Machine Unlearning in Learned Databases: An Experimental Analysis

MEGHDAD KURMANJI, University of Warwick, UK
ELENI TRIANTAFILLOU, Google DeepMind, UK
PETER TRIANTAFILLOU, University of Warwick, UK

Machine learning models based on neural networks (NNs) are enjoying ever-increasing attention in the Database (DB) community, both in research and practice. However, an important issue has been largely overlooked, namely the challenge of dealing with the inherent, highly dynamic nature of DBs, where data updates are fundamental, highly-frequent operations (unlike, for instance, in ML classification tasks). Although some recent research has addressed the issues of maintaining updated NN models in the presence of new data insertions, the effects of data deletions (a.k.a., "machine unlearning") remain a blind spot. With this work, for the first time to our knowledge, we pose and answer the following key questions: What is the effect of unlearning algorithms on NN-based DB models? How do these effects translate to effects on key downstream DB tasks, such as cardinality/selectivity estimation (SE), approximate query processing (AQP), data generation (DG), and upstream tasks like data classification (DC)? What metrics should we use to assess the impact and efficacy of unlearning algorithms in learned DBs? Is the problem of (and solutions for) machine unlearning in DBs different from that of machine learning in DBs in the face of data insertions? Is the problem of (and solutions for) machine unlearning for DBs different from unlearning in the ML literature? what are the overhead and efficiency of unlearning algorithms (versus the naive solution of retraining from scratch)? What is the sensitivity of unlearning on batching delete operations (in order to reduce model updating overheads)? If we have a suitable unlearning algorithm (forgetting old knowledge), can we combine it with an algorithm handling data insertions (new knowledge) en route to solving the general adaptability/updatability requirement in learned DBs in the face of both data inserts and deletes? We answer these questions using a comprehensive set of experiments, various unlearning algorithms, a variety of downstream DB tasks (such as SE, AQP, and DG), and an upstream task (DC), each with different NNs, and using a variety of metrics (model-internal, and downstream-task specific) on a variety of real datasets, making this also a first key step towards a benchmark for learned DB unlearning.

CCS Concepts: • **Computing methodologies** → **Machine learning approaches**; • **Information systems** → *Data management systems*.

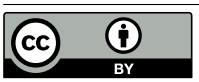
Additional Key Words and Phrases: Learned database systems, data deletions, machine unlearning

ACM Reference Format:
Meghdad Kurmanji, Eleni Triantafillou, and Peter Triantafillou. 2024. Machine Unlearning in Learned Databases: An Experimental Analysis. *Proc. ACM Manag. Data* 2, 1 (SIGMOD), Article 49 (February 2024), 26 pages. https://doi.org/10.1145/3639304

1 INTRODUCTION

ML has been recently enjoying increasing attention from the DB community. ML models are being developed to aid DB systems in performing better on a large variety of tasks, including

Authors' addresses: Meghdad Kurmanji, Meghdad.Kurmanji@warwick.ac.uk, University of Warwick, UK; Eleni Triantafillou, etriantafillou@google.com, Google DeepMind, UK; Peter Triantafillou, p.triantafillou@warwick.ac.uk, University of Warwick, UK.



ACM 2836-6573/2024/2-ART49
https://doi.org/10.1145/3639304

cardinality/selectivity estimation (SE) [15, 32, 44, 46, 48, 49, 53], Approximate Query Processing (AQP) [16, 27, 28, 39], query cost estimation [37, 52], learned indices [4, 5, 22, 31], Data Generation (DG) [3, 33, 47], workload forecasting [54], DB tuning [26, 43], etc.

One critical challenge learned DB systems are facing is adapting to the dynamic nature of database systems. Analytical DBs face frequent insertions, while transactional DBs face frequent insertions, deletions, and in-place updates (the latter can be modelled as a deletion followed by an insertion). The need for updatability/adaptability of the learned DB models is thus a fundamental requirement for learned DBs. Recently, this has started being addressed in-depth. Specifically, Li et al. [25] have introduced a method to update learned Cardinality Estimators (CE) when there is a drift in data or workload. Although a step forward, their framework can only update workload-driven models and for CE applications only. More recently, [23] have provided a solution based on transfer learning and knowledge-distillation to update learned DBs in the presence of data insertions carrying out-of-distribution (OOD) data. These papers clearly show that in the presence of distribution drifts, appropriate methods must be developed for model updating. However, these solutions are limited to data insertions and have not been studied under data deletions.

In this paper, we initiate the study of data deletion in NN-based learned DBs. Specifically, we study its effect on the learned DB components and inform the community of our findings vis-a-vis lessons learned from related research in ML and from insertion updates in DBs. In ML, data deletion materializes as the problem of "Machine Unlearning". This is the problem of removing information related to a part of a dataset from a trained model, without hurting the information about the rest of the data [9]. This has also been referred to as “forgetting”, “scrubbing”, “removing”, and “deletion”. We will use these terms interchangeably. Unlearning is enjoying increasing attention in the ML research community. A recent machine unlearning competition, launched on Kaggle, attracted nearly 1200 participating teams [42].

Machine unlearning in ML research is motivated by the need to deal with out-of-date, noisy, poisoned, or outlier data. Another important reason for machine unlearning is to protect users’ privacy and guarantee the "right to be forgotten". Machine unlearning due to DB deletes is qualitatively and quantitatively different from the setup studied in the ML literature, as (i) deletes are very commonplace, sometimes even more frequent than queries themselves, and (ii) typically, “downstream tasks” are different (e.g., AQP, SE, DG, DC, etc.). Nonetheless, the aims are the same: to update a trained model in such a way that the “effect of deleted data is removed from the trained model”, while, at the same time, the model does not lose any knowledge it has learned about non-deleted data. Consider an AQP engine like DBEst++ [27] or a cardinality estimator like Naru/NeuroCard [48, 49]. These models essentially learn the data’s underlying probability distributions and perform query inference based on these. When a cohort of data is removed, the models should be updated to reflect the correct densities (and/or correlations) for accurate inference. That is, we need to ensure that the updated model makes correct predictions when querying either the deleted rows and/or the remaining rows.

Albeit an important problem, removing information from a trained neural network (without damaging the accuracy of the remaining data) is unfortunately a very challenging task. Ideally, one could remove the to-be-forgotten data from the DB and retrain a new model from scratch. However, as also has been shown by [23], training neural networks is prohibitively expensive.

Furthermore, it is not easy to measure how well a model has truly forgotten the deleted cohort of data after an unlearning (forgetting) algorithm is applied, as defining an appropriate set of evaluation metrics is an open problem in and of itself. A contribution of our work is to employ metrics for unlearning that are appropriate in DBs, drawing where possible inspiration from the ML literature that queries various aspects of a model’s outputs like the error (accuracy) for classification tasks, the loss values, or the entropy of its predictions. We will elaborate on this in Section 6.

2 MACHINE UNLEARNING IN LEARNED DB

Our study aims to be general enough so that its conclusions are pertinent to different NNs and to different DB downstream tasks for which these NNs were designed. (The selected NNs are not meant to imply any judgment on their being the best for selected tasks. Rather they are meant to boost the generality of drawn conclusions).

Consider a database relation R with a set of columns $\{C_1, C_2, ..., C_m\}$, with tuples $\{< c_1^i, c_2^i, \ldots, c_m^i >\}_{i=1}^{|R|}$. This can be a raw table or the result of a join query. Also, let $f(\cdot;\theta)$ be a neural network with a set of trainable parameters θ that parameterize a function f. f then could be used for any downstream task like AQP, SE, or DG, etc. Since here we wish to be general enough to handle different tasks/applications and different types of NNs, f might be trained with different objectives.

Assume we have a dataset $D = \{< c_1^i, c_2^i, \ldots, c_m^i >\}_{i=1}^{|R|}$ of tabular data with $|R|$ rows with m columns. We will denote by θ^o the parameters of the ("original", i.e. before unlearning) model trained on D, using a stochastic algorithm $\mathcal{A}$ like Stochastic Gradient Descent. That is, $\theta^o = \mathcal{A}(D, \theta^r)$, with θ^r denoting the random weights that the network is initialized with. Now, at unlearning time, assume we are given a partition of D into two disjoint sets of data rows: the "delete-set" (a.k.a "forget-set" or "deleted data"), D_d, and the "retain-set", D_r, where $D_r = D \setminus D_d$; that is, the retain-set is the complement of the delete-set in D. Informally, the goal of machine unlearning is to transform the model's parameters into a new set of parameters $\theta^u = \mathcal{U}(D_r, D_f, \theta^o)$ that do not contain any "knowledge" about D_d but retains all "knowledge" about D_r, where $\mathcal{U}$ denotes an unlearning algorithm that has access to the original parameters and the retain and forget sets.

Formally defining unlearning and quantifying success is a challenging problem in and of itself [41]. A common viewpoint for the goal of unlearning in the literature is that we desire the weights θ^u to be indistinguishable from the oracle weights θ^* [11], or the outputs $f(\cdot;\theta^u)$ to be indistinguishable from the outputs $f(\cdot;\theta^*)$ [12], where $\theta^* = \mathcal{A}(D_r, \theta^r)$ denotes the weights of the "oracle" of retraining from scratch using only D_r. More precisely, previous work defines the goal of (exact) unlearning as achieving $\mathbb{P}(\theta^u) = \mathbb{P}(\theta^*)$, with the probability distribution here being over model weights resulting from training with different random seeds (i.e. starting from different initial random weights θ^r and seeing a different ordering of mini-batches) or as achieving $\mathbb{P}(f(.;\theta^u)) = \mathbb{P}(f(.;\theta^*))$. For deep neural networks, the only known family of exact unlearning is based on retraining from scratch. To mitigate its inefficiency, recent work turned to approximate unlearning: achieving $\mathbb{P}(\theta^u) \approx \mathbb{P}(\theta^*)$ or $\mathbb{P}(f(.;\theta^u)) \approx \mathbb{P}(f(.;\theta^*))$, sometimes accompanied by theoretical guarantees about the quality of that approximation.

In this work, we translate these definitions into a set of metrics that is suitable for DB tasks. Specifically, in learned DBs, generative models are frequently derived, that model the distribution in D, and subsequently we are interested in performance in downstream tasks. To be comprehensive in our study, we thus create two sets of metrics for quantifying unlearning quality: 1) with respect to the "internal state" of the model (i.e. its estimate of the probability distribution) and 2) with respect to the performance on downstream tasks. For completeness, in addition, we will also study discriminative models using a NN for an upstream DC task. To quantify the success of unlearning in terms of downstream task performance, we utilize task-specific errors, where lower is better. Intuitively, we want the model to always make correct predictions (i.e. predictions matching the ground truth) for any query, whether the query relates to retained or deleted data.

3 LITERATURE REVIEW

3.1 Machine Learning for Databases

Many ML-based DB components have emerged recently, using different models for different applications like Database indices [4, 5, 22, 31], learned cardinality/selectivity estimators [15, 32, 44,

46, 48, 49, 53] and approximate query processors [27, 28, 39], query optimization and join ordering [21, 30], cost estimation [37, 52], and auto-tuning databases [26, 43, 50]. However, these do not provide any insights or solutions to adapt to changes due to DB data updates. The recent works in [23, 25] tackled distribution changes for CE tasks and CE/AQP/DG tasks respectively. ML methods have been used also for deriving samples of join query results without executing the join [35]. However, these approaches cannot account for data deletions, which as we shall see pose different updatability problems to NN-based learned DBs.

As in [23, 25], we also consider updates in batches - the isolated effect of single data rows in learned models is negligible. In environments like Online Transactional Processing (OLTP), however, update frequencies are higher and data may be deleted/inserted in single records. In such a setting the approach would be: The set of updates in a batch would be preprocessed and the final insert/delete records/operations would be identified. Delete records would comprise the forget set for unlearning. Insert records would comprise the new batch in [23, 25]). As the results in Section 7 show, unlearning can work well with smaller or larger batches and in conjunction with data insertions.

3.2 Machine Unlearning

The problem of machine unlearning was first introduced in [2], where they provide an exact forgetting algorithm for statistical query learning. [1] proposed a training framework by sharding data and creating multiple models, enabling exact unlearning of certain data partitions. [9] was the first paper to introduce a probabilistic definition for machine unlearning which was inspired by Differential Privacy [6], and formed the origin of the idea of the model indistinguishability definition discussed above. More recently, [14, 17, 34, 45] built upon this framework and introduced unlearning methods that can provide theoretical guarantees under certain assumptions. [29] surveyed methods for linear classification, showing different certifiability versus efficiency trade-offs.

Recently, approximate unlearning methods were developed that can be applied to deep neural networks. [11] proposed an information-theoretic procedure for removing information about D_d from the weights of a neural network and [10, 12] proposed methods to approximate the weights that would have been obtained by unlearning via a linearization inspired by NTK theory [18] in the first case, and based on Taylor expansions in the latter.

However, [11, 12] scale poorly with the size of the training dataset, as computing the forgetting step scales quadratically with the number of training samples. [10] addresses this issue, albeit under a different restrictive assumption. Specifically, they assume that a large dataset is available for pre-training that will remain "static", in the sense that no forgetting operations will be applied on it; an assumption that unfortunately can't always be made in practice.

Some recent works suggest modifying the original model's training for better unlearning in the future. [40] propose a regularizer to reduce 'verification error' making unlearning easier. However, it may impact model performance. [51] introduce a process with quantized gradients and randomized smoothing to avoid future unlearning, but large changes in data distribution, as a result of deletion, may exceed the 'deletion budget' invalidating assumptions. More recent works try to directly identify the parameters in the original model that are significantly influenced by the forget-set, aiming to modify these parameters to eliminate the impact of the forget-set. [8, 36] leverage Fisher information scores to identify the important parameters for the forget-set. [36] take a straightforward approach by fine-tuning the model on the retain-set while keeping the remaining parameters frozen. In contrast, [8] tries to 'dampen' those parameters while minimizing adverse effects on those essential to the retain-set. [19] introduce a 'sparsity-aware' unlearning technique, integrating unlearning through fine-tuning on the retain-set with a sparsification policy employing model pruning techniques. [7], on the other hand, proposes a 'saliency-aware' unlearning

approach, utilizing the loss function's gradient to learn a mask identifying 'salient' parameters related to the forget-set. These parameters are subsequently unlearned using existing unlearning baselines such as 'random labelling'. Despite the diversity in these methods, a common challenge persists—efficiently identifying and modifying the important parameters tied to the forget-set without adversely affecting those essential to the retain-set.

On the other hand, SCRUB [24] is a recent machine unlearning method for computer vision image classification tasks that scales better than previous works without making restrictive assumptions. SCRUB reveals different requirements and metrics for different unlearning applications (e.g., removing biases, correcting erroneous mislabelling or attack-poisoned labels, and preserving user privacy). SCRUB is shown to be the most consistently well-performing approach on a wide range of architectures, datasets and metrics.

4 LEARNING TASKS (APPLICATIONS)

We will study unlearning in the context of four well-studied, key tasks for learned DBs and data analytics (AQP, SE, DG, DC), using each a different NN type (as used in previous work). We now give an overview of each of these tasks.

4.1 Downstream DB Applications

Approximate Query Processing. AQP approximates the results of aggregation queries. This is a key task in analytical DBs, particularly for very large tables, where obtaining exact results can be prohibitively expensive [16, 23, 27, 39].

Selectivity Estimation. SE refers to the process of estimating the number of rows that a query will return. This is key for query optimization in RDBMSs, as it helps the query planner make informed decisions about best execution plans [23, 48, 49].

Data Generation. (Synthetic) data generation involves creating artificial data that mimics the characteristics of real-world data. DG is important for overcoming issues of insufficient or sensitive real data, thus dealing with privacy concerns or data scarcity issues.

Data Classification. DC has numerous real-world applications. The integration of classifiers within DBMSs is key in enhancing business intelligence and analytical services. For example, Microsoft SQL Server provides a 'data discovery and classification' feature. Similarly, Google's BigQuery empowers data mining processes through a repertoire of classifiers. Such classifiers typically use a categorical attribute whose values define the different classes according to which tuples are classified.

4.2 Machine Learning Models

Mixture Density Networks (MDNs) for AQP. MDNs consist of an NN to learn feature vectors and a mixture model to learn the *probability density function* (pdf) of data. Ma et al. [27] propose DBest++ which uses MDNs with Gaussian nodes to perform AQP. For the Gaussian Mixture, the last layer of MDN consists of three sets of nodes $\{\omega_i, \mu_i, \sigma_i\}_{i=1}^{M}$ that form the pdf according to Eq. 1, where M is the number of Gaussian components in the mixture.

Let y be the dependent variable (a target numerical attribute), $\mathbf{x}$ the vector $(x_1, ..., x_n)$ of independent variables (a set of categorical attributes). We also define f to be a NN that takes the encoded input vectors $\mathbf{x}$ and transforms them into learned "feature vectors" h. The likelihood under the mixture of Gaussians is then given by:

$$\hat{P}(y|h) = \sum_{i=1}^{M} \omega_i \mathcal{N}(h; \mu_i, \sigma_i) \tag{1}$$

where

$$\omega = g_1(h;\theta^{\omega}); \mu = g_2(h;\theta^{\mu}); \sigma = g_3(h;\theta^{\sigma}); h = f(\mathbf{x};\theta^{base}) \quad (2)$$

where each of g_1, g_2 and g_3 is a single-layer network that produces the weight ("mixing proportion") ω_i, the mean μ_i and standard deviation σ_i, respectively, for each of the i Gaussians in the mixture. Since the mixing proportions should sum up to 1, g_1 has a Softmax activation function, whereas g_2 and g_3 use a ReLU activation. We train all parameters of the system, (θ^{base}, θ^{ω}, θ^{μ}, θ^{σ}) jointly, by minimizing the negative log of the likelihood in Eq. 1.

Deep Autoregressive Networks for SE. The Naru and NeuroCard cardinality/selectivity estimators [48, 49] use deep autoregressive networks (DARNs) to approximate a fully factorized data density. DARNs are generative models capable of learning full conditional probabilities of a sequence using a masked autoencoder via maximum likelihood. Once the conditionals are available, the joint data distribution can be represented by the product rule as follows:

$$\hat{P}(x_1, x_2, \ldots, x_m) = \hat{P}(x_1)\hat{P}(x_2|x_1)\ldots\hat{P}(x_m|x1, \ldots, x_{m-1})$$

where x_i is an attribute in a relation R with m columns and the probability of the i^{th} conditional, $\hat{P}(x_i|x1, \ldots, x_{i-1})$, is parameterized via a neural network $f(\cdot;\theta_i)$. Naru and NeuroCard use cross-entropy between input and conditionals as the loss function, to train the parameters of the NNs $\theta_1, \ldots, \theta_n$.

Variational Autoencoders for DG. VAEs [**?**] are a commonly-used model for data generation. A VAE is a probabilistic auto-encoder, that uses a pair of neural networks to parameterize an encoder and a decoder module. In DB systems, [39] used VAEs to build AQP engines, [15] exploited them for CE, and [47] introduced a synthetic tabular data generator called **Tabular-VAE** (TVAE). VAEs are trained using a different loss function, known as Evidence-Lower-Bound (ELBO) loss (which amounts to a lower-bound estimation of the likelihood). Here we will use **TVAE** for learned synthetic tabular data generation, which is of particular importance in privacy-sensitive environments, or when data is scarce for data augmentation purposes, or when wishing to train models over tables and accessing raw data is costly.

Deep NNs for Classification for DC. Our previous three applications are based on *generative models*, trained in an unsupervised/semi-supervised fashion. To complete the picture, we add a data classification (DC) task, using a *discriminative* NN model. Note that there is no downstream task. Traditionally, DC over tabular data has been tackled using decision trees, random forests, support vector machines, etc. However, deep NNs have emerged as a powerful alternative that can effectively learn complex relationships and patterns in tabular data. Deep NNs for tabular DC leverage various architectures (e.g., feedforward NNs, convolutional NNs (CNNs), or recurrent NNs (RNNs)). Gorishniy et al. [13] have shown that, for tabular data, ResNet-like architectures are strong performers. Residual Neural Network (ResNet) [20], is a family of deep learning architectures that have significantly advanced the field.

5 UNLEARNING METHODS

In this section, we describe key baselines and a state-of-the-art method that has been proposed by the unlearning research in the ML community, in the context of classification tasks in computer vision. Each of these offers a different procedure to utilize the retain-set D_r and / or the delete-set D_d to achieve unlearning.

Retrain. The "oracle" solution is to remove the to-be-forgotten data from the DB and retrain a new model on only D_r. Retraining neural networks "from scratch", however, is prohibitively expensive and not practical in many settings [23].

Stale. This leaves the model stale as it was trained on the original data D. For DB tasks this is more complex as most of the learned DB components comprise learned NNs, and non-learned

modules such as the auxiliary meta-data. The stale baseline here updates the meta-data (like table cardinalities, or frequency tables), and leaves the learned model stale.

Fine-tune. This simple baseline fine-tunes the original model for a small number of epochs on the remaining rows. Concretely, it continues training the NN with data from D_r only, steering the model towards forgetting the delete-set. In ML classification tasks, it has been shown that this method will not erase the information completely [11]. Interestingly, fine-tuning has also been suggested by some of the learned DB components to support insertion updates [27, 49]. There, fine-tuning is performed on the new data to force the model to learn it. This method has been shown not to perform well with OOD data insertions [23].

NegGrad. An interesting idea is to continue training the model, but instead of using gradient descent on only D_r as in `Fine-tune`, use gradient *ascent* on only D_d (or equivalently, gradient descent on D_d with a negated gradient; earning the nickname NegGrad). The intuition for this is to attempt to "delete" D_d by maximizing the loss on that data, aiming to "undo" the process that the network had previously undergone to learn that data.

NegGrad+. NegGrad may degrade the performance on the retain set, since it has no incentive to protect useful information from being deleted when performing the gradient ascent. Intuitively, if retained data is "similar" to data in the delete-set, then `NegGrad`'s objective of maximizing the loss on D_d may indirectly also lead to maximizing the loss on (parts of) D_r, which is of course undesirable. To address this, we use a stronger baseline that simultaneously performs gradient ascent (as in `NegGrad`) on the delete-set and gradient descent (as in `Fine-tune`) on the retain-set, aiming to strike a good balance between maximizing the loss on D_d but keeping it small on D_r. We refer to this stronger baseline as `NegGrad+`. Formally, it obtains the unlearned weights θ^u by initializing them from θ^o and then minimizing the following w.r.t θ^u:

$$\beta \frac{1}{|D_r|} \sum_{x^r \in D_r} l(f(x^r; \theta^u)) - (1-\beta) \frac{1}{|D_f|} \sum_{x^d \in D_d} l(f(x^d; \theta^u)) \tag{3}$$

Where l is a task-specific loss function like negative likelihood and $\beta \in [0, 1]$ is a hyperparameter. Note that we can recover Fine-tune by setting β to 1 and NegGrad by setting β to 0.

SCRUB. This is a more sophisticated state-of-the-art algorithm for unlearning in image classification [24]. SCRUB makes use of a teacher-student framework where the teacher is the original model, trained on the entire dataset, and the student is initialized from the teacher and is subsequently trained to keep only the information from the teacher that pertains to the retain set. Specifically, the student is trained with two objectives: one that minimizes the distance between the teacher and the student for the retain-set, and one that maximizes this distance for the delete-set, leading to surgically removing information about the delete-set. Similarly to NegGrad+, SCRUB aims to find a sweet spot between deleting information about D_d while retaining information about D_r but does in the form of (positive and negative, for D_r and D_d, respectively) knowledge distillation from the "all-knowing" original model. Formally, SCRUB obtains the unlearned (student) weights θ^u by initializing them from the original (teacher) weights θ^o and then minimizing the following objective w.r.t θ^u:

$$\beta \frac{1}{|D_r|} \sum_{x^r \in D_r} d(x^r; \theta^o, \theta^u) - (1-\beta) \frac{1}{|D_f|} \sum_{x^d \in D_d} d(x^d; \theta^o, \theta^u) \tag{4}$$

where d is a measure of the distance between the student and teacher models, to be defined in an application-dependent manner. While SCRUB is originally defined for classification, using KL-divergence for d, here we propose an adaptation of it for two of our DB tasks: SE and AQP.

For SE, where we use deep autoregressive models to estimate the joint (over columns) probability density $\hat{P}$ of the data, we define d as the KL divergence between the estimated probabilities of the teacher and student. Formally,

$$d(x;\theta^o,\theta^u) = KL(\hat{P}(x;\theta^o)||\hat{P}(x;\theta^u)) \tag{5}$$

For AQP, where Mixture Density Networks are used, it is not straightforward how to apply SCRUB. We propose the following form for d to capture the discrepancy between the mixture distributions of the teacher and student:

$$d(x;\theta^o,\theta^u) = \frac{1}{M}\sum_{i}^{M}(MSE(\mu_i^o,\mu_i^u) + MSE(\sigma_i^o,\sigma_i^u) + KL(\omega_i^o||\omega_i^u)) \tag{6}$$

where for a given x, ω is obtained as $g_1(f(x;\theta^{base});\theta^{\omega})$, and analogously for μ and σ, as explained for MDNs in Section 4. *MSE* is Mean Squared Error, M is the number of components in the mixture, and the parameters θ (each of θ^u, θ^o, correspondingly) are the set:

$$\theta = \{\theta^{base},\theta^{\omega},\theta^{\mu},\theta^{\sigma}\} \tag{7}$$

SISA. Sharded, Isolated, Sliced, and Aggregated (SISA) is an ensemble method that first splits the data into p disjoint partitions, and further slices each partition into s slices. SISA trains a constituent model for each partition, by incrementally incorporating slices of that partition. During unlearning, when a delete-set is requested to be unlearned, SISA finds all the models that have been trained with the examples from the delete-set, removes those examples from the dataset and retrains the affected models from scratch. As such, SISA is an 'exact unlearning' method, like Retrain, unlike the rest of the baselines we consider which perform 'approximate unlearning'.

Since SISA was originally built for classification tasks, it uses a majority vote aggregation. However, for most of the applications that we study, including AQP, SE and DG this aggregation is not applicable. Therefore, we design new aggregations: For AQP and SE where the models evaluate a workload of `sum` or `count` queries, we calculate the result of the query using each model and sum up the results. For DG, we generate samples using each model, and concatenate all the samples to form the final synthetic data.

6 MEASURING THE IMPACT OF DELETION

As mentioned earlier, this is an open problem of its own right in the ML literature. And for learned DBs, as we discuss below, several adjustments need to be made.

In the ML literature unlearning is mostly studied for classification problems, and the accuracy of the retain-set and the test-set are used to measure the model's performance on the remaining data and its generalization, respectively. We desire an unlearning algorithm that successfully 'forgets' the delete-set without deteriorating either of these. We measure the 'forget quality' using the accuracy on the delete-set which is ideally close to the delete-set accuracy of the `Retrain` oracle (that truly never trained on the delete-set).

The commonly-studied applications for unlearning in the ML literature (i.e., classification) concern the direct output of learned models (i.e., upstream tasks). Our setting is different; our learned DB models are developed for *downstream tasks*, that are not the immediate output of the trained model. For instance, the AQP engine in [27] infers the query answer using an integral estimation over the learned Gaussians produced by MDNs. Similarly, the cardinality estimators in [48, 49] perform a progressive sampling over the learned DARN to infer the cardinality. Given this, we will care both about what the model has learned and unlearned, as well as the accuracy of the downstream tasks. We establish a distinction between, on the one hand, evaluating the quality of unlearning with respect to the internal state of the model itself (how well has it forgotten the

requested data) and, on the other hand, how its performance on the downstream tasks of interest is affected. This distinction is interesting from a scientific perspective, as it enables the study of several research questions concerning unlearning in generative pretrained models: Can we achieve the desired outcome on a set of downstream tasks without having optimally amended the model's internal state in light of deletions? What is the relationship between unlearning quality upstream and downstream? In this work, we initiate this investigation and propose a set of metrics for each of these evaluation facets.

To study the downstream task's performance, we use the original metrics that have been used to evaluate the task, with a small change. Inspired by the accuracy evaluation in unlearning in ML, we divide the evaluation workload into two separate groups that target the delete-set and the retain-set. More specifically, for AQP and CE, one workload only queries the deleted rows, and the other workload queries the remaining rows. In both cases, the lower the error, the better (unlike other ML applications where higher error is desirable / indicates better forgetting). Intuitively, considering a COUNT query for example, when the user requests a query that targets the deleted part, the engine should correctly answer 0. For DG, the generated data could be used for different tasks. We follow the evaluation in [47] where we use a classification task and evaluate on the test-set and measure forgetting via accuracy on the delete-set.

Additionally, we evaluate unlearning in the model's internal state in two ways. First, by inspecting the likelihood, we can assess what data is "likely", or "compatible" with the model's internal understanding of the distribution. Indeed, likelihood is often the training objective of learned DB models and, in fact, in some cases, like in MDNs, the exact likelihood is available through the mixture of Gaussians. Intuitively, we would like the model to assign a higher likelihood to the retain-set on average, and a lower likelihood to the delete-set. Second, since learned DB systems are usually generative models that learn the data density, we can directly inspect the learned probability distribution via sampling, before and after deletion. Intuitively, we want the unlearning process to modify the learned distribution in such a way that it accurately reflects the updated true "state of the world" after the deletions.

Finally, we use Membership Inference Attacks (MIAs) to asses the forgetability of the models. In an MIA, the adversary tries to infer whether a data point has been involved in training a model. We focus on the common black-box attack setting where the attacker observes only outputs of the model. Designing MIAs generally, and especially as a metric for forgetability, is an open research problem. Nevertheless, we take two attacks that have been used in the unlearning literature and apply them to our DC models.

7 EXPERIMENTAL EVALUATION

We empirically evaluate and analyze the aforementioned unlearning algorithms, in the context of several applications and evaluation metrics. We consider two scenarios: Deletion in "one-go", where a single round of deletion is performed, as well as "sequential deletion", where several deletion requests are carried out sequentially. The code and information for reproducibility and availability purposes can be found at https://github.com/meghdadk/DB_unlearning.git.

7.1 Experimental Setup

7.1.1 Datasets. We have used three real-world tabular datasets, typically found in the literature for our downstream tasks, namely: Census: 48k rows, Forest: 580K rows, and DMV: 11M rows.

7.1.2 Delete Operations. Deletes have the following skeletons:

```
1. DELETE FROM Table WHERE L <= att <= U
2. DELETE FROM Table WHERE (L <= att <= U) and
```

$$(row\text{-}index \ \% \ 2 \ == \ 0)$$

where att is a numerical attribute, L and U define the range of values that will be deleted, and $row\text{-}index$ is a primary incremental index started from 1. In the former, a whole data subspace is deleted ("full deletion") whereas in the latter, only some parts within a subspace are deleted ("selective deletion").

For Census, att refers to the *age* attribute, where $L = 30$ and $U = 35$. For Forest, att refers to $Elevation$ and $L = 2500$, $U = 2700$. For DMV, att is $max\text{-}gross\text{-}weight$, $L = 7000$, and $U = 8200$. In all cases, full (selective) deletion queries delete 10% (5%) of the dataset.

7.1.3 Queries and Metrics. For the AQP and SE tasks, after a delete is performed, two types of queries are generated: Query-Retain (QR) targets only the remaining (non-deleted) data. On the other hand, Query-Delete (QD) queries only target deleted data. For QR, we report the "relative error" (relative to the ground truth) to evaluate AQP and SE tasks. For QD, however, the ground truth is always 0 for both tasks (since the sum, mean and count over deleted rows are 0); thus, we report the "absolute error" instead of the relative error, to avoid a division by 0. Furthermore, for DBEst++ MDNs, as their mixture of Gaussians provides the exact likelihood, we also report the average likelihood for QD and QR.

For the DG task we evaluate the model's data generation quality via the accuracy of an XGboost classifier trained on the synthetic samples generated by the model (TVAE) after training, as in [47]. We hold out 30% of the synthetic table as the test set and train a classifier with XGBoost. Then we predict the classes of the held-out data test set. We report the *macro f1-score* for the classifier. For Census, Forest and DMV, we use: *income*, *cover-type*, and *fuel-type*, as the target class, respectively. Here we created a smaller version of DMV with only 1M records, as training TVAE on the whole DMV is very time/resource-consuming. For this smaller DMV, instead of forming the deletion query via a range, as shown above, we delete all rows that satisfy $State$ = 'OK'.

For the DC task, we use: *marital-status*, *cover-type*, and *fuel-type* for Census, Forest and DMV, as the target class, respectively. We split the tables into a 80%-10%-10% splits of train-validation-test.

7.1.4 Workloads. Each model is evaluated using 2000 randomly generated QR and 2000 QD queries. For Naru/NeuroCard, we use their generator to synthesize QR and QD queries: It randomly selects the number of filters per query. For Forest, this number is randomly selected from the range [3,8], for Census from [5,12], and for DMV from [5,12] Then, it uniformly selects a row of the table and randomly assigns operators [=, >=, <=] to the columns corresponding to the selected filters. Columns with a domain less than 10 are considered categorical and only equality filters are used.

For DBest++, we randomly select a *lower-bound* and a *higher-bound* for the range filter and uniformly select a category from the categorical column for the equality filter.

SELECT AGG(·) FROM $Table$ WHERE F_k AND F_n

where AGG is an aggregation function over a numerical attribute, and F_k (F_n) is a filter over a categorical (numerical) attribute. Specifically, F_n is a range operation and the aggregation function is `COUNT`, `SUM`, or `AVG`. We select the following columns from each dataset: Census:[`age, country`]; Forest:[`slope, elevation`]; DMV:[`body type, max gross weight`]; where the first/second attribute is categorical/numeric. Naru/NeuroCard is a cardinality estimator and we only evaluate `COUNT` queries for it.

7.1.5 Models' configurations. For DBEst++, for Census, we build an MDN with 2 fully connected layers of size 128 each, and use a MoG of size 30 for the last layer. For Forest and DMV, we use the same number of fully connected layers with the same size, but with 80 MoG for the last layer. We train the `Original` and `Retrain` models for 50 epochs, with a learning rate (lr) of 0.001 (decaying

with a rate of 0.1 at epochs 10, 20, 30), and a batch size (bs) of 128, using the Adam optimizer. For Naru/NeuroCard, and TVAE, we use the same configuration as in the original work and tune hyper-parameters for smaller errors, for the datasets that have not been used in the original work. For the unlearning baselines, we tune all the hyper-parameters of the models including lr, bs, decay rate, optimizer, and the method's specific hyper-parameters, to achieve the smallest retain error and forget error.

For SISA, we set $p = 10$ and $s = 10$ for the DC task, and $p = 5$ and $s = 5$ for other applications. For DC, we use a smaller neural network with 3 fully connected layers. For AQP/DBEst++ application we use an MDN with fully connected layers of size 64 and 15 MoG for Census and 30 for Forest and DMV. For SE/Naru-NeuroCard and DG/TVAE we decrease the models' depth by 1 layer for each of the encoder and the decoder. Finally, for better efficiency of SISA, we sort each dataset based on the column that is queried for deletion.

7.2 Deletion in "One-Go" (Large Batch)

Deletion (in "one-go"' – i.e. using a large batch of deletions) is studied using both downstream task and model-internal metrics.

7.2.1 Results on Downstream Tasks. **AQP/DBEst++.** These results are reported in Tables 1 and 2. Overall, the main conclusions from Table 1 are as follows: We first notice that `NegGrad` performs well for QD but poorly for QR, which is expected as it has no incentive to retain knowledge of the retain-set, only to erase knowledge for the delete-set. For QR-count, `Retrain`, `Fine-tune`, and `SCRUB`, all perform similarly (with mostly overlapping CIs) `NegGrad+` follows, with a slightly worse performance for Census and Forest, but strong performance on DMV. For QR-sum, the findings are very similar.

For QD, as expected, `NegGrad` performs very well, as it was designed to update the model specifically for deletes. With respect to `Stale`, we would expect it to have very poor performance for deleted data, as this has not been updated to learn it. This is indeed the case. For QD-count: We note that `NegGrad+` is a top performer, hand-in-hand with `Retrain` and that `Fine-tune` does very well (very close to `Retrain`). `SCRUB` follows in performance, but in absolute terms, it achieves small errors (e.g., being much closer to `Retrain` than to `Stale`. And for DMV `SCRUB`'s performance has overlapping CIs with `Retrain`. For QD-sum, the conclusions are very similar to those of QD-count. `NegGrad` may be a top performer for QD, but note that its performance for QR is dismal.

For selective deletion (Table 2), perhaps surprisingly, `Stale` appears to be doing better than would be expected. However, recall that: (i) queries here cover both remaining and deleted data simultaneously, due to the interleaved manner with which the retain and forget sets are defined, and `Stale` is expected to do well for retained data, and (ii) `Stale` may be leaving the model unchanged but it does update other statistics (such as frequency tables) used in the end to predict the aggregate. Given these facts, the good performance on this interleaved set is not so surprising. This finding highlights why we used downstream task errors *and* internal model metrics. Specifically, looking at Table 2 we clearly see that `Stale` has a much higher likelihood for deleted data than the other unlearned models. This is so because `Stale` has not unlearned and deleted data is still part of its training set. We will discuss this mismatch between internal state and downstream task metrics in later sections.

SE/Naru-NeuroCard. We have reported the results in Table 3. The results are for both full and selective deletion. As mentioned earlier, Naru-NeuroCard involve a sampling process during inference that automatically zeroes-out the queries on the delete-set as the sample does not find any record from that region of the table to sample from. We observe that `Fine-tune` is the top performer. In fact, fine-tuning the model results in better accuracy than retraining from scratch. This is an

Table 1. Full deletion results for AQP/DBEst++. For QR we show the average relative-error and 95 % CI. For QD we show the average absolute-error and 95 % CI. "count" ("sum") refer to queries with COUNT (SUM) aggregation function. Likelihood numbers show the average likelihood the MDN model predicts for the retained and the deleted rows using Eq. 1.

dataset	model	Query-Retain - QR		Query-Delete - QD		likelihoods	
		count	sum	count	sum	remain	delete
Census	Retrain	16.26±1.06	16.53±1.11	0.09±0.04	2.29±0.83	0.88	0.04
	Stale	22.44±1.38	23.03±1.54	26.62±6.60	824.70±211.52	0.74	0.98
	Fine-tune	16.88±1.10	17.58±1.39	0.83±0.19	24.69±5.15	0.86	0.13
	NegGrad+	20.20±1.39	19.46±1.46	0.27±0.02	11.34±6.81	0.87	0.07
	NegGrad	153.27±15.78	141.54±13.31	0.00±0.00	0.00±0.00	0.22	0.00
	SCRUB	17.39±1.14	17.83±1.35	1.94±0.51	57.48±14.23	0.85	0.18
Forest	Retrain	9.50±1.02	9.25±0.84	0.00±0.00	0.69±0.13	1.36	0.01
	Stale	48.12±34.32	24.59±3.37	4.23±0.22	9148.42±512.56	1.24	0.49
	Fine-tune	10.15±1.08	9.96±0.98	0.01±0.00	14.68±5.97	1.36	0.01
	NegGrad+	13.55±1.10	13.81±1.07	0.00±0.00	1.75±3.20	1.36	0.00
	NegGrad	137.93±37.00	157.15±62.51	0.00±0.00	0.63±0.88	1.01	0.00
	SCRUB	11.16±1.47	10.29±0.93	0.05±0.00	72.93±7.49	1.36	0.03
DMV	Retrain	85.06±35.36	82.96±21.45	0.46±0.17	2448.82±711.35	85.09	8.27
	Stale	56.05±11.46	58.42±7.77	2.13±0.70	13367.03±2780.23	40.78	42.90
	Fine-tune	106.07±49.79	89.86±26.12	1.10±0.39	5545.70±1251.51	39.68	16.42
	NegGrad+	81.52±25.01	119.98±91.18	0.52±0.15	3457.04±1071.17	41.65	4.58
	NegGrad	8083.59±527.80	1131923741.20±64946.12	0.00±0.00	0.00±0.00	2.78	0.00
	SCRUB	69.49±14.05	102.54±70.92	0.50±0.25	2642.01±487.11	46.06	12.77

interesting finding of its own right, as sometimes fine-tuning may reinforce old knowledge while unlearning as well. Similarly, NegGrad+ also performs great. SCRUB is also a top performer, except for full deletion on DMV. Stale also does well (as also observed above for AQP). But, note that Stale performs poorly for full and selective deletion for DMV. And as before, its performance for full deletion is worse than that for selective deletion. Again, it is crucial to evaluate across models, downstream tasks, internal model metrics, and datasets for a complete picture. Finally, as before, NegGrad degrades the model drastically.

DG/TVAE. Finally, in Table 4 we show the results stemming from the classification accuracy over the synthetic data generated by TVAE across different unlearning methods and datasets. We do not report SCRUB as it is unknown how to apply its loss and integrate it with a TVAE. We also do not show numbers for NegGrad as its performance was very poor, suffering from exploding gradients, even with very small learning rates and a very small number of iterations. The first main observation is that all methods perform very close to each other. Even Stale. This experiment helps bring to the surface additional interesting issues. Why do *all* methods, even Stale, perform equally? This occurs because the delete operation in this particular case happens to not affect the classification task. We investigated this by computing the Pearson correlation coefficient values among the dependent and independent variables (omitted for space reasons) and we found that essentially all correlations remain unaffected before and after deletion. Please note that this reflects a perfectly reasonable real-world scenario as not all delete operations affect downstream task accuracy. Nonetheless, an additional concern for such cases is whether the ML models themselves were affected at all by the deletion. And this is why model-internal metrics should be used to reveal such differences. In fact, as we shall show in the next subsection, there are clear differences between models of different methods with respect to the effect of even such deletion operations; and, it is shown that the Stale model fails to register the deletion.

Table 2. Selective deletion results for AQP/DBEst++. In this setting queries cover ranges of remained and deleted values and the ground truth is therefore never zero. So, we show the average relative-error and 95 % CI. The likelihood numbers show the average likelihood the model predicts for the retained rows and the deleted rows using Eq. 1.

dataset	model	Queries		likelihoods	
		count	sum	remain	delete
Census	Retrain	15.36±1.09	14.12±0.97	0.78	0.61
	Stale	17.68±1.21	16.15±0.99	0.77	0.98
	Fine-tune	16.02±1.08	14.69±0.97	0.78	0.64
	NegGrad+	15.31±1.01	14.16±0.89	0.78	0.59
	NegGrad	166.76±19.62	157.35±18.32	0.19	0.00
	SCRUB	15.66±1.07	14.48±0.87	0.77	0.65
Forest	Retrain	8.59±1.00	10.65±1.18	1.25	0.27
	Stale	14.64±1.65	15.47±1.62	1.20	0.49
	Fine-tune	9.68±1.14	11.45±1.26	1.25	0.28
	NegGrad+	10.87±1.10	12.71±1.45	1.27	0.17
	NegGrad	162.41±92.24	147.14±38.94	0.96	0.00
	SCRUB	9.97±1.24	11.31±1.35	1.25	0.29
DMV	Retrain	92.28±41.29	60.70±12.69	46.93	31.32
	Stale	82.33±48.65	54.10±7.16	40.97	42.91
	Fine-tune	104.31±71.19	60.22±16.38	40.79	30.64
	NegGrad+	66.30±17.51	57.98±9.63	37.71	23.93
	NegGrad	288478.30±152969.35	221261.35±119654.25	0.01	0.00
	SCRUB	157.40±96.48	64.61±14.19	41.75	31.96

7.2.2 Results using Model-Internal Metrics. As we have seen above, while investigating the effect of various unlearning methods, looking at only downstream tasks results can be misleading or reveal only part of the full picture. Such results depend on underlying data distributions, on the particular delete operations, and on the actual analytics task at hand. For example, the data subspace affected by a delete and its correlation to the dependent variable may be such that downstream task accuracy results are unaffected, showing all methods (even poor unlearning methods, or even `Stale`) to perform very well. In our example tasks studied here, all ML models are generative in nature. Hence, they could generate data items, according to the distributions they have learned. Then one can compare these distributions against the original (ground truth) data distributions and do so before and after deletions. In this section, we highlight results using such distributions. Note that additional model-internal metrics can be utilized. One such example is using a model's likelihood numbers, as we have done above for the AQP/DBEst++ case where the underlying MDN model provides such likelihoods. For space reasons, we show these distributions for the AQP/DBEst++ case for the Census dataset in Figure 1 (results for the other datasets are very similar). In addition to the visual understanding provided by these histograms, we also measure the divergence between the distributions produced by unlearning algorithms and the real distributions after deletions. We use the Jensen-Shannon divergence (a symmetrical version of KL divergence) between these distributions. JS-divergence values range from 0 to $ln\ 2$ (ca. 0.69). The results are reported in Table 5.

Table 3. Full and selective deletion for SE/Naru-NeuroCard. Numbers are average relative-error and 95 % confidence interval for remain queries-QR (full deletion) and queries covering remain and deleted data (selective deletion).

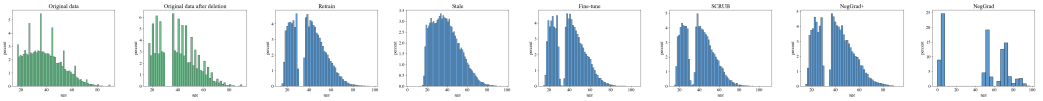
dataset	model	Full Deletion	Selective Deletion
Census	Retrain	16.87±1.08	14.32±1.07
	Stale	16.20±0.86	13.81±0.99
	Fine-tune	10.23±0.71	9.48±0.79
	NegGrad+	10.42±0.70	9.19±0.79
	NegGrad	97.62±26.17	107.35±43.48
	SCRUB	11.71±0.74	10.93±0.87
Forest	Retrain	24.96±4.44	21.66±1.99
	Stale	23.78±4.61	20.78±1.71
	Fine-tune	21.48±3.67	18.34±1.64
	NegGrad+	22.43±4.61	19.38±2.05
	NegGrad	74.23±18.83	76.81±17.15
	SCRUB	23.12±4.78	19.94±1.86
DMV	Retrain	7.37±0.58	8.68±0.61
	Stale	13.63±0.61	11.95±1.00
	Fine-tune	5.13±0.56	5.77±0.66
	NegGrad+	5.34±0.56	9.06±0.70
	NegGrad	2892.63±3283.17	40.90±2.63
	SCRUB	18.67±1.87	10.25±0.91

Table 4. Full and selective deletion for DG/TVAE. 'retain synth' refers to the f1-score of the xgboost classifier trained with synthetic data and evaluated on the held-out test-set. 'delete synth' refers to the f1-score of the xgboost classifier trained with the synth data and evaluate on the deleted rows.

dataset	model	Full Deletion		Selective Deletion	
		retain synth	delete synth	retain synth	deleted synth
Census	Retrain	61.13±0.84	62.84±3.99	61.33±0.93	69.79±0.99
	Stale	61.27±0.19	69.33±0.71	60.77±0.95	70.10±0.33
	Fine-tune	60.11±1.18	68.84±0.49	61.32±1.18	70.18±0.21
	NegGrad+	60.41±0.83	69.09±1.01	61.35±0.65	69.80±1.07
Forest	Retrain	41.46±1.00	16.39±0.87	44.57±0.77	21.55±2.66
	Stale	46.94±0.80	32.31±1.83	46.15±0.46	32.49±1.68
	Fine-tune	48.39±1.49	28.85±2.04	48.89±0.73	34.65±1.07
	NegGrad+	49.95±0.78	31.16±1.16	46.32±1.01	33.74±1.13
DMV	Retrain	41.69±0.72	36.19±3.76	41.02±0.40	43.64±1.62
	Stale	41.12±0.51	36.99±4.47	41.12±0.48	35.70±5.44
	Fine-tune	41.34±0.51	36.08±1.19	40.39±0.29	37.59±5.05
	NegGrad+	40.72±0.56	37.40±4.12	41.22±0.72	52.21±6.96

Note that all values are very close to zero, showing strong unlearning performance for the model internally.

In the results shown in Figure 1, the first two subplots show the real data distribution before and after the deletion. The other subplots show the distribution of the samples generated by each model after unlearning. The goal is to match the ground truth histogram that is obtained after



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. 1. Histogram of "age" for Census. Native-Country="United States". AQP/DBEst++. "Full" deletion. (a) Original data (b) Original data after deletion (c) `Retrain` (d) `Stale` (e) `Fine-tune` (f) `SCRUB` (g) `NegGrad+` (h) `NegGrad`

Table 5. JS divergence between distributions of the original data and the synthetic data generated after deletion.

method	Full			Selective		
	Census	Forest	DMV	Census	Forest	DMV
Train	0.0156	0.0085	0.0275	0.0096	0.0697	0.0174
Retrain	0.0118	0.0086	0.0015	0.0053	0.0147	0.0315
Finetune	0.0198	0.0470	0.0782	0.0019	0.0726	0.0216
NegGrad+	0.0062	0.0255	0.0158	0.0019	0.6407	0.0264
NegGrad	0.3084	0.3346	0.0874	0.1001	0.6926	0.6891
SCRUB	0.0278	0.0092	0.0017	0.0015	0.0639	0.0661

deletion, shown in subplot (b). One can easily see that `Stale` does not unlearn, as is expected. Furthermore, `NegGrad` is shown to introduce unwanted artifacts in the distribution of the rest of the data as well, showing its limitations. All other methods are shown to unlearn the part of the data that is deleted. This scenario and discussion raise the following key issue: As not all deletions will affect downstream task accuracy, one may not wish to apply any unlearning algorithms as downstream accuracy will be unaffected. So a "detector" for such cases is warranted. Naturally, if the same model is being used for different downstream tasks, such task-specific detectors may not be appropriate, as downstream task accuracy for other tasks may be affected.

7.3 Sequential Deletion (Smaller Batches)

In the above experiments, we performed data deletion in "one-go". Here, we study a different setting where deletes are executed sequentially (in smaller batches). We aim to address two questions in this section. First, how does accuracy evolve when models are unlearned sequentially? Second, is it better to execute deletion requests on-demand, or to group them and perform unlearning at once? For these experiments, we use the "Full" delete operation described earlier for each dataset. We split it into 5 smaller deletes. At every step, we perform unlearning on the models as updated in the previous step. Figures 2 to 4 show how the median error of downstream tasks evolves in the different settings. Overall, three main conclusions can be drawn from these figures. First, `Fine-tune`, `NegGrad+` and `SCRUB` have errors for QR queries that are close to those of `Retrain`. Second, these errors do not accumulate across sequential deletes. `NegGrad` and `Stale` conversely show an increase of error in consecutive steps. Third, the errors on QD queries show that `NegGrad` performs consistently well in terms of forgetting (as expected) along with `Retrain`, `Fine-tune`, and `NegGrad+`, while, `Stale` again shows a growing error. These conclusions addressed our first question w.r.t the evolution of errors.

Figures 5 and 6 address the second question for AQP/DBEst++ and SE/Naru-NeuroCard for the errors of the queries on the remaining rows. Results for accuracy on deleted rows are very similar and deleted for space reasons. We also have plotted the horizontal line of 1, to signal the point of

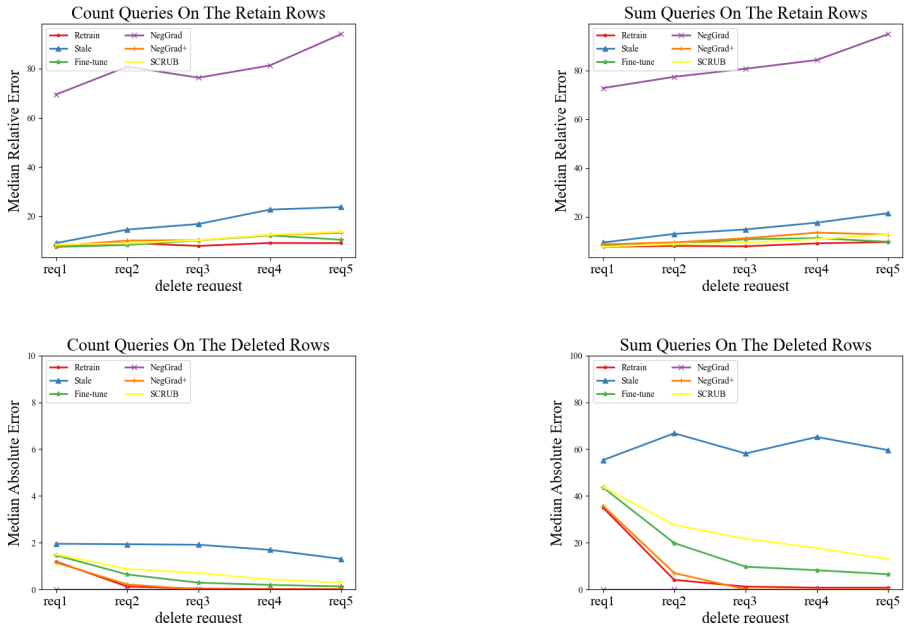


Fig. 2. Error evolution. Deleting sequentially. Census dataset, AQP/DBEst++ model.

no difference between errors in sequential vs the one-go settings. The results reveal interesting behaviors. First, looking at Figures 5 and 6, we see that for the Census datasets and for AQP/DBEst++ and SE/Naru-NeuroCard there is hardly any difference between sequential vs one-go deletion for all methods. Looking at the Figures for Forest dataset, however, we see emerging differences. Namely, for AQP/DBEst++ and Forest, one-go deletion for `Retrain` and `NegGrad+` emerges as preferable. And this does not hold for Forest and SE/Naru-NeuroCard (Figure 6). This highlights the fact that conclusions depend on downstream tasks and datasets. Interestingly, `Fine-tune` appears to be less affected than `Retrain` across these tasks and datasets. This is important as `Fine-tune` has proved to be a very strong performer from all previous experiments, competitive to `Retrain`. It is obviously also faster than `Retrain`. And now we see that it appears to be more robust than `Retrain` with respect to how often it should be run. So, it appears to be even less expensive.

7.4 Insertions and Deletions

Learned DB updatability in general requires support for both data update operations, insertions, and deletions. We have shown thus far that `Fine-tune` (and `NegGrad+`) perform well for unlearning in a learned DB model. However, recent research in learned DB updatability for data insertions in [23] showed that fine-tuning is not accurate and more complex algorithms are needed.. Thus, it is unknown how these different mechanisms for insertions and deletions would interact with one another. In other words, can this simple fine-tuning method for unlearning perform well even in conjunction with a method for data insertions? We answer this question now using the code for [23] and combining it with fine-tuning for deletions, as follows: we assume insertions and deletions come in batches. First, we apply the fine-tuning method for deleting a subspace (e.g., all tuples with age values in [20,30]). Then we apply the method in [23] on the fine-tuned model, for inserting back all tuples in the same age group. Ideally, we would get to a distribution very close to the original, before any deletions and insertions. Table 6 shows the results. After training the original model in Step 1, we perform the deletions in Step 2. Finally, we insert the deleted data again in Step 3. We

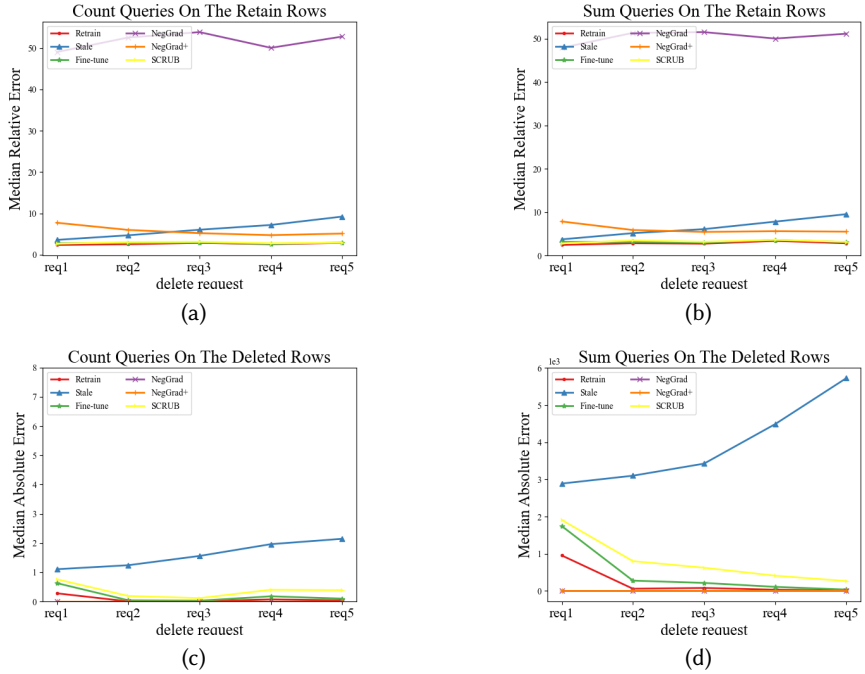


Fig. 3. Error evolution. Deleting sequentially. Forest dataset, AQP/DBEst++ model.

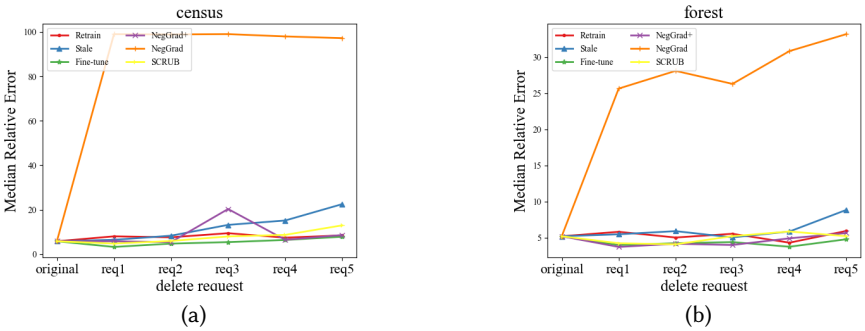
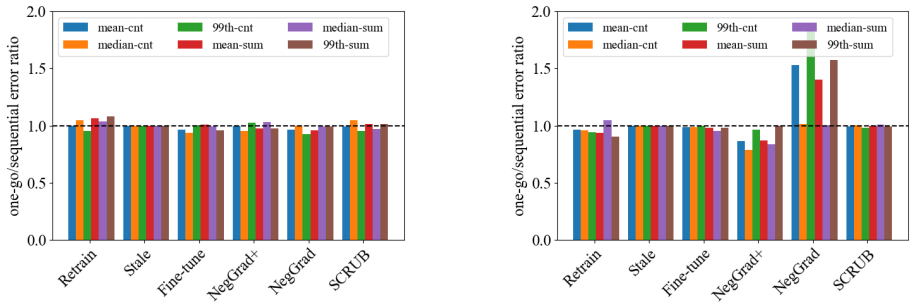


Fig. 4. Error evolution. Sequential deletion. Census dataset, SE/Naru-NeuroCard.

show average relative errors for SUM and COUNT queries after each step. We also report the JS divergence between the original data distribution and the generated sample distribution after each step to illustrate that internally, the models at each step stay close to the original data. Note that both errors and JS divergences are very small after Steps 2 and 3.

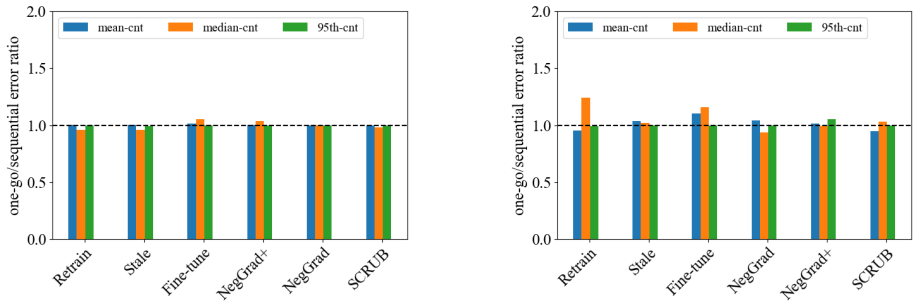
7.5 A Data Classification (Upstream) Task

We report results for the DC task. Following the ML unlearning literature like [24], we perform both class unlearning (delete all examples/tuples of a class) and selective unlearning (deleted only a subset of the class examples/tuples). For Census, for class unlearning, we remove all tuples of class ' Divorced'. And for selective unlearning, we delete only tuples whose age value is between 30 and 40 *from all classes*. In both cases, the delete-set amounts to 10% of the whole data. Table 7 shows



(a) (b)

Fig. 5. Ratio of relative errors: deleting in one-go over sequential deletion. AQP/DBEst++ on QR. a) Census, b) Forest.



(a) (b)

Fig. 6. Ratio of relative errors: deleting in one-go over sequential deletion. SE/Naru-NeuroCard in Forest on QR. a) Census, b) Forest.

Table 6. Combining Deletions and Insertions. Numbers show average relative error ± 95 % CIs. NB: Naru/NeuroCard does not support sum queries (shown as NA).

Model	Step	Operation	Queries		JS-divergence
			Count	Sum	
DBEst++	1	Train	15.05±1.09	15.62±1.08	0.0136
	2	Delete	17.13±1.39	16.00±0.98	0.0129
	3	Insert	17.10±1.08	17.79±1.19	0.0044
NeuroCard	1	Train	20.85±1.12	NA	0.0934
	2	Delete	10.15±0.71	NA	0.1003
	3	Insert	19.50±0.89	NA	0.1056

the classification accuracy for Census - the results for DMV and Forest are very similar. We can see that Fine-tune continues to enjoy high performance.

7.6 Efficiency

Unlearning algorithms are motivated by wanting to avoid the high costs of retrain-from-scratch. We thus evaluated the efficiency of the approximate unlearning algorithms studied. Table 8 shows the speed-up of each algorithm over Retrain for our three downstream tasks and their NN models. (The results are very similar for the upstream DC task). We can see that speedups, especially for

Table 7. DC task results with a ResNet architecture. The numbers are average accuracy ± 95% CIs.

Dataset	Method	Class Unlearning			Selective Unlearning		
		Test	Retain	Forget	Test	Retain	Forget
Census	Original	83.69±1.58	88.65±0.41	64.72±2.17	83.48±0.60	85.87±0.39	81.87±0.84
	Retrain	79.78±1.00	93.12±0.17	0.00±0.00	84.94±0.51	86.19±0.44	80.66±1.12
	Fine-tune	79.03±1.37	92.27±0.96	0.00±0.00	84.34±1.84	85.16±1.52	80.56±4.18
	NegGrad	52.39±8.37	60.79±1.24	0.00±0.00	1.56±1.51	1.49±1.81	0.99±1.61
	NegGrad+	76.65±5.83	89.51±4.13	0.00±0.00	79.51±8.22	80.80±7.08	76.66±8.93
	SCRUB	78.70±0.49	92.19±0.48	0.00±0.00	83.53±0.47	85.15±0.41	80.62±0.87

Table 8. Speed-up of unlearning algorithms over the 'Retrain' oracle. Stale is obviously not included.

method	AQP/DBest++						SE/Naru-NeuroCard						DG/TVAE					
	Full			Selective			Full			Selective			Full			Selective		
	Census	Forest	DMV	Census	Forest	DMV	Census	Forest	DMV	Census	Forest	DMV	Census	Forest	DMV	Census	Forest	DMV
Retrain	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Fine-tune	11.47	17.05	11.36	28.87	18.37	9.46	1.81	2.00	2.30	2.66	1.94	4.5	16.27	11.86	17.04	19.42	12.57	18.31
NegGrad+	10.90	10.28	6.67	11.55	10.61	5.70	1.86	2.09	2.28	2.5	2.21	2.45	16.27	15.98	43.08	19.42	15.11	19.88
NegGrad	10.38	10.23	15.22	12.83	9.69	5.64	2.4	5.86	5.67	5.71	5.22	2.88	NA	NA	NA	NA	NA	NA
SCRUB	3.63	5.79	1.90	3.72	6.19	1.55	1.56	1.58	1.40	2.15	1.47	2.03	NA	NA	NA	NA	NA	NA

the `Fine-tune` and `Negrad+` are high. Note that these speedups can become much higher when much larger datasets are used (as training times depend on dataset sizes).

7.7 SISA Results

In this section, we summarize `SISA`'s results, reported in Table 9 to Table 12. We only report results for Census dataset due to space, but the conclusions hold for Forest and DMV.

For AQP/DBEst++, `SISA` consistently demonstrates comparable or improved error rates in comparison to regular training. Retraining `SISA` generally results in lower errors for deleted queries compared to regular retraining. When examining likelihoods, it's important to note that direct comparisons with regular settings may not be valid due to different model architectures. Nevertheless, the tables reveal that retraining `SISA` leads to lower average likelihoods for deleted rows, showcasing the impact of unlearning.

`SISA` shows significantly higher relative error during training compared to regular training in SE/Naru-NeuroCard. Our detailed experiments revealed that constituent models perform better on their own data chunks but struggle with larger data ranges. The introduced sparsity from creating embeddings for the entire table while training on a small shard decreases performance. Additionally, aggregating results across models further increases error.

In DG/TVAE, `SISA`'s classification performance is comparable to regular training, but retraining during unlearning leads to a slight decrease in accuracy for both retained and deleted sets.

For DC, `SISA` for training and unlearning has a slight decrease in test and retain accuracy, compared to regular training. This difference persists when comparing regular retraining and SISA retraining. However, surprisingly, the forget accuracy after unlearning SISA is quite similar to the regular retraining from scratch.

Finally, we report `SISA`'s speed-up both in the training phase compared to the regular training, as well as in the unlearning phase compared to regular retraining. Overall, while training `SISA` is slower than regular training, during unlearning, retraining `SISA` is considerably faster than regular retraining. The speed-up is especially remarkably higher in the DC task where we have a higher number of partitions and slices.

Table 9. Full/Selective deletion results for AQP/DBEst++ and SISA. Corresponding to Table 1 and Table 2.

model	Query-Retain - QR		Query-Delete - QD		likelihoods		Speedup	Queries		likelihoods		Speedup
	count	sum	count	sum	remain	delete		sum	count	remain	delete	
SISA-Train	14.74±3.15	12.92±3.39	NA	NA	NA	NA	0.77	11.68±2.36	11.31±2.66	NA	NA	0.77
SISA-Retrain	13.82±3.12	15.97±3.91	0.00±0.00	2.17±1.57	8.03	0.03	10.02	16.54±2.91	13.87±2.73	3.88	2.96	9.81

Table 10. The full and selective deletion results for SE/Naru-NeuroCard and SISA. Corresponding to Table 3.

dataset	model	Full Deletion	Selective Deletion	Speedup (full, selective)
Census	SISA-Train	62.81±2.95	62.79±2.95	0.91
	SISA-Retrain	48.43±2.31	45.47±2.21	(8.4, 7.9)

Table 11. Full and selective deletion results for DG/TVAE and SISA. Corresponding to Table 4.

dataset	model	Full Deletion		Selective Deletion		Speedup (full, selective)
		retain synth	delete synth	retain synth	deleted synth	
Census	SISA-Train	61.81±2.17	NA	NA	NA	0.84
	SISANARetrain	54.09±0.73	18.80±0.11	55.52±1.74	19.81±1.67	(7.84, 7.01)

Table 12. SISA for the DC task. Corresponding to Table 7

Method	Class Unlearning			Selective Unlearning			Speedup
	Test	Retain	Forget	Test	Retain	Forget	
SISA-Train	81.27±0.01	87.64±0.01	42.31±0.09	81.27±0.01	81.55±0.01	81.07±0.00	0.81
SISA-Retrain	77.81±0.01	90.33±0.00	0.00±0.00	80.11±0.01	80.05±0.02	80.72±0.00	22.01

7.8 Membership Inference Attacks

We perform two types of MIAs to evaluate unlearning quality across baselines. The first is a loss-based attack [24]. The second is a confidence-based attack [11]. The core idea behind these attacks is to train a binary classifier to learn to distinguish between the members (retain-set or forget-set) and the non-members (validation-set). However, achieving a high accuracy in this classifier doesn't guarantee successful membership detection if the input distributions are very different. To address this, the validation set should follow the distribution of the forget-set, while we must additionally ensure that there is no overlap of rows between those two sets.

Applying MIAs to generative models is an underexplored problem since these models do not have a clear output signal like loss or confidence that indicates memorization. Therefore, we only applied the above MIAs to the DC task.

Given the above explanations, we split the table into train, test, and validation sets. Then we randomly unlearn 5% of the train-set using different unlearning baselines. The results are reported in Table 13. In both attacks, the attack's accuracy on the original model is higher than the unlearned models, except for `NegGrad` which results in a very high confidence-based MIA accuracy.

7.9 Trade-offs

Machine Unlearning algorithms trade off three aspects: model utility, forget quality, and speed. In our experiments, model utility is quantified as the models' performance (accuracy) on downstream tasks. Forget quality is captured using likelihood, sample distribution, MIA, as well as accuracy on the forget set. Our results throughout illustrate what axes of these trade-offs are improved or sacrificed by each unlearning algorithm.

Table 13. Membership Inference Attack

Method	Test Accuracy	Retain Accuracy	Forget Accuracy	loss-MIA	confidence-MIA
Original	82.11±0.57	91.15±1.96	89.43±2.39	57.48±03.18	60.64±2.82
Retrain	82.48±1.45	90.32±2.73	72.18±3.08	49.25±02.91	54.45±3.72
Fine-tune	83.84±0.91	85.68±0.32	76.40±1.00	51.31±04.10	52.22±3.32
NegGrad	8.32±4.76	8.68±5.38	7.39±3.75	52.52±02.82	66.60±78.10
NegGrad+	72.72±26.68	72.82±25.75	63.52±24.86	50.00±03.36	48.59±11.74
SCRUB	84.29±0.65	84.44±0.27	76.27±3.09	47.10±01.99	49.12±5.03
SISA-Train	83.32±0.27	82.92±0.12	76.05±0.64	49.60±0.21	57.40±0.34
SISA-Retrain	83.04±0.22	82.72±0.37	75.40±0.87	49.60±0.66	57.72±0.46

In the exact unlearning algorithms, while `Retrain` does a perfect job of maintaining utility and forgetting at the same time, it could be very slow, as evidenced by the speedups associated with some approximate unlearning algorithms. The `SISA` version for exact unlearning can also unlearn quickly and with high quality, but the utility is not always guaranteed (as is the case for SE/Naru-NeuroCard (e.g., Table 10)).

For approximate unlearning algorithms, while all of them show speed-ups over `Retrain`, their utility vs. forgetting trade-offs are different. `NegGrad` usually does a top job of unlearning w.r.t. different metrics (likelihoods and accuracy on the deleted data). However, it catastrophically damages utility (e.g., Table 1). `NegGrad+` and `SCRUB` show better trade-offs between forgetting, utility and speed-up over `Retrain`. `Fine-tune`, on the other hand, consistently performs well by efficiently forgetting, without damaging utility, and while offering significant speedups over `Retrain`.

Comparing `SISA` exact unlearning vs the top performers of approximate unlearning, we note the following: First, in general, `SISA` speedups highly depend on how many of its models need to be retrained. This, in turn, depends on how data partitions are defined and how the forget set at any given time is distributed over these partitions. Our results show several cases where `Fine-tune` speedups are significantly higher than `SISA`'s. With respect to model utility also, for some tasks, `SISA` may underperform, likely owing to the aggregation of involved models whereby errors may accumulate. For forgetting quality, `SISA` is generally a top performer.

8 LESSONS LEARNED

Is machine unlearning in learned DBs different than updating learned DBs with new data insertions? Our investigation has revealed an interesting, perhaps surprising finding: when it comes to deleting data in learned database systems, with commonly-used models on commonly-studied tasks, simple methods (like `Fine-tune` and `NegGrad+`) do very well. This finding is interesting because it is in stark contrast with two other observations: First, in the context of learned database systems, *inserting* data is a hard problem and simple fine-tuning approaches evidently fail badly [23]. When it comes to the first discrepancy, one hypothesis is that, in the current scenarios studied in learned databases, inserting data is a harder problem than deleting data. One contributing factor to this is that the new "knowledge" that is inserted may interfere with old "knowledge". Simple methods, like `Fine-tune`, interfere with old knowledge by fine-tuning for new data, while "catastrophically forgetting" old data. The issue of catastrophic forgetting in the machine learning community is a challenging one and requires dedicated solutions that are more sophisticated than simple fine-tuning. Instead, removing knowledge does not face this difficulty. However, it is still an open problem to identify whether there are other unique difficulties associated with deleting knowledge from learned components of database systems that were not surfaced in our initial investigation

Table 14. Results for image classification (IC) on Cifar5 and Lacuna5 datasets. There are 5 classes and 100 samples in each class [11]. We unlearn class 0. An All-CNN network [38] with 3 CNN layers (much shallower than the usual models for IC tasks, which have 20+ layers). Fine-tune and SCRUB perform similarly in terms of delete-error – a very different conclusion compared to the IC results with deeper networks [24]. In terms of test- and retain-error, the results are mixed. This is an important finding: For shallower networks, the effect of more sophisticated unlearning algorithms is much less pronounced. For shallower networks, for IC tasks `Fine-tune` does as well as the others across 3 errors. But, for these shallow networks test and retain-errors are very bad, so deeper networks are needed.

model	Cifar5			Lacuna5		
	test-error	retain-error	delete-error	test-error	retain-error	delete-error
original	42.75±1.1	37.75±0.7	71.0±1.9	49.2±1.2	43.5±0.9	54.0±1.0
Retrain	43.0±1.3	30.2±0.0	100.0±0.0	46.5±1.4	44.7±0.0	100.0±0.0
Fine-tune	40.5±0.7	30.5±0.8	98.0±1.0	41.7±1.8	39.5±2.1	100.0±0.0
SCRUB	44.5±2.1	39.7±1.8	97.0±0.8	37.0±1.4	35.7±1.5	100.0±0.0

into the topic. For instance, a systematic understanding of how different aspects of the deletion problem (e.g. size and homogeneity of delete-set, relationship between delete-set and retain-set) affect results is an open problem for future research.

Is machine unlearning in learned DBs different than machine unlearning in image classification? In the context of image classification (IC), which is the common testbed for unlearning algorithms in the ML community, the simple approach of fine-tuning yields poor results [11, 12, 24]. And, interestingly, (our adaptation of) `SCRUB`, a top-performing unlearning method for IC substantially outperforming `Fine-tune` (and `NegGrad+`) on IC tasks, does not generally do better than `Fine-tune` for our DB tasks.

An important difference between the setup we study in this paper compared to the standard unlearning benchmarks in IC tasks is that the latter uses significantly deeper neural networks, with significantly more trainable parameters. Given this, we hypothesize that perhaps the success of simple approaches like fine-tuning in our case is (at least partially) due to the shallower nature of the neural networks we use. More concretely, we have 2, 5, and 4 hidden layers in our networks for the MDN in AQP/DBEst++, for the DARN in SE/Naru-NeuroCard, and for the TVAE in DG, respectively. While, for instance, [24] uses models with more than 20 hidden layers for image classification. To investigate this, we ran an experiment on the computer vision tasks with shallower models than those typically used (in Table 14), and indeed we observed that this intervention led to a significantly reduced gap in the performance of the state-of-the-art algorithms over fine-tuning. This is an important finding for this community, as it indicates that transitioning to using deeper networks in the future may come with tough growing pains for data deletion in learned data systems. It is also an important finding for ML unlearning research, as it ties unlearning performance to characteristics of the network architecture (shallower vs deeper) - a connection not previously made.

On internal-model accuracy and downstream task accuracy for unlearning. We have also learned that looking at only downstream task accuracy may be misleading. As our experiments have shown, downstream accuracy is very much dependent on dataset characteristics, delete operation characteristics, relationships between deleted data and retained data, and the downstream task itself. In some cases, even `Stale` performs well, for example. It is instructive, therefore, to use model-internal metrics (like likelihood, in cases the ML models provide them, and generated learned distributions of ML models and related distribution-distances like JS divergence) in addition which can help explain downstream task accuracy. Looking at such internal-model metrics is also very

valuable in any experimental analysis, where inevitably only a select set of experiments are (can be) performed. Thus, looking only at downstream task accuracy may be dependent on data and/or model and/or delete operation peculiarities and may be misleading.

On accuracy, overheads, and frequency of unlearning. The ideal unlearning method would ensure high accuracy on retained and deleted data at very low overheads. This is the raison d' etre of this research field; that is, to avoid running `Retrain` which (especially for very large datasets) can be prohibitively expensive. The experiments of sequential vs one-go deletions showed that different methods have different sensitivities on datasets and downstream tasks. `Fine-tune` appears to be less sensitive to these, affording the luxury of running it less frequently (i.e. on larger batches of deletion requests). So `Fine-tune` (and `NegGrad+`), in addition to their accuracy being very competitive under all tasks/models and datasets studied, and to the fact that they are very efficient methods (high speedups versus Retrain), they can also be run less frequently, as errors are shown not to be accumulating over time.

On combining algorithms for continuous learning and unlearning. The simplicity of `Fine-tune` as an unlearning algorithm does not adversely interact with more complex algorithms needed to ensure continuous learning, as new data insertions occur. We have seen that these can be combined (even deleting and inserting data in overlapping data spaces) providing a comprehensive highly accurate solution for both continuous learning and unlearning.

9 CONCLUSION

We have presented the first study of unlearning in learned database systems. This is a crucial ingredient for successfully updating models in NN-based learned DBs in the face of frequent data updates that characterize DBs. And is the only ingredient currently completely lacking thorough research and findings. Our investigation covered three different downstream tasks (AQP, SE, and DG), each employing a different generative neural-network-based model, and one upstream task based on a discriminative NN model (DC), and across three different datasets. It studied different unlearning methods, ranging from simple baselines, such as `Fine-tune` and `NegGrad`, more sophisticated methods, such as `NegGrad+`, and a state-of-the-art unlearning algorithm, `SCRUB`, adapted from the machine unlearning literature. Our investigation proposed and studied appropriate metrics including downstream-task specific, as well as model-internal specific metrics in order to substantiate and interpret results. Our results answer a large number of related key questions with respect to key learned DBs. They also point to different conclusions compared to those from research in insertion updates in learned DBs and from the ML community's findings for unlearning in image classification tasks. The work puts forth the basic skeleton (for instance, unlearning algorithm baselines, performance metrics, and downstream and upstream tasks) as well as related findings en route to a much-needed benchmark for machine unlearning in learned DBs.

ACKNOWLEDGMENTS

This work is partially sponsored by Huawei IRC and by EPSRC while doing a PhD at the University of Warwick

REFERENCES

[1] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 141–159.

[2] Yinzhi Cao and Junfeng Yang. 2015. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 463–480.

[3] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F Stewart, and Jimeng Sun. 2017. Generating multi-label discrete patient records using generative adversarial networks. In *Machine learning for healthcare conference*.

PMLR, 286–305.

[4] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, et al. 2020. ALEX: an updatable adaptive learned index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 969–984.

[5] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. 2020. Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. *arXiv preprint arXiv:2006.13282* (2020).

[6] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[7] Chongyu Fan, Jiancheng Liu, Yihua Zhang, Dennis Wei, Eric Wong, and Sijia Liu. 2023. SalUn: Empowering Machine Unlearning via Gradient-based Weight Saliency in Both Image Classification and Generation. *arXiv preprint arXiv:2310.12508* (2023).

[8] Jack Foster, Stefan Schoepf, and Alexandra Brintrup. 2023. Fast Machine Unlearning Without Retraining Through Selective Synaptic Dampening. *arXiv preprint arXiv:2308.07707* (2023).

[9] Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou. 2019. Making ai forget you: Data deletion in machine learning. *Advances in neural information processing systems* 32 (2019).

[10] Aditya Golatkar, Alessandro Achille, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. 2021. Mixed-privacy forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 792–801.

[11] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. 2020. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9304–9312.

[12] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. 2020. Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations. In *European Conference on Computer Vision*. Springer, 383–398.

[13] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2021. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems* 34 (2021), 18932–18943.

[14] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. 2019. Certified data removal from machine learning models. *arXiv preprint arXiv:1911.03030* (2019).

[15] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep learning models for selectivity estimation of multi-attribute queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1035–1050.

[16] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2019. DeepDB: Learn from Data, not from Queries! *Proceedings of the VLDB Endowment* 13, 7 (2019).

[17] Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. 2021. Approximate data deletion from machine learning models. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2008–2016.

[18] Arthur Jacot, Franck Gabriel, and Clément Hongler. 2018. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems* 31 (2018).

[19] Jinghan Jia, Jiancheng Liu, Parikshit Ram, Yuguang Yao, Gaowen Liu, Yang Liu, Pranay Sharma, and Sijia Liu. 2023. Model Sparsity Can Simplify Machine Unlearning. In *Annual Conference on Neural Information Processing Systems*.

[20] S Jian, H Kaiming, R Shaoqing, and Z Xiangyu. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision & Pattern Recognition*. 770–778.

[21] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2018. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677* (2018).

[22] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*. 489–504.

[23] Meghdad Kurmanji and Peter Triantafillou. 2023. Detect, Distill and Update: Learned DB Systems Facing Out of Distribution Data. In *Proceedings of the 2023 International Conference on Management of Data (to appear)*.

[24] Meghdad Kurmanji, Peter Triantafillou, Jamie Hayes, and Eleni Triantafillou. 2023. Towards Unbounded Machine Unlearning. *arXiv preprint arXiv:2302.09880, to appear in NeurIPS 2024, Conference in Neural Information Processing Systems* (2023).

[25] Beibin Li, Yao Lu, and Srikanth Kandula. 2022. Warper: Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts. In *Proceedings of the 2022 International Conference on Management of Data*.

[26] Guoliang Li, Xuanhe Zhou, Shifu Li, and Bo Gao. 2019. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2118–2130.

[27] Qingzhi Ma, Ali Mohammadi Shanghooshabad, Mehrdad Almasi, Meghdad Kurmanji, and Peter Triantafillou. 2021. Learned Approximate Query Processing: Make it Light, Accurate and Fast.. In *CIDR*.

[28] Qingzhi Ma and Peter Triantafillou. 2019. Dbest: Revisiting approximate query processing engines with machine learning models. In *Proceedings of the 2019 International Conference on Management of Data*. 1553–1570.

[29] Ananth Mahadevan and Michael Mathioudakis. 2021. Certifiable machine unlearning for linear models. *arXiv preprint arXiv:2106.15093* (2021).
[30] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. 2019. Neo: A learned query optimizer. *arXiv preprint arXiv:1904.03711* (2019).
[31] Vikram Nathan, Jialin Ding, Mohammad Alizadeh, and Tim Kraska. 2020. Learning multi-dimensional indexes. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 985–1000.
[32] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-loss: Learning cardinality estimates that matter. *Proceedings of the VLDB Endowment* (2021).
[33] Noseong Park, Mahmoud Mohammadi, Kshitij Gorde, Sushil Jajodia, Hongkyu Park, and Youngmin Kim. 2018. Data synthesis based on generative adversarial networks. *arXiv preprint arXiv:1806.03384* (2018).
[34] Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. 2021. Remember what you want to forget: Algorithms for machine unlearning. *Advances in Neural Information Processing Systems* 34 (2021), 18075–18086.
[35] Ali Mohammadi Shanghooshabad, Meghdad Kurmanji, Qingzhi Ma, Michael Shekelyan, Mehrdad Almasi, and Peter Triantafillou. 2021. Pgmjoins: Random join sampling with graphical models. In *Proceedings of the 2021 International Conference on Management of Data*. 1610–1622.
[36] Jiaeli Shi, Najah Ghalyan, Kostis Gourgoulias, John Buford, and Sean Moran. 2023. DeepClean: Machine Unlearning on the Cheap by Resetting Privacy Sensitive Weights using the Fisher Diagonal. *arXiv preprint arXiv:2311.10448* (2023).
[37] Tarique Siddiqui, Alekh Jindal, Shi Qiao, Hiren Patel, and Wangchao Le. 2020. Cost models for big data query processing: Learning, retrofitting, and our findings. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 99–113.
[38] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. 2014. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806* (2014).
[39] Saravanan Thirumuruganathan, Shohedul Hasan, Nick Koudas, and Gautam Das. 2020. Approximate query processing for data exploration using deep generative models. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1309–1320.
[40] Anvith Thudi, Gabriel Deza, Varun Chandrasekaran, and Nicolas Papernot. 2022. Unrolling sgd: Understanding factors influencing machine unlearning. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 303–319.
[41] Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. 2022. On the necessity of auditable algorithmic definitions for machine unlearning. In *31st USENIX Security Symposium (USENIX Security 22)*. 4007–4022.
[42] Eleni Triantafillou, Fabian Pedregosa, Jamie Hayes, Peter Kairouz, Isabelle Guyon, Meghdad Kurmanji, Gintare Karolina Dziugaite, Peter Triantafillou, Kairan Zhao, Lisheng Sun Hosoya, Julio C. S. Jacques Junior, Vincent Dumoulin, Ioannis Mitliagkas, Sergio Escalera, Jun Wan, Sohier Dane, Maggie Demkin, and Walter Reade. 2023. NeurIPS 2023 - Machine Unlearning. https://kaggle.com/competitions/neurips-2023-machine-unlearning
[43] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1009–1024.
[44] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2020. Are We Ready For Learned Cardinality Estimation? *arXiv preprint arXiv:2012.06743* (2020).
[45] Yinjun Wu, Edgar Dobriban, and Susan Davidson. 2020. Deltagrad: Rapid retraining of machine learning models. In *International Conference on Machine Learning*. PMLR, 10355–10366.
[46] Ziniu Wu, Parimarjan Negi, Alizadeh Mohammad, Tim Kraska, and Madden Samuel. 2023. FactorJoin: A New Cardinality Estimation Framework for Join Queries. 1, 1 (2023).
[47] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. 2019. Modeling tabular data using conditional gan. *Advances in Neural Information Processing Systems* 32 (2019).
[48] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: one cardinality estimator for all tables. *arXiv preprint arXiv:2006.08109* (2020).
[49] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep unsupervised cardinality estimation. *arXiv preprint arXiv:1905.04278* (2019).
[50] Ji Zhang, Yu Liu, Ke Zhou, Guoliang Li, Zhili Xiao, Bin Cheng, Jiashu Xing, Yangtao Wang, Tianheng Cheng, Li Liu, et al. 2019. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In *Proceedings of the 2019 International Conference on Management of Data*. 415–432.
[51] Zijie Zhang, Yang Zhou, Xin Zhao, Tianshi Che, and Lingjuan Lyu. 2022. Prompt certified machine unlearning with randomized gradient smoothing and quantization. *Advances in Neural Information Processing Systems* 35 (2022), 13433–13455.

[52] Johan Kok Zhi Kang, Sien Yi Tan, Feng Cheng, Shixuan Sun, and Bingsheng He. 2021. Efficient Deep Learning Pipelines for Accurate Cost Estimations Over Large Scale Query Workload. In *Proceedings of the 2021 International Conference on Management of Data*. 1014–1022.

[53] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2020. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *arXiv preprint arXiv:2011.09022* (2020).

[54] Yonghua Zhu, Weilin Zhang, Yihai Chen, and Honghao Gao. 2019. A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment. *EURASIP Journal on Wireless Communications and Networking* 2019, 1 (2019), 1–18.

Received July 2023; revised October 2023; accepted November 2023