_{\text{new}} = v + (1-d)\theta_{\text{new}} - \text{hvp}/s$
    $\Delta = \Delta + \theta_{\text{new}}/r$
return Δ

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 $\tilde{H}_t^{-1} v$ as our final estimate of $H^{-1}v$ in $\mathcal{O}(rtp)$ of time. The pseudo-code in Algorithm 1 summarizes how we compute the second-order update.

D. Proofs for Certified Unlearning

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

Theorem 1. *Assume that $\|x_i\|_2 \leq 1$ for all data points and the gradient $\nabla\ell(z, \theta)$ is γ_z-Lipschitz with respect to z at θ^* and γ-Lipschitz with respect to θ. Further let $\tilde{Z}$ change the features $j, \ldots, j+F$ 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*

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

Proof: By definition, we have

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

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

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

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

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

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

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

□

We proceed to prove the update bounds of Theorem 1. The 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

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

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

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

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

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

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

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

□

Proof: (Case 2) For the second-order update of our approach, 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

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

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

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

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

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

and then for the Hessians obtain

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

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

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

□

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 $\mathcal{U}$ performs ϵ-certified unlearning for $\mathcal{A}$.*
2) *If $p \sim \mathcal{N}(0, c\epsilon'/\epsilon)^d$ for some $c > 0$ then $\mathcal{U}$ 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 and Roth [36].

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

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

   If we now apply Theorem 2, we can finalize the proof for the first case.
2) The second proof is similar to Theorem 3.22 in Dwork and Roth [36] 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.

□

E. Relation to Differential Privacy

Our definition of certified unlearning shares interesting similarities with the concept of differential privacy [13] 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*

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

holds for all $\mathcal{T} \subset \Theta$ and datasets D, D' that differ in one sample. That is, we consider $|D| = |D'|$ where one sample has been replaced. This is denoted as "bounded differential privacy" in literature [50, 51].

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 no model updates are performed during unlearning.

Consequently, certified unlearning can be obtained through DP, yet the learning model's performance may suffer when enforcing strong privacy guarantees [see 12, 52]. In Section VII, 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, our approach to certified unlearning can be seen as a compromise between the high privacy guarantees of DP and the optimal performance achievable through costly re-training.

F. 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 calculating the inverse Hessian for

TABLE V: 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×
First-Order	1.1×10^4	0.04ms	38.0×	1.0×10^2	0.10ms	67.1×	1.0×10^2	4.87ms	603.7×
Second-Order	6.5×10^4	5.42s	0.3×	1.3×10^4	0.23ms	29.2×	7.8×10^4	0.03s	98.0×

each intermediate step, which increases the computational effort. An empirical comparison of our approach with multiple update steps is presented in Appendix G3.

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 [53–55] which states that applying an ϵ-DP algorithm n times results in a $n\epsilon$-DP algorithm.

G. Evaluation of Certified Unlearning

1) Fidelity evaluation with loss: In Section VII, we evaluate the fidelity of the retrained model with different approaches using the difference in test loss presented in a scatter plot. Figure 11 shows the plots for the Adult and Spam 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.

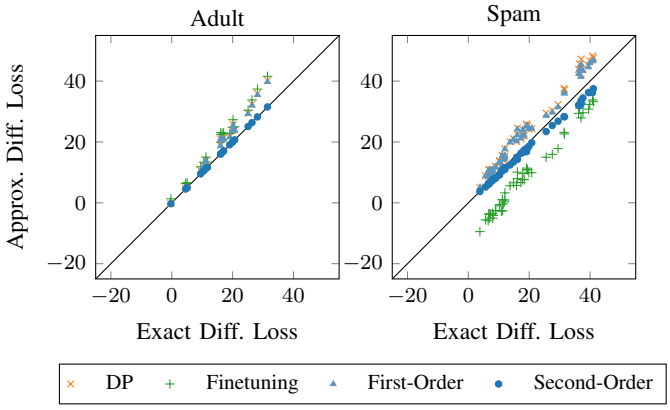

Fig. 11: Difference in test loss between retraining and unlearning when removing or changing random combinations of features. For perfect unlearning the results would lie on the identity line.

2) Efficiency evaluation: In Section VII, we present the runtime evaluation only for the Malware dataset. We show the evaluations for the remaining datasets in Table V. 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.

3) Sequential unlearning steps: So far, we have presented our approach as one-shot unlearning, that is, all perturbed samples are collected in the set $\tilde{Z}$ and the update is performed on one step. The theoretical analysis in Section VI, 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. To evaluate this setting, we repeat the previous experiment but now split the update into 10 steps of equal size.

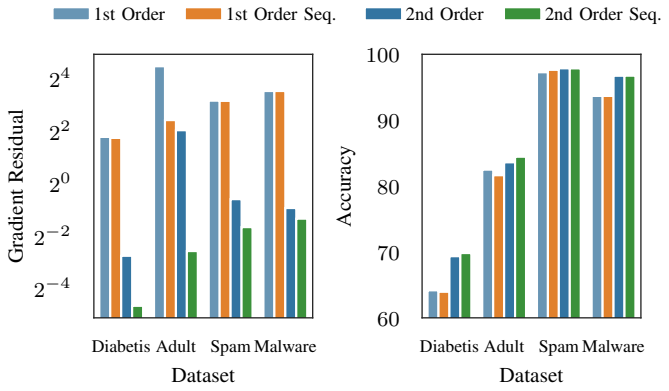

Fig. 12: Average accuracy of the corrected models (in %) when removing 100 features at once or sequentially in 10 steps.

Figure 12 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 Appendix F, this increase in privacy budget comes with an increase in runtime for the second-order

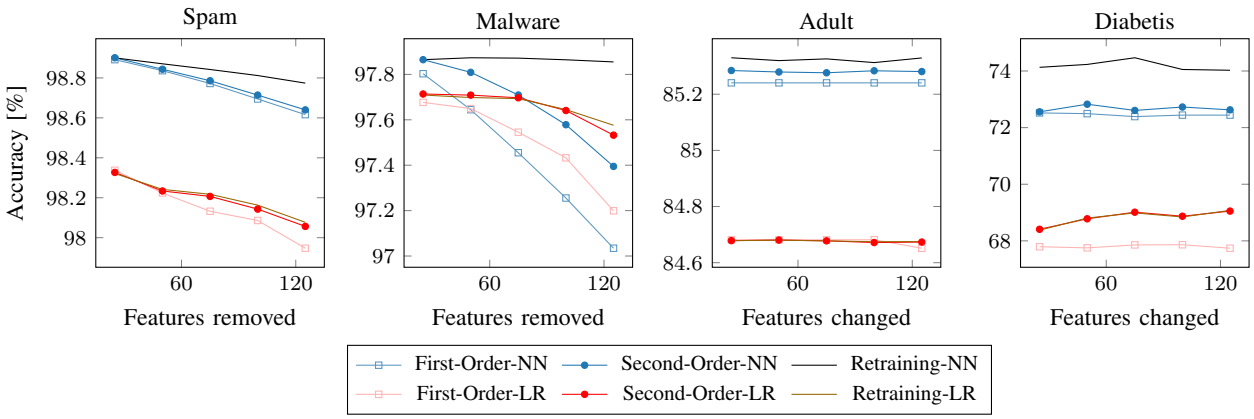

Fig. 13: Fidelity (accuracy) of neural network and logistic regression for varying number of affected features (higher values are better).

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.

H. Neural Networks in First Unlearning Scenario

Our first unlearning scenario in Section VII-A focuses on a logistic regression model. This learning model was chosen, because it provides a strongly convex loss function and thus allows the application of certified unlearning. However, linear models are inherently limited in their capabilities, so we also employ a neural network to the four datasets considered in Section VII-A for comparison. Note that neural networks generally do not have a convex loss and therefore do not provide theoretical guarantees for unlearning in our framework.

In particular, we train a fully connected neural network with two hidden layers consisting of 100 neurons for each of the datasets. We then remove and replace, respectively, sensitive features using unlearning as described in Section VII-A. As the loss of the network is not convex, we cannot use the gradient residual norm to determine its efficacy or calibrate the noise for certified unlearning, as done for the logistic regression. Consequently, we drop the noise term from the loss in this experiment for the neural network and the logistic regression. Still, we can investigate the accuracy after unlearning to get a rough reference for the general performance of our approach on neural networks in this scenario.

Figure 13 shows the accuracy of the neural network and the logistic regression on the four dataset when unlearning features. Both models perform very well in this scenario and enable to unlearn 120 features without significant changes of the accuracy. For both models, the accuracy drops by less than 1% point on all datasets when corrections are performed using our first-order and second-order update. This strong performance demonstrates the capability of our approach for unlearning with high fidelity. As we shown in Section VII-A, however, when theoretical guarantess are enforced, the decrease in accuracy is more pronounced as noise needs to be added to the model parameters to achieve certification.

We also find that the neural network has a higher accuracy compared to the logistic regression on all datasets. The non-linear network is capable of better modeling the underlying data and thus attains a more accurate prediction. However, the difference to the logistic regression is marginal and remains below 5%. For three of the four datasets (Spam, Malware, and Adult), it is even less than 1%. This result provides an important insight for the unlearning scenario: The logistic regression and the neural network achieve a comparable accuracy on the four datasets. Hence, if privacy guarantees are necessary and a small degradation in performance can be tolerated, the logistic regression model is actually preferable to the neural network, despite its inherent limitations.

I. Poisoning Experiment Details

In addition to the description in Section VII-C, we provide more details about the dataset, CNN architecture and used unlearning parameters in this section.

The CIFAR10 dataset [56] contains 50,000 images with $32 \times 32 \times 3$ 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 [57] with a learning rate of 1×10^{-4}.

For our approach, we use an unlearning rate τ of 2×10^{-5} for the first-order 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} [see 47]. The 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 computation of the inverse Hessian. Agarwal et al. [47] originally suggest to repeat the algorithm multiple times and average the results for a more stable solution but we find that a single repetition is sufficient in our experiments. Additionally, 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.