Provenance-based Model Maintenance: Implications for Privacy

Yinjun Wu, Val Tannen and Susan B. Davidson
University of Pennsylvania
{wuyinjun, val, susan}@seas.upenn.edu

Abstract

With the increasing need for Machine-learning-as-a-service (MaaS) online systems, effectively maintaining and reusing machine learning models in light of changes to the underlying data has become a big concern. In particular, it is extremely challenging to refresh existing models after the removal of training samples, which is called "machine unlearning". Addressing this challenge not only requires an efficient solution, but must comply with emerging privacy issues, e.g. GDPR, which implies that the removed samples must be fully erased from the models so that they cannot be leaked to an adversary. We review two provenance-based solutions, PrIU and DeltaGrad, and show how they can guard against "model inversion attacks", which reconstruct the removed training samples from the updated models after the unlearning process. Since PrIU and DeltaGrad support a limited class of models, we envision a system that can unlearn general *models in an efficient and secure manner and outline possible technical challenges for building this system.*

1 Introduction

The problem of incrementally updating model parameters after the deletion of a small set of training samples has attracted increasing attention in machine learning over the past few years. It arises in applications such as refreshing model parameters after sensitive training samples are removed (the *GDPR* issue [1]), reducing bias in statistics [2], and quantifying uncertainty [3].

It is also used for quantifying the importance of a training sample using measures such as the *Data Shapley value* [4]. A key step in evaluating this type of measure is to *remove* a subset of training samples and calculate the updated ML model parameters. The most straightforward way to do this is to reconstruct the ML model from scratch after the samples have been deleted. However, recalculating from scratch is prohibitively expensive, especially when the training data is frequently updated, and so the question is whether the model can be updated in real time.

From the perspective of a database researcher, this problem seems very similar to the well-studied problem of *materialized view maintenance* [5, 6] (see Figure **??**). In materialized view maintenance, we have input relations over which a view is constructed using relational algebra operators (left side of the figure). In machine learning (right side of the figure), the analogy to input relations is the training data, and the operations forming the "view" (the model) is the learning algorithm. The question is whether techniques that have been developed for materialized view maintenance can be used for what we will call *model-maintenance*.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

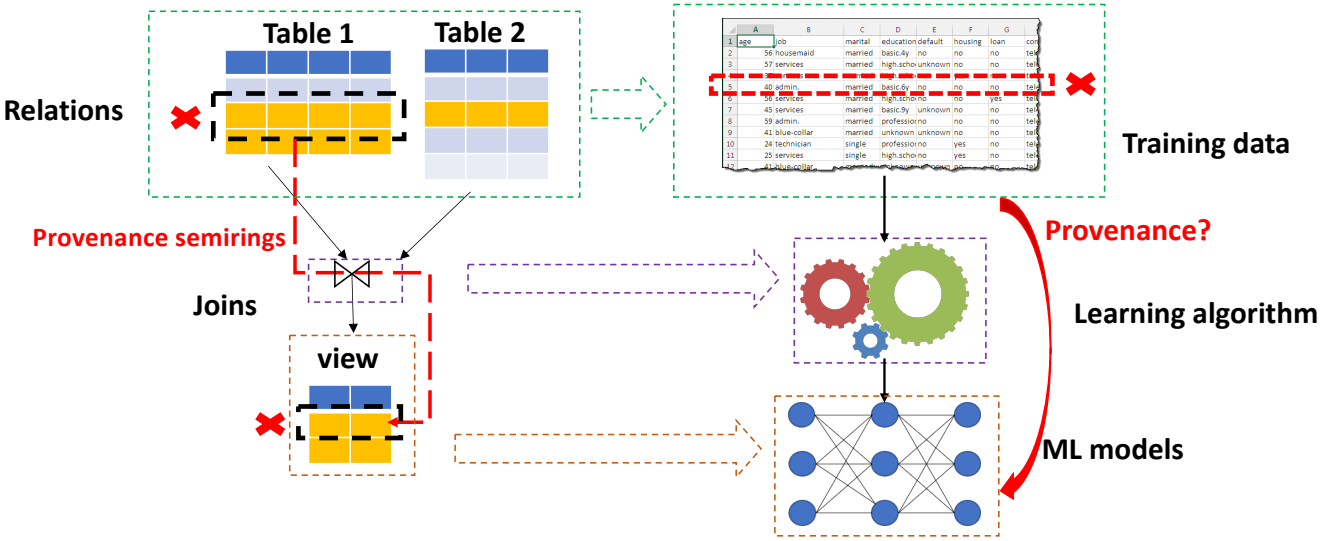

Figure 1: Parallels between materialized view maintenance and model maintenance

One very effective technique for materialized view maintenance is based on provenance, in particular the *provenance semiring* framework [7]. In this setting, by tracing the use of input tuples for each tuple in the view using *provenance polynomials*, the deletion of an input tuple can be propagated to a view tuple by seeing how and if it is used in the view tuple's provenance polynomial.

Of course, a machine learning algorithm is much more complicated than a relational algebra expression. However, recent work has extended the provenance semiring framework to linear algebra operations [8], opening the door to using provenance to reason over ML algorithms based on those operations, such as *linear regression* and *logistic regression* in which non-linear operations are linearized using piecewise linear interpolation [9]. In this paper, we show how provenance can be used for incrementally updating machine learning models for linear or logistic regression in our PrIU system [9]. In particular we show how provenance information carried by the linear operations can be cached during the model training process and then reused for speeding up the model maintenance. For more complex models which use non-linear operations, provenance is not yet defined. However, building on the ideas used in PrIU of caching essential intermediate results, we therefore show how caching can be used to incrementally update more complex models (such as neural networks) in our DeltaGrad system [10].

We then explore the connection of our provenance-based technique to other *machine unlearning* techniques (e.g., [11]). A major concern in machine unlearning is that the unlearned model may suffer from *model inversion attacks* [12], in which the adversary is able to restore the deleted data items (a.k.a *private* data) from the resulting model either using just the model (a *black box* attack) or using auxiliary information such as the model type, model parameters, or even the remaining training samples (a *white box* attack). In contrast to other machine unlearning techniques, we show that a benefit of our provenance-based technique is that it can avoid such attacks with low overhead and without loss of prediction performance.

The remainder of this paper is organized as follows: In Section 2 we give background information on provenance semirings and deletion propagation. In Section 3 we discuss how to extend these ideas to incremental model maintenance for linear and logistic regression models in our system PrIU, and for more complex models in our system DeltaGrad. We then discuss the problem of model inversion attacks in Section 4, and show how our framework can be used to guard against them.

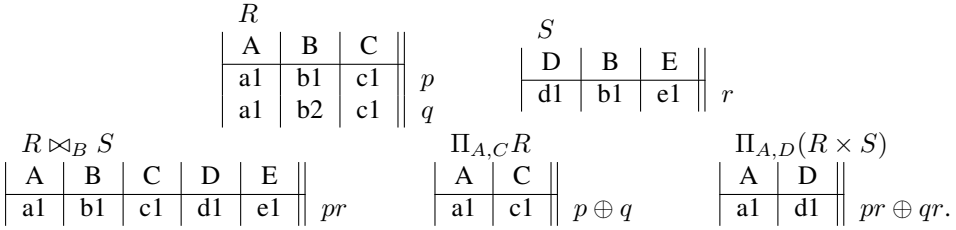

Figure 2: Provenance propagation

2 Background

We start by describing the provenance semiring framework in relational databases, and how provenance can be used to incrementally update views. We then discuss how the framework has been extended to include linear algebra operations

Provenance Semirings. In the provenance semiring framework [7], input data to a query is annotated with provenance tokens which are propagated through the algebraic operators performed on the data, e.g. select, project, join, union, and more recently aggregation [13]. As the provenance annotations propagate, they are combined using two abstract operations. The provenance operation $\otimes$ signifies *joint use* of data. For example, when tuple t_1, respectively t_2, are annotated with provenance p, respectively q, then the tuple obtained by joining t_1 and t_2 is annotated with provenance $p \otimes q$. The other provenance operation, $\oplus$, signifies *alternative use* of data. For example, when tuple t_1, respectively t_2, are annotated with provenance p, respectively q, such that t_1 and t_2 project to the same tuple t, then the resulting tuple t is annotated with provenance $p \oplus q$. In addition, an abstract "zero-polynomial" 0_{prov} is used to annotate *absent* tuples and an abstract "unit-polynomial" 1_{prov} is used to annotate "neutral" data whose provenance is ignored (it's of no interest in a specific analysis). If P is the mathematical space of provenance annotations, standard equivalences of positive relational algebra imply that $(P, \oplus, \otimes, 0_{prov}, 1_{prov})$ is a *commutative semiring* [7]. It follows that the expressions obtained as provenance is propagated through a query can be put in standard polynomial form, where the indeterminates (the variables) are provenance tokens that annotate the tuples in the data given as input to the query. Therefore, in the semiring framework provenance is captured by *provenance polynomials*.

Example 1: As an example, suppose we have two tables, R and S, whose tuples are annotated with provenance tokens p, q, and r, as shown in Figure 2. In the same figure we show the provenance polynomials in indeterminates p, q, r that are produced by provenance propagation in the outputs of queries $R \bowtie_B S$, $\Pi_{A,C}R$, and $\Pi_{A,D}(R \times S)$. As is customary in algebra, we omit the multiplication-like symbol $\otimes$ when we write polynomials in commutative indeterminates.

Deletion Propagation. One of the benefits of the semiring framework is that deletions of tuples in the input tuples propagate straightforwardly to deletions (or modifications of provenance) of output tuples by partially evaluating provenance polynomials using familiar algebraic rules.

When an input tuple is deleted, the effect on the output can be efficiently calculated by setting its token to 0_{prov} in the output's provenance polynomials, signifying absence. For example, if the first tuple in R, (a1, b1, c1) with provenance token p, were deleted, then the provenance of the (only) tuple in $R \bowtie_B S$ would become $0_{prov} \otimes r = 0_{prov}$, indicating that it no longer appears – intuitively, both tuples are needed for that tuple to be present. On the other hand, the tuple in $\Pi_{A,C}R$ would still appear but with a different provenance $0_{prov} \oplus q = q$ – intuitively, at least one of p and q is needed for the result to be present. Similarly, deleting the

first tuple in R will still leave the tuple (a1,d1) in the output of $\Pi_{A,D}(R \times S)$, but its provenance would become $(0_{prov} \otimes r) \oplus (q \otimes r) \ = \ 0_{prov} \oplus (q \otimes r) \ = \ qr$.

Extension to Linear Algebra Operations Recently, the semiring framework has been extended to include basic linear algebra operations: matrix addition and multiplication [8]. In this extension, the provenance polynomials play the role of abstract scalars and an abstract version of multiplication with scalars plays the role of annotating matrices (in particular, vectors) with provenance. Matrices form a non-commutative, many-sorted ring. The structure obtained by combining the ring of matrices with multiplication by scalars from the semiring of provenance polynomials is a *semialgebra* [8]. This framework is flexible enough that for input matrices we can annotate both rows (samples) and columns (features) with arbitrary provenance tokens.

As an example, suppose that p,q,r,s are provenance tokens that annotate four different samples in a training dataset. We denote the multiplication with scalars (i.e., the annotation with provenance polynomias) by the abstract operation $*$. Using the work in [8], our methods will show that vectors of interest (such as the vector of model parameters) can be expressed with provenance annotated expressions such as:

$$\mathbf{w} = (p^2 q * \mathbf{u}) + (qr^4 * \mathbf{v}) + (ps * \mathbf{z}) \tag{1}$$

Here, **u**,**v**,**z** are numerical vectors signifying contributions to the answer **w** and they are annotated with the provenance polynomials p^2q, qr^4, ps.

Now suppose the sample (input row) annotated with r is deleted while those annotated p,q,and s are retained but we decide not to track them anymore. As we did in the paragraph on deletion propagation, we can express the updated value of **w** under this deletion by setting r to 0_{prov} which signifies absence, and p,q,s to 1_{prov}, which signifies "neutral" presence in Equation (1). Again, the resulting expressions can be simplified using familiar algebraic manipulations. As expected, $0_{prov} \otimes r^4 = 0_{prov}$ as well as $0_{prov} * \mathbf{v} = \mathbf{0}$ (the all-zero vector). Moreover, $1_{prov} \otimes 1_{prov} = 1_{prov}$ and $1_{prov} * \mathbf{z} = \ \mathbf{z}$. It follows that under this deletion $\mathbf{w} = \ \mathbf{u} + \mathbf{z}$.

3 Overview of PrIU and DeltaGrad

We start with preliminaries on Stochastic Gradient Descent (SGD) before showing the explicit use of provenance in PrIU and the implicit use of provenance in DeltaGrad for incrementally updating machine learning models in DeltaGrad.

3.1 Preliminaries on SGD

By assuming that SGD is used for model training, PrIU and DeltaGrad can incrementally update the "gradients" at each SGD step. Suppose the training dataset is $D_{\text{train}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ and model parameter is **w**, at the step t of SGD **w** is computed by evaluating the gradients on a randomly sampled mini-batch of the training dataset, i.e.:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \cdot \text{Grad}(\mathbf{w}_t; \mathscr{B}_t), \tag{2}$$

where $\text{Grad}(\mathbf{w}_t; \mathscr{B}_t)$ represents the gradient evaluated on a mini-batch $\mathscr{B}_t$.

Suppose a subset of training samples, R, is removed from the training dataset. Then to compute the updated model parameter, Equation (2) is modified as:

$$\mathbf{w}_{t+1}^U = \mathbf{w}_t^U - \eta_t \cdot \text{Grad}(\mathbf{w}_t; \mathscr{B}_t - R) \tag{3}$$

where $\mathbf{w}^U$ represents the updated model parameter and $\mathscr{B}_t - R$ represents the remaining samples in $\mathscr{B}_t$ after the removal of R.

Note that both Grad$(\mathbf{w}_t; \mathscr{B}_t)$ in Equation (2) and Grad$(\mathbf{w}_t; \mathscr{B}_t - R)$ in Equation (3) can be regarded as the integration of two components: the model parameter $\mathbf{w}_t$ (or $\mathbf{w}_t^U$) and the data part (the sample $(\mathbf{x}_i, y_i)$). For example, for linear regression models, the loss function $F(\mathbf{w}_t; (\mathbf{x}_i, y_i))$ using L2 regularization is

$$F(\mathbf{w}_t, (\mathbf{x}_i, y_i)) = (y_i - \mathbf{x}_i^\top \mathbf{w})^2 + \frac{\lambda}{2}\|\mathbf{w}_t\|^2,$$

Using this, Grad$(\mathbf{w}_t; \mathscr{B}_t)$ and Grad$(\mathbf{w}_t; \mathscr{B}_t - R)$ can be rewritten as:

$$\text{Grad}(\mathbf{w}_t; \mathscr{B}_t) = (\lambda \mathbf{I} + \frac{2}{|\mathscr{B}_t|}\sum\nolimits_{i \in \mathscr{B}_t} \mathbf{x}_i \mathbf{x}_i^\top)\mathbf{w}_t - \frac{2}{|\mathscr{B}_t|}\sum\nolimits_{i \in \mathscr{B}_t} \mathbf{x}_i y_i, \tag{4}$$

$$\text{Grad}(\mathbf{w}_t; \mathscr{B}_t - R) = (\lambda \mathbf{I} + \frac{2}{|\mathscr{B}_t - R|}\sum\nolimits_{i \in \mathscr{B}_t - R} \mathbf{x}_i \mathbf{x}_i^\top)\mathbf{w}_t^U - \frac{2}{|\mathscr{B}_t - R|}\sum\nolimits_{i \in \mathscr{B}_t - R} \mathbf{x}_i y_i \tag{5}$$

The data part in Grad$(\mathbf{w}_t; \mathscr{B}_t)$ (Equation (4)) consists of two terms:

$$D_1(\mathscr{B}_t) = \sum\nolimits_{i \in \mathscr{B}_t} \mathbf{x}_i \mathbf{x}_i^\top,\ D_2(\mathscr{B}_t) = \sum\nolimits_{i \in \mathscr{B}_t} \mathbf{x}_i y_i.$$

Similarly, the data part in Grad$(\mathbf{w}_t; \mathscr{B}_t - R)$ (Equation (5) can be expressed as:

$$D_1(\mathscr{B}_t - R) = \sum\nolimits_{i \in \mathscr{B}_t - R} \mathbf{x}_i \mathbf{x}_i^\top,\ D_2(\mathscr{B}_t - R) = \sum\nolimits_{i \in \mathscr{B}_t - R} \mathbf{x}_i y_i.$$

3.2 PrIU

As described above, by decomposing the gradient formulas Grad$(\mathbf{w}_t; \mathscr{B}_t)$ or Grad$(\mathbf{w}_t; \mathscr{B}_t - R)$ into a data part and a model parameter part, we can *capture the provenance of the data part to track how the changes of the data part lead to the updates of the model parameters at each SGD iteration*.

Specifically, for the above example, suppose each training sample $(\mathbf{x}_i, y_i)$ is given a unique provenance token, p_i. Then we can generate the following provenance-aware formula for $D_1(\mathscr{B}_t)$ and $D_2(\mathscr{B}_t)$:

$$\text{Prov}(D_1(\mathscr{B}_t)) = \sum\nolimits_{i \in \mathscr{B}_t} p_i^2 * \mathbf{x}_i \mathbf{x}_i^\top,\ \text{Prov}(D_2(\mathscr{B}_t)) = \sum\nolimits_{i \in \mathscr{B}_t} p_i^2 * \mathbf{x}_i y_i$$

To obtain the values of $D_1(\mathscr{B}_t - R)$ and $D_2(\mathscr{B}_t - R)$, we can set the provenance token p_i to 0_{prov} for each $i \in R$ to zero out the removed training samples, and set the other provenance tokens to 1_{prov}, i.e.:

$$D_1(\mathscr{B}_t - R) = \sum\nolimits_{i \in \mathscr{B}_t - R} \mathbf{x}_i \mathbf{x}_i^\top = [\sum\nolimits_{i \in \mathscr{B}_t - R} p_i^2 * \mathbf{x}_i \mathbf{x}_i^\top]_{p_i = 1_{prov}} + [\sum\nolimits_{i \in \mathscr{B}_t \bigcap R} p_i^2 * \mathbf{x}_i \mathbf{x}_i^\top]_{p_i = 0_{prov}}$$

$$D_2(\mathscr{B}_t - R) = \sum\nolimits_{i \in \mathscr{B}_t - R} \mathbf{x}_i y_i = [\sum\nolimits_{i \in \mathscr{B}_t - R} p_i^2 * \mathbf{x}_i y_i]_{p_i = 1_{prov}} + [\sum\nolimits_{i \in \mathscr{B}_t \bigcap R} p_i^2 * \mathbf{x}_i y_i]_{p_i = 0_{prov}}$$

This can be implemented by reusing the cached terms, $D_1(\mathscr{B}_t)$ and $D_2(\mathscr{B}_t)$ and subtracting the terms corresponding to the removed samples in R, i.e.:

$$D_1(\mathscr{B}_t - R) = \sum\nolimits_{i \in \mathscr{B}_t - R} \mathbf{x}_i \mathbf{x}_i^\top = D_1(\mathscr{B}_t) - \sum\nolimits_{i \in \mathscr{B}_t \bigcap R} \mathbf{x}_i \mathbf{x}_i^\top$$

$$D_2(\mathscr{B}_t - R) = \sum\nolimits_{i \in \mathscr{B}_t - R} \mathbf{x}_i y_i = D_2(\mathscr{B}_t) - \sum\nolimits_{i \in \mathscr{B}_t \bigcap R} \mathbf{x}_i y_i$$

This is considerably more efficient than recomputing $D_1(\mathscr{B}_t - R)$ and $D_2(\mathscr{B}_t - R)$ from scratch if the size of R is much smaller than the total number of training samples.

So far, we have discussed how to efficiently update Grad($\mathbf{w}_t; \mathscr{B}_t$) to Grad($\mathbf{w}_t; \mathscr{B}_t - R$) after the removal of a training sample set for SGD formulas where the data part and the model parameters are *linearly combined*. However, in general cases the data part and the model parameters in Grad($\mathbf{w}_t; \mathscr{B}_t$) can be integrated in an arbitrary way, making it difficult to use provenance. For example, for *logistic regression with L2 regularization* the loss function, $F(\cdot)$, and Grad($\mathbf{w}_t; \mathscr{B}_t$), can be instantiated as:

$$F\left(\mathbf{w}_t, (\mathbf{x}_i, y_i)\right) = \ln(1 + \exp\{-y_i \mathbf{w}_t^\top \mathbf{x}_i\}) + \frac{\lambda}{2}\|\mathbf{w}_t\|^2 \tag{6}$$

$$\text{Grad}(\mathbf{w}_t; \mathscr{B}_t) = \lambda \mathbf{w}_t - \frac{1}{|\mathscr{B}_t|} \sum\nolimits_{i \in \mathscr{B}_t} y_i \mathbf{x}_i (1 - \frac{1}{1 + \exp\{-y_i \mathbf{w}_t^\top \mathbf{x}_i\}}) \tag{7}$$

in which the model parameter $\mathbf{w}_t$ and the data part are combined with non-linear operations (i.e., $\exp$ and division). To be able to use provenance for rapid updates, we therefore linearize the non-linear operations in Equation (7) using *piecewise linear interpolation*, which leads to the following "Linearized Grad($\mathbf{w}_t; \mathscr{B}_t$)":

$$\text{Linearized Grad}(\mathbf{w}_t; \mathscr{B}_t) = \lambda \mathbf{w}_t - \frac{1}{|\mathscr{B}_t|} (\sum\nolimits_{i \in \mathscr{B}_t} a_{i,t} \mathbf{x}_i \mathbf{x}_i^\top \mathbf{w}_t + \sum\nolimits_{i \in \mathscr{B}_t} b_{i,t} y_i \mathbf{x}_i) \tag{8}$$

in which the two coefficients, $a_{i,t}$ and $b_{i,t}$, are generated after piece-wise linear interpolation is applied. As a consequence, Equation (8) shares a similar form with Equation (4), hence data provenance can also be used in Equation (8) for incrementally updating logistic regression models. Note that due to the linearization operations, the incrementally updated model parameters may not be the same as the ones retrained from scratch. However, we can prove that the difference between the two model parameters (quantified by L2 norm) is very small (see detailed analysis in [9]).

3.3 DeltaGrad

General machine learning models such as neural networks can be arbitrarily complex, making it challenging to separate the data part from the model parameters in the gradient expression. Therefore, instead of *explicitly* employing data provenance as in PrIU we proposed a second method, DeltaGrad, which updates model parameters by *implicitly* leveraging data provenance.

We start by rewriting the SGD update rule in Equation (3) as follows:

$$\mathbf{w}_{t+1}^U = \mathbf{w}_t^U - \eta_t \cdot \text{Grad}(\mathbf{w}_t^U; \mathscr{B}_t - R) = \mathbf{w}_t^U - \eta_t \cdot [\text{Grad}(\mathbf{w}_t^U; \mathscr{B}_t) - \text{Grad}(\mathbf{w}_t^U; R)] \tag{9}$$

In this update rule, after the removal of R instead of directly evaluating the gradient on the remaining samples in $\mathscr{B}_t$, i.e., $\mathscr{B}_t - R$, we subtract the gradient on the removed samples (i.e., $\mathscr{B}_t \bigcap R$) from the gradient on the full mini-batch (i.e., the term Grad($\mathbf{w}_t^U; \mathscr{B}_t$)). It is worth noting that Grad($\mathbf{w}_t^U; \mathscr{B}_t$) has the same form as Grad($\mathbf{w}_t; \mathscr{B}_t$) in Equation (2) except for the differences on the dependent model parameters (i.e. $\mathbf{w}_t$ and $\mathbf{w}_t^U$ resp.). We therefore *cache the term, Grad($\boldsymbol{w}_t; \mathscr{B}_t$) during the model training phase*, i.e. the evaluation of Equation (2), so that it can be reused for accelerating the evaluation of Grad($\mathbf{w}_t^U; \mathscr{B}_t$).

Specifically, we estimate Grad($\mathbf{w}_t^U; \mathscr{B}_t$) by estimating the difference between Grad($\mathbf{w}_t; \mathscr{B}_t$) and Grad($\mathbf{w}_t^U; \mathscr{B}_t$), which can be computed using the Cauchy mean value theorem:

$$\text{Grad}(\mathbf{w}_t^U; \mathscr{B}_t) - \text{Grad}(\mathbf{w}_t; \mathscr{B}_t) = \mathbf{H}([\mathbf{w}_t, \mathbf{w}_t^U]; \mathscr{B}_t) \left(\mathbf{w}_t^U - \mathbf{w}_t\right)$$

where $\mathbf{H}([\mathbf{w}_t, \mathbf{w}_t^U]; \mathscr{B}_t)$ represents the Hessian matrix integrated between $\mathbf{w}_t$ and $\mathbf{w}_t^U$ given a mini-batch $\mathscr{B}_t$. Since the explicit evaluation of the Hessian matrix is extremely time-consuming, we adapt the L-BFGS algorithm [14] to approximately evaluate the Hessian-vector product $\mathbf{H}([\mathbf{w}_t, \mathbf{w}_t^U]; \mathscr{B}_t) \left(\mathbf{w}_t^U - \mathbf{w}_t\right)$. In this modified version of the L-BFGS algorithm, the input consists of the vector $(\mathbf{w}_t^U - \mathbf{w}_t)$, the history model parameters, $\mathbf{w}$ and $\mathbf{w}^U$

from previous SGD iterations, as well as the corresponding gradients (i.e., $\text{Grad}(\mathbf{w}_t; \mathscr{B}_t)$ and $\text{Grad}(\mathbf{w}_t^U; \mathscr{B}_t)$) from those iterations. As a result, $\text{Grad}(\mathbf{w}_t^U; \mathscr{B}_t)$ is approximately evaluated with the following formula:

$$\text{Grad}(\mathbf{w}_t^U; \mathscr{B}_t) \approx \text{Grad}(\mathbf{w}_t; \mathscr{B}_t) + g^{\text{L-BFGS}}(\mathbf{w}_t^U - \mathbf{w}_t, \{(\mathbf{w}_{t_r}^U, \mathbf{w}_{t_r}, \text{Grad}(\mathbf{w}_{t_r}^U; \mathscr{B}_{t_r}), \text{Grad}(\mathbf{w}_{t_r}; \mathscr{B}_{t_r}))\}_{r=1}^m) \quad (10)$$

in which $g^{\text{L-BFGS}}(\cdot)$ denotes the L-BFGS algorithm and the history model parameters, $\mathbf{w}_{t_r}$, and the corresponding gradients, $\text{Grad}(\mathbf{w}_{t_r}; \mathscr{B}_{t_r})$, are regarded as the "implicit" provenance.

Note that the computation of $\text{Grad}(\mathbf{w}_t^U; \mathscr{B}_t)$ can lead to approximation errors. Therefore, in order to guarantee that the updated model parameters calculated by this approximate gradient are not far away from the expected ones, we compute $\text{Grad}(\mathbf{w}_t^U; \mathscr{B}_t)$ from scratch in the first few SGD iterations and periodically compute it from scratch afterwards. It can be shown that the updated model parameters calculated in this way are very close to the ones retrained from scratch [10].

4 Security concern: Model inversion attacks

In the previous section, we showed how provenance can be used both explicitly (in the case of linear and logistic regression) and implicitly (in the case of more complex models such as neural networks) to incrementally update machine learning models. Going beyond incremental updates, we now discuss how a provenance-based framework can be used to defend against an emerging security concern: *model inversion attacks*.

4.1 Preliminaries

The recently established General Data Protection Regulation (GDPR) guidelines [1] state that users have the right to have private data items removed from the entities storing those items. However, this is not as simple as just deleting the data items: If they have been used as training data in state-of-the-art machine learning systems, the effect of these data items must also be "erased" from the models that these systems have learned and rely on. Otherwise, the systems may be subject to *model inversion attacks*, a type of attack in which the adversary is able to restore the private data items from the machine learning models without accessing the data items [12]. Therefore, the ultimate goal of *unlearning a machine learning model* is to give an updated model in which the private data items have been "forgotten", i.e. the model behaves as if the private data items never appeared in the training set thus safeguarding against model inversion attacks. Therefore, in addition to *efficiency* (i.e. the time to update the model) and *performance* guarantees (i.e. the predictive power of the updated model), an ideal unlearning algorithm should guard against model inversion attacks.

It is worth noting that, depending on the adversary's knowledge, the vulnerability of the model can vary. Generally speaking, there are two model inversion attack settings: *black-box* and *white-box* [12]. In the black-box setting, the adversary can only use the model output to launch the attack, whereas in the white-box setting the adversary can also use auxiliary information such as the model type, model parameters, or even the remaining training samples (in the extreme case). It is therefore much more challenging to defend against white-box attacks than black-box attacks.

4.2 Vulnerability of current machine unlearning methods

Current machine unlearning methods fall into one of several different categories: 1) Retraining-based methods; 2) Methods based on differential privacy; and 3) One-step update methods. We give an overview of each, and then compare them as well as our provenance-based approach with respect to *efficiency* (i.e. the time to update the model), *performance* (i.e. the predictive power of the updated model), and their ability to guard against model-inversion attacks. A summary of this comparison can be found in Table 2.

Retraining-based methods One straightforward machine unlearning strategy is to retrain the models from scratch, which can fully erase the removed training samples from the models and maintain the model prediction performance. However, this is very inefficient when frequent unlearning requests occur. To mitigate this, [11] proposes to shard the training set into smaller partitions, construct one local model for each partition and only retrain the local model if a deletion request hits the corresponding partition. Note that this method is not efficient if training samples from each partition are removed simultaneously, leading to the reconstruction of all the models.

Differential privacy-based methods. These methods, [15, 16], build on the classical notion of differential privacy [17]. The goal is to update the model in a single step, and then add some carefully designed random noise to the incrementally updated model so that it is indistinguishable from the one retrained from scratch, to which the same level of random noise has been added. In particular, [15] leverages the following Newton update mechanism for updating linear models (e.g., linear regression models and logistic regression models):

$$\mathbf{w}_*^U = \mathbf{w}_* + \mathbf{H}^{-1}(\mathbf{w}_*; D_{\text{remaining}}) \cdot \text{Grad}(\mathbf{w}_*; R), \tag{11}$$

in which $\mathbf{H}^{-1}(\mathbf{w}_*; D_{\text{remaining}})$ represents the inversed Hessian matrix (i.e. the second order gradient) on the remaining training set, $D_{\text{remaining}}$. Then $\mathbf{w}_*^U$ is perturbed with some randomly drawn noise vector $\mathbf{b}$ to hide the gradient information of the removed training samples. Despite the efficiency and perfect privacy guarantees of this type of solution, they suffer from the fact that the added noise may hurt the model prediction performance [18, 19].

One-step update methods. The second type of method incrementally updates the model in one-step but does not introduce extra noise [20, 18]). Specifically, these solutions can be represented by the following abstract formula:

$$\mathbf{w}_*^U = \mathbf{w}_* + G(R, D_{\text{remaining}}) \tag{12}$$

in which G is a function taking the removed training set R and the remaining training set $D_{\text{remaining}}$ as arguments. For example, Equation (12) could be the one-step Newton update (Equation (11)) in which the function G could be expressed as:

$$G(R, D_{\text{remaining}}) = \mathbf{H}^{-1}(\mathbf{w}_*; D_{\text{remaining}}) \cdot \text{Grad}(\mathbf{w}_*; R)$$

As observed in [21], the product between the inverse of the Hessian matrix, $\mathbf{H}^{-1}(\mathbf{w}_*; D_{\text{remaining}})$, and the vector $\nabla F(\mathbf{w}_*, R)$ in the above formula can be effectively evaluated using conjugate gradients [22] or the stochastic estimation method of [23]. To further speed up the above computations, by leveraging the fact that R is far smaller than $D_{\text{remaining}}$, $\mathbf{H}^{-1}(\mathbf{w}_*; D_{\text{remaining}})$ could be regarded as the low-rank updates on $\mathbf{H}^{-1}(\mathbf{w}_*; D_{\text{remaining}} + R)$, which is the inverse of the Hessian matrix on the full training set and thus can be cached beforehand. According to [18], such low-rank updates could be effectively computed by employing the Sherman-Morrison-Woodbury formula [24]. Note that the models incrementally updated in this manner are also very close to the retrained ones [21].

Despite the efficiency and predictive performance, this type of method suffers from model inversion attacks. In what follows, we describe at least two scenarios in which in which model inversion attacks can occur.

Scenario 1 Consider the following extreme scenario where everything about the model except for the removed sample set R is revealed to the adversary, which includes the remaining training samples $D_{\text{remaining}}$, the original model parameter, $\mathbf{w}_*$ and the incrementally updated model parameter, $\mathbf{w}_*^U$). However, in this case, the value of $G(R, D_{\text{remaining}})$ could be obtained by calculating the difference between $\mathbf{w}_*$ and $\mathbf{w}_*^U$, and thus R could be reconstructed by solving the following optimization problem:

$$\text{argmin}_{R'} \|G(R', D_{\text{remaining}}) - G(R, D_{\text{remaining}})\| = \text{argmin}_{R'} \|G(R', D_{\text{remaining}}) - (\mathbf{w}_*^U - \mathbf{w}_*)\| \tag{13}$$

It is worth noting that this extreme scenario could indeed occur in practice. First of all, different versions of models could be accessed by the adversary simultaneously. For example, machine learning models are increasingly used inside DBMSs, e.g., learned indexes [25] and learning-based query optimizers [26]. The models are typically constructed by taking the data in the DBMS as the training dataset. Due to updates on the underlying data, the models could also be updated, thus leaving different copies of the model snapshots, which could be logged inside the DBMS. Plus, due to the reproducibility requirements, the input training samples might also be stored for later use, which thus might be accessed by the adversary.

Scenario 2 In what follows, let us consider a more practical scenario where the knowledge of the adversary is limited. Specifically, we assume that only the updated model $\mathbf{w}_*^U$ is revealed to the adversary, which is similar to the standard assumption of the white-box attack. Then given $\mathbf{w}_*^U$, we assume that there is a strong white-box model inversion attack tool that the adversary could employ to reconstruct the remaining training samples, $D_{\text{remaining}}$ [12]. Note that in this scenario, the original model parameter $\mathbf{w}_*$ is not available to the adversary. Therefore, it is not enough to directly employ Equation (4.2) for reconstructing the removed training sample set, R. Instead, the adversary could **jointly** construct $\mathbf{w}_*$ during the derivation of R by leveraging the dependence of $\mathbf{w}_*$ on R (recall that $\mathbf{w}_*$ is trained on the full training set $D_{\text{remaining}} + R$). This could be formalized as the following bi-level optimization problem:

$$\begin{aligned} &R = \text{argmin}_{R'} \|G(R', D_{\text{remaining}}) - (\mathbf{w}_*^U - \mathbf{w}_*)\|, \\ &\text{where } \mathbf{w}_* = \min_{\mathbf{w}} \{F(\mathbf{w}; D_{\text{remaining}}) + F(\mathbf{w}; R')\} \end{aligned} \tag{14}$$

which could be effectively solved by using the optimization method proposed in [27].

Provenance-based approach. In contrast, both of our methods, PrIU and DeltaGrad, can resist the above scenarios. To see this, consider the following abstract formula representing how PrIU and DeltaGrad incrementally update the model:

$$\text{Grad}(\mathbf{w}_t^U; D_{\text{remaining}}) \approx \text{Prov}(\mathbf{w}_t^U; D_{\text{remaining}} + R) - \text{Grad}(\mathbf{w}_t^U; R) \tag{15}$$

in which,

$$\text{Prov}(\mathbf{w}_t^U; D_{\text{remaining}} + R) \approx \text{Grad}(\mathbf{w}_t^U; D_{\text{remaining}} + R)$$

Assuming that the the provenance term $\text{Prov}(\mathbf{w}_t^U; D_{\text{remaining}} + R)$ and the removed training sample set R are not accessible, then the adversary must solve the following formula to recover R:

$$\text{Grad}(\mathbf{w}_t^U; D_{\text{remaining}}) \approx \text{Grad}(\mathbf{w}_t^U; D_{\text{remaining}} + R) - \text{Grad}(\mathbf{w}_t^U; R) \tag{16}$$

which, however, holds for an arbitrary set of samples R' instead of simply holding for R. As a consequence, the best that the adversary can do is to randomly guess what R is.

Table 2: Comparison of state-of-the-art machine unlearning methods

	Privacy	Prediction performance	Efficiency
Retraining from scratch	✓	✓	
Partition-based Retraining [11]	✓	✓	
Differential-privacy-based methods	✓		✓
One-step-update methods		✓	✓
Provenance-based methods	✓	✓	✓

A summary of the comparison between current machine unlearning strategies can be found in Table 2. We next show how our provenance-based method can be used to avoid model-inversion attacks while achieving high prediction performance and efficiency.

4.3 Adding provenance support for secure machine unlearning

In future work, our goal is to develop a machine unlearning system that can achieve *real-time and secure updates on general machine learning models* such that the updated models are *almost identical to the retrained models*, thus not hurting the prediction power. The strategy to achieve this goal could vary depending on the threat model. However, we expect that in the worst scenario mentioned in Section 4.2, adding provenance support to the machine unlearning system is essential.

Specifically, we plan to generalize the idea of PrIU and DeltaGrad for general neural network models, such that the footprint of the removed training samples is erased from the model parameters collected across all the SGD iterations, and the updated model parameter at each SGD iteration is almost identical to the one for retraining the model from scratch. Given this, despite the knowledge of the adversary about the model and the remaining training samples, it would be almost impossible to reconstruct the removed training samples since the remaining information on the updated training trajectory does not include the removed training samples. This can therefore bring a better security guarantee than the One-step-update methods (as introduced in Section 4.2) in which the function G still encodes information about the removed samples, leading to the leakage of those samples.

Figure 3 illustrates how the envisioned provenance-enabled model unlearning system would work. In this figure there are two main components: the "Training loop component" and the "Updating loop component". In the "Training loop component" necessary provenance information is collected during the training process on the neural network models, where SGD is assumed to be the default training method (similar to PrIU and DeltaGrad). The dominant provenance information would be different versions of model snapshots (i.e., the model parameters and the computed gradients) captured at each iteration. Such provenance information would then be stored in secure storage, which would then be retrieved for incrementally computing the updated model in the "Updating loop component" after sensitive training data is deleted. As analyzed in the previous subsection, this could fully erase the footprint of the removed training samples from the model parameters at each SGD step, thus guarding against the model inversion attacks.

Discussion Note that there are several existing solutions that also aim to delete the footprint of the removed sensitive training samples from the entire training trajectory. For example, [28] proposes an efficient way to incrementally update K-means models through caching and reusing the information of all the clusters at each training iteration for model updates. Such cached cluster information could be therefore considered as provenance information. However, to facilitate effective unlearning, some necessary modifications are applied to the K-means models, which may degrade the model prediction performance.

Challenges and research opportunities Several challenges remain for developing a provenance-enabled machine unlearning system for general neural network models. First of all, general neural network models are non-convex, which is beyond the model classes that PrIU or DeltaGrad support. Therefore, we would envision that a new provenance-based unlearning method is necessary to support incremental updates on general non-convex neural network models. Inspired by PrIU and DeltaGrad, the models incrementally updated in this way could be approximately close to the retrained models, but with rigorous theoretical guarantees on the smallness of the approximation errors. One potential idea to design this new unlearning method is to relax the strong-convexity assumption on the model class in DeltaGrad such that general non-convex models could be handled. The main bottleneck is the strong dependency of DeltaGrad on the L-BFGS algorithm, in which the strongly convex objective functions are essential. We notice that this assumption has been relaxed in many extended versions of the L-BFGS algorithm (e.g., [29]), which could be potentially adapted for extending DeltaGrad.

In addition, due to the high complexity of state-of-the-art neural neural networks, the model snapshots at each SGD iteration could be extremely large, incurring a prohibitively high overhead for the entire unlearning system. For example, the ResNet18 network for vision tasks has around 11 million parameters [30] while the GPT-3 model [31] for natural language processing tasks has around 175 billion parameters. The problem would become

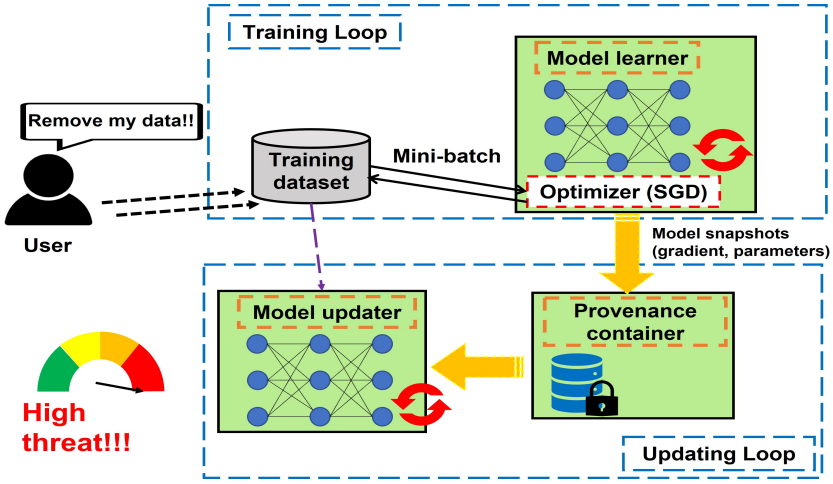

Figure 3: Provenance-enabled model unlearning system

even worse if their parameters and gradients were required to be repetitively recorded in the provenance-enabled model unlearning system. As a consequence, it is essential to build a version control system on large neural network models such that the model snapshots could be effectively captured, stored and retrieved.

One expected characteristic of such a version control system is that the models from different versions (i.e. from different SGD iterations) could be compressed to reduce the storage overhead. Ideally, the compression should be lossless so that the variations between different versions can be captured without adding extra approximation errors after those models are uncompressed during the model update phase. Otherwise, it could potentially amplify the approximation errors brought by the unlearning algorithm itself, thus hurting the quality of the unlearned models.

The version control problem for machine learning models has been recently studied in [32]. Specifically, the difference between the models from different SGD iterations is calculated first, which can be represented by a set of matrices. Then each element of each matrix, i.e., one float number, is approximately represented with k-byte (k=16 or 8) integers through the quantization operations. This is then followed by compressing the higher-order bits of the quantized representations across different versions of the models throughout the SGD iterations, which can yield significant savings (see the experiments of [32]). However, since the quantization operations can produce approximation errors, as discussed above, when this solution is applied to the machine unlearning pipeline such errors may lead to significant deviations of the incrementally updated models from the retrained ones.

Another requirement for the provenance-enabled model unlearning system, as shown in Figure 3, is that the collected provenance information be cached in secure storage. Otherwise, the adversary could easily reconstruct the removed training samples from the cached provenance of those samples by using the attack paradigm presented in Section 4.2. Finally, it is worth noting that the chance of the the security threat mentioned in Section 4.2 actually occuring may be quite low in practice due to the limited knowledge of the adversary on the models. For instance, in typical online systems, the deployed machine learning models are released as an open API [1] where the adversary only has black-box access, meaning that only the model output given one input sample can be obtained through the API. In this scenario, it might be inappropriate to use provenance-based unlearning methods due to their relatively high overhead with respect to other unlearning methods. It would be interesting to explore the applicability of existing unlearning methods under different threat assumptions and rank them based on their risk of leaking the removed training samples.

[1] see e.g., Google prediction API: https://cloud.google.com/ai-platform/prediction/docs/reference/rest/v1/projects/predict

5 Conclusions

In this paper, we reviewed our provenance-based techniques, PrIU and DeltaGrad, for incrementally updating machine learning models and showed the connection to incrementally updating database views. We then studied the privacy implications of machine unlearning techniques, and analyzed the capability of PrIU and DeltaGrad as well as other state-of-the-art unlearning techniques on defending against the model inversion attack. Our analysis reveals that provenance is essential for the unlearning process to guard against this type of attack without hurting performance and the model prediction power. Based on this observation, we envision a provenance-based unlearning system, which could effectively unlearn general machine learning models in a secure manner. We also outlined critical technical challenges and potential solutions, paving the way towards building such systems.

Acknowledgements

This material is based upon work that is in part supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0047. Partial support was provided by NSF Awards 1547360 and 1733794. Tannen's work at the National University of Singapore was supported in part by the Kwan Im Thong Hood Cho Temple/Avalokiteśvara.

References

[1] P. Voigt, and A. Von dem Bussche The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

[2] M. H. Quenouille, Notes on bias in estimation. *Biometrika*, 43 (3/4), pp.353-360, 1956.

[3] D. N. Politis, and J. P. Romano and M. Wolf Subsampling. *Springer Science & Business Media*, 1999.

[4] A. Ghorbani, and J. Zou. Data shapley: Equitable valuation of data for machine learning. *In International Conference on Machine Learning*, pp. 2242-2251. PMLR, 2019.

[5] A. Gupta, and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2), pp.3-18., 1995

[6] A. Labrinidis, and Y. Sismanis. View Maintenance, pages 3326–3328. *Springer*, Boston, MA, 2009.

[7] T. J. Green, G. Karvounarakis, and V. Tannen, Provenance semirings. *In Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* pp. 31-40, 2007.

[8] Z. Yan, V. Tannen, and Z. G. Ives. Fine-grained Provenance for Linear Algebra Operators. In *TaPP*. 2016.

[9] Y. Wu, V. Tannen, and S. B. Davidson. Priu: A provenance-based approach for incrementally updating regression models. *In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 447-462. 2020.

[10] Y. Wu, E. Dobriban, and S. B. Davidson. DeltaGrad: Rapid retraining of machine learning models. *In International Conference on Machine Learning*, pp. 10355-10366. PMLR, 2020.

[11] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot. Machine unlearning. *In 2021 IEEE Symposium on Security and Privacy (SP)*, pp. 141-159, 2021.

[12] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. *In Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pp. 1322-1333. 2015.

[13] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. *In Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 153-164. 2011.

[14] A. Mokhtari, and A. Ribeiro. Global convergence of online limited memory BFGS. *The Journal of Machine Learning Research* 16, no. 1 (2015): 3151-3181.

[15] C. Guo, T. Goldstein, A. Hannun, and L. Van Der Maaten.Certified Data Removal from Machine Learning Models. *In International Conference on Machine Learning*, pp. 3832-3842. PMLR, 2020.

[16] S. Neel, A. Roth, and S. Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning. *In Algorithmic Learning Theory*, pp. 931-962. PMLR, 2021.

[17] C. Dwork, and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* 9, no. 3-4 (2014): 211-407.

[18] Z. Izzo, M. A. Smart, K. Chaudhuri, and J. Zou. Approximate data deletion from machine learning models. *In International Conference on Artificial Intelligence and Statistics*, pp. 2008-2016. PMLR, 2021.

[19] 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 Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 363-375. 2020.

[20] A. Golatkar, A. Achille, and S. Soatto. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. *In European Conference on Computer Vision*, pp. 383-398. Springer, Cham, 2020.

[21] P. W. Koh, and P. Liang. Understanding black-box predictions via influence functions. *In International Conference on Machine Learning*, pp. 1885-1894. PMLR, 2017.

[22] J. Martens.Deep learning via hessian-free optimization. In *ICML*, vol. 27, pp. 735-742. 2010.

[23] N. Agarwal, B. Bullins, and E. Hazan. Second-order stochastic optimization in linear time. *stat* 1050 (2016): 15.

[24] J. Sherman, and W. J. Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics* 21, no. 1 (1950): 124-127.

[25] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. *In Proceedings of the 2018 International Conference on Management of Data*, pp. 489-504. 2018.

[26] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul. Neo: A Learned Query Optimizer. *Proceedings of the VLDB Endowment* 12, no. 11.

[27] J. Shu, Q. Xie, L. Yi, Q. Zhao, S. Zhou, Z. Xu, and D. Meng. Meta-Weight-Net: Learning an Explicit Mapping For Sample Weighting. *Advances in Neural Information Processing Systems* 32 (2019): 1919-1930.

[28] A. Ginart, M. Guan, G. Valiant, and J. Zou. Making AI Forget You: Data Deletion in Machine Learning. *Advances in neural information processing systems* (2019).

[29] D. Li, and M. Fukushima. A modified BFGS method and its global convergence in nonconvex minimization. *Journal of Computational and Applied Mathematics* 129, no. 1-2 (2001): 15-35.

[30] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

[31] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan et al. Language Models are Few-Shot Learners. *arXiv e-prints (2020)*: arXiv-2005.

[32] H. Miao, A. Li, L. S. Davis, and A. Deshpande. Towards unified data and lifecycle management for deep learning. *In 2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pp. 571-582. IEEE, 2017.