Conflict Resolution for Improving ML Accuracy

Wenfei Fan[1,2], Xiaoyu Han[3], Hufsa Khan[1], Weilong Ren[1], Yaoshu Wang[1], Min Xie[1], Zihuan Xu[1]
[1]Shenzhen Institute of Computing Sciences [2]University of Edinburgh [3]Fudan University
{wenfei, hufsa, renweilong, yaoshuw, xiemin, xuzihuan}@sics.ac.cn, xyhan22@m.fudan.edu.cn

***Abstract*—This paper investigates how to make practical use of conflict resolution (CR) to enhance the accuracy of ML classifiers $\mathcal{M}$ on relational data. We show that applying CR to influential attributes and/or tuples can substantially improve the performance of downstream model $\mathcal{M}$. Based on this, we formulate two problems for identifying influential attributes and tuples in the data to maximize model accuracy. Although we show that both problems are intractable, we develop effective algorithms to pinpoint these critical factors. To mitigate the impact of noise introduced by imprecise CR methods, we propose a creator-critic framework that iteratively applies CR and trains $\mathcal{M}$ with the corrected data. We prove that the creator-critic process guarantees convergence to a more accurate model. Using real-life datasets, we experimentally verify that our approach improves the relative accuracy of various ML classifiers by an average of 43.1%, up to 72.5%.**

I. Introduction

Given a dataset $\mathcal{D}$, *conflict resolution* (*CR*) [1] aims to detect and correct inconsistencies in a single tuple and across multiple tuples, *e.g.,* mismatching area code and city in an address, or an actor born after the movie's release. It returns an improved dataset $\mathcal{D}_c$ for subsequent queries and applications. There has been a large body of work on CR, by using logic rules [1]–[13], ML [14]–[18], and hybrid of the two [19], [20].

There has been recent interest in *data cleaning for machine learning (ML)* [21]–[36]. The process takes an ML model $\mathcal{M}$ and a dataset $\mathcal{D}$ as input, where $\mathcal{D}$ is used to train and evaluate $\mathcal{M}$. The goal is to derive a "cleaned" dataset $\mathcal{D}_\mathcal{M}$ from $\mathcal{D}$ such that the accuracy of the downstream model $\mathcal{M}$, when trained with $\mathcal{D}_\mathcal{M}$, is maximally improved compared to training with $\mathcal{D}$. The need for this is evident. It is reported that 80% of data scientists' time is spent on cleaning datasets for ML applications [37]–[39]. Using the Cleanlab data cleaning tool [40], a Berkeley lab was able to improve the accuracy of ML models by 15% and reduces the training time by 33%.

Unlike traditional cleaning, which produces a one-size-fits-all dataset $\mathcal{D}_c$, data cleaning for ML tailors a dataset $\mathcal{D}_\mathcal{M}$ specifically to improve the model accuracy of a given model $\mathcal{M}$, by fixing inconsistencies in the training data. However, computing $\mathcal{D}_\mathcal{M}$ is a delicate process, as it depends on factors such as the type of model $\mathcal{M}$, the distribution of data in $\mathcal{D}$, and the effectiveness of the CR methods used. It is reported in [24] that CR "is more likely to have an insignificant impact and unlikely to have a negative impact on ML classification models".

We replicated the experiments in [24] and found that (1) CR only slightly improves the accuracy of $\mathcal{M}$ when applied to the entire training data, consistent with the findings in [24]; however, (2) CR can significantly enhance $\mathcal{M}$'s accuracy when applied to a carefully selected subset of the data (see Section VI).

tid	category (A_1)	gender (A_2)	education (A_3)	workhour (A_4)	occupation (A_5)	income (Y)
t_1	Never-worked	M	HS-grad	0	Exec-managerial	$< 50K$
t_2	Private	M	Bachelor	30	Transport-moving	$\leq 50K$
t_3	Local-gov	F	PhD	70	Prof.	$> 50K$
t_4	Local-gov	F	HS-grad	40	Prof.	$> 50K$
t_5	State-gov	F	Bachelor	40	Adm-clerical	$\leq 50K$
t_6	State-gov	M	Bachelor	5	Tech-support	$> 50K$
...	...	...	...	...	...	...

TABLE I: An Adult Table

Example 1: Consider Table I of real-life schema Adult [41] with category, gender, education, workhour, occupation as the attributes and income as the label. Values in red are erroneous, *e.g.,* t_1[occupation] should be `None` instead of `Exec-managerial`, t_4[education] should be `PhD`, not `HS-grad`, and t_6[workhour] should be `40`, not `5`. Such erroneous values appear to conflict with other correct values in either the same tuple or across multiple tuples, *e.g.,* (a) category and workhour of t_6 are conflicting since `State-gov` employees usually work >30 hours (similarly for category and occupation of t_1); and (b) t_3 and t_4 are inconsistent on education, given their similar behavior in other attributes. These conflicts can mislead the ML model and deteriorate its accuracy, *e.g.,* the model may learn that `Exec-managerial` comes with low salary (by t_1), which is usually not true. As will be seen in Section VI, fixing carefully selected attributes and tuples alone improves the accuracy of some classification models $\mathcal{M}$ by 43.1% on average. In contrast, if we apply CR to the entire training dataset, the accuracy of $\mathcal{M}$ improves only by 3.5%. □

This raises several questions. How can we effectively use CR to enhance ML model accuracy? Which conflicts should we fix to achieve this? When a CR method is *imprecise* (*i.e.,* its accuracy is not 100% and may introduce noise during cleaning), can we still utilize it while mitigating its negative impact?

Contributions & Organization. This paper addresses these questions, to make practical use of CR to improve ML model accuracy, focusing on differential classifiers for relational data.

(1) A framework (Section III). To utilize (imprecise) CR methods, we propose CR4ML, a creator-critic framework. Given a model $\mathcal{M}$ and a dataset $\mathcal{D}$, CR4ML iteratively cleans $\mathcal{D}$ and trains $\mathcal{M}$. In the k-th round, the critic identifies influential attributes and tuples (Sections IV and V), resolves conflicts, and obtains a cleaner dataset $\mathcal{D}_k$. The creator then incrementally trains $\mathcal{M}$ with $\mathcal{D}_k$. We show that CR4ML guarantees convergence to a stable state, producing a dataset $\mathcal{D}_\mathcal{M}$, and provides an accuracy approximation bound for $\mathcal{M}$ trained on $\mathcal{D}_\mathcal{M}$.

One may plug any (not-perfect) CR method in CR4ML, as CR4ML can mitigate the impact of noise introduced by CR.

(2) Influential attributes (Section IV). We formulate a problem

that, given a set $\mathcal{D}$ of schema $\mathcal{R}$, a model $\mathcal{M}$ and a validation set $\mathcal{C}$ (selected by the critic of CR4ML), identifies a set of influential attributes of $\mathcal{R}$ such that resolving their conflicts maximumly improves the accuracy of $\mathcal{M}$ on $\mathcal{C}$. We show that its decision problem is intractable. Nonetheless, we develop a genetic algorithm that employs the boosting strategy [42] to select a sub-optimal solution, without frequently retraining $\mathcal{M}$.

(3) Influential tuples (Section V). We formulate another problem that, given $\mathcal{M}$ and $\mathcal{C}$, pinpoints tuples of $\mathcal{D} \setminus \mathcal{C}$ on which CR can maximumly improve the accuracy of $\mathcal{M}$ on $\mathcal{C}$. Although the problem is also intractable, we identify a set $\mathcal{T}$ of influential tuples by employing influence functions [41], [43]. We resolve conflicts in $\mathcal{T}$, estimate the update to $\mathcal{M}$ and validate the model on $\mathcal{C}$, without actually retraining $\mathcal{M}$.

(4) Experimental evaluation (Section VI). Using different datasets, models and CR methods, we find the following. On average, (a) CR4ML improves the (relative) accuracy of various models by 43.1%. (b) By resolving conflicts in both influential attributes and tuples, CR4ML is 39.6% more accurate than cleaning the entire $\mathcal{D}$. (c) It is fast given an efficient and accurate CR, *e.g.,* it takes 451.6s on $\mathcal{D}$ of 13K tuples with CR method Rock [44]. (d) The more accurate CR is, the better CR4ML improves the accuracy of $\mathcal{M}$; nonetheless, CR4ML works with not-so-accurate CR and can mitigate its noise.

We discuss preliminaries in Section II, related work in Section VII and future work in Section VIII; proofs are in [45].

Novelty. The novelty of CR4ML lies in its practical approach to applying CR to enhance the accuracy of downstream ML models, including (a) a creator-critic framework that, given a dataset $\mathcal{D}$, a classifier $\mathcal{M}$ and a (imprecise) CR method $\mathcal{A}_{\mathsf{CR}}$, automatically detects a set of impactful data that can be confidently fixed by $\mathcal{A}_{\mathsf{CR}}$, improving the accuracy of $\mathcal{M}$, (b) the formulation of two problems for identifying attributes and tuples for $\mathcal{A}_{\mathsf{CR}}$ to maximize the accuracy of $\mathcal{M}$, along with the intractability of both problems; and (c) effective methods for identifying the impactful attributes and tuples, respectively.

A related topic is *data cleaning for AutoML* [46]–[50], which aims to automatically equip ML pipelines with embedded data cleaning to maximize the accuracy of $\mathcal{M}$. In this paper, we focus on data cleaning for ML since it not only helps train a better $\mathcal{M}$ but also produces a cleaned dataset $\mathcal{D}_{\mathcal{M}}$ for other use; for example, $\mathcal{D}_{\mathcal{M}}$ can serve as a better starting point for data preparation for AutoML [35].

II. Influential Attributes and Tuples

We start with basic notations. Consider a database schema $\mathcal{R} = (R_1, \ldots, R_m)$, where each R_j is a relation schema $R(A_1, \ldots, A_n, L)$, each A_i is an attribute, and L is an attribute denoting the classification label. A dataset $\mathcal{D}$ of $\mathcal{R}$ is $(D_1, \ldots, D_m)$, where D_j is a relation of R_j. Following [51], we assume *w.l.o.g.* that each tuple t in $\mathcal{D}$ represents an entity.

Accuracy. For an ML model $\mathcal{M}$ and a dataset $\mathcal{D}$, we use $\mathsf{acc}(\mathcal{M}, \mathcal{D})$ to denote the accuracy of $\mathcal{M}$ that is trained (resp. evaluated) with a training set $\mathcal{D}_{\mathsf{train}}$ (resp. testing set $\mathcal{D}_{\mathsf{test}}$) of $\mathcal{D}$; here $\mathcal{D}_{\mathsf{test}}$ will not be used in training $\mathcal{M}$ (thus no data leakage). Following [22], [32], [33], we assume that $\mathcal{D}_{\mathsf{test}}$ is clean, while $\mathcal{D}_{\mathsf{train}}$ may be erroneous; this is a common setting as $\mathcal{D}_{\mathsf{test}}$ is relatively small and can be manually checked by humans while $\mathcal{D}_{\mathsf{train}}$ may be too large for humans to handle. Moreover, one may extract a coreset as $\mathcal{D}_{\mathsf{test}}$ (see Section III).

We measure $\mathsf{acc}(\mathcal{M}, \mathcal{D})$ by macro-averaged F_1 over all classes [52], *i.e.,* $\mathsf{acc}(\mathcal{M}, \mathcal{D}) = \frac{1}{N} \sum_l 2 \times \frac{\mathsf{rec}_l \times \mathsf{pre}_l}{\mathsf{rec}_l + \mathsf{pre}_l}$, where N is the number of classes, l is a classification label, rec_l is the ratio of l-class tuples (*i.e.,* tuples with label l) correctly predicted by $\mathcal{M}$ to all l-class tuples, and pre_l is the ratio of l-class tuples correctly predicated by $\mathcal{M}$ to all tuples with predicated class l.

Influential data. Unlike previous works (*e.g.,* [24], [27]) that apply CR methods to the entire $\mathcal{D}$, we clean $\mathcal{D}$ more selectively for ML models. As will be shown in Section VI, applying CR to a subset of impactful data in $\mathcal{D}$ improves $\mathcal{M}$'s accuracy more effectively than cleaning the full dataset. This is because: (a) a CR method's confidence/accuracy varies across different conflicts in $\mathcal{D}$, and (b) resolving certain conflicts impacts $\mathcal{M}$'s training differently. We argue that both $\mathcal{M}$ and CR should be considered when cleaning $\mathcal{D}$ to improve $\mathcal{M}$'s accuracy.

Given a dataset $\mathcal{D}$ of $\mathcal{R}$, a classifier $\mathcal{M}$ and a CR method $\mathcal{A}_{\mathsf{CR}}$, we denote by $\mathcal{D}_{\mathsf{CR}}$ the cleaned $\mathcal{D}$ via $\mathcal{A}_{\mathsf{CR}}$. The difference between $\mathsf{acc}(\mathcal{M}, \mathcal{D})$ and $\mathsf{acc}(\mathcal{M}, \mathcal{D}_{\mathsf{CR}})$ is the accuracy improvement of $\mathcal{M}$. In particular, if $\mathcal{A}_{\mathsf{CR}}$ only cleans a subset $\mathcal{S}$ of attributes in all tuples of $\mathcal{D}$ (resp. all attributes of a subset $\mathcal{T}$ of tuples in $\mathcal{D}$), we denote the cleaned set by $\mathcal{D}^{\mathcal{S}}_{\mathsf{CR}}$ (resp. $\mathcal{D}^{\mathcal{T}}_{\mathsf{CR}}$).

Below we define *influential attributes* and *influential tuples*.

Influential attributes. Given a dataset $\mathcal{D}$ of schema $\mathcal{R}$, we say that $\mathcal{S} \subseteq \mathcal{R}$ is a set of *influential attributes* for $\mathcal{M}$ *w.r.t.* $\mathcal{D}$ if for any set $\mathcal{S}' \subseteq \mathcal{R}$, $\mathsf{acc}(\mathcal{M}, \mathcal{D}^{\mathcal{S}}_{\mathsf{CR}}) \geq \mathsf{acc}(\mathcal{M}, \mathcal{D}^{\mathcal{S}'}_{\mathsf{CR}})$. Intuitively, it means that cleaning the $\mathcal{S}$-attribute values in $\mathcal{D}$ maximumly improves the accuracy of $\mathcal{M}$, compared with other attributes.

In Example 1, the set $\mathcal{S}$ consists of education (A_3) and occupation (A_5), since they contain erroneous values which have a strong impact on income, *e.g.,* $t_1[A_5]$ may mislead $\mathcal{M}$ to conclude that `Exec-managerial` is a low-paying job; and $t_4[A_3]$ may deceive $\mathcal{M}$ that `HS-grad` has high income.

Influential tuples. A subset $\mathcal{T}$ of $\mathcal{D}$ is a set of *influential tuples* for $\mathcal{M}$ *w.r.t.* $\mathcal{D}$ if for any $\mathcal{T}' \subseteq \mathcal{D}$, $\mathsf{acc}(\mathcal{M}, \mathcal{D}^{\mathcal{T}}_{\mathsf{CR}}) \geq \mathsf{acc}(\mathcal{M}, \mathcal{D}^{\mathcal{T}'}_{\mathsf{CR}})$. Intuitively, cleaning tuples in $\mathcal{T}$ maximumly improves the accuracy of $\mathcal{M}$, compared with other tuples in $\mathcal{D}$.

In Example 1, erroneous values in $\mathcal{T} = \{t_1, t_6\}$ reduce the accuracy of $\mathcal{M}$, regardless of whether they are in influential attributes (*e.g.,* t_1) or non-influential ones (*e.g.,* t_6), *e.g.,* $\mathcal{M}$ may be misled that `Exec-managerial` are poorly paid by t_1, and inactive workers get high salary by t_6. In practice, tuples from under-represented classes are more likely to be influential, since disturbance on them mostly affects the data distribution.

III. A Creator-Critic Framework

This section proposes CR4ML, a creator-critic framework that integrates ML training and conflict resolution, to ensure that the ML model is more robust and accurate on unseen data.

Overview. CR4ML takes as input a differentiable classifier $\mathcal{M}$,

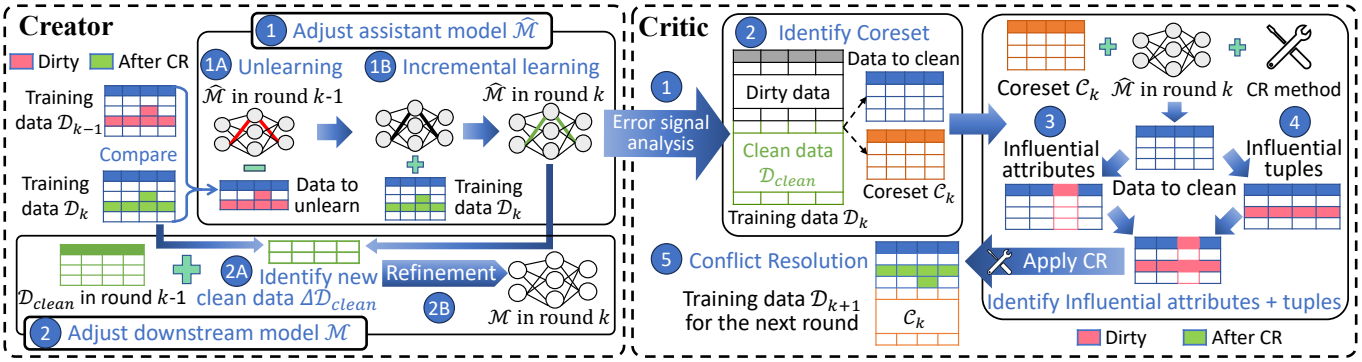


Fig. 1: Overview of CR4ML

a CR method $\mathcal{A}_{\mathsf{CR}}$, a schema $\mathcal{R}$, a dirty dataset $\mathcal{D}$ of $\mathcal{R}$, and a number e of epochs. Here $\mathcal{A}_{\mathsf{CR}}$ may be imprecise; it may use rules, ML, or a hybrid of the two. It is to mitigate the impact of noise introduced by $\mathcal{A}_{\mathsf{CR}}$ that CR4ML is developed.

CR4ML iterates a creator and a critic in *rounds* (Figure 1). In the k-th round, it works on a dirty dataset $\mathcal{D}_k$ and aims to produce a cleaner set $\mathcal{D}_{k+1}$ from $\mathcal{D}_k$ by using $\mathcal{A}_{\mathsf{CR}}$, so that $\mathcal{D}_{k+1}$ provides more clean tuples to train $\mathcal{M}$. In particular, CR4ML integrates CR and ML training, where the creator focuses on ML training and the critic targets conflict resolution. Initially, we set $\mathcal{D}_0 = \mathcal{D}$. Besides, we maintain the following datasets/models during the iterative process of CR4ML:

- A set $\mathcal{D}_{\mathsf{clean}}$, accumulating clean tuples obtained so far.
- Model $\mathcal{M}$, the targeted downstream model, which is only trained on the clean data $\mathcal{D}_{\mathsf{clean}}$ in each round.
- Model $\hat{\mathcal{M}}$, an assistant model with the same architecture and initial parameters as $\mathcal{M}$. It is incrementally trained on (dirty) $\mathcal{D}_k$ in each round, to identify (a) influential data in $\mathcal{D}_k$ to clean and (b) new clean data to be added to $\mathcal{D}_{\mathsf{clean}}$.

This process continues until $\mathcal{D}_k$ cannot be further cleaned; the set $\mathcal{D}_k$ and the tuned $\mathcal{M}$ are returned as the results. More specifically, in the k-th round, CR4ML works as follows.

The creator is responsible for *training models* $\mathcal{M}$ and $\hat{\mathcal{M}}$:

(1) Given the assistant model $\hat{\mathcal{M}}$ that has been trained on data $\mathcal{D}_{k-1}$ in the (k-1)-th round, we first update $\hat{\mathcal{M}}$ on $\mathcal{D}_k$ (*i.e.,* the dataset cleaned from $\mathcal{D}_{k-1}$) to identify influential data.

(2) We *identify new clean data from* $\mathcal{D}_k$ using the updated $\hat{\mathcal{M}}$, add it into $\mathcal{D}_{\mathsf{clean}}$ and conduct *model refinement* on $\mathcal{M}$.

The critic is responsible for *cleaning data* in five steps:

(1) Compute the *error signal* of each tuple in $\mathcal{D}_k$ using $\hat{\mathcal{M}}$.

(2) Find a relatively clean *coreset* [53] from $\mathcal{D}_k$ for validation.

(3) Identify *influential attributes* via error signals and coresets.

(4) Identify *influential tuples* in $\mathcal{D}_k$ similarly to step (3).

(5) Use $\mathcal{A}_{\mathsf{CR}}$ to resolve conflicts in influential attributes/tuples, and return a cleaner $\mathcal{D}_{k+1}$ to train $\mathcal{M}$ and $\hat{\mathcal{M}}$ in next round.

Below we detail the creator and critic (Section III-A), and present the workflow and properties of CR4ML (Section III-B).

A. Model Training and Conflict Resolution

Creator. The creator incrementally trains (a) $\hat{\mathcal{M}}$ with the updated $\mathcal{D}_k$ such that $\hat{\mathcal{M}}$ can distinguish dirty/clean tuples, and (b) $\mathcal{M}$ with clean data so that it can generalize to unseen data.

- *Input*: Models $\hat{\mathcal{M}}$ and $\mathcal{M}$ trained in (k-1)-th round, $\mathcal{D}_k$ and $\mathcal{D}_{k\text{-}1}$ from the (k-1)-th and (k-2)-th rounds, respectively, $\mathcal{D}_{\mathsf{clean}}$ accumulated so far, and a number e of epochs.
- *Output*: An extended $\mathcal{D}_{\mathsf{clean}}$ with more clean data, and updated model $\hat{\mathcal{M}}$ (resp. $\mathcal{M}$) trained on $\mathcal{D}_k$ (resp. $\mathcal{D}_{\mathsf{clean}}$).

Below we detail the training of $\hat{\mathcal{M}}$ and $\mathcal{M}$.

(1) Assistant model $\hat{\mathcal{M}}$. The training for $\hat{\mathcal{M}}$ consists of two parts: (A) *model unlearning* to eliminate the negative effect of dirty data in $\mathcal{D}_{k-1}$, on which $\hat{\mathcal{M}}$ is previously trained; and (B) *incremental learning* to update $\hat{\mathcal{M}}$ with cleaned tuples in $\mathcal{D}_k$.

(1A) Model unlearning. Denote by $\Delta\mathcal{D}_k = \{t \mid t \in \mathcal{D}_{k-1}$ and $t \notin \mathcal{D}_k\}$ the identified dirty tuples to be cleaned in round k. We use the influence function [43] to debias the negative effects of $\Delta\mathcal{D}_k$ learned by $\hat{\mathcal{M}}$ in round k-1, *i.e.,* mimicking the scenario that we retrain $\hat{\mathcal{M}}$ with the data $\mathcal{D}_{k-1} \setminus \Delta\mathcal{D}_k$, removing dirty tuples in $\Delta\mathcal{D}_k$ from $\mathcal{D}_{k-1}$, in the (k-1)-th round.

More specifically, for each tuple $t \in \Delta\mathcal{D}_k$ with attributes x and label y, we can upweight t by an infinitesimal amount ϵ; the influence of upweighting t on parameters of $\hat{\mathcal{M}}$ is

$$\mathcal{I}_{\mathsf{up,params}}(t) \stackrel{\mathsf{def}}{=} \frac{\mathrm{d}\theta_{\epsilon,t}}{\mathrm{d}\epsilon}\Big|_{\epsilon=0} = -H_\theta^{-1} \nabla_{\theta_{k-1}} l(\hat{\mathcal{M}}(x;\theta_{k-1}), y),$$

where θ_k is the parameter of $\hat{\mathcal{M}}$ in the k-th round, $\nabla_{\theta_{k-1}}$ is the partial derivation of θ_{k-1} *w.r.t.* ϵ, H_θ is the Hessian Matrix, and $l(\hat{\mathcal{M}}(x;\theta_k), y)$ is the loss of tuple t in the k-th round. Note that unlearning a tuple $t \in \mathcal{D}_{k-1}$ is the same as upweighting it by $\epsilon = -\frac{1}{|D_{k-1}|}$, such that θ_k in the k-th round can be updated as $\theta_k = \theta_{k-1} - \frac{1}{|D_{k-1}|} \sum_{t \in \Delta\mathcal{D}_k} \mathcal{I}_{\mathsf{up,params}}(t)$.

(1B) Incremental learning. Prior incremental strategies [54], [55] focus on retaining the performance of $\hat{\mathcal{M}}$ on both old and new data, *i.e.,* $\mathcal{D}_{k-1}$ and $\mathcal{D}_k$. In contrast, since $\mathcal{D}_k$ is cleaned from $\mathcal{D}_{k-1}$, we adopt a simple strategy that continuously fine-tunes $\hat{\mathcal{M}}$ with updated $\mathcal{D}_k$ in e epochs, where $\hat{\mathcal{M}}$ inherits parameters from model unlearning. It works well since the negative effects of dirty data in $\mathcal{D}_{k-1}$ have already been removed.

(2) Downstream model $\mathcal{M}$. We train $\mathcal{M}$ by (A) *identifying new clean data* and (B) *refining model* on accumulated $\mathcal{D}_{\mathsf{clean}}$.

(2A) Identifying new clean data. The creator first identifies new clean data $\Delta\mathcal{D}_{\mathsf{clean}}$ from $\mathcal{D}_k$ with the help of assistant model $\hat{\mathcal{M}}$. To do this, we initialize $\Delta\mathcal{D}_{\mathsf{clean}}$ to be empty and

compute the dynamic loss $L_k(t)$ for each tuple t in $\mathcal{D}_k$ [56], using the exponential moving average (EMA):

$$L_k(t) = \lambda \cdot l(\hat{\mathcal{M}}(x;\theta_k), y) + (1-\lambda) \cdot L_{k-1}(t),$$

where $t = (x, y) \in \mathcal{D}_k$, $\lambda \in [0, 1]$ is a discounting factor, and $L_k(t_i)$ is the accumulated loss of t from 0 to k epochs. Intuitively, the larger $L_k(t)$, the more likely t is dirty.

For each (uncleaned) tuple $t \in \mathcal{D}_k$, we monitor the dynamic loss $L_k(t)$ of $\hat{\mathcal{M}}$. Given two thresholds η_1 and η_2 for identifying clean training data for $\mathcal{M}$ (see below), we add a tuple t into $\Delta\mathcal{D}_{\mathsf{clean}}$ if one of the following is satisfied:

- $L_{k-1}(t) - L_k(t) \geq \eta_1$, indicating that $\mathcal{A}_{\mathsf{CR}}$ has effectively enhanced t's quality and thus t can be considered as clean.
- $L_k(t) \leq \eta_2$, indicating that regardless of whether t has been cleaned by $\mathcal{A}_{\mathsf{CR}}$, it is likely to be clean due to its low $L_k(t)$.

Let Q_1^1 and Q_3^1 (resp. Q_1^2 and Q_3^2) be the first and third quartiles of the distribution of $L_{k-1}(t)-L_k(t)$ (resp. $L_k(t)$) in $\mathcal{D}$, respectively [57]. We set $\eta_1 = Q_3^1 + 1.5 \cdot \mathsf{IQR}^1$ and $\eta_2 = Q_1^2 - 1.5 \cdot \mathsf{IQR}^2$, where $\mathsf{IQR}^1 = Q_3^1 - Q_1^1$, and $\mathsf{IQR}^2 = Q_3^2 - Q_1^2$.

(2B) Model refinement. CR4ML adopts Stochastic Gradient Descent (SGD) [58] for iterative model refinement, to leverage insights from both new clean data $\Delta\mathcal{D}_{\mathsf{clean}}$ and the accumulated clean data $\mathcal{D}_{\mathsf{clean}}$. We adapt SGD for continuous integration of clean data, such that $\mathcal{M}$ can be progressively improved with more clean data accumulated (details shown in [45]).

Finally, we accumulate $\mathcal{D}_{\mathsf{clean}} = \Delta\mathcal{D}_{\mathsf{clean}} \cup \mathcal{D}_{\mathsf{clean}}$.

Critic. Based on updated $\hat{\mathcal{M}}$ from the creator, the critic analyzes the error signals and retrieves a relatively clean subset from $\mathcal{D}_k$ as the *coreset* [53], [59]. The critic then uses the coreset as validation data to identify influential attributes (see Section IV) and influential tuples (Section V) from $\mathcal{D}_k$ on which the CR method $\mathcal{A}_{\mathsf{CR}}$ is applied. It returns a cleaner set $\mathcal{D}_{k+1}$ for incrementally training $\hat{\mathcal{M}}$ and $\mathcal{M}$ in the next round.

- *Input:* Updated $\hat{\mathcal{M}}$, $\mathcal{D}_k$ from the $(k-1)$-th round, and $\mathcal{A}_{\mathsf{CR}}$.
- *Output:* A cleaner set $\mathcal{D}_{k+1}$ for the next round.

Specifically, the critic consists of the following five steps.

(1) Error signal analysis. Intuitively, the error signal of a tuple $t \in \mathcal{D}_k$, denoted by $\mathsf{err}(t)$, reflects the potential inaccuracy possibly introduced by training $\hat{\mathcal{M}}$ with $t \in \mathcal{D}_k$. To compute the error signal of t, we transform the dynamic loss $L_k(t)$ to a softmax probability, *i.e.,* we compute $\mathsf{err}(t)$ as follows:

$$\mathsf{err}(t) = \mathcal{P}(t) = \frac{\exp[-\pi L_k(t)]}{\sum_{t' \in \mathcal{D}_k} \exp[-\pi L_k(t')]},$$

where π is a parameter controlling the probability distribution.

(2) Coreset identification. Based on the error signals, the critic identifies a coreset $\mathcal{C}_k \subseteq \mathcal{D}_k$. Following [56], we add a tuple $t \in \mathcal{D}_k$ into $\mathcal{C}_k$ with probability $\mathcal{P}(t) = \mathsf{err}(t)$, such that $\mathcal{C}_k$ retains the underlying distribution and the diversity of $\mathcal{D}_k$. Note that if a dirty tuple is added into $\mathcal{C}_k$, it has a high probability to be excluded from $\mathcal{C}_{k+1}$ in the next round, and therefore, it can be eventually detected and cleaned.

(3) Identifying influential attributes. Given $\mathcal{C}_k$, the critic finds a set of influential attributes in $\mathcal{R}$ so that if we resolve conflicts in those attributes using $\mathcal{A}_{\mathsf{CR}}$ and fine-tune $\hat{\mathcal{M}}$ on $\mathcal{D}_k \setminus \mathcal{C}_k$, the accuracy of $\hat{\mathcal{M}}$ is maximumly improved on $\mathcal{C}_k$. However, this problem is NP-hard and frequently fine-tuning $\hat{\mathcal{M}}$ is costly. We will develop an effective solution in Section IV.

(4) Pinpointing influential tuples. Detecting influential tuples is also NP-hard, which is costly since we need to frequently update the model (due to data modification) to decide what tuples are truly influential. We will deal with this in Section V.

(5) CR. We adopt $\mathcal{A}_{\mathsf{CR}}$ to resolve the conflicts in the influential attributes of all tuples and all the attributes of influential tuples, yielding a cleaner set $\mathcal{D}_{k+1}$ for training in the next round.

Example 2: Consider $\mathcal{D} = \{t_1\text{-}t_6\}$ in Table I. In the first round, the creator trains $\hat{\mathcal{M}}$ using $\mathcal{D}_0 = \mathcal{D}$ and outputs $\hat{\mathcal{M}}$ to the critic, which executes the five steps. It (1) computes $\mathsf{err}(t)$ of each tuple t and (2) samples tuples with probability $\mathsf{err}(t)$. After $\mathcal{C}_0 = \{t_2, t_3, t_5\}$ is identified as the coreset, the critic (3) identifies influential attributes, *e.g.,* $\{A_3, A_5\}$, and (4) influential tuples, *e.g.,* $\mathcal{T} = \{t_1\}$, using $\mathcal{C}_0$ as the validation data. Finally, (5) the critic cleans influential data via $\mathcal{A}_{\mathsf{CR}}$ and gets cleaner $\mathcal{D}_1$ by updating $t_1[A_5] = \texttt{None}$ and $t_4[A_3] = \texttt{PhD}$. □

B. Workflow and Property

As shown in Figure 2, CR4ML takes as input $\mathcal{M}$, $\mathcal{R}$, $\mathcal{D}$, $\mathcal{A}_{\mathsf{CR}}$, and the number e of epochs. It outputs a cleaned $\mathcal{D}_{\mathcal{M}}$ (*i.e.,* $\mathcal{D}_{\mathsf{clean}}$) and a fine-tuned $\mathcal{M}$ with $\mathcal{D}_{\mathcal{M}}$. It initializes $\mathcal{D}_0 = \mathcal{D}$, $\hat{\mathcal{M}} = \mathcal{M}$, $\mathcal{D}_{\mathsf{clean}} = \Delta\mathcal{D}_{\mathsf{clean}} = \emptyset$ (line 1). It trains $\hat{\mathcal{M}}$ and $\mathcal{M}$ with the creator, and cleans $\mathcal{D}$ with the critic, in rounds (lines 2-15). In each round, the creator first computes $\Delta\mathcal{D}_k$, the differences between $\mathcal{D}_k$ and $\mathcal{D}_{k-1}$ (line 3). Then it eliminates the effect of $\Delta\mathcal{D}_k$ from $\hat{\mathcal{M}}$ via model unlearning (line 4) and incrementally trains $\hat{\mathcal{M}}$ with $\mathcal{D}_k$ in e epochs (line 5). It computes the dynamic loss (line 6), and identifies new clean data $\Delta\mathcal{D}_{\mathsf{clean}}$ (line 7). If $\Delta\mathcal{D}_{\mathsf{clean}} \neq \emptyset$, it trains $\mathcal{M}$ with $\mathcal{D}_{\mathsf{clean}}$ and $\Delta\mathcal{D}_{\mathsf{clean}}$ in e epochs (lines 8-9). It updates $\mathcal{D}_{\mathsf{clean}} = \mathcal{D}_{\mathsf{clean}} \cup \Delta\mathcal{D}_{\mathsf{clean}}$, and sets $\Delta\mathcal{D}_{\mathsf{clean}} = \emptyset$ (line 10).

The critic is then executed to find influential data in $\mathcal{D}_k$. To this end, the critic first analyzes the error signals (line 11), based on which it identifies a relatively clean subset $\mathcal{C}_k$ of $\mathcal{D}_k$ as the coreset (line 12). The coreset $\mathcal{C}_k$ is used as the validation data to identify a subset $\mathcal{S}$ of influential attributes and a subset $\mathcal{T}$ of influential tuples (lines 13-14). The CR method $\mathcal{A}_{\mathsf{CR}}$ is then applied on these attributes and tuples to get a better dataset $\mathcal{D}_{k+1}$ for the next round (line 15).

The process proceeds until it is *stable*, *i.e.,* $\mathcal{D}_k$ stabilizes (lines 16-18). We take $\mathcal{D}_k$ as $\mathcal{D}_{\mathsf{clean}}$ and fine-tune $\mathcal{M}$ on $\mathcal{D}_{\mathsf{clean}}$ (line 17) and finally, return improved $\mathcal{M}$ and $\mathcal{D}_{\mathsf{clean}}$ (line 18).

Termination. CR4ML converges at a stable state if $\mathcal{A}_{\mathsf{CR}}$ is α-accurate ($\alpha \in (0.5, 1]$), *i.e.,* it correctly resolves conflicts with probability α; the cleaned data does not raise error signal.

Theorem 1: *Given an α-accurate $\mathcal{A}_{\mathsf{CR}}$,* CR4ML *guarantees to return (a) a cleaned set $\mathcal{D}_{\mathsf{clean}}$ as $\mathcal{D}_{\mathcal{M}}$, and (b) a model $\mathcal{M}$ trained with $\mathcal{D}_{\mathsf{clean}}$ such that* $\mathsf{acc}(\mathcal{M}, \mathcal{D}_{\mathsf{clean}}) > \mathsf{acc}(\mathcal{M}, \mathcal{D})$. □

Proof sketch: Observe the following: (1) $\mathcal{A}_{\mathsf{CR}}$ is α-accurate ($\alpha > 50\%$); meaning that it cleans data and approaches error-free CR that resolves all conflicts through iterations; (2) the convergence of SGD is ensured by unbiased gradient estimates and Robbins-Monro step-size conditions; and (3) training models with clean data, rather than dirty data, approximates the true data distribution and improves test accuracy [21], [60], [61]. Taken together, these guarantee that by iteratively applying $\mathcal{A}_{\mathsf{CR}}$ and optimizing model parameters, CR4ML terminates with $\mathcal{D}_{\mathcal{M}}$, which improves the accuracy of $\mathcal{M}$. □

Theorem 1 shows that CR4ML works with any CR tool $\mathcal{A}_{\mathsf{CR}}$ as long as $\mathcal{A}_{\mathsf{CR}}$ is α-accurate and $\alpha > 0.5$, which is common in practice. It iteratively fine-tunes $\mathcal{M}$ and cleans data with $\mathcal{A}_{\mathsf{CR}}$, and mitigates the impact of imprecise $\mathcal{A}_{\mathsf{CR}}$. One may assess the accuracy α of $\mathcal{A}_{\mathsf{CR}}$ with a test that computes its precision and recall on a small set of manually labeled data.

While identifying the optimal influential attributes and tuples is NP-hard, CR4ML provides an approximation bound for the expected accuracy of the model trained on the cleaned data.

Theorem 2: *Given an α-accurate $\mathcal{A}_{\mathsf{CR}}$ ($\alpha > 0.5$), CR4ML achieves an expected submodular improvement in model accuracy. The expected accuracy $\mathbb{E}(\mathsf{acc}(\mathcal{M}, \mathcal{D}_{\mathsf{clean}}))$ of the model $\mathcal{M}$ trained on the cleaned data $\mathcal{D}_{\mathsf{clean}}$ is guaranteed to reach at least 63% of the optimal expected accuracy.* □

Proof sketch: Observe the following: (1) Since $\mathcal{A}_{\mathsf{CR}}$ is α-accurate ($\alpha > 50\%$), each iteration resolves conflicts and progressively yields a cleaner dataset, such that the model's accuracy improves as more influential attributes and tuples are cleaned. (2) The accuracy gains remain stable across iterations, guaranteeing consistent improvement, as supported by the convergence of the SGD training process [62], [63]. (3) The gain functions are both monotonic (*i.e.,* more cleaning always improves accuracy) and submodular (*i.e.,* marginal gains decrease as more data is cleaned). These enable the use of submodular optimization results [64], ensuring that a greedy algorithm achieves at least 63% of the optimal accuracy improvement. □

Remark. (1) When model $\mathcal{M}$ (*e.g.,* FT-Transformer [65]) is non-convex, H_θ^{-1} does not exist due to negative eigenvalues in H_θ. We follow [43] and add a small dampening coefficient to the matrix's diagonal to obtain H_θ^{-1}. Moreover, we show in Theorem 1 [45] that CR4ML consistently reduces dynamic loss and enhances model accuracy under non-convex conditions. (2) CR4ML obtains the coreset as validation data rather than for training as in previous works (*e.g.,* [23]), by adapting the idea in [53] from images to relations. (3) Rather than using influence functions to directly rank tuples, CR4ML applies it to estimate the parameter change of models, based on which CR4ML mitigates the effect of dirty tuples on models (*i.e.,* model unlearning) and identifies influential tuples (Section V).

Example 3: Continuing with Example 2, given that that tuples t_1 and t_4 are updated in the second round ($\Delta\mathcal{D}_1 = \{t_1, t_4\}$), the creator eliminates their effect from $\hat{\mathcal{M}}$ by model unlearning and incrementally learns $\hat{\mathcal{M}}$ on $\mathcal{D}_1$. Given $\hat{\mathcal{M}}$, the critic recomputes error signals, based on which a new $\mathcal{C}_1$ is identified.

Input: An ML model $\mathcal{M}$, a data schema $\mathcal{R}$, a set $\mathcal{D}$ of $\mathcal{R}$, a CR method $\mathcal{A}_{\mathsf{CR}}$, the number e of epochs.
Output: A fine-tuned ML model $\mathcal{M}$ and a cleaned set $\mathcal{D}_{\mathsf{clean}}$.

1. $k := 0$; $\mathcal{D}_0 := \mathcal{D}$; $\hat{\mathcal{M}} := \mathcal{M}$; $L_k := \emptyset$; $\{\mathcal{D}_{\mathsf{clean}}, \Delta\mathcal{D}_{\mathsf{clean}}\} := \emptyset$;
2. **while** true **do**
   /* Creator */
3. $\Delta\mathcal{D}_k := \mathsf{CheckUpdates}(\mathcal{D}_k, \mathcal{D}_{k-1})$;
4. Model unlearning of $\hat{\mathcal{M}}$ with $\Delta\mathcal{D}_k$;
5. Incrementally fine-tune $\hat{\mathcal{M}}$ with $\mathcal{D}_k$ in e epochs;
6. Compute $L_k(t)$ for all $t \in \mathcal{D}_k/\mathcal{D}_{\mathsf{clean}}$;
7. $\Delta\mathcal{D}_{\mathsf{clean}} := \mathsf{CleanDataIdentify}(\mathcal{D}_k/\mathcal{D}_{\mathsf{clean}}, L_k, L_{k-1})$;
8. **if** $\Delta\mathcal{D}_{\mathsf{clean}} \neq \emptyset$ **then**
9. Train $\mathcal{M}$ with $\mathcal{D}_{\mathsf{clean}}$ and $\Delta\mathcal{D}_{\mathsf{clean}}$ in e epochs;
10. $\mathcal{D}_{\mathsf{clean}} := \mathcal{D}_{\mathsf{clean}} \cup \Delta\mathcal{D}_{\mathsf{clean}}$; $\Delta\mathcal{D}_{\mathsf{clean}} := \emptyset$;
    /* Critic */
11. $\mathsf{err} := \mathsf{ErrSignalsAnalysis}(\hat{\mathcal{M}}, \mathcal{D}_k)$;
12. $\mathcal{C}_k := \mathsf{CoresetIdentify}(\mathcal{D}_k, \mathsf{err})$;
13. $\mathcal{S} := \mathsf{InfluentialAttributes}(\mathcal{D}_k, \mathcal{R}, \hat{\mathcal{M}}, \mathcal{A}_{\mathsf{CR}}, \mathcal{C}_k, \mathcal{D}_{\mathsf{clean}})$;
14. $\mathcal{T} := \mathsf{InfluentialTuples}(\mathcal{D}_k, \hat{\mathcal{M}}, \mathcal{A}_{\mathsf{CR}}, \mathcal{C}_k, \mathcal{D}_{\mathsf{clean}})$;
15. $\mathcal{D}_{k+1} := \mathcal{A}_{\mathsf{CR}}(\mathcal{D}_k, \mathcal{T}, \mathcal{S})$;
16. **if** $\mathcal{D}_{k+1}$ remains unchanged from $\mathcal{D}_k$ **then**
17. $\mathcal{D}_{\mathsf{clean}} := \mathcal{D}_k$; Train $\mathcal{M}$ with $\mathcal{D}_{\mathsf{clean}}$ in e epochs;
18. **return** $(\mathcal{M}, \mathcal{D}_{\mathsf{clean}})$;
19. $k := k + 1$;

Fig. 2: The workflow of CR4ML

Assume that η_1=0.04, η_2=0.02, $L_1(t_1)$=0.04, $L_0(t_1)$=0.08, $L_1(t_5)$=0.01. Then t_1 and t_5 are added to $\Delta\mathcal{D}_{\mathsf{clean}}$ since (a) t_1 is enhanced by $\mathcal{A}_{\mathsf{CR}}$ with high loss reduction (*i.e.,* $L_1(t_1) - L_0(t_1) \geq \eta_1$), and (b) t_5 has a low loss (*i.e.,* $L_1(t_5) \leq \eta_2$). After identifying $\Delta\mathcal{D}_{\mathsf{clean}}$, $\mathcal{M}$ is trained with $\Delta\mathcal{D}_{\mathsf{clean}}$ and $\mathcal{D}_{\mathsf{clean}}$ (=$\emptyset$), and we update $\mathcal{D}_{\mathsf{clean}} = \mathcal{D}_{\mathsf{clean}} \cup \Delta\mathcal{D}_{\mathsf{clean}}$. After identifying influential data as before, we obtain a cleaned $\mathcal{D}_2$ by fixing $t_6[A_4] = \texttt{40}$. Since all conflicts are now resolved, $\mathcal{D}_2$ is stable and we incrementally train $\mathcal{M}$ on $\mathcal{D}_{\mathcal{M}} = \mathcal{D}_2$. □

Complexity. CR4ML takes $O(|\mathcal{R}| \cdot |\mathcal{D}_k| \cdot (|\mathcal{D}_k| \cdot c_{\mathsf{ul}} + 2e \cdot c_{\mathsf{ft}} + 5|\mathcal{D}_k| + c_{\mathsf{IA}} + c_{\mathsf{IT}} + c_{\mathsf{CR}}) + e \cdot c_{\mathsf{ft}})$ time, where c_{ul} is the cost of eliminating the effect of a dirty tuple in $\mathcal{D}_{k-1}$ from $\hat{\mathcal{M}}$, c_{ft} is the cost of fine-tuning $\hat{\mathcal{M}}$ (resp. $\mathcal{M}$) with $\mathcal{D}_k$ (resp. $\mathcal{D}_{\mathsf{clean}} \cup \Delta\mathcal{D}_{\mathsf{clean}}$) in one epoch, e is the number of epochs, c_{IA} is the cost of finding a set of influential attributes in $\mathcal{R}$, c_{IT} is the cost of getting a set of influential tuples in $\mathcal{D}_k$, and c_{CR} is the cost of cleaning $\mathcal{D}_k$ by $\mathcal{A}_{\mathsf{CR}}$. Here c_{CR} is often the most costly factor. We will see in Section VI that the CR4ML is fast when the embedded $\mathcal{A}_{\mathsf{CR}}$ and $\mathcal{M}$ are efficient.

The worst-case complexity above assumes that CR4ML takes $|\mathcal{R}| \cdot |\mathcal{D}_k|$ rounds, cleaning one value per round. In practice, CR4ML needs much less rounds, *e.g.,* 7 rounds on average (Section VI), since in each round it cleans all dirty values for newly identified influential data, not just one.

IV. Identifying influential attributes

This section formulates the influential attribute identifying problem, shows its intractability, and develops an algorithm.

Problem. For each dataset $\mathcal{D}_k$ of schema $\mathcal{R}$, we split it into a training set $\mathcal{B}_k$ and a validation set $\mathcal{C}_k$; here $\mathcal{C}_k$ is the coreset identified via error signals, and $\mathcal{B}_k = \mathcal{D}_k \setminus \mathcal{C}_k$. Abusing notation, we also use $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k)$ to denote the accuracy of

$\hat{\mathcal{M}}$ trained/fine-tuned (resp. evaluated) with $\mathcal{B}_k$ (resp. $\mathcal{C}_k$).

Given $\mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k)$, a CR method $\mathcal{A}_{\mathsf{CR}}$, and a model $\hat{\mathcal{M}}$, we identify a set $\mathcal{S}$ of influential attributes, such that if we resolve conflicts in the $\mathcal{S}$-attributes in $\mathcal{B}_k$, the accuracy of $\hat{\mathcal{M}}$ on $\mathcal{C}_k$ is maximumly improved. Denote the cleaned $\mathcal{B}_k$ by $\mathcal{B}_k^{\mathcal{S}}$. Then we obtain a cleaned version $\mathcal{D}_k^{\mathcal{S}} = (\mathcal{B}_k^{\mathcal{S}}, \mathcal{C}_k)$ of $\mathcal{D}_k$.

The *problem of identifying influential attributes* (IA) is:

- *Input*: $\mathcal{R}, \mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k), \hat{\mathcal{M}}$ and $\mathcal{A}_{\mathsf{CR}}$ as described above.
- *Output*: A set $\mathcal{S}$ of influential attributes from schema $\mathcal{R}$.
- *Objective*: Maximize $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) - \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k)$.

NP-hardness. It is nontrivial to find the optimal set of influential attributes. The decision problem, also referred to as IA, is to decide, given $\mathcal{R}, \mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k), \hat{\mathcal{M}}$, $\mathcal{A}_{\mathsf{CR}}$ and a bound δ, whether there exists a set $\mathcal{S}$ of attributes such that $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) - \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) \geq \delta$. This is intractable, even when training and testing $\hat{\mathcal{M}}$ takes polynomial time.

Theorem 3: *Problem* IA *is* NP*-hard.* □

Proof sketch: We prove the NP-hardness of IA by reduction from 3-SAT, which is known to be NP-complete [66]. Given a 3-SAT instance ϕ, we construct a schema $\mathcal{R}$ and a set $\mathcal{D}_k$ of $\mathcal{R}$ such that each $A \in \mathcal{R}$ encodes a variable in ϕ; we also develop a model $\hat{\mathcal{M}}$ and a CR method $\mathcal{A}_{\mathsf{CR}}$. We show that there exists a set $\mathcal{S} \subseteq \mathcal{R}$ with $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) - \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) \geq \delta$ if and only if there exists a truth assignment that satisfies ϕ. □

Naive approach. A naive approach is to greedily and iteratively add attributes A in $\mathcal{R}$ into $\mathcal{S}$, one attribute per round, so that cleaning the A-attributes of $\mathcal{B}_k$ via $\mathcal{A}_{\mathsf{CR}}$ can maximumly improve $\hat{\mathcal{M}}$'s accuracy on $\mathcal{C}_k$ in each round. Unfortunately, this method is costly, since $\hat{\mathcal{M}}$ needs to be retrained $O(|\mathcal{R}||\mathcal{S}|)$ times, where $|\mathcal{S}|$ (resp. $|\mathcal{R}|$) is the size of $\mathcal{S}$ (resp. $\mathcal{R}$).

Our approach. Adapting the boosting strategy [42], we do not retrain $\hat{\mathcal{M}}$ every time when an attribute is added. Instead, we first pre-clean $\mathcal{B}_k$ as $\mathcal{B}_k^{\mathcal{R}}$, by cleaning all attributes in $\mathcal{R}$ via $\mathcal{A}_{\mathsf{CR}}$; for a set $\mathcal{S}$ of attributes, we train a lightweight tree-based model $\hat{\mathcal{M}}_{\mathsf{add}}$ (*e.g.,* GBDT [42]) with the cleaned $\mathcal{S}$-attributes of $\mathcal{B}_k^{\mathcal{R}}$ (*i.e.,* a projected set of $\mathcal{B}_k^{\mathcal{R}}$ with schema $\mathcal{S}$), where $\hat{\mathcal{M}}_{\mathsf{add}}$ aims to fit the misalignment between the actual labels and the predicated labels by $\hat{\mathcal{M}}$. We combine (two weaker) $\hat{\mathcal{M}}$ and $\hat{\mathcal{M}}_{\mathsf{add}}$ as an integrated and stronger model $\hat{\mathcal{M}}_{\mathsf{int}}$ by weighted averaging of predicated labels from $\hat{\mathcal{M}}$ and $\hat{\mathcal{M}}_{\mathsf{add}}$, and validate $\hat{\mathcal{M}}_{\mathsf{int}}$ in $\mathcal{C}_k$ without retaining $\hat{\mathcal{M}}$. The set $\mathcal{S}$ that gives the most accurate $\hat{\mathcal{M}}_{\mathsf{int}}$ on $\mathcal{C}_k$ is returned as the solution.

However, since there are $|\mathcal{R}|!$ attribute combinations for $\mathcal{S}$, it is still costly to train numerous $\hat{\mathcal{M}}_{\mathsf{add}}$. To reduce the search space, we develop a genetic algorithm to find a sub-optimal set $\mathcal{S}$, by running a generation step and a selection step iteratively.

(a) Set generation. In each round, we generate a set $\mathcal{G} = \{\mathcal{S}_1, \ldots, \mathcal{S}_i, \ldots\}$ of at most $\frac{m \cdot (m-1)}{2}$ attribute sets, where m is a bound picked by users. In the first round, each $\mathcal{S}_i$ in $\mathcal{G}$ consists of a single attribute in $\mathcal{R}$ while in subsequent rounds, $\mathcal{G}$ is generated from the sets in previous rounds via two operators: (a) mutation, that randomly replaces an attribute (*e.g.,* replace A_2 in $\{A_1, A_2\}$ with A_3 and get $\{A_1, A_3\}$), and (b) crossover that merges two sets into one (*e.g.,* merge $\{A_1, A_2\}$ and $\{A_1, A_3\}$ into $\{A_1, A_2, A_3\}$). Following [67], [68], each iteration starts with crossover and invokes mutation with a probability.

(b) Set selection. We then adopt the boosting strategy [42] to evaluate each set in $\mathcal{G}$ and retain at most m sets in $\mathcal{G}$ for set generation in the next round. Specifically, for each $\mathcal{S}_i$ in $\mathcal{G}$, we train the corresponding $\hat{\mathcal{M}}_{\mathsf{add}}$, combine it with $\hat{\mathcal{M}}$ to get integrated $\hat{\mathcal{M}}_{\mathsf{int}}$, and evaluate the accuracy of $\hat{\mathcal{M}}_{\mathsf{int}}$ with $\mathcal{C}_k$. At most m sets with the highest accuracy are retained in $\mathcal{G}$. Moreover, we also maintain the set $\mathcal{S}_{\mathsf{best}}$ with the highest accuracy so far, and return it when we reach the maximum number of rounds. Note that the best accuracy of $\hat{\mathcal{M}}$ is improved monotonically, since (a) $\mathcal{C}_k$ is fixed, and (b) $\mathcal{S}_{\mathsf{best}}$ is updated only when there exists another set with a higher accuracy.

Algorithm. We develop Algorithm AlgIA for IA, as shown in Figure 3. AlgIA takes as input $\mathcal{R}, \mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k), \hat{\mathcal{M}}$, $\mathcal{A}_{\mathsf{CR}}$, the clean data $\mathcal{D}_{\mathsf{clean}}$ and two thresholds $\mathsf{ite}_{\mathsf{max}}$ and m (the number of rounds and the maximum number of candidate sets in each round, respectively). It returns a set $\mathcal{S}$ of influential attributes.

AlgIA first initializes the set $\mathcal{G}$ of candidate attribute sets and the attribute set $\mathcal{S}_{\mathsf{best}}$ (initially empty) with the highest accuracy $\mathsf{acc}_{\mathsf{best}}$ observed so far (line 1), and pre-cleans $\mathcal{B}_k$ as $\mathcal{B}_k^{\mathcal{R}}$ via $\mathcal{A}_{\mathsf{CR}}$ (line 2), where we only clean tuples in $\mathcal{B}_k \backslash \mathcal{D}_{\mathsf{clean}}$. Then AlgIA iteratively conducts two steps (lines 3-12). (1) *Selection* (lines 4-10). For each $\mathcal{S}_i$ in $\mathcal{G}$, it trains the lightweight $\hat{\mathcal{M}}_{\mathsf{add}}$ (line 5), combines it with $\hat{\mathcal{M}}$ as $\hat{\mathcal{M}}_{\mathsf{int}}$ (line 6), and evaluates the accuracy of $\hat{\mathcal{M}}_{\mathsf{int}}$ on $\mathcal{C}_k$ (line 7). If $\mathsf{acc}(\hat{\mathcal{M}}_{\mathsf{int}}, \mathcal{C}_k) > \mathsf{acc}_{\mathsf{best}}$, we take $\mathcal{S}_i$ as $\mathcal{S}_{\mathsf{best}}$ (line 8). If none of the sets in $\mathcal{G}$ outperforms $\mathcal{S}_{\mathsf{best}}$, it indicates that $\mathcal{S}_{\mathsf{best}}$ is a good one and AlgIA converges (lines 9-10). (2) *Generation* (lines 11-12). At most m sets in $\mathcal{G}$ with the most accurate $\hat{\mathcal{M}}_{\mathsf{int}}$ are then used to generate at most $\frac{m \cdot (m-1)}{2}$ attribute sets for the next iteration, via mutation and crossover. Finally, when it reaches the maximum number of iterations (*i.e.,* $\mathsf{ite}_{\mathsf{max}}$), AlgIA returns $\mathcal{S}_{\mathsf{best}}$ (line 13).

Example 4: Consider $\mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k)$ of $\mathcal{R}$ in Table I, where $m = 8$. Initially, $\mathcal{G} = \{\{A_1\}, \{A_2\}, \{A_3\}, \{A_4\}, \{A_5\}\}$ and $\mathcal{S}_{\mathsf{best}} = \emptyset$. Algorithm AlgIA first pre-cleans all A_1-A_5 attributes of $\mathcal{B}_k$ via $\mathcal{A}_{\mathsf{CR}}$ to get $\mathcal{B}_k^{\mathcal{R}}$. In the first iteration, AlgIA trains a model $\hat{\mathcal{M}}_{\mathsf{add}}$ for each projected set of $\mathcal{B}_k^{\mathcal{R}}$ on $\{A_i\}$ $(1 \leq i \leq 5)$. Assume that cleaning A_5 gives the best $\mathsf{acc}_{\mathsf{best}}$. We update $\mathcal{S}_{\mathsf{best}}$ as $\{A_5\}$. All sets in $\mathcal{G}$ are retained for set generation in next iteration, *e.g.,* $\{A_3\}$ and $\{A_5\}$ are merged into $\{A_3, A_5\}$. If cleaning $\{A_3, A_5\}$ gives an accuracy higher than $\mathsf{acc}_{\mathsf{best}}$, we update $\mathcal{S}_{\mathsf{best}} = \{A_3, A_5\}$. This process proceeds until either $\mathcal{S}_{\mathsf{best}}$ stabilizes or AlgIA reaches $\mathsf{ite}_{\mathsf{max}}$ rounds. □

Complexity. AlgIA takes $O(c_{\mathsf{CR}} + \mathsf{ite}_{\mathsf{max}} \cdot m^2 \cdot (c_{\mathsf{train}} + c_{\mathsf{val}} + \log(m)) + c_{\mathsf{cross}} + c_{\mathsf{muta}}))$ time, where $\mathsf{ite}_{\mathsf{max}}$ and m are given above, c_{CR} is the cost of cleaning $\mathcal{B}_k$ by $\mathcal{A}_{\mathsf{CR}}$, c_{train} is for training $\hat{\mathcal{M}}_{\mathsf{add}}$, c_{val} is for validating $\hat{\mathcal{M}}_{\mathsf{int}}$ on $\mathcal{C}_k$, $O(m^2 \cdot \log(m))$ is for selecting the top-m sets from $\mathcal{G}$, and c_{cross} (resp. c_{muta}) is for a crossover (resp. mutation) operation in $\mathcal{G}$. AlgIA is fast (see Section VI); one may use a smaller $\mathsf{ite}_{\mathsf{max}}$ for better efficiency, without degrading much the accuracy.

Input: A schema $\mathcal{R}$, a dataset $\mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k)$ of schema $\mathcal{R}$, an ML model $\hat{\mathcal{M}}$, a CR method $\mathcal{A}_{\mathsf{CR}}$, the clean data $\mathcal{D}_{\mathsf{clean}}$, and thresholds $\mathsf{ite}_{\mathsf{max}}$ and m.
Output: A set $\mathcal{S}$ of influential attributes.

1. $\mathcal{G} := \{\{A\} \mid A \in \mathcal{R}\}$; $\mathsf{idx} := 1$; $\mathcal{S}_{\mathsf{best}} := \emptyset$; $\mathsf{acc}_{\mathsf{best}} := 0$;
2. $\mathcal{B}_k^{\mathcal{R}} :=$ cleaning tuples in $\mathcal{B}_k \setminus \mathcal{D}_{\mathsf{clean}}$ via $\mathcal{A}_{\mathsf{CR}}$;
3. **while** $\mathsf{idx} \leq \mathsf{ite}_{\mathsf{max}}$ **do**
   /* candidate set selection */
4. **for each** $\mathcal{S}_i$ in $\mathcal{G}$ **do**
5. $\hat{\mathcal{M}}_{\mathsf{add}} := \mathsf{train}(\mathcal{B}_k^{\mathcal{R}}[\mathcal{S}_i])$;
6. $\hat{\mathcal{M}}_{\mathsf{int}} := \mathsf{boosting}(\hat{\mathcal{M}}, \hat{\mathcal{M}}_{\mathsf{add}})$;
7. **if** $\mathsf{acc}(\hat{\mathcal{M}}_{\mathsf{int}}, \mathcal{C}_k) > \mathsf{acc}_{\mathsf{best}}$ **do**
8. $\mathcal{S}_{\mathsf{best}} := \mathcal{S}_i$; $\mathsf{acc}_{\mathsf{best}} := \mathsf{acc}(\hat{\mathcal{M}}_{\mathsf{int}}, \mathcal{C}_k)$;
9. **if** $\mathcal{S}_{\mathsf{best}}$ is not updated **do**
10. **break**
    /* candidate set generation */
11. $\mathcal{G} := \mathsf{select}(\mathcal{G}, m)$;
12. $\mathcal{G} := \mathsf{generate}(\mathcal{G}, m)$; $\mathsf{idx} := \mathsf{idx} + 1$;
13. **return** $\mathcal{S}_{\mathsf{best}}$;

Fig. 3: Algorithm AlgIA

V. Pinpointing Influential Tuples

This section formulates the influential tuple identifying problem, proves its intractability, and develops a method.

Problem. Given a dataset $\mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k)$, a CR method $\mathcal{A}_{\mathsf{CR}}$ and a model $\hat{\mathcal{M}}$, we aim to identify the set $\mathcal{T}$ of *influential tuples* of $\mathcal{B}_k$, such that if we resolve conflicts in such tuples via $\mathcal{A}_{\mathsf{CR}}$, the accuracy of $\hat{\mathcal{M}}$ is maximumly improved when evaluated with $\mathcal{C}_k$. Denote the cleaned $\mathcal{B}_k$ by $\mathcal{B}_k^{\mathcal{T}}$. Then we obtain a cleaned version$\mathcal{D}_k^{\mathcal{T}} = (\mathcal{B}_k^{\mathcal{T}}, \mathcal{C}_k)$ of $\mathcal{D}_k$.

The *problem of pinpointing influential tuples* (IT) is:

- *Input*: $\mathcal{R}, \mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k), \hat{\mathcal{M}}$, $\mathcal{A}_{\mathsf{CR}}$, $\mathcal{D}_{\mathsf{clean}}$ as in Section III.
- *Output*: A set $\mathcal{T}$ of influential tuples of $\mathcal{B}_k$.
- *Objective*: Maximize $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}) - \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k))$.

Here $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}})$ (resp. $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k)$) is the accuracy of $\hat{\mathcal{M}}$ that is trained with $\mathcal{B}_k^{\mathcal{T}}$ (resp. $\mathcal{B}_k$) and is evaluated with $\mathcal{C}_k$.

NP-hardness. The problem is hard. The decision problem, also referred to as IT, is to determine, given $\mathcal{R}, \mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k), \hat{\mathcal{M}}$, $\mathcal{A}_{\mathsf{CR}}$ and a bound δ, whether there exists a set $\mathcal{T}$ of tuples such that $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}) - \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) \geq \delta$.

Theorem 4: *Problem* IT *is* NP*-hard.* □

Proof sketch: We show its intractability by reduction from 3-SAT. Given a 3-SAT instance ϕ with n variables, we define a schema $\mathcal{R}$ and construct a set $\mathcal{D}_k$ of $\mathcal{R}$, where each tuple in $\mathcal{D}_k$ encodes a variable in ϕ. We develop $\hat{\mathcal{M}}$ and $\mathcal{A}_{\mathsf{CR}}$ and show that there exists a set $\mathcal{T} \subseteq \mathcal{D}_k$ s.t. $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}) - \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) \geq \delta$ if and only if there exists a truth assignment that satisfies ϕ. □

Naive approach. One might want to adopt a greedy algorithm to solve IT as follows. (1) Clean a dirty tuple t in $\mathcal{B}_k$ via $\mathcal{A}_{\mathsf{CR}}$, and denote the cleaned $\mathcal{B}_k$ as $\mathcal{B}_k^t$. (2) Retrain a model $\hat{\mathcal{M}}$, denoted by $\hat{\mathcal{M}}_t$, on each $\mathcal{B}_k^t$. (3) Iteratively find the most influential tuple t' in $\mathcal{B}_k$ such that cleaning t' in $\mathcal{B}_k$ improves the accuracy of $\hat{\mathcal{M}}$ on $\mathcal{C}_k$ the most, add t' to $\mathcal{T}$, and update $\mathcal{D}_k$ and $\hat{\mathcal{M}}$ to be $\mathcal{D}_k^{t'}$ and $\hat{\mathcal{M}}_{t'}$, respectively, until $\hat{\mathcal{M}}$ cannot be further improved. (4) Set $\mathcal{T}$ is returned as the set of influential tuples.

Input: A schema $\mathcal{R}$, a dataset $\mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k)$ of $\mathcal{R}$, an ML model $\hat{\mathcal{M}}$, a CR method $\mathcal{A}_{\mathsf{CR}}$, the cleaned data sets $\mathcal{D}_{\mathsf{clean}}$, and a number T_{max}.
Output: A set $\mathcal{T}$ of at most T_{max} influential tuples.

1. $\mathcal{T} := \emptyset$; $\mathcal{B}_k :=$ cleaning all tuples in $\mathcal{B}_k$ via $\mathcal{A}_{\mathsf{CR}}$;
2. **for** $t \in \mathcal{B}_k \setminus \mathcal{D}_{\mathsf{clean}}$ **do**
3. $t_{\mathsf{old}} :=$ the version of t with which $\mathcal{M}$ is trained;
4. $t_{\mathsf{new}} :=$ the new version of t after pre-cleaning via $\mathcal{A}_{\mathsf{CR}}$;
5. **if** $t_{\mathsf{old}} \neq t_{\mathsf{new}}$**do**
6. $\Delta\theta_t := \frac{1}{|\mathcal{D}_k|}(H_\theta^{-1}(\nabla_\theta l(t_{\mathsf{new}}, \theta) - \nabla_\theta l(t_{\mathsf{old}}, \theta)))$;
7. $\hat{\mathcal{M}}_t := \mathsf{update}(\hat{\mathcal{M}}, \Delta\theta_t)$;
   $\Delta\mathsf{acc}_t := \mathsf{acc}(\hat{\mathcal{M}}_t, C_k) - \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{C}_k)$;
8. **if** $\Delta\mathsf{acc}_t > 0$ **do**
9. $\mathcal{T} := \mathsf{update}(\mathcal{T}, t, T_{\mathsf{max}})$;
10. **return** $\mathcal{T}$;

Fig. 4: Algorithm AlgIT

However, this algorithm needs $O(c_{\mathsf{CR}} + |\mathcal{T}| \cdot (|\mathcal{D}_k| \cdot |\theta| \cdot c_{\mathsf{iter}} + c_{\Delta\mathsf{acc}_t} + \log(\mathcal{T})))$ time, where $|\theta|$ and c_{iter} represents the number of $\hat{\mathcal{M}}$'s parameters and training iterations, respectively, c_{CR} is the cost of cleaning $\mathcal{B}_k$ by $\mathcal{A}_{\mathsf{CR}}$, $c_{\Delta\mathsf{acc}_t}$ is for computing the accuracy improvement of $\hat{\mathcal{M}}$ on $\mathcal{C}_k$ by cleaning a tuple $t \in \mathcal{B}_k$, and $O(\log(\mathcal{T}))$ is the cost for maintaining the tuples in $\mathcal{T}$.

Our approach. In contrast, we identify a set $\mathcal{T}$ of influential tuples by estimating the effect of cleaning tuples in $\mathcal{B}_k$ via $\mathcal{A}_{\mathsf{CR}}$ without actually retraining $\hat{\mathcal{M}}$. To do so, we adopt the influence function [41], [43] to estimate the change to the parameters of model $\hat{\mathcal{M}}$ when tuples in $\mathcal{B}_k$ are removed or modified.

Specifically, cleaning a tuple $t \in \mathcal{B}_k$ via $\mathcal{A}_{\mathsf{CR}}$ can be regarded as removing an old version t_{old} of t from $\mathcal{D}$ and adding a new version t_{new} to $\mathcal{D}$. Denote by θ_{old} and θ_{new} the parameters of $\hat{\mathcal{M}}$ trained with the set containing t_{old} and t_{new}, respectively. For each $t = (x, y)$, where x and y are the attributes and label of t, respectively, we can upweight t by an infinitesimal amount ϵ, such that removing (resp. adding) t from $\mathcal{D}$ is the same as upweighting its old version t_{old} (resp. t_{new}) by $-\epsilon = -\frac{1}{|\mathcal{D}_k|}$ (resp. ϵ). As shown in [43], the change of $\hat{\mathcal{M}}$'s parameters, *i.e.,* $\Delta\theta_t = \theta_{\mathsf{new}} - \theta_{\mathsf{old}}$, by cleaning t via $\mathcal{A}_{\mathsf{CR}}$ can be estimated in a closed form: $-H_\theta^{-1}(\nabla_\theta l(t_{\mathsf{new}}, \theta) - \nabla_\theta l(t_{\mathsf{old}}, \theta))$, where $\theta = \theta_{\mathsf{old}}$, ∇_θ is the partial derivation of θ *w.r.t.* ϵ, H_θ is the Hessian Matrix, and $l(t, \theta)$ is the loss of tuple t.

When a tuple t in $\mathcal{B}_k$ is cleaned via $\mathcal{A}_{\mathsf{CR}}$, we approximate its impact on $\hat{\mathcal{M}}$ by adjusting the parameters of $\hat{\mathcal{M}}$ as $\theta_{\mathsf{new}} (= \theta_{\mathsf{old}} + \Delta\theta_t)$ and then validate $\hat{\mathcal{M}}$ on $\mathcal{C}_k$, without retaining $\hat{\mathcal{M}}$. Based on the accuracy improvement, we can derive the set $\mathcal{T}$.

Algorithm. We develop Algorithm AlgIT for IT, as shown in Figure 4. AlgIT takes as input $\mathcal{R}, \mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k), \hat{\mathcal{M}}, \mathcal{A}_{\mathsf{CR}}$, the clean data $\mathcal{D}_{\mathsf{clean}}$, and the maximum number T_{max} of tuples. It returns a set $\mathcal{T}$ of at most T_{max} influential tuples.

More specifically, AlgIT initializes $\mathcal{T}$ as an empty set, and cleans all tuples in $\mathcal{B}_k \setminus \mathcal{D}_{\mathsf{clean}}$ via $\mathcal{A}_{\mathsf{CR}}$ (line 1). For each tuple t in $\mathcal{B}_k$, it checks whether t is cleaned by $\mathcal{A}_{\mathsf{CR}}$ (lines 3-5). If so (line 5), it uses the influence function to estimate the impact of cleaning $t \in \mathcal{B}_k$ on $\hat{\mathcal{M}}$. To do this, AlgIT produces a copy $\hat{\mathcal{M}}_t$ of $\hat{\mathcal{M}}$ by updating the parameters of $\hat{\mathcal{M}}$ with the change $\Delta\theta_t$, estimated with the influence function (line 6).

Datasets	$\|\mathcal{D}\|$	$\|\mathcal{R}\|$	Noise Ratio (%)	# of classes
Adult [41]	48,842	15	10	2
Nursery [70]	12,960	8	10	4
Bank [71]	49,732	16	10	2
Default [71]	30,000	24	10	2
German [71], [72]	1,000	20	10	2
RoadSafety [73]	363,243	67	10	3

TABLE II: The tested datasets

$\mathcal{M}$	$\mathcal{A}_{\mathsf{CR}}$	German	Adult	Nursery	Default	Bank
LR	Base	0.481	0.653	0.727	0.523	0.487
	$\mathsf{Oracle}_{\mathsf{full}}$	0.691	0.784	0.901	0.673	0.684
MLP	Base	0.578	0.617	0.775	0.478	0.53
	$\mathsf{Oracle}_{\mathsf{full}}$	0.788	0.801	1	0.678	0.746
FTT	Base	0.434	0.47	0.259	0.46	0.471
	$\mathsf{Oracle}_{\mathsf{full}}$	0.51	0.493	0.26	0.484	0.496

TABLE III: acc (Base) and acc($\mathsf{Oracle}_{\mathsf{full}}$)

Then it computes $\Delta\mathsf{acc}_t = \mathsf{acc}(\hat{\mathcal{M}}_t, C_k) - \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{C}_k)$ (line 7), the accuracy improvement of cleaning t on $\mathcal{C}_k$, and adds t to $\mathcal{T}$ if $\Delta\mathsf{acc}_t > 0$ and $\Delta\mathsf{acc}_t$ is among the top-T_{max} highest for tuples in $\mathcal{T}$ (lines 8-9). Finally, $\mathcal{T}$ is returned (line 10).

Note that there is a trade-off between the efficiency and accuracy with T_{max}. Given a smaller T_{max}, model $\hat{\mathcal{M}}$ is often more accurate, as will be seen in Section VI. This is because influence function estimates impact to $\hat{\mathcal{M}}$ when a single tuple in $\mathcal{D}_k$ is removed or modified [43], but not the joint influence when multiple tuples in $\mathcal{D}_k$ are changed. Thus a smaller T_{max} indicates more fine-tuning rounds of CR4ML, which more accurately estimates the joint influence, but incurs a larger cost.

Remark. We need to compute the inverse Hessian Matrix $-H_\theta^{-1}$ for $\Delta\theta_t$, which has a complexity of $O(|\mathcal{D}_k|\cdot|\theta|^2+|\theta|^3)$. We can reduce this complexity to $O(|\mathcal{D}_k|\cdot|\theta|)$ by estimating Hessian-Vector Products via iterative solvers like Pearlmutter's method [43], [69], or even to $O(r\cdot|\theta|)$ via recent works [69], where r is a small constant (typically 10–20). The computation of $\Delta\theta_t$ can be restricted to only a few layers responsible for the predictions, further reducing $|\theta|$ and thus the total cost.

Example 5: Continuing with $\mathcal{D}_{\mathsf{clean}} = \emptyset$ and $T_{\mathsf{max}} = 1$, assume that $\hat{\mathcal{M}}$ is trained with $\mathcal{B}_k = \{t_1\text{-}t_6\}$ and after pre-cleaning, t_1 and t_4 are updated. Their Δacc on $\mathcal{C}_k$ are 0.03 and 0.01, respectively. Thus, the set $\mathcal{T} = \{t_1\}$ is returned as output. □

Complexity. AlgIT takes $O(c_{\mathsf{CR}}+|\mathcal{T}|\cdot(c_{\Delta\theta_t}+c_{\hat{\mathcal{M}}_t}+c_{\Delta\mathsf{acc}_t}+\log(T_{\mathsf{max}})))$ time, where c_{CR} and $c_{\Delta\mathsf{acc}_t}$ are described above, $c_{\Delta\theta_t}$ is for assessing $\hat{\mathcal{M}}$'s parameter changes by influence function (*e.g.,* $O(r\cdot|\theta|)$ as mentioned above), $c_{\hat{\mathcal{M}}_t}$ is for the updating $\hat{\mathcal{M}}$ to $\hat{\mathcal{M}}_t$, and $O(\log(T_{\mathsf{max}}))$ is for maintaining the top-T_{max} tuples in $\mathcal{T}$. We will test the effect of T_{max} in Section VI.

VI. Experimental Study

This section experimentally evaluates the (1) effectiveness and (2) efficiency of CR4ML for different (a) ML differential classifiers, (b) CR methods, and (c) configurable parameters.

Experimental setting. We start with our settings.

Datasets. We used 6 real-life datasets (Table II; see [45]), each randomly split into $\mathcal{D}_{\mathsf{train}}$ and $\mathcal{D}_{\mathsf{test}}$ in 8 : 2 ratio for training and testing, respectively, with seed $= 42$ for reproduction.

Following [41], we treat the released version of each dataset as "clean" $\mathcal{D}$, and inject noises (conflicts) in all attributes of $\mathcal{D}$ (influential or not). This is because: (a) there is no "good" public benchmark with ground truth of real inconsistencies, *e.g.,* [47] tested CleanML benchmarks [74] and found that they contain few real inconsistencies that have significant impact on the downstream models and (b) labeling real datasets is costly, *e.g.,* "it takes 3 people across 6 months to finish the label process" for 16 benchmarks of categorical deduplication [35].

We consider three noise injectors NI: (1) Self-injector (SI) that randomly injects conflicts. (2) DeepFool [75], [76] that injects conflicts targeted for degrading given ML models. (3) Pattern injector (PI) that injects conflicts following systematic patterns (*e.g.,* violating functional dependencies [77]). Since the average error rates used in [12], [15], [17], [35] are around 10%, we set the noise ratio as 10% following prior work.

ML models $\mathcal{M}$. In addition to (a) logistic regression (LR) [78] on which [21], [25], [36], [50] primarily focus, we evaluated two differential ML classification models: (b) a shallow neural network model MLPClassifier [78] and (c) a representative table representation model FT-Transformer (FTT) [65]. We use one-hot encoding to encode relation data for its popularity.

CR methods $\mathcal{A}_{\mathsf{CR}}$. We tested five CR methods $\mathcal{A}_{\mathsf{CR}}$: (1) RB that uses Raha [15] (an error detection method) to detect conflicts and Baran [17] (an error correction method) to correct conflicts, where we set its labeling budget to 20 as in [17]; (2) Rock, a data cleaning system [44] for conflict detection and correction; (3) HoloClean [12], a data cleaning system, where we fed it with the same set of rules (transformed to denial constraints) as Rock, and we randomly selected a maximal subset of rules to maximize its memory usage; (4) RT uses Raha [15] and T5 [79] (a pretrained language model) to detect and resolve conflicts, respectively; and (5) Mode [22], [25] fills in null values with the most frequent domain values. As reported in [17], [44], Rock and RB are considered accurate $\mathcal{A}_{\mathsf{CR}}$.

Baselines. We implemented CR4ML in python and used the following baselines: (1) Base, a "lower-bound" baseline that cleans nothing. (2) $\mathsf{Oracle}_{\mathsf{full}}$, an "upper-bound" baseline that resolves conflicts by experts with 100% accuracy. (3) $\mathsf{CR}_{\mathsf{full}}$, an approach that applies $\mathcal{A}_{\mathsf{CR}}$ to the entire training data. (4) $\mathsf{Rank}_{\mathsf{top}}$, an approach that applies $\mathcal{A}_{\mathsf{CR}}$ to the top-5 (resp. top-30%) attributes (resp. tuples) in the training data ranked via feature importance [78] (resp. influence function [43]). (5) Picket [26], which detects corrupted tuples via reconstruction loss and directly removes dirty tuples from the training data. (6) DiffPrep-fix (resp. DiffPrep-flex) [50], a data preprocessing pipeline that selects suitable cleaning operators for differential ML models with (resp. without) pre-defined transformation orders. (7) SAGA [36], a framework that automatically generates the top-K effective data cleaning pipelines; we set K as 1. (8) ActiveClean [21], an approach that treats data cleaning and model training in a form of SGD. (9) CPClean [22], a data imputation method for KNN models. (10) BoostClean [25], a data cleaning approach that enhances ML models by boosting.

Here (a) BoostClean and CPClean fix missing values only;

thus we only tested their efficiency; (b) we fed ActiveClean with dirty training data as for CR4ML; and (c) we configured DiffPrep-fix, DiffPrep-flex, SAGA, CPClean and BoostClean with selected cleaning tools as in their released codes.

While Base, $\mathsf{Oracle}_{\mathsf{full}}$, $\mathsf{CR}_{\mathsf{full}}$ and Picket are independent of $\mathcal{M}$, the others are $\mathcal{M}$-aware, *i.e.,* they target a given $\mathcal{M}$.

Configurable parameters. By default, we set (1) for CR4ML, $e = 3$ for the number of epochs; (2) for AlgIA, $\mathsf{ite}_{\mathsf{max}} = 2$ as the maximum number of rounds and $m = |\mathcal{R}|$ as the bound on candidate sets; and (3) for AlgIT, $T_{\mathsf{max}} = 2\%|\mathcal{D}|$. We used LR as $\mathcal{M}$, SI as the noise injector NI and Rock as $\mathcal{A}_{\mathsf{CR}}$. For baselines, we set their parameters following their recommendations in [21], [25], [26], [36], [50]. We focus on the effect of CR on the performance of ML models and leave exhaustive tuning strategies (*e.g.,* [80]) to future work.

Configuration. We ran experiments on a machine powered by 504GB RAM and 104 Intel(R) Xeon(R) Gold 5320 CPUs @2.20GHz. Each test was run 3 times; the average is reported.

Experimental findings. We next report our findings.

Exp-1 Effectiveness. We evaluated the accuracy of CR4ML and baselines with various models and CR methods. Following [22], we use the *relative accuracy* of a baseline X, defined as

$$\mathsf{ACC}_{\mathsf{re}}(X) = \frac{\mathsf{acc}(X) - \mathsf{acc}(\mathsf{Base})}{\mathsf{acc}(\mathsf{Oracle}_{\mathsf{full}}) - \mathsf{acc}(\mathsf{Base})},$$

where $\mathsf{acc}(X) = \mathsf{acc}(\mathcal{M}, \mathcal{D}_{\mathsf{CR}})$ and $\mathcal{D}_{\mathsf{CR}}$ is the set cleaned by X, *e.g.,* applying a CR method on entire data (*e.g.,* $\mathsf{CR}_{\mathsf{full}}$) or only influential data (*e.g.,* CR4ML). Since $\mathsf{Oracle}_{\mathsf{full}}$ (resp. Base) is the "upper bound" (resp. "lower bound") of cleaning, (with accuracy in Table III), $\mathsf{ACC}_{\mathsf{re}}(X) \in (-\infty, 1]$ quantifies the improvement in accuracy relative to the upper bound; here $\mathsf{acc}(\mathsf{Oracle}_{\mathsf{full}})$ for complex models (*e.g.,* FTT) may be low due to overfitting. Note that $\mathsf{ACC}_{\mathsf{re}}(X) < 0$ if the accuracy of cleaning via X is even worse than cleaning nothing, *i.e.,* Base.

Accuracy vs. baselines. As shown in Figures 5(a)-5(c), CR4ML has the highest $\mathsf{ACC}_{\mathsf{re}}$. Besides, we find the following.

(1) The released code of all $\mathcal{M}$-aware baselines can only support LR. CR4ML outperforms all these baselines for LR *e.g.,* its $\mathsf{ACC}_{\mathsf{re}}$ is 45.7% on average, as opposed to 7.7%, -1.1%, -20.4% and -1.8% of DiffPrep-fix, DiffPrep-flex, ActiveClean and SAGA, respectively. This is because CR4ML (a) incrementally identifies dirty data by monitoring and analyzing fine-grained error signals, and (b) selectively resolves conflicts only in influential data that positively affect $\mathcal{M}$, *e.g.,* CR4ML fixed $1‰|\mathcal{D}|$ influential tuples on Adult, which improves $\mathsf{ACC}_{\mathsf{re}}$ by 2.63% alone. In contrast, the $\mathcal{M}$-aware baselines attempt to clean the entire $\mathcal{D}$ without carefully considering how each correction affects the model performance. This justifies the need for cleaning influential data alone, instead of the entire $\mathcal{D}$.

(2) CR4ML beats Base, $\mathsf{CR}_{\mathsf{full}}$, $\mathsf{Rank}_{\mathsf{top}}$ and Picket for all models; on average its $\mathsf{ACC}_{\mathsf{re}}$ is 43.1%, 39.6%, 31.8%, and 78.8% higher than the four baselines, up to 72.5%, 79.9%, 165.7% and 90.7%, respectively. These further show the benefits of cleaning influential data; moreover, guided by the influence function and boosting strategy, CR4ML selectively resolves conflicts that (a) it can confidently address, and (b) the cleaning of such conflicts will maximumly improve the accuracy of $\mathcal{M}$, *e.g.,* the $\mathsf{ACC}_{\mathsf{re}}$ of CR4ML and $\mathsf{CR}_{\mathsf{full}}$ is 43.1% and 3.5%, respectively. This is because CR4ML considers the effect of conflict resolution and model performance in a fine-grained manner, and iteratively identifies influential data to fix, rather than relying on one-time bet as in $\mathsf{Rank}_{\mathsf{top}}$. It also mitigates the impact of noises introduced by imprecise CR methods, by considering their joint influence with the model accuracy.

(3) The $\mathsf{ACC}_{\mathsf{re}}$ of CR4ML for LR, MLP and FTT is 54.8%, 42.5% and 24.2%, by correcting 23.1%, 21.2% and 12.4% of the injected conflicts, respectively. Thus CR4ML is effective in both simple models like LR and complex models like FTT.

Case study on CleanML *benchmarks.* We evaluated CR4ML on CleanML benchmarks [74] (not shown), which contain real-world conflicts but lack ground truth (*i.e.,* $\mathsf{Oracle}_{\mathsf{full}}$). Thus, we report the accuracy of $\mathcal{M}$ instead of $\mathsf{ACC}_{\mathsf{re}}$. It improves the accuracy, *e.g.,* by 6.1% on Company, at least 2× higher than the result reported in [47]. This shows that CR4ML is promising in improving $\mathcal{M}$'s accuracy by fixing real-world conflicts.

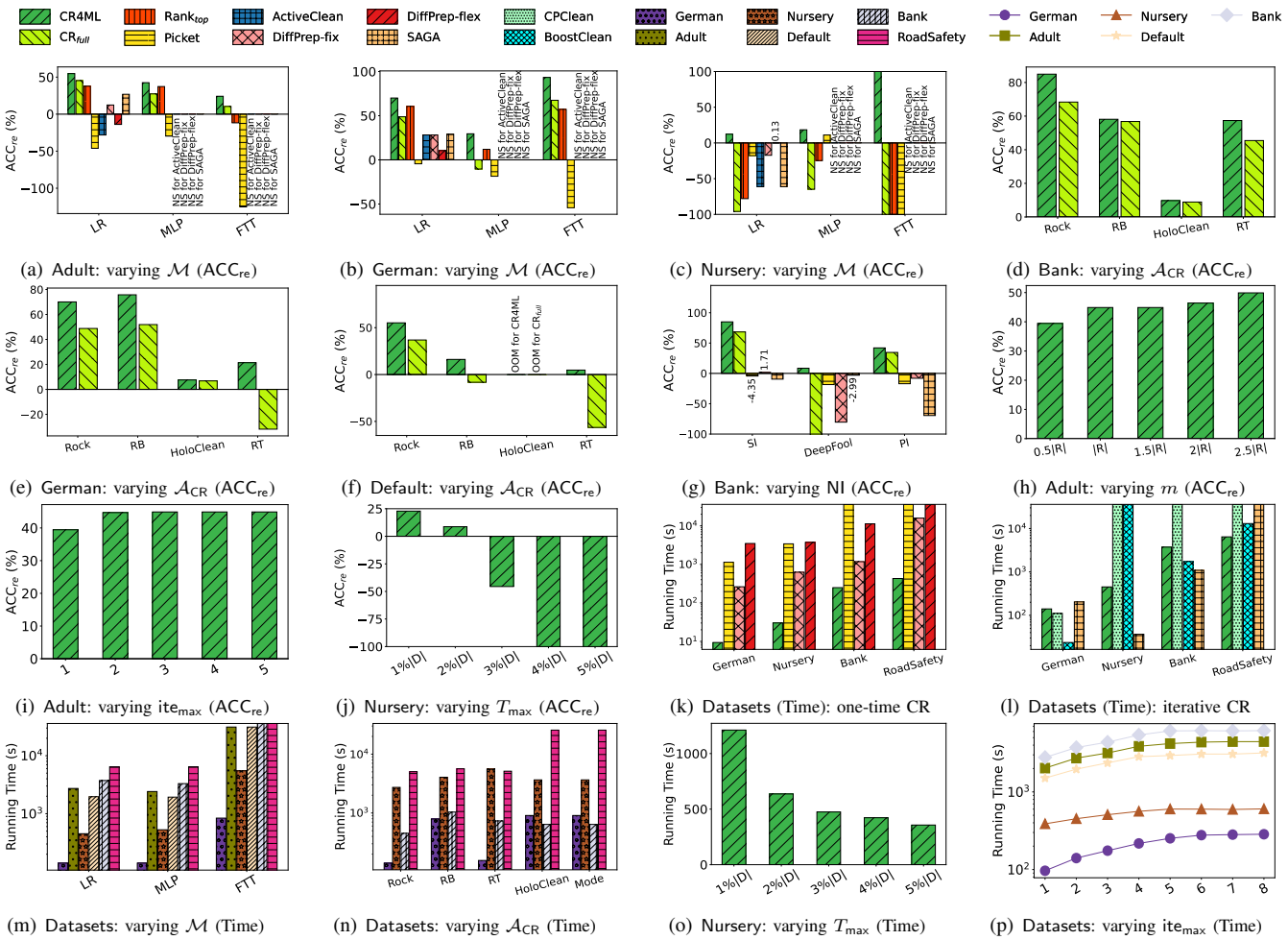
We next tested the impact of parameters on the accuracy.

Varying $\mathcal{A}_{\mathsf{CR}}$. We tested the impact of $\mathcal{A}_{\mathsf{CR}}$ (except Holoclean on Default, whose features cannot fit in memory). Besides, Picket, DiffPrep-fix, DiffPrep-flex, ActiveClean and SAGA are not compared since they do not rely on any particular CR.

As shown in Figures 5(d)-5(f), (1) CR4ML performs better with more accurate $\mathcal{A}_{\mathsf{CR}}$, *e.g.,* the $\mathsf{ACC}_{\mathsf{re}}$ of CR4ML with Rock is 61.4% higher on average than Holoclean. This is because more accurate CR correctly resolves more influential conflicts *e.g.,* CR4ML with Rock (resp. Holoclean) correctly resolves 27.8% (resp. 2.8%) of conflicts. (2) Given accurate $\mathcal{A}_{\mathsf{CR}}$, CR4ML improves the $\mathsf{ACC}_{\mathsf{re}}$ better than directly cleaning entire $\mathcal{D}$, *e.g.,* CR4ML with Rock improves $\mathsf{ACC}_{\mathsf{re}}$ of $\mathcal{M}$ by 70% on German instead of 48.8% by cleaning the entire $\mathcal{D}$. This further verifies the effectiveness of selective conflict resolution. (3) CR4ML works with imprecise $\mathcal{A}_{\mathsf{CR}}$ and mitigates its noises, *e.g.,* on average the $\mathsf{ACC}_{\mathsf{re}}$ of CR4ML with RT is 27.8%, as opposed to -14.3% by cleaning entire $\mathcal{D}$ with RT.

Varying NI. We tested the impact of different noise injectors NI in Figure 5(g). The $\mathsf{ACC}_{\mathsf{re}}$ of CR4ML is sensitive to NI, *e.g.,* it is 85%, 8.5% and 42% on Bank under SI, DeepFool and PI, respectively, since injectors like DeepFool are $\mathcal{M}$-aware and harm the accuracy of $\mathcal{M}$ by injecting noises with small training losses, such that the injected noises are hard to detect. Nevertheless, CR4ML consistently beats the baselines, *e.g.,* its $\mathsf{ACC}_{\mathsf{re}}$ is 45.2% on average, 58.4% higher than Picket, the highest one among baselines under various noise injectors. This is because CR4ML considers the joint influence of data cleaned and carefully identifies the influential data, instead of only focusing on the training loss of individual tuples.

We also evaluated the impact of noises in influential vs. non-influential data, by injecting the same ratio (5%) of noises into influential (resp. non-influential) data in the "clean" version



(a) Adult: varying $\mathcal{M}$ ($\mathsf{ACC_{re}}$) (b) German: varying $\mathcal{M}$ ($\mathsf{ACC_{re}}$) (c) Nursery: varying $\mathcal{M}$ ($\mathsf{ACC_{re}}$) (d) Bank: varying $\mathcal{A}_{\mathsf{CR}}$ ($\mathsf{ACC_{re}}$)

(e) German: varying $\mathcal{A}_{\mathsf{CR}}$ ($\mathsf{ACC_{re}}$) (f) Default: varying $\mathcal{A}_{\mathsf{CR}}$ ($\mathsf{ACC_{re}}$) (g) Bank: varying NI ($\mathsf{ACC_{re}}$) (h) Adult: varying m ($\mathsf{ACC_{re}}$)

(i) Adult: varying $\mathsf{ite_{max}}$ ($\mathsf{ACC_{re}}$) (j) Nursery: varying T_{max} ($\mathsf{ACC_{re}}$) (k) Datasets (Time): one-time CR (l) Datasets (Time): iterative CR

(m) Datasets: varying $\mathcal{M}$ (Time) (n) Datasets: varying $\mathcal{A}_{\mathsf{CR}}$ (Time) (o) Nursery: varying T_{max} (Time) (p) Datasets: varying $\mathsf{ite_{max}}$ (Time)

Fig. 5: Performance evaluation ("NS" stands for "not support")

of $\mathcal{D}$, resulting in "dirty" $\mathcal{D}_{\mathsf{inf}}$ (resp. $\mathcal{D}_{\mathsf{non\text{-}inf}}$). Comparing the accuracy of $\mathcal{M}$ trained on $\mathcal{D}_{\mathsf{inf}}$ and $\mathcal{D}_{\mathsf{non\text{-}inf}}$, we find that noises in influential data significantly degrades the accuracy of $\mathcal{M}$, while noises in non-influential data has little impact on $\mathcal{M}$, *e.g.,* the accuracy drops by 5.4% (resp. 0%) on Bank when $\mathcal{M}$ is trained on $\mathcal{D}_{\mathsf{inf}}$ (resp. $\mathcal{D}_{\mathsf{non\text{-}inf}}$). This reinforces the importance of identifying and resolving conflicts in influential data.

Varying m. We varied the number m of attribute sets from $0.5|\mathcal{R}|$ to $2.5|\mathcal{R}|$ in Figure 5(h). The $\mathsf{ACC_{re}}$ of CR4ML increases when m gets larger, *e.g.,* it is improved by 17.6% when m is changed from $0.5|\mathcal{R}|$ to $2|\mathcal{R}|$. This is because a larger m allows more explorations in the search space, increasing the chance of identifying a good set of influential attributes. Nonetheless, a fairly small m often suffices, *e.g.,* the $\mathsf{ACC_{re}}$ of $\mathcal{M}$ with $m = |\mathcal{R}|$ is just 1.5% lower than with $m = 2|\mathcal{R}|$.

Varying $\mathsf{ite_{max}}$. We varied $\mathsf{ite_{max}}$ from 1 to 5 (Section IV). As shown in Figure 5(i), the $\mathsf{ACC_{re}}$ initially increases and then stabilizes when $\mathsf{ite_{max}}$ exceeds 2. Indeed, while CR4ML checks more candidate attribute sets when $\mathsf{ite_{max}}$ increases, a good set of influential attributes can be identified with a small $\mathsf{ite_{max}}$.

Varying T_{max}. We tested the impact of the number T_{max} of influential tuples on CR4ML, varying it from $1\%|\mathcal{D}|$ to $5\%|\mathcal{D}|$ in Figure 5(j). To better illustrate its effect, we disabled Algorithm AlgIA (*i.e.,* we cleaned influential tuples alone) and thus, the $\mathsf{ACC_{re}}$ in Figure 5(j) is lower than the full setting (*e.g.,* Figures 5(a)-5(c)), since the performance of CR4ML is maximized when both influential tuples and attributes are cleaned.

As shown there, when T_{max} increases, the $\mathsf{ACC_{re}}$ of CR4ML decreases, *e.g.,* its $\mathsf{ACC_{re}}$ decreases from 22.9% to -45.6% when T_{max} varies from $1\%|\mathcal{D}|$ to $3\%|\mathcal{D}|$. This is because with a large T_{max}, the estimation of joint influence of tuples in CR4ML is less accurate, and thus the assistant model $\hat{\mathcal{M}}$ may provide the final model $\mathcal{M}$ with inappropriate tuples for incremental training. However, since a small T_{max} also results in longer runtime (see Exp-2), we suggest to set T_{max} that strikes a good balance between effectiveness and efficiency.

Varying e. We also varied the number e of epochs for incremental learning (not shown). The $\mathsf{ACC_{re}}$ of CR4ML is not very sensitive to e when it reaches a certain threshold (*e.g.,* the default). This is because $\mathcal{M}$ can fit $\mathcal{D}$ within a few epochs.

Exp-2 Efficiency. Below we first compared the overall time of CR4ML and baselines. We then evaluated the impact of various parameters on the efficiency of CR4ML. We report the performance of CR4ML for both one-round and multi-round

iterations, and compared it with the non-iterative baselines (Picket, DiffPrep-fix and DiffPrep-flex) and iterative ones (BoostClean, CPClean and SAGA), respectively; we did not compare with ActiveClean, since it needs experts to resolve conflicts and is hard to estimate its time. We omitted a baseline if it ran out of memory (OOM) or could not finish in 10 hours.

We compared CR4ML with baselines in Figures 5(k)-5(l).

Non-iterative baselines. One-round execution of CR4ML beats all non-iterative baselines, *e.g.,* 16.8X, 121.6X and 96.5X faster than DiffPrep-fix, DiffPrep-flex and Picket on average, respectively. Its $\mathsf{ACC_{re}}$ is 24%, 38.8% and 57% higher than the three for LR, verifying that CR4ML can make practical use of CR. Better still, multi-round CR4ML is even comparable to some non-iterative baselines, *e.g.,* CR4ML takes 140.5s on German with 15 rounds, and is 8.1X, 1.9X and 24.8X faster than Picket, DiffPrep-fix and DiffPrep-flex, respectively.

Iterative baselines. Multi-round CR4ML is faster than or comparable to iterative baselines, *e.g.,* it only takes 451.6s on Nursery and is at least 66.4X faster than CPClean. This is because CR4ML only detects and resolves conflicts in influential data, and stops as soon as all such conflicts are resolved. Indeed, CR4ML converges after 7 rounds on average. Note that BoostClean is often the fastest one since it enhances $\mathcal{M}$ by training weaker classifiers, without directly cleaning $\mathcal{D}$ using $\mathcal{A}_{\mathsf{CR}}$, which dominates the computational cost. This said, CR4ML is able to not only improve the accuracy of $\mathcal{M}$ but also tune datasets for other use, *e.g.,* training other models.

In short, CR4ML is practical since (a) it is fast given efficient $\mathcal{M}$ and $\mathcal{A}_{\mathsf{CR}}$, and (b) its creator-critic framework stops as soon as $\mathcal{A}_{\mathsf{CR}}$ resolves conflicts in all influential data.

Scalability on $|\mathcal{D}|$. To test the scalability of CR4ML on large datasets, we produced an enlarged Adult with 1M tuples by repeatedly sampling tuples in Adult. We find that CR4ML is able to handle large $\mathcal{D}$ with a reasonable T_{max}, *e.g.,* with T_{max} = $10\%|\mathcal{D}|$, CR4ML takes 14,909s to get $\mathsf{ACC_{re}} = 66.9\%$, which is only 0.1% less than the $\mathsf{ACC_{re}}$ of CR4ML with T_{max} = $2\%|\mathcal{D}|$, while all baselines fail due to OOM or time limitation.

Varying $\mathcal{M}$. We tested the impact of ML models $\mathcal{M}$ on the efficiency. As shown in Figure 5(m), CR4ML takes longer when $\mathcal{M}$ is more complex, *e.g.,* with FTT it takes 5,487.2s on Nursery, 12.1X longer than with LR. This is because with a complex $\mathcal{M}$, CR4ML needs more time to train $\mathcal{M}$, *e.g.,* to compute the Hessian matrix for $\mathcal{M}$. This said, CR4ML is fast given fast $\mathcal{M}$, *e.g.,* with LR it only takes 140.5s on German.

Varying $\mathcal{A}_{\mathsf{CR}}$. The efficiency of CR4ML is heavily dependent on embedded $\mathcal{A}_{\mathsf{CR}}$ as shown in Figure 5(n), *e.g.,* it takes 1,801.8s (resp. 5,669.2s) on average when $\mathcal{A}_{\mathsf{CR}}$ is Rock (resp. HoloClean). When $\mathcal{A}_{\mathsf{CR}}$ is fast, so is CR4ML, *e.g.,* CR4ML with Mode takes only 6.3s and 307s on German and Adult, respectively, compared to 156.1s and 5,575.5s with RB.

Varying T_{max}. We disabled AlgIA and varied the number T_{max} of influential tuples, from $1\%|\mathcal{D}|$ to $5\%|\mathcal{D}|$. As shown in Figure 5(o), it takes less time when T_{max} gets larger, from 1211.6s to 356.9s on Nursery. This is because with a larger T_{max}, algorithm AlgIT may find more tuples to fix, and thus make $\mathcal{M}$ converge quicker to a local optima. However, larger T_{max} may make CR4ML less accurate as shown in Exp-1 (see also Section V) and thus, we set $T_{\mathsf{max}} = 2\%|\mathcal{D}|$ in our default setting.

Varying $\mathsf{ite_{max}}$. We varied the maximum number $\mathsf{ite_{max}}$ of iterations for algorithm AlgIA from 1 to 8. As shown in Figure 5(p), the cost of CR4ML first increases and then stabilizes, *e.g.,* it increases from 2,025s to 4,379s on Adult when $\mathsf{ite_{max}}$ varies from 1 to 6, and then remains stable when $\mathsf{ite_{max}} \geq 6$. This is because given a larger $\mathsf{ite_{max}}$, CR4ML explores more attribute combinations to fine-tune the models, and takes longer as expected. This said, a small $\mathsf{ite_{max}}$ suffices for AlgIA to find a good $\mathcal{S}$ and converge, as shown in Exp-1.

The impact of m is similar and hence is not shown.

Varying e. We also varied the number e of epochs for incremental learning from 1 to 5 (not shown). When e gets larger, the training time of $\mathcal{M}$ increases slightly, *e.g.,* when e varies from 1 to 5, CR4ML only takes 1.06X longer to converge. This tells us that it only needs a few epochs such that $\mathcal{M}$ fits the newly cleaned tuples in $\mathcal{D}$, *i.e.,* a small e suffices; moreover, the result also verifies that incremental training is efficient.

Parameter setting. We find $e = 3$, $m = |\mathcal{R}|$, $\mathsf{ite_{max}} = 2$ and $T_{\mathsf{max}} = 2\%|\mathcal{D}|$ typically strike a balance of efficiency and accuracy. For CR tools, Rock and RB work well for various $\mathcal{M}$.

Summary. We find the following. (1) On average, CR4ML improves $\mathsf{ACC_{re}}$ of differential classification models by 43.1%, *e.g.,* it improves the $\mathsf{ACC_{re}}$ of LR by 45.7%, versus -22.8%, -20.4%, 7.7%, -1.1% and -1.8% of Picket, ActiveClean, DiffPrep-fix, DiffPrep-flex and SAGA, respectively. This verifies that CR can indeed improve the ML models. CR4ML with accurate CR works better or comparably to AutoML methods (*e.g.,* DiffPrep-fix). (2) CR4ML is 39.6% (resp. 31.8%) more accurate than cleaning the entire dataset (resp. the top-ranked attributes/tuples); the latter may degrade the accuracy of models. These justify the need for selectively cleaning. (3) CR4ML improves $\mathcal{M}$ better with more accurate CR, *e.g.,* the accuracy of $\mathcal{M}$ improved by CR4ML with Rock is 61.4% higher than with HoloClean on average for LR. This said, it also works with imprecise CR by mitigating the impact of noises that CR may introduce. (4) CR4ML is fast when it is equipped with an efficient CR method, *e.g.,* with Rock, 5 rounds of CR4ML are 18.2X faster than non-iterative Picket and iterative CPClean, with 58.2% higher $\mathsf{ACC_{re}}$. It scales well, *e.g.,* it takes 14,909s on a dataset with 1M tuples for LR, with $\mathsf{ACC_{re}} = 66.9\%$.

VII. Related Work

We categorize the related work as follows.

Data cleaning for ML. The prior work improves the accuracy of white-box and black-box models by cleaning training data.

White-box. ActiveClean [21] treats data cleaning and model training as a form of SDG. CPClean [22] studies the impact of data incompleteness on the quality of KNN models. Good-

Core [23] selects a coreset over incomplete data through gradient approximation. [81] investigates whether Naive Bayes classifiers trained on possible worlds of incomplete data produce consistent predictions on a fixed test dataset. [82] learns an accurate ML model on incomplete training data without imputing missing values. ZORRO [83] learns linear models over possible worlds of uncertain data. CleanML [24] empirically studies the impact of data cleaning on ML classification on structured data. It concludes that CR "is more likely to have insignificant impact and unlikely to have negative impact".

Black-box. BoostClean [25] automatically selects an ensemble of detecting and repairing operations from a library. Picket [26] proposes a framework to safeguard against data corruptions during both training and deployment of ML models. MLClean [27] proposes a framework to simultaneously handle data cleaning, unfairness mitigation and sanitation. Snorkel [28] allows users to train ML models via data programming with labeling functions. Coco [29], [30] assesses unfriendly tuples for models by using constraints defined with arithmetic expressions on numerical attributes. Amalur [31] unifies data integration and ML. Complaint-driven methods [32], [33] leverage users' complaints to remove/revise a minimum set of training examples. Explanation-based [34] models treat query explanation as a hyper-parameter tuning problem; it adopts Random tree and Bayesian optimization to solve it. CategDedup [35] empirically studies the impact of categorical duplicates on ML models. DLearn [84] employs integrity constraints to learn accurate pattern models on dirty data without cleaning the data. COMET [85] incrementally selects features to clean by simulating errors and estimating their impact on model accuracy.

This work differs from the prior work in the following. (1) Instead of one-shot cleaning workflows for $\mathcal{M}$ (*e.g.,* MLClean [27]), we propose CR4ML, a creator-critic framework for differentiable models $\mathcal{M}$, to iteratively conduct CR and model training. It selectively resolves conflicts on influential data, rather than indistinguishably applying CR to the entire dataset as practiced by all previous works except [85]. (2) CR4ML differs from the selective cleaning in [85] as follows: (a) We do not require a noise injector that replicates the noise pattern of dirty data, (b) we account for the impact of noise that may be introduced by imprecise CR methods, and (c) we perform CR at both tuple and attribute levels, not just the attribute level as in [85]. (3) CR4ML gives a concrete solution to improving $\mathcal{M}$ via CR, beyond an empirical analysis as in [35]. (4) CR4ML does not restrict the choice of (perfect) CR methods as in [21], [25], [85]; in fact, it is to mitigate the impact of noise introduced by imprecise CR that CR4ML is developed. (5) CR4ML detects and resolves conflicts in the data, rather than deleting dirty tuples entirely as in [26]. (6) CR4ML does not require user interaction in the cleaning/training process, as opposed to Snorkel [28] and complaint-driven methods [32], [33]. (7) CR4ML aims to enhance the accuracy of differential models $\mathcal{M}$ by detecting and resolving conflicts in dirty data, rather than investigating the stability of $\mathcal{M}$ trained on incomplete data as in [81], [82], or learning linear models on uncertain data as in [83]. (8) CR4ML focuses on improving the accuracy of a specific downstream ML model, rather than learning patterns over possible worlds as in [84].

Data cleaning for AutoML. This approach automatically selects ML pipelines to maximumly improve the accuracy of ML models, in which data cleaning serves as a component.

White-box. DiffML [46] enables differentiable ML pipelines by assigning adjustable parameters to each training record and each error detection/repairing approach in a pool of data cleaning tools. AutoClean [47] re-evaluates several CleanML benchmarks using AutoML systems and extends AutoSklearn [48] with more advanced cleaning strategies for both white-box and black-box models; it concludes that most existing benchmarks contain few errors that substantially impact the downstream models, and thus, calls on the community for benchmarks with impactful real-world errors so as to study data cleaning for ML/AutoML on the attribute level.

Black-box. AlphaClean [49] studies parameter tuning for data cleaning pipelines to optimize an objective function. AutoSklearn [48] improves AutoML by assessing its historical performance on similar datasets. DiffPrep [50] proposes an automatic data preprocessing method for any dataset and differentiable model. SAGA [36] automatically generates the top-K most effective data cleaning pipelines.

More generally, [86] shows how different aspects of data quality propagate through various stages of ML development; [87] reviews recent progress in data cleaning for ML and presents a holistic cleaning framework; and Rein [41] introduces a comprehensive benchmark to evaluate the impact of data cleaning methods on various ML models. None of the previous work considers how to make practical use of CR methods to improve the accuracy of ML classifications.

This work differs from the prior works on AutoML in the following. (1) Rather than replying on other components (*e.g.,* feature engineering [47], [48]) in AutoML, we study the impact of CR itself on ML; moreover, our results can help AutoML prepare data for feature engineering and smoothing, beyond serving as embedded data cleaning. (2) CR4ML allows users to plug in any CR method with $\alpha > 0.5$, as opposed to those (*e.g.,* [46], [50]) that only support restrictive CR tools.

VIII. Conclusion

We have shown how CR can be effectively utilized to improve the accuracy of downstream ML models. (1) CR can substantially enhance differential classification models when applied correctly, only to influential attributes and tuples. (2) We propose CR4ML, a creator-critic framework that can works even with not-so-accurate CR tools $\mathcal{A}_{\mathsf{CR}}$, mitigating the impact of noise introduced by $\mathcal{A}_{\mathsf{CR}}$. (3) Although identifying influential attributes and tuples is intractable, CR4ML provides effective methods with approximation bounds. Our experimental study has verified that CR4ML is promising in practice.

Topics for future work include extending CR4ML to (a) improve non-differential classifiers, and (b) support other data cleaning operations, *e.g.,* label cleaning and outlier repairing.

REFERENCES

[1] W. Fan, F. Geerts, N. Tang, and W. Yu, "Conflict resolution with data currency and consistency," *J. Data and Information Quality*, vol. 5, no. 1-2, pp. 6:1–6:37, 2014.

[2] M. Arenas, L. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," in *PODS*, 1999, pp. 68–79.

[3] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *ACM Trans. Database Syst.*, vol. 33, no. 2, pp. 6:1–6:48, 2008.

[4] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma, "Improving data quality: Consistency and accuracy," in *VLDB*, 2007, pp. 315–326.

[5] S. Giannakopoulou, M. Karpathiotakis, and A. Ailamaki, "Cleaning denial constraint violations through relaxation," in *SIGMOD*, 2020, pp. 805–815.

[6] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin, "On the relative trust between inconsistent data and inaccurate constraints," in *ICDE*. IEEE, 2013, pp. 541–552.

[7] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas, "Guided data repair," *PVLDB*, vol. 4, no. 5, pp. 279–289, 2011.

[8] L. Bertossi, *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers, 2011.

[9] X. Ding, H. Wang, J. Su, M. Wang, J. Li, and H. Gao, "Leveraging currency for repairing inconsistent and incomplete data," *TKDE*, 2020.

[10] F. Geerts, G. Mecca, P. Papotti, and D. Santoro, "The llunatic data-cleaning framework," *PVLDB*, vol. 6, no. 9, pp. 625–636, 2013.

[11] A. Gilad, D. Deutch, and S. Roy, "On multiple semantics for declarative database repairs," in *SIGMOD*, 2020, pp. 817–831.

[12] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "HoloClean: Holistic data repairs with probabilistic inference," *PVLDB*, vol. 10, no. 11, pp. 1190–1201, 2017.

[13] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu, "Towards certain fixes with editing rules and master data," *VLDB J.*, vol. 21, no. 2, pp. 213–238, 2012.

[14] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas, "HoloDetect: Few-shot learning for error detection," in *SIGMOD*, 2019, pp. 829–846.

[15] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Raha: A configuration-free error detection system," in *SIGMOD*, 2019, pp. 865–882.

[16] L. Visengeriyeva and Z. Abedjan, "Metadata-driven error detection," in *SSDBM*, 2018, pp. 1:1–1:12.

[17] M. Mahdavi and Z. Abedjan, "Baran: Effective error correction via a unified context representation and transfer learning," *PVLDB*, vol. 13, no. 12, pp. 1948–1961, 2020.

[18] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid, "Don't be scared: Use scalable automatic repairing with maximal likelihood and bounded changes," in *SIGMOD*. ACM, 2013.

[19] W. Fan, P. Lu, and C. Tian, "Unifying logic rules and machine learning for entity enhancing," *Sci. China Inf. Sci.*, vol. 63, no. 7, 2020.

[20] W. Fan, Z. Han, W. Ren, D. Wang, Y. Wang, M. Xie, and M. Yan, "Splitting tuples of mismatched entities," *Proc. ACM Manag. Data*, 2024.

[21] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, "ActiveClean: Interactive data cleaning for statistical modeling," *PVLDB*, vol. 9, no. 12, pp. 948–959, 2016.

[22] B. Karlas, P. Li, R. Wu, N. M. Gürel, X. Chu, W. Wu, and C. Zhang, "Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions," *CoRR*, vol. abs/2005.05117, 2020. [Online]. Available: https://arxiv.org/abs/2005.05117

[23] C. Chai, J. Liu, N. Tang, J. Fan, D. Miao, J. Wang, Y. Luo, and G. Li, "GoodCore: Data-effective and data-efficient machine learning through coreset selection over incomplete data," *Proc. ACM Manag. Data*, 2023.

[24] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang, "CleanML: A study for evaluating the impact of data cleaning on ML classification tasks," in *ICDE*. IEEE, 2021, pp. 13–24.

[25] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu, "BoostClean: Automated error detection and repair for machine learning," *CoRR*, vol. abs/1711.01299, 2017.

[26] Z. Liu, Z. Zhou, and T. Rekatsinas, "Picket: Self-supervised data diagnostics for ML pipelines," *CoRR*, vol. abs/2006.04730, 2020. [Online]. Available: https://arxiv.org/abs/2006.04730

[27] K. H. Tae, Y. Roh, Y. H. Oh, H. Kim, and S. E. Whang, "Data cleaning for accurate, fair, and robust models: A big data - AI integration approach," in *DEEM@SIGMOD 2019*. ACM, 2019, pp. 5:1–5:4.

[28] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. A. Fries, S. Wu, and C. Ré, "Snorkel: Rapid training data creation with weak supervision," *PVLDB*, vol. 11, no. 3, pp. 269–282, 2017.

[29] A. Fariha, A. Tiwari, A. Meliou, A. Radhakrishna, and S. Gulwani, "Coco: Interactive exploration of conformance constraints for data understanding and data cleaning," in *SIGMOD*. ACM, 2021, pp. 2706–2710.

[30] A. Fariha, A. Tiwari, A. Radhakrishna, S. Gulwani, and A. Meliou, "Conformance constraint discovery: Measuring trust in data-driven systems," in *SIGMOD*. ACM, 2021, pp. 499–512.

[31] R. Hai, C. Koutras, A. Ionescu, Z. Li, W. Sun, J. van Schijndel, Y. Kang, and A. Katsifodimos, "Amalur: Data integration meets machine learning," in *ICDE*. IEEE, 2023, pp. 3729–3739.

[32] W. Wu, L. Flokas, E. Wu, and J. Wang, "Complaint-driven training data debugging for query 2.0," in *SIGMOD*. ACM, 2020, pp. 1317–1334.

[33] L. Flokas, W. Wu, Y. Liu, J. Wang, N. Verma, and E. Wu, "Complaint-driven training data debugging at interactive speeds," in *SIGMOD*, 2022, pp. 369–383.

[34] B. Lockhart, J. Peng, W. Wu, J. Wang, and E. Wu, "Explaining inference queries with Bayesian optimization," *PVLDB*, vol. 14, no. 11, pp. 2576–2585, 2021.

[35] V. Shah, T. Parashos, and A. Kumar, "How do categorical duplicates affect ML? A new benchmark and empirical analyses," *PVLDB*, vol. 17, no. 6, pp. 1391–1404, 2024.

[36] S. Siddiqi, R. Kern, and M. Boehm, "SAGA: A scalable framework for optimizing data cleaning pipelines for machine learning applications," *Proc. ACM Manag. Data*, 2024.

[37] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang, "The data civilizer system," in *CIDR*. www.cidrdb.org, 2017.

[38] A. Agrawal, R. Chatterjee, C. Curino, A. Floratou, N. Godwal, M. Interlandi, A. Jindal, K. Karanasos, S. Krishnan, B. Kroth, J. Leeka, K. Park, H. Patel, O. Poppe, F. Psallidas, R. Ramakrishnan, A. Roy, K. Saur, R. Sen, M. Weimer, T. Wright, and Y. Zhu, "Cloudy with high chance of DBMS: A 10-year prediction for enterprise-grade ML," *CoRR*, vol. abs/1909.00084, 2019.

[39] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer, "Enterprise data analysis and visualization: An interview study," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2917–2926, 2012.

[40] "Why users love cleanlab," 2025, https://cleanlab.ai/love/.

[41] M. Abdelaal, C. Hammacher, and H. Schöning, "REIN: A comprehensive benchmark framework for data cleaning methods in ML pipelines," in *EDBT*. OpenProceedings.org, 2023, pp. 499–511.

[42] T. Zhang, Z. A. Zhang, Z. Fan, H. Luo, F. Liu, Q. Liu, W. Cao, and L. Jian, "OpenFE: Automated feature generation with expert-level performance," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 2023, pp. 41 880–41 901.

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

[44] X. Bao, Z. Bao, Q. Duan, W. Fan, H. Lei, D. Li, W. Lin, P. Liu, Z. Lv, M. Ouyang, J. Peng, J. Zhang, R. Zhao, S. Tang, S. Zhou, Y. Wang, Q. Wei, M. Xie, J. Zhang, X. Zhang, R. Zhao, and S. Zhou, "Rock: Cleaning data by embedding ml in logic rules," in *SIGMOD (industrial track)*, 2024.

[45] "Code, datasets and full version," 2025, https://github.com/HatsuneHan/CR4ML-ICDE26.

[46] B. Hilprecht, C. Hammacher, E. S. dos Reis, M. Abdelaal, and C. Binnig, "DiffML: End-to-end differentiable ML pipelines," in *DEEM@SIGMOD*. ACM, 2023, pp. 7:1–7:7.

[47] F. Neutatz, B. Chen, Y. Alkhatib, J. Ye, and Z. Abedjan, "Data cleaning and AutoML: Would an optimizer choose to clean?" *Datenbank-Spektrum*, 2022.

[48] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *NIPS*, 2015, pp. 2962–2970.

[49] S. Krishnan and E. Wu, "AlphaClean: Automatic generation of data cleaning pipelines," *CoRR*, vol. abs/1904.11827, 2019. [Online]. Available: http://arxiv.org/abs/1904.11827

[50] P. Li, Z. Chen, X. Chu, and K. Rong, "DiffPrep: Differentiable data preprocessing pipeline search for learning over tabular data," *Proc. ACM Manag. Data*, vol. 1, no. 2, pp. 183:1–183:26, 2023.

[51] E. F. Codd, "Extending the database relational model to capture more meaning," *TODS*, vol. 4, no. 4, pp. 397–434, 1979.

[52] "Scikit-learn metrics," 2025, https://scikit-learn.org/stable/modules/model_evaluation.html.

[53] Z. Borsos, M. Mutny, and A. Krause, "Coresets via bilevel optimization for continual learning and streaming," in *NeurIPS*, 2020.

[54] Z. Wang and Y. Shen, "Incremental learning for multi-interest sequential recommendation," in *ICDE*. IEEE, 2023, pp. 1071–1083.

[55] H. Liu, S. Di, and L. Chen, "Incremental tabular learning on heterogeneous feature space," *Proc. ACM Manag. Data*, vol. 1, no. 1, pp. 18:1–18:18, 2023.

[56] T. Zhou, S. Wang, and J. A. Bilmes, "Robust curriculum learning: from clean label detection to noisy label self-correction," in *ICLR*, 2021.

[57] D. C. Hoaglin and B. Iglewicz, "Fine-tuning some resistant rules for outlier labeling," *Journal of the American statistical Association*, vol. 82, no. 400, pp. 1147–1149, 1987.

[58] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[59] B. Mirzasoleiman, J. A. Bilmes, and J. Leskovec, "Coresets for data-efficient training of machine learning models," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 6950–6960.

[60] B. Mirzasoleiman, K. Cao, and J. Leskovec, "Coresets for robust training of deep neural networks against noisy labels," in *NeurIPS*, 2020.

[61] Y. Song, T. Kim, S. Nowozin, S. Ermon, and N. Kushman, "Pixeldefend: Leveraging generative models to understand and defend against adversarial examples," in *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2018.

[62] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao, "Optimal distributed online prediction using mini-batches." *Journal of Machine Learning Research*, vol. 13, no. 1, 2012.

[63] P. Toulis, T. Horel, and E. M. Airoldi, "The proximal robbins–monro method," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 83, no. 1, pp. 188–212, 2021.

[64] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Mathematical programming*, vol. 14, pp. 265–294, 1978.

[65] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," *NeurIPS*, vol. 34, pp. 18 932–18 943, 2021.

[66] S. A. Cook, "The complexity of theorem-proving procedures," in *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*, 2023, pp. 143–152.

[67] C. Darwin, "Origin of the species," in *British Politics and the Environment in the Long Nineteenth Century*. Routledge, 2023, pp. 47–55.

[68] O. H. Babatunde, L. Armstrong, J. Leng, and D. Diepeveen, "A genetic algorithm-based feature selection," Edith Cowan University, Tech. Rep., 2014.

[69] Z. Hammoudeh and D. Lowd, "Training data influence analysis and estimation: A survey," *Machine Learning*, 2024.

[70] Y. Seldin and N. Tishby, "Multi-classification by categorical features via clustering," in *ICML*, ser. Proceedings of Machine Learning Research. PMLR, 2008, pp. 920–927.

[71] Z. Wang, Y. Zhou, M. Qiu, I. Haque, L. Brown, Y. He, J. Wang, D. Lo, and W. Zhang, "Towards fair machine learning software: Understanding and addressing model bias through counterfactual thinking," *arXiv preprint arXiv:2302.08018*, 2023.

[72] S. Guha, F. A. Khan, J. Stoyanovich, and S. Schelter, "Automated data cleaning can hurt fairness in machine learning-based decision making," in *ICDE*. IEEE, 2023, pp. 3747–3754.

[73] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *Advances in Neural Information Processing Systems*, 2022.

[74] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang, "CleanML: A benchmark for joint data cleaning and machine learning [experiments and analysis]," *CoRR*, vol. abs/1904.09483, 2019.

[75] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *ICCV*, 2016.

[76] Z. He, C. Ouyang, L. Alzubaidi, A. Barros, and C. Moreira, "Investigating imperceptibility of adversarial attacks on tabular data: An empirical analysis," *arXiv preprint arXiv:2407.11463*, 2024.

[77] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *TODS*, 2008.

[78] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[79] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, pp. 140:1–140:67, 2020.

[80] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, 2017.

[81] S. Bian, X. Ouyang, Z. Fan, and P. Koutris, "Naive Bayes classifiers over missing data: Decision and poisoning," *arXiv preprint arXiv:2303.04811*, 2023.

[82] C. Zhen, N. Aryal, A. Termehchy, and A. S. Chabada, "Certain and approximately certain models for statistical learning," *SIGMOD*, vol. 2, no. 3, pp. 1–25, 2024.

[83] J. Zhu, S. Feng, B. Glavic, and B. Salimi, "Learning from uncertain data: From possible worlds to possible models," *arXiv preprint arXiv:2405.18549*, 2024.

[84] J. Picado, J. Davis, A. Termehchy, and G. Y. Lee, "Learning over dirty data without cleaning," in *SIGMOD*, 2020, pp. 1301–1316.

[85] S. Mohammed, F. Naumann, and H. Harmouch, "Step-by-step data cleaning recommendations to improve ml prediction accuracy," *arXiv preprint arXiv:2503.11366*, 2025.

[86] C. Renggli, L. Rimanic, N. M. Gürel, B. Karlas, W. Wu, and C. Zhang, "A data quality-driven view of MLOps," *IEEE Data Eng. Bull.*, vol. 44, no. 1, pp. 11–23, 2021.

[87] F. Neutatz, B. Chen, Z. Abedjan, and E. Wu, "From cleaning before ML to cleaning for ML," *IEEE Data Eng. Bull.*, vol. 44, no. 1, pp. 24–41, 2021.

[88] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

APPENDIX

A. Datasets

- **Adult:** It contains demographic information of individuals. It is to determine whether a person's income exceeds 50K per year. The class labels are $\leq$50k or $>$50k.
- **Nursery:** It includes the family and financial circumstances of children. The prediction is to assign a ranking label to each child, including not recommend, recommend, strongly recommend, priority, and top priority.
- **Bank:** It contains the demographic, financial and social information of bank clients. The goal is to predict whether a client will apply for a term deposit (yes or no).
- **Default:** It records default payments of customers in Taiwan from April 2005 to September 2005. The ML prediction is to estimate whether a customer will default payments in the next month. The class labels are Yes or No.
- **German:** It contains financial information of bank account holders. The prediction is to determine the creditworthiness of a holder. The class labels are Good or Bad.
- **RoadSafety:** It contains detailed information on road traffic incidents in the UK from 1979 to 2015. The prediction task is to determine the severity of an accident. The class labels include slight, serious, and fatal.

B. More about model training

Below we present the incremental training for $\hat{\mathcal{M}}$ and $\mathcal{M}$.

(1) Assistant model $\hat{\mathcal{M}}$. The training for $\hat{\mathcal{M}}$ consists of two parts: (1) *model unlearning* and (2) *incremental learning*, to eliminate the negative effect of dirty tuples in $\mathcal{D}_{k-1}$ (on which $\hat{\mathcal{M}}$ is previously trained) and to update $\hat{\mathcal{M}}$ with cleaned tuples in $\mathcal{D}_k$, respectively.

(1A) Model unlearning. Denote by $\Delta\mathcal{D}_k$ the set of tuples in $\mathcal{D}_{k-1}$ that are cleaned by $\mathcal{A}_{\mathsf{CR}}$ in $\mathcal{D}_k$: $\Delta\mathcal{D}_k = \{t \mid t \in \mathcal{D}_{k-1} \text{ and } t \notin \mathcal{D}_k\}$; *i.e.,* $\Delta\mathcal{D}_k$ contains biased/dirty tuples. To debias the negative effects of $\Delta\mathcal{D}_k$ from $\hat{\mathcal{M}}$, we adopt the influence function of [43], an effective tool to estimate parameter change of a model when tuples are removed/modified in the training data. Intuitively, it mimics the scenario that we retrain $\hat{\mathcal{M}}$ with a smaller training data, $\mathcal{D}_{k-1} \setminus \Delta\mathcal{D}_k$, in the $(k-1)$-th round, by removing dirty tuples in $\Delta\mathcal{D}_k$ from $\mathcal{D}_{k-1}$. To achieve this, for each $t = (x, y) \in \Delta\mathcal{D}_k$, where x and y are the attributes and label of t, respectively, we upweight t by an infinitesimal amount ϵ; the influence of upweighting t on parameters of $\hat{\mathcal{M}}$ is

$$\mathcal{I}_{\mathsf{up,params}}(t) \stackrel{\mathsf{def}}{=} \frac{\mathrm{d}\theta_{\epsilon,t}}{\mathrm{d}\epsilon}\Big|_{\epsilon=0} = -H_\theta^{-1}\nabla_{\theta_{k-1}} l(\hat{\mathcal{M}}(x;\theta_{k-1}), y),$$

where θ_k is the parameter of $\hat{\mathcal{M}}$ in the k-th round, $\nabla_{\theta_{k-1}}$ is the partial derivation of θ_{k-1} *w.r.t.* ϵ, H_θ is the Hessian Matrix, and $l(\hat{\mathcal{M}}(x;\theta_k), y)$ is the loss of tuple t in the k-th round. Note that removing t from the training set is the same as upweighting it by $\epsilon = -\frac{1}{|D_{k-1}|}$, such that the parameter θ_k of $\hat{\mathcal{M}}$ in the k-th round can be updated as $\theta_k = \theta_{k-1} - \frac{1}{|D_{k-1}|}\sum_{t\in\Delta\mathcal{D}_k}\mathcal{I}_{\mathsf{up,params}}(t)$. In particular, when $\hat{\mathcal{M}}$ is non-convex, H_θ may contain negative eigenvalues and H_θ^{-1} does not exist. To cope with this, we follow [43] and add a small dampening coefficient to the matrix's diagonal to ensure H_θ^{-1} exists.

(1B) Incremental learning. Previous incremental strategies [54], [55] mainly focus on retaining the performance of $\hat{\mathcal{M}}$ on both old and new datasets, *i.e.,* $\mathcal{D}_{k-1}$ and $\mathcal{D}_k$. However, since $\mathcal{D}_k$ is cleaned from $\mathcal{D}_{k-1}$, we adopt a simple strategy that continuously fine-tunes $\hat{\mathcal{M}}$ with updated $\mathcal{D}_k$ in e epochs, where $\hat{\mathcal{M}}$ inherits the parameters from the model unlearning step. This strategy works well since the negative effects of dirty data in $\mathcal{D}_{k-1}$ have already been removed.

(2) Downstream model $\mathcal{M}$. We train $\mathcal{M}$ on accumulated $\mathcal{D}_{\mathsf{clean}}$ by (1) *identifying new clean data* and (2) *model refinement* as follows.

(2A) Identifying new clean data. The creator first identifies new clean data $\Delta\mathcal{D}_{\mathsf{clean}}$ from $\mathcal{D}_k$ with the help of $\hat{\mathcal{M}}$. To do this, we initialize $\Delta\mathcal{D}_{\mathsf{clean}}$ to empty and compute the dynamic loss $L_k(t)$ for each tuple t in $\mathcal{D}_k$ [56], using the exponential moving average (EMA):

$$L_k(t) = \lambda \cdot l(\hat{\mathcal{M}}(x;\theta_k), y) + (1-\lambda)\cdot L_{k-1}(t),$$

where $t = (x, y) \in \mathcal{D}_k$, $\lambda \in [0, 1]$ is a discounting factor, and $L_k(t_i)$ is the accumulated loss of t from 0 to k epochs. Intuitively, the larger $L_k(t)$, the more likely t is dirty. For each tuple $t \in \mathcal{D}_k$, we monitor the dynamic loss $L_k(t)$ of $\hat{\mathcal{M}}$. Given thresholds η_1 and η_2, we add a tuple t into $\Delta\mathcal{D}_{\mathsf{clean}}$ if one of the following is satisfied:

- $L_{k-1}(t) - L_k(t) \geq \eta_1$, indicating that $\mathcal{A}_{\mathsf{CR}}$ has effectively enhanced t's quality and thus t can be considered as a clean tuple.
- $L_k(t) \leq \eta_2$, suggesting that no matter whether t has been cleaned by $\mathcal{A}_{\mathsf{CR}}$ or not, it is likely to be clean due to its low $L_k(t)$.

(2B) Model refinement. CR4ML adopts Stochastic Gradient Descent (SGD) [58] for iterative model refinement, to leverage insights from both new clean data $\Delta\mathcal{D}_{\mathsf{clean}}$ and the accumulated clean data $\mathcal{D}_{\mathsf{clean}}$. We adapt SGD for continuous integration of clean data, such that $\mathcal{M}$ can be progressively improved with more clean data accumulated.

We apply full-gradient computation to the cleaned dataset $\mathcal{D}_{\mathsf{clean}}$, denoted as $g_{\mathsf{clean}}(\cdot)$, taking into account all data points in $\mathcal{D}_{\mathsf{clean}}$:

$$g_{\mathsf{clean}}(\theta_{k-1}) = \frac{1}{|\mathcal{D}_{\mathsf{clean}}|}\sum_{(x,y)\in\mathcal{D}_{\mathsf{clean}}} \nabla l(\mathcal{M}(x;\theta_{k-1}), y).$$

The idea is to ensure a continuous training of $\mathcal{M}$ by effectively incorporating the data cleaned after the critic (see below).

We calculate the gradient from the newly cleaned data $\Delta\mathcal{D}_{\mathsf{clean}}$ after each round of critic as follows, denoted by $g_{\Delta\mathsf{clean}}(\cdot)$:

$$g_{\Delta\mathsf{clean}}(\theta_{k-1}) = \frac{1}{|\Delta\mathcal{D}_{\mathsf{clean}}|}\sum_{(x,y)\in\Delta\mathcal{D}_{\mathsf{clean}}} \frac{\nabla l(\mathcal{M}(x;\theta_{k-1}), y)}{p(x,y)}.$$

Here $p(x, y)$ denotes the probability for tuple (x, y) in $\mathcal{D}_k$ to

be picked to clean (see below). We use $p(x, y)$ to adjust any bias that may arise from non-uniform sampling or selective cleaning. By weighting the gradient contribution of each newly cleaned tuple based on the inverse of its selection probability, the gradient computation accurately reflects the entire dataset, and thus provides an unbiased foundation for the subsequent updates of model parameters.

We compute the parameter updates by taking into account the gradients from both data sources, which are weighted by their respective proportions in the entire dataset $\mathcal{D}_k$, as follows:

$$g(\theta_{k-1}) = \frac{|\mathcal{D}_{\mathsf{clean}}|}{|\mathcal{D}_k|} g_{\mathsf{clean}}(\theta_{k-1}) + \frac{|\mathcal{D}_k| - |\mathcal{D}_{\mathsf{clean}}|}{|\mathcal{D}_k|} g_{\Delta\mathsf{clean}}(\theta_{k-1}).$$

Intuitively, the weighting scheme balances the influence of (a) the stable, already cleaned data $\mathcal{D}_{\mathsf{clean}}$, and (b) the dynamic, newly cleaned data sampled from $\mathcal{D}_k \setminus \mathcal{D}_{\mathsf{clean}}$ on the model evolution.

Using the combined gradient g, we update the parameters of $\mathcal{M}$:

$$\theta_k = \theta_{k-1} - \zeta_k g(\theta_{k-1}),$$

where ζ_k denotes the learning rate for round k, modulating the extent of each update in response to the calculated gradient.

To ensure the convergence of SGD, the learning rate ζ_k is adjusted at each round following the Robbins-Monro conditions [58], *i.e.,* on the one hand, ζ_k does not decrease too rapidly to allow sufficient exploration of the parameter space and on the other hand, it decreases fast enough to ensure convergence to an optimum.

Finally, we accumulate $\mathcal{D}_{\mathsf{clean}}$, *i.e.,* $\mathcal{D}_{\mathsf{clean}} = \Delta\mathcal{D}_{\mathsf{clean}} \cup \mathcal{D}_{\mathsf{clean}}$.

C. Proof of Theorem 1

Theorem 1: *Given an α-accurate $\mathcal{A}_{\mathsf{CR}}$,* CR4ML *guarantees to terminate and returns (a) a cleaned dataset $\mathcal{D}_{\mathsf{clean}}$ as $\mathcal{D}_{\mathcal{M}}$, and (b) model $\mathcal{M}$ trained with $\mathcal{D}_{\mathsf{clean}}$ such that* $\mathsf{acc}(\mathcal{M}, \mathcal{D}_{\mathsf{clean}}) > \mathsf{acc}(\mathcal{M}, \mathcal{D})$. □

Proof: We start with three observations.

- (F1). The CR method $\mathcal{A}_{\mathsf{CR}}$ is α-accurate ($\alpha \in (0.5, 1]$), *i.e.,* it correctly fixes erroneous data with probability α. We assume that the data correctly fixed by $\mathcal{A}_{\mathsf{CR}}$ raises no error signals by the ML model $\mathcal{M}$ as in practice, and hence remains unchanged. Intuitively, iteratively applying $\mathcal{A}_{\mathsf{CR}}$ approaches the "perfect method" Oracle, which correctly fixes all errors in the data.
- (F2). The Stochastic Gradient Descent (SGD) process converges on unbiased gradient estimates with a step size sequence $\{\gamma_t\}$ under the Robbins-Monro conditions, as verified by [62], [63].
- (F3). Training $\mathcal{M}$ with clean data makes a closer approximation of the true data distribution and better test accuracy than training with dirty data, as observed in [21], [60], [61].

Based on the observations, the proof consists of three steps.

- (S1) **Iterative data cleaning**: For part (a) of Theorem 1, we show that iterative applications of $\mathcal{A}_{\mathsf{CR}}$ reduce the error rate in the dataset (F1) such that $\mathcal{D}_{\mathcal{M}}$ obtained more accurately reflects the true data distribution and improves the accuracy of $\mathcal{M}$ (F3).
- (S2) **Convergence by unbiased gradient estimation**: For part (b) of the theorem, we prove the convergence of model refinement with progressively cleaned data by (F2); that is, CR4ML ensures unbiased gradient estimation in SGD when incorporating gradients from both the accumulated and newly cleaned datasets. CR4ML approaches this by adjusting gradients based on the probability of the data points being cleaned.
- (S3) **Accuracy improvement**: Also for part (b) of the theorem, we show that $\mathcal{M}$ trained with $\mathcal{D}_{\mathsf{clean}}$ becomes more accurate. We first examine model convergence results in both convex and non-convex optimization scenarios. We then explore the impact of CR4ML's cleaning on the accuracy of $\mathcal{M}$ in terms of dynamic loss reduction. We show that the selection criteria of $\Delta\mathcal{D}_{\mathsf{clean}}$ yield lower dynamic losses, thereby demonstrating the improved model accuracy of $\mathcal{M}$ trained with $\mathcal{D}_{\mathcal{M}}$.

Below we provide the details of each step in the proof.

(S1) **Iterative data cleaning**: Given an initial dataset $\mathcal{D}$ with an error rate e_0, we inductively show that iterative applications of α-accurate $\mathcal{A}_{\mathsf{CR}}$ can reduce the error rate below a threshold ϵ in k iterations, where k is determined by α, ϵ and $\mathbb{E}[p(x, y)]$ (see below).

For a dataset $\mathcal{D}_k$ in the k-th iteration, the likelihood of selecting its erroneous data points for cleaning is:

$$\mathbb{E}[p(x, y)] = \frac{1}{|\mathcal{D}_k|} \sum_{(x,y)\in\mathcal{D}_k} p(x, y),$$

where $|\mathcal{D}_k|$ is the size of $\mathcal{D}_k$, and $p(x, y)$ is determined by AlgIA and AlgIT. Assuming that the error rate is e_k after k iterations, then by incorporating $\mathbb{E}[p(x, y)]$, the error rate e_{k+1} is

$$e_{k+1} = e_k \cdot (1 - \alpha \cdot \mathbb{E}[p(x, y)]),$$

where α is the accuracy of $\mathcal{A}_{\mathsf{CR}}$. To satisfy $e_k < \epsilon$, the equation:

$$e_0 \cdot (1 - \alpha \cdot \mathbb{E}[p(x, y)])^k < \epsilon$$

determines the necessary number k of iterations. That is,

$$k > \frac{\log(\epsilon/e_0)}{\log(1 - \alpha \cdot \mathbb{E}[p(x, y)])}.$$

This shows the connection between the number k of iterations and (a) the average error selection probability $\mathbb{E}[p(x, y)]$, and (b) the accuracy α of CR method $\mathcal{A}_{\mathsf{CR}}$. For example, when $\alpha = 0.6$ and $\mathbb{E}[p(x, y)] = 0.6$, to reduce $e_0 = 0.1$ below $\epsilon = 0.01$, it typically requires about 4 iterations. That is, we need k rounds in the creator-critic framework of CR4ML to get the cleaned dataset $\mathcal{D}_{\mathcal{M}}$. In practice, CR4ML typically converges in much fewer rounds.

(S2) **Convergence via unbiased gradient estimation**: CR4ML converges when the gradient estimation during the training process is unbiased. The gradient from the already cleaned data $\mathcal{D}_{\mathsf{clean}}$ in the k-th round, denoted by $g_{\mathsf{clean}}(\theta_{k-1})$, is computed as follows:

$$g_{\mathsf{clean}}(\theta_{k-1}) = \frac{1}{|\mathcal{D}_{\mathsf{clean}}|} \sum_{(x,y)\in\mathcal{D}_{\mathsf{clean}}} \nabla l(\mathcal{M}(x;\theta_{k-1}),y),$$

where $\nabla l(\mathcal{M}(x;\theta_{k-1}),y)$ denotes the gradient of the loss function l *w.r.t.* model $\mathcal{M}$'s parameters θ_{k-1} in the (k-1)-th round, evaluated at the predicted label $\mathcal{M}(x;\theta_{k-1})$ and the truth label y of a data point (x,y). That is, the continuous training of $\mathcal{M}$ integrates all tuples in $\mathcal{D}_{\mathsf{clean}}$ and is unbiased by the use of the full-gradient method.

Denote by $g_{\Delta\mathsf{clean}}(\theta_{k-1})$ the gradient from the newly cleaned data $\Delta\mathcal{D}_{\mathsf{clean}}$ in the k-th round. Given probability $p(x,y)$ for selecting data to clean, CR4ML conducts gradient estimation as follows:

$$g_{\Delta\mathsf{clean}}(\theta_{k-1}) = \frac{1}{|\Delta\mathcal{D}_{\mathsf{clean}}|} \sum_{(x,y)\in\Delta\mathcal{D}_{\mathsf{clean}}} \frac{\nabla l(\mathcal{M}(x;\theta_{k-1}),y)}{p(x,y)}.$$

To show the unbiasedness of $g_{\Delta\mathsf{clean}}(\theta_{k-1})$, we consider its expectation *w.r.t.* the data selection process for cleaning:

$$\mathbb{E}[g_{\Delta\mathsf{clean}}(\theta_{k-1})] = \mathbb{E}\left[\frac{1}{|\Delta\mathcal{D}_{\mathsf{clean}}|} \sum_{(x,y)\in\Delta\mathcal{D}_{\mathsf{clean}}} \frac{\nabla l(\mathcal{M}(x;\theta_{k-1}),y)}{p(x,y)}\right].$$

Given that each data point (x,y) is independently chosen from $\mathcal{D}_k \setminus \mathcal{D}_{\mathsf{clean}}$ with probability $p(x,y)$, the expected contribution of the gradient for any selected data is adjusted by its selection probability. This ensures that the overall expectation reflects the true gradient contribution from the subset $\mathcal{D}_k \setminus \mathcal{D}_{\mathsf{clean}}$:

$$\mathbb{E}\left[\frac{\nabla l(\mathcal{M}(x;\theta_{k-1}),y)}{p(x,y)}\right] = \sum_{(x,y)\in\mathcal{D}_k\setminus\mathcal{D}_{\mathsf{clean}}} \nabla l(\mathcal{M}(x;\theta_{k-1}),y).$$

In light of this, the combined gradient $g(\theta_{k-1})$ for model parameter update integrates gradients from both sources, appropriately weighted by their proportions in the total dataset $\mathcal{D}_k$:

$$g(\theta_{k-1}) = \frac{|\mathcal{D}_{\mathsf{clean}}|}{|\mathcal{D}_k|} g_{\mathsf{clean}}(\theta_{k-1}) + \frac{|\mathcal{D}_k| - |\mathcal{D}_{\mathsf{clean}}|}{|\mathcal{D}_k|} g_{\Delta\mathsf{clean}}(\theta_{k-1}).$$

Given the unbiasedness of $g_{\mathsf{clean}}(\theta_{k-1})$ and the proven unbiased expectation of $g_{\Delta\mathsf{clean}}(\theta_{k-1})$, the expectation of the combined gradient $E[g(\theta_{k-1})]$ accurately reflects the true gradient of the entire dataset $\mathcal{D}_k$ (note that $|\mathcal{D}_k| = |\mathcal{D}_{\mathsf{clean}}| = |\mathcal{D}_{\mathcal{M}}|$ in the end). This ensures unbiased gradient estimation and model parameter updates:

$$\mathbb{E}[g(\theta_{k-1})] = \frac{1}{|\mathcal{D}_{\mathcal{M}}|} \sum_{(x,y)\in\mathcal{D}_{\mathcal{M}}} \nabla l(\mathcal{M}(x;\theta_{k-1}),y).$$

Consequently, model parameters are updated according to:

$$\theta_k = \theta_{k-1} - \zeta_k g(\theta_{k-1}),$$

where the learning rate ζ_k satisfies Robbins-Monro conditions. From this it follows CR4ML's convergence:

$$\sum_{k=1}^{\infty} \zeta_k = \infty, \quad \sum_{k=1}^{\infty} \zeta_k^2 < \infty.$$

That is, CR4ML ensures stable convergence in model training and data processing of the creator-critic process.

(S3) **Accuracy improvement**: We next show that when CR4ML converges, $\mathcal{M}$ is more accurate than $\mathcal{M}$ trained with the original dataset $\mathcal{D}$, given that $\mathcal{M}$ is trained with progressively cleaned datasets. We prove that the process reduces dynamic loss by considering convex and non-convex optimization cases.

Convex optimization. In convex optimization scenarios, CR4ML ensures that model $\mathcal{M}$ converges at the global optimum θ^* via gradient descent. Employing unbiased gradient updates $g(\theta_{k-1})$, $\mathcal{M}$ iteratively refines its parameters with the cleaned data in $\mathcal{D}_{\mathsf{clean}}$. As more data is cleaned and integrated into the training process, $\mathcal{M}$ gradually approaches θ^*, demonstrating convergence as:

$$\lim_{k\to\infty} \theta_k = \theta^*.$$

Non-convex optimization. In non-convex optimization, gradient descent on $\mathcal{M}$ typically converges at local optima or saddle points. However, by (F3), CR4ML enhances the exploration of the loss function's landscape by progressively incorporating cleaned data during training. This increases the chances of finding lower local minima, and improves the accuracy of $\mathcal{M}$ trained with uncleaned data in non-convex settings. We show this by dynamic loss analysis.

Given a dataset $\mathcal{D}$, let $D^{(k)}_{\mathsf{clean}}$ and $\Delta\mathcal{D}^{(k)}_{\mathsf{clean}}$ denote the dataset D_{clean} and $\Delta\mathcal{D}_{\mathsf{clean}}$ after k rounds, respectively. The conditions for a data point t to be added to $\Delta\mathcal{D}^{(k)}_{\mathsf{clean}}$ are either $L_{k-1}(t) - L_k(t) \geq \eta_1$ or $L_k(t) \leq \eta_2$, where $L_k(t)$ is the dynamic loss of t in round k.

The total dynamic loss of $\mathcal{M}$ trained with $\mathcal{D}^{(k)}_{\mathsf{clean}}$ at the k-th round is:

$$\mathcal{L}^{(k)}_{\mathsf{clean}} = \sum_{t\in\mathcal{D}^{(k)}_{\mathsf{clean}}} L_k(t),$$

while the total dynamic loss of $\mathcal{M}$ trained with $\mathcal{D}$ is:

$$\mathcal{L}^{(k)}_{\mathcal{D}} = \sum_{t\in\mathcal{D}} L_k(t).$$

In the dynamic loss analysis, $\mathcal{D}^{(k)}_{\mathsf{clean}} = \bigcup_{i=0}^{k-1} \Delta\mathcal{D}^{(i)}_{\mathsf{clean}}$, where $\Delta\mathcal{D}^{(i)}_{\mathsf{clean}}$ consists of tuples t categorized by their original state in dataset $\mathcal{D}$, either clean or dirty, processed in the i-th round.

Inherently clean tuples are incorporated into $\Delta\mathcal{D}^{(k)}_{\mathsf{clean}}$ when they meet the condition $L_k(t) \leq \eta_2$, due to their inherent cleanliness and alignment with the model's accuracy criteria, thus resulting in a low dynamic loss. Their impact on the dynamic loss $\mathcal{L}^{(k)}_{\mathsf{clean}}$ aligns with their effect on the dynamic loss $\mathcal{L}^{(k)}_{\mathcal{D}}$, since their contribution to the model's learning is both stable and predictable.

On the other hand, newly-cleaned dirty tuples are selected for $\Delta\mathcal{D}^{(k)}_{\mathsf{clean}}$ when they satisfy $L_{k-1}(t) - L_k(t) \geq \eta_1$, indicating a significant reduction in the dynamic loss by $\mathcal{A}_{\mathsf{CR}}$. Such tuples decrease $\mathcal{L}^{(k)}_{\mathsf{clean}}$ more substantially compared to $\mathcal{L}^{(k)}_{\mathcal{D}}$. Denote the quantity of such tuples by d. Its reduction in dynamic loss is quantified as:

$$\Delta\mathcal{L}^{(k)}_{\mathsf{clean}} = \sum_{i=1}^{k} \sum_{t\in\Delta\mathcal{D}^{(i)}_{\mathsf{clean}}} (L_{i-1}(t) - L_i(t)) \geq \eta_1 \cdot d \geq 0.$$

Therefore, suppose that CR4ML terminates at round k. Then we have the following, taking into account the reduction in cleaning loss and the contribution of inherently clean data:

$$\mathcal{L}^{(k)}_{\mathsf{clean}} + \Delta\mathcal{L}^{(k)}_{\mathsf{clean}} \leq \mathcal{L}^{(k)}_{\mathcal{D}}.$$

Thus $\mathcal{L}^{(k)}_{\mathsf{clean}} \leq \mathcal{L}^{(k)}_{\mathcal{D}} - \eta_1 \cdot d$. From this it follows that $\mathsf{acc}(\mathcal{M}, \mathcal{D}_{\mathsf{clean}}) > \mathsf{acc}(\mathcal{M}, \mathcal{D})$, *i.e.,* part (b) of Theorem 1 holds. □

D. Proof of Theorem 2

Theorem 2: *Given an α-accurate $\mathcal{A}_{\mathsf{CR}}$ with $\alpha > 0.5$,* CR4ML *achieves an expected submodular improvement in model accuracy. The expected accuracy $\mathbb{E}(\mathsf{acc}(\mathcal{M}, \mathcal{D}_{\mathsf{clean}}))$ of the model $\mathcal{M}$ trained on cleaned data $\mathcal{D}_{\mathsf{clean}}$ is guaranteed to reach at least 63% of the optimal expected accuracy* □

Proof: We develop the proof under the following assumptions:

- (A1) **Data consistency of the coreset** $\mathcal{C}_k$: As outlined in Section III, we assume that in each iteration, the coreset $\mathcal{C}_k$ is a representative subset of the cleaned dataset $\mathcal{D}_{\mathsf{clean}}$, sharing the same underlying data distribution. This consistency is supported by the α-accuracy of $\mathcal{A}_{\mathsf{CR}}$ in providing more cleaned data and by the coreset selection strategy [56].
- (A2) **Stability of model accuracy gain**: We assume that the incremental accuracy gain of the ML models remains stable across iterations. This means that as influential attributes and tuples are progressively cleaned, the marginal improvement in accuracy does not fluctuate significantly due to training randomness. This assumption is supported by studies on the convergence and stability of Stochastic Gradient Descent (SGD) [62], [63].
- (A3) **Predictive accuracy of auxiliary models $\hat{\mathcal{M}}_{\mathsf{int}}$ and $\hat{\mathcal{M}}_t$**: We assume that accuracy improvements in models $\hat{\mathcal{M}}_{\mathsf{int}}$ and $\hat{\mathcal{M}}_t$ are proportional to those in the model $\hat{\mathcal{M}}$, *i.e.,* greater gains in $\hat{\mathcal{M}}_{\mathsf{int}}$ or $\hat{\mathcal{M}}_t$ correspond to greater gains in $\hat{\mathcal{M}}$. This assumption relies on the robustness of boosting for effectively capturing influential attribute effects [42] and on the established accuracy of influence functions for assessing individual tuple impacts [41], [43].

We proceed with the proof through the following steps:

- (S1) **Define gain functions**: We define gain functions to measure the *expected accuracy improvement* in the auxiliary models when influential attributes or tuples are cleaned. By focusing on the expected gains, we capture the average impact of CR, with the stability from A2 ensuring the consistency despite randomness in training and data selection. Additionally, by Assumption A1, the coreset $\mathcal{C}_k$ reliably represents $\mathcal{D}_{\mathsf{clean}}$, providing an effective basis for evaluating improvements in both $\hat{\mathcal{M}}$ and $\mathcal{M}$.
- (S2) **Prove monotonicity and submodularity**: We verify the monotonicity of the gain functions through the positive contributions of influential attributes and tuples selected by algorithms AlgIA and AlgIT, respectively, ensuring that the accuracy does not decrease as more data points are cleaned. The submodularity is verified by the diminishing returns property, *i.e.,* as additional tuples are cleaned, the marginal gain in accuracy decreases due to fewer remaining errors in the dataset for the $\mathcal{A}_{\mathsf{CR}}$ to correct.
- (S3) **Approximation guarantee**: Utilizing the submodularity and monotonicity, we apply results from submodular optimization to obtain the approximation guarantee.

Below we provide the details of each step in the proof.

(S1) **Gain functions**. We define functions $g(\mathcal{S})$ and $g(\mathcal{T})$ to measure the expected accuracy improvement in the auxiliary models $\hat{\mathcal{M}}_{\mathsf{int}}$ and $\hat{\mathcal{M}}_t$ when influential attributes $\mathcal{S}$ or tuples $\mathcal{T}$ are cleaned. Let G denote the expected accuracy improvement in $\hat{\mathcal{M}}$.

Attribute gain function. For a set of influential attributes $\mathcal{S}$ selected by AlgIA in the k-th round, the gain function $g(\mathcal{S})$ is defined as:

$$g(\mathcal{S}) = \mathbb{E}\left[\mathsf{acc}\left(\hat{\mathcal{M}}_{\mathsf{int}}(\mathcal{S}), \mathcal{C}_k\right) - \mathsf{acc}\left(\hat{\mathcal{M}}_{\mathsf{int}}, \mathcal{C}_k\right)\right],$$

where $\hat{\mathcal{M}}_{\mathsf{int}}(\mathcal{S})$ denotes the model $\hat{\mathcal{M}}_{\mathsf{int}}$ trained on data after cleaning $\mathcal{S}$, and $\mathsf{acc}\,(\cdot, \mathcal{C}_k)$ is the model accuracy on $\mathcal{C}_k$.

Tuple gain function. Similarly, for an influential tuple set $\mathcal{T}$ selected by AlgIT in the k-th round, the gain function $g(\mathcal{T})$ is defined as:

$$g(\mathcal{T}) = \mathbb{E}\left[\mathsf{acc}\left(\hat{\mathcal{M}}_t(\mathcal{T}), \mathcal{C}_k\right) - \mathsf{acc}\left(\hat{\mathcal{M}}_t, \mathcal{C}_k\right)\right],$$

where $\hat{\mathcal{M}}_t(\mathcal{T})$ denotes $\hat{\mathcal{M}}_t$ trained after cleaning tuples in $\mathcal{T}$.

Gain function for $\hat{\mathcal{M}}$. By Assumption A3, the expected accuracy gain in $\hat{\mathcal{M}}$ is proportional to the gains in auxiliary models:

$$G(\mathcal{S}) \propto g(\mathcal{S}), \quad G(\mathcal{T}) \propto g(\mathcal{T}),$$

where $G(\mathcal{S})$ and $G(\mathcal{T})$ represent expected accuracy gains in $\hat{\mathcal{M}}$ from cleaning attributes in $\mathcal{S}$ and tuples in $\mathcal{T}$, respectively.

(S2) **The monotonicity and submodularity of** G. We analyze the behavior under incremental data cleaning and show that iterative cleaning leads to diminishing expected accuracy gains such that G satisfies the monotonicity and submodularity.

Monotonicity of G: For any sets $A \subseteq B \subseteq E$, where E represents all possible influential attributes or tuples, we show that:

$$G(A) \leq G(B).$$

Since the CR method $\mathcal{A}_{\mathsf{CR}}$ is α-accurate with $\alpha > 0.5$, it ensures that the error rate in the dataset is non-increasing. As shown in Figures 3 and 4, AlgIA (resp. AlgIT) selects influential attributes (resp. tuples) that contribute positively to the accuracy of $\hat{\mathcal{M}}_{\mathsf{int}}$ (resp. $\hat{\mathcal{M}}_t$). By S1, the auxiliary models $\hat{\mathcal{M}}_{\mathsf{int}}$ and $\hat{\mathcal{M}}_t$ reliably reflect the accuracy improvement trends of $\hat{\mathcal{M}}$, maintaining an expected non-decreasing accuracy in $\hat{\mathcal{M}}$ when additional data points are cleaned. Therefore, cleaning subset B provides at least as much accuracy improvement as cleaning subset A, confirming the monotonicity of the gain function G in our framework.

Submodularity of G: We show that the gain of cleaning an additional influential attribute or tuple $t \notin B$ for subsets $A \subseteq B \subseteq E$ satisfies:

$$G(A \cup \{t\}) - G(A) \geq G(B \cup \{t\}) - G(B).$$

When t is an influential attribute, cleaning data points in $\mathcal{D} - \mathcal{D}_{\mathsf{clean}}$ based on the influential attributes identified in $B \cup \{t\}$ yields a non-increasing accuracy improvement compared to $A \cup \{t\}$. Since $A \subseteq B$ implies that $\mathcal{D}$ at the stage of B has fewer uncleaned tuples than at the stage of A, the impact of cleaning t is reduced since fewer errors remain in the attribute

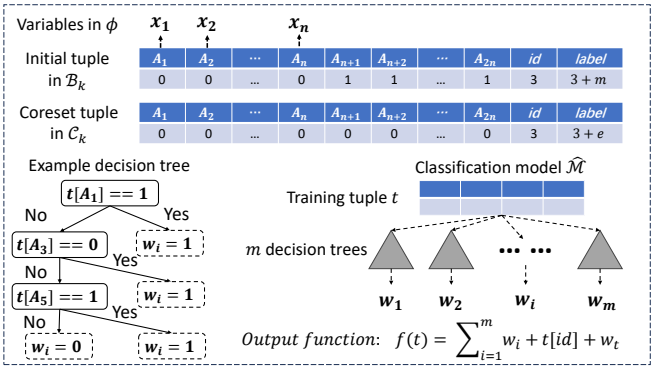


Fig. 6: Problem IA reduction example

e for $\mathcal{A}_{\mathsf{CR}}$ to correct.

When t is an influential tuple, cleaning data points in $\mathcal{D} - \mathcal{D}_{\mathsf{clean}}$ associated with the stage either $A \cup \{t\}$ or $B \cup \{t\}$ produces the same accuracy improvement. Since cleaning the same tuple results in a fixed accuracy gain, the improvement remains identical.

Therefore, in both cases, we have:

$$G(A \cup \{t\}) - G(A) \geq G(B \cup \{t\}) - G(B).$$

By Assumption A2 (The stability of model accuracy gain), these incremental gains remain stable across iterations, ensuring a consistent trend of diminishing returns of G, and thus is submodular.

(S3) **Approximation guarantee**. Since G is non-negative, monotonic, and submodular, classical submodular optimization results [64] guarantee a $(1 - 1/e) > 63\%$ approximation bound with the greedy strategies of AlgIA and AlgIT, where $e \approx 2.718$ is the base of the natural logarithm. This ensures that the expected improvement in accuracy achieved by $\hat{\mathcal{M}}$ on the cleaned dataset $\mathcal{D}_{\mathsf{clean}}$ is at least 63% of the optimal achievable improvement.

Consequently, the expected accuracy of $\hat{\mathcal{M}}$ on $\mathcal{D}_{\mathsf{clean}}$, combining its pre-cleaning accuracy and the achieved improvement, satisfies:

$$\mathbb{E}(\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_{\mathsf{clean}})) \geq (1 - 1/e) \cdot \mathsf{acc}_{\mathsf{opt}}(\hat{\mathcal{M}}),$$

where $\mathsf{acc}_{\mathsf{opt}}(\hat{\mathcal{M}})$ denotes the optimal expected accuracy that is achievable by $\hat{\mathcal{M}}$ through data cleaning.

To extend this result to the target model $\mathcal{M}$, Assumption A1 ensures that the coreset $\mathcal{C}_k$ reliably represents $\mathcal{D}_{\mathsf{clean}}$, allowing accuracy improvements observed in $\hat{\mathcal{M}}$ to approximate those expected on $\mathcal{D}_{\mathsf{clean}}$. Furthermore, Assumption A2 guarantees stability in incremental accuracy gains across iterations, such that these improvements generalize consistently to $\mathcal{M}$.

Thus, the expected accuracy of $\mathcal{M}$ satisfies the following:

$$\mathbb{E}(\mathsf{acc}(\mathcal{M}, \mathcal{D}_{\mathsf{clean}})) \geq (1-1/e) \cdot \mathsf{acc}_{\mathsf{opt}}(\mathcal{M}) > 63\% \cdot \mathsf{acc}_{\mathsf{opt}}(\mathcal{M}).$$

This result confirms that the expected accuracy of $\mathcal{M}$ trained on the cleaned dataset $\mathcal{D}_{\mathsf{clean}}$ achieves at least 63% of the optimal accuracy. This completes the proof of Theorem 2. □

E. Proof of Theorem 3

Theorem 3: *Problem* IA *is NP-hard.* □

Proof: We prove the NP-hardness of problem IA by reduction from the 3-SAT problem, which is NP-complete (cf. [88]). The 3-SAT problem is to decide, given a Boolean formula ϕ in the conjunctive normal form with n variables and m clauses, each containing three literals (a variable x or its negation $\neg x$), whether there exists a truth value assignment to the variables that makes ϕ true. The reduction from a 3-SAT instance to an IA instance is given as follows.

(1) Schema $\mathcal{R}$: We use a relation schema $(A_1, A_2, \ldots, A_{2n}, \mathsf{id}, \mathsf{label})$, where for all $i \in [1, 2n]$, A_i is a Boolean attribute, id is in the range $[1, m]$, and label is in the range $[1, 2m]$.

(2) Dataset $D_k = (\mathcal{B}_k, \mathcal{C}_k)$: Construct relations $\mathcal{B}_k$ and $\mathcal{C}_k$ of the schema above, each having m tuples. For tuples t in $\mathcal{B}_k$, $t[\mathsf{id}]$ takes values from 1 to m, respectively, $t[A_i] = 0$ ($i \in [1, n]$), $t[A_j] = 1$ ($j \in [n+1, 2n]$), and $t[\mathsf{label}] = t[\mathsf{id}] + m$. Intuitively, we use attributes $A_1, A_2, \ldots, A_n$ of tuples in $\mathcal{B}_k$ to encode the n variables of ϕ.

For tuples s in $\mathcal{C}_k$, their id attributes are instantiated in the same way as their counterparts in $\mathcal{B}_k$. However, we set $s[A_i] = 0$ ($i \in [1, 2n]$), and $s[\mathsf{label}] = s[\mathsf{id}] + e$, where e is the number of satisfied clauses in ϕ when all variables are set false.

Intuitively, we will use a CR method $\mathcal{A}_{\mathsf{CR}}$ to simulate the truth assignment of ϕ, an ML model $\hat{\mathcal{M}}$ to check the satisfaction of the clauses of ϕ, and the correspondence between attributes A_i and A_{n+i} ($i \in [1, n]$) in $\mathcal{B}_k$ to identify influential attributes.

(3) CR method $\mathcal{A}_{\mathsf{CR}}$: We use a rule-based method $\mathcal{A}_{\mathsf{CR}}$ to adjust attributes in $\mathcal{B}_k$. It employs a single rule that makes use of a set $\mathcal{S}$ of influential attributes. The rule states that for each tuple $t \in \mathcal{B}_k$ and each $i \in [1, n]$, if the attribute A_i is identified as an influential attribute, then $t[A_i]$ takes the value of $t[A_{i+n}]$; while for $i > n$, $t[A_i]$ remains unchanged. That is, $\mathcal{A}_{\mathsf{CR}}$ aligns correlated attributes such that the attribute pair A_i and A_{i+n} have the same value.

(4) Model $\hat{\mathcal{M}}$: We use a classification tree-based (regression) model $\hat{\mathcal{M}}$ to evaluate clause satisfaction based on the attribute values in $\mathcal{D}_k$. It consists of m decision trees, each for a clause C_i in ϕ. The i-th tree yields $w_i = 1$ if clause C_i is satisfied, otherwise $w_i = 0$. Figure 6 shows an example decision tree for the clause $C_i = (x_1 \vee \neg x_3 \vee x_5)$.

Specifically, the model $\hat{\mathcal{M}}$ predicts labels for each tuple $t \in D_k$ using the regression function $f(t) = \sum_{i=1}^{m} w_i + t[\mathsf{id}] + w_t$, where w_t is a trainable parameter, making f differentiable on the input tuple t. The label is predicated as $\lfloor f(t) \rfloor$.

Observe that before applying $\mathcal{A}_{\mathsf{CR}}$ to $\mathcal{B}_k$, for any tuple $t \in \mathcal{B}_k$, the function $f(t)$ is evaluated as $e + t[\mathsf{id}] + w_t$. Since $t[\mathsf{label}] = t[\mathsf{id}] + m$ for tuples $t \in \mathcal{B}_k$, the training process adjusts w_t to $m - e$ such that $f(t) = t[\mathsf{label}]$. When $\hat{\mathcal{M}}$ predicts any tuple $s \in \mathcal{C}_k$ in the validation process, $f(s) = e + s[\mathsf{id}] + m - e$, which is not equal to $s[\mathsf{label}] = s[\mathsf{id}] + e$; as a consequence, initially $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) = 0$.

(5) Parameter δ: Define $\delta = 1$, which is the maximum possible

improvement under an initial $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) = 0$.

The reduction can be obviously constructed in PTIME. We next show that there exists an attribute set $\mathcal{S}$ that makes $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) = 1$ if and only if the 3-SAT instance ϕ is satisfiable.

$\Rightarrow$ First, we assume that there exists a set $\mathcal{S}$ such that $\mathcal{A}_{CR}$ on $\mathcal{S}$ makes $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) \geq \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$. More specifically, the application of $\mathcal{A}_{CR}$ ensures that for any tuple t in $\mathcal{B}_k^{\mathcal{S}}$ and for any attribute $A_i \in \mathcal{S}$, $t[A_i] = 1$; and for any attribute $A_j \notin \mathcal{S}$, $t[A_j] = 0$. In fact, $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) \geq \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$ implies that $\mathcal{M}$ classifies all tuples in $\mathcal{C}_k$ correctly. Given $f(s) = \sum_{i=1}^{m} w_i + s[\mathsf{id}] + w_t$ for each $s \in \mathcal{C}_k$ and $s[\mathsf{label}] = e + s[\mathsf{id}]$, it necessitates $w_t = 0$. Recall that during training on tuples t in $\mathcal{B}_k^{\mathcal{S}}$ using mean squared error, the optimal value for w_t is determined by the formula $w_t = \frac{\sum_{t \in \mathcal{B}_k^{\mathcal{S}}} \left(t[\mathsf{label}] - t[\mathsf{id}] - \sum_{i=1}^{m} w_i\right)}{m}$. If this optimal w_t is found to be 0, it implies that $\sum_{i=1}^{m} w_i = m$, indicating that each of the m clauses has been satisfied by the current attribute settings in $\mathcal{B}_k^{\mathcal{S}}$.

We give the truth assignment in ϕ as follows. For each tuple t in $\mathcal{B}_k^{\mathcal{S}}$ and $i \in [1, n]$, if $t[A_i] = 1$, then variable x_i is assigned true. Conversely, if $t[A_i] = 0$, then x_i is set to false.

One can verify that this truth assignment satisfies ϕ. Observe that the truth assignment is determined by $\hat{\mathcal{M}}$'s decision trees, each aligned with a specific clause C_i from ϕ. A tree yields $w_i = 1$ iff for any tuple $t \in \mathcal{B}_k^{\mathcal{S}}$ and $i \in [1, n]$, at least one $t[A_i]$ value meets the decision criterion, equivalently satisfying C_i by ensuring a literal's truth. This leads to $\sum_{i=1}^{m} w_i = m$, with $w_i = 1$ for all trees, which verifies ϕ's satisfiability via the truth assignment derived from $\mathcal{S}$.

$\Leftarrow$ Conversely, we show that the existence of a satisfying truth assignment for ϕ guarantees the existence of a set $\mathcal{S}$ of attributes such that $\mathcal{A}_{\mathsf{CR}}$ on $\mathcal{S}$ in $\mathcal{B}_k$ ensures $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) \geq \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$. Given a satisfying truth assignment for ϕ, the set $\mathcal{S}$ is identified as follows: For each variable x_i in ϕ, if x_i is true in the truth assignment, then we include the attribute A_i in the influential attribute set $\mathcal{S}$. Then we perform $\mathcal{A}_{\mathsf{CR}}$ on $\mathcal{S}$ to obtain $\mathcal{D}_k^{\mathcal{S}}$, where for each $t \in \mathcal{B}_k^{\mathcal{S}}$ and $A_i \in \mathcal{S}$, $t[A_i] = 1$, while $t[A_j] = 0$ for $A_j \notin \mathcal{S}$. Suppose by contradiction that $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{S}}) < \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$, *i.e.,* some $\mathcal{C}_k$ tuples are incorrectly classified. Given $f(s) = e + s[\mathsf{id}] + w_t$ for $s \in \mathcal{C}_k$ where $s[\mathsf{label}] = e + s[\mathsf{id}]$, w_t cannot be 0 in order to produce incorrect predictions. A non-zero w_t contradicts the premise that ϕ is satisfiable. Indeed, during the training, for a tuple $t \in \mathcal{B}_k^{\mathcal{S}}$ with $t[\mathsf{label}] = t[\mathsf{id}] + m$, $\hat{\mathcal{M}}$ outputs $f(t) = m + t[\mathsf{id}] + w_t$; this indicates that w_t must be 0 to train an accurate classifier $\hat{\mathcal{M}}$. □

F. Proof of Theorem 4

Theorem 4: *Problem* IT *is NP-hard.* □

Proof: The NP-hardness of IT is also shown by reduction from the 3-SAT problem. The reduction is given as follows. Consider a 3-SAT instance ϕ that has n variables and m conjunctive clauses.

(1) Schema $\mathcal{R}$: We use a ternary schema $(A_1, \mathsf{id}, \mathsf{label})$, where A_1 is a Boolean, id is in the range $[1, n]$, and label is in the range $[1, n + m]$.

(2) Dataset $\mathcal{D}_k = (\mathcal{B}_k, \mathcal{C}_k)$. We create relations $\mathcal{B}_k$ and $\mathcal{C}_k$ of the schema above, each having n tuples. For tuples t in $\mathcal{B}_k$, $t[\mathsf{id}]$ takes values from 1 to n, respectively, $t[A_1] = 0$, and $t[\mathsf{label}] = t[\mathsf{id}] + m$. Intuitively, we use the n tuples in $\mathcal{B}_k$ to encode the n variables of ϕ.

For tuples s in $\mathcal{C}_k$, their id attributes are instantiated in the same way as their counterparts in $\mathcal{B}_k$. However, we set $s[A_1] = 1$, and $s[\mathsf{label}] = s[\mathsf{id}] + e$, where e is the number of satisfied clauses in ϕ when all variables in the 3-SAT instance ϕ are set true.

As will be seen shortly, we will use a CR method $\mathcal{A}_{\mathsf{CR}}$ to simulate the truth assignment of ϕ, an ML model $\hat{\mathcal{M}}$ to check the satisfaction of the clauses of ϕ, and the pairwise correspondence between tuples in $\mathcal{B}_k$ and those in $\mathcal{C}_k$ to identify influential tuples.

(3) CR method $\mathcal{A}_{\mathsf{CR}}$: We use $\mathcal{A}_{\mathsf{CR}}$ that, for each tuple t in $\mathcal{B}_k$, if t is identified as influential, then $t[A_1]$ takes the value of $s[A_1]$, where s is the tuple in $\mathcal{C}_k$ such that $t[\mathsf{id}] = s[\mathsf{id}]$. That is, $\mathcal{A}_{\mathsf{CR}}$ ensures that for any pair $t \in \mathcal{B}_k$ and $s \in \mathcal{C}_k$ of tuples, if they have the same id, then the two must have the same A_1-attribute value.

(4) Model $\hat{\mathcal{M}}$: Design a classification tree-based (regression) model $\hat{\mathcal{M}}$ that uses a decision tree for each 3-SAT clause, where each tree decides whether its clause is satisfied based on the A_1 values. The input to the model regression function is an array of A_1 values in the size of n, arranged in order by the tuple identifiers; that is, $\hat{\mathcal{M}}$ processes a batch d of n tuples simultaneously. The trees yield $w_i = 1$ for satisfied clauses C_i, 0 otherwise. Figure 7 shows an example for the clause $C_i = (x_1 \vee \neg x_3 \vee x_5)$, where $d[i]$ indicates the i-th A_1 value of the input d. The regression function $f(d) = \sum_{i=1}^{m} w_i + w_t$ aggregates the trees' outputs and $\hat{\mathcal{M}}$ predicts the label of each tuple t within the batch by $f(d) + t[\mathsf{id}]$, where w_t is the trainable parameter, making f differentiable on the input d.

Observe that before applying $\mathcal{A}_{\mathsf{CR}}$ on $\mathcal{B}_k$, for a tuple $t \in \mathcal{B}_k$ with label $t[\mathsf{id}] + m$, its predicted label is $f(d) + t[\mathsf{id}] = e' + w_t + t[\mathsf{id}]$. Here e' is the number of satisfied clauses of ϕ under a truth assignment that assigns false to all the variables of ϕ. Based on the mean squared error, the training adjusts w_t to $m - e' \neq 0$. When $\hat{\mathcal{M}}$ predicts any tuple $s \in \mathcal{C}_k$ by batching n tuples in $\mathcal{C}_k$ into an input d', the predicted label of s is $f(d') + s[\mathsf{id}] = e + m - e' + s[\mathsf{id}]$, which is not equal to $s[\mathsf{id}] + e$; as a result, we have that $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) = 0$.

(5) Parameter δ: Choose $\delta = 1$ as the target for maximum accuracy improvement under an initial $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) = 0$.

The reduction can be constructed in PTIME. We next show that there exists a set $\mathcal{T} \subseteq \mathcal{B}_k$ of influential tuples such that $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}) \geq \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$ iff ϕ is satisfiable.

$\Rightarrow$ First, assume that there exists a set $\mathcal{T}$ of tuples in $\mathcal{B}_k$ such that $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}) \geq \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$. Then the application of $\mathcal{A}_{\mathsf{CR}}$ ensures that for any tuple t in $\mathcal{T}$, if there exists a tuple s in $\mathcal{C}_k$ such that $t[\mathsf{id}] = s[\mathsf{id}]$, then $t[A_1]$ takes the value

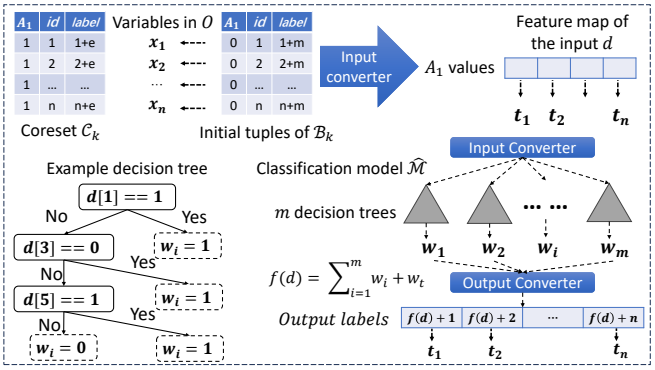


Fig. 7: Problem IT reduction example

of $s[A_1]$. Since $\hat{\mathcal{M}}$ can correctly output all labels in $\mathcal{C}_k$, the parameter w_t must be 0. During training with mean squared error, the optimal w_t equals $\frac{\sum_{\forall t \in \mathcal{B}_k^{\mathcal{T}}} \left(t[\mathsf{label}] - t[\mathsf{id}] - \sum_{i=1}^{m} w_i\right)}{n} = \frac{\sum_{\forall t \in \mathcal{B}_k^{\mathcal{T}}} \left(m - \sum_{i=1}^{m} w_i\right)}{n}$. When $w_t = 0$, $\sum_{i=1}^{m} w_i = m$; this leads to a satisfying assignment for all m clauses.

Specifically, the truth assignment is defined as follows. For each tuple t in $\mathcal{B}_k^{\mathcal{T}}$ with $t[\mathsf{id}]$=i, if $t[A_1]$=1, then x_i is assigned true. Conversely, if $t[A_1]$ remains 0, then x_i is set to false.

One can verify that this truth assignment satisfies ϕ. Indeed, observe that the truth assignment is determined by $\hat{\mathcal{M}}$'s decision trees, each corresponding to a clause C_i from ϕ. A tree yields $w_i = 1$ iff at least one A_1 value in the input d meets the decision criterion; in other words, it satisfies clause C_i. Since $\sum_{i=1}^{m} w_i = m$, all the clauses of ϕ are satisfied by the truth assignment.

$\Leftarrow$ Conversely, assume that ϕ is satisfied by a certain truth assignment for $x_1, \ldots, x_n$. Then we identify a set $\mathcal{T}$ of influential tuples in $\mathcal{B}_k$ as follows. If a variable x_i is true, we add $t \in \mathcal{B}_k$ to $\mathcal{T}$ if $t[\mathsf{id}] = i$. Then we apply $\mathcal{A}_{\mathsf{CR}}$ on $\mathcal{T}$ to obtain $\mathcal{D}_k^{\mathcal{T}}$, such that $t[A_1] = 1$ for all tuples t in $\mathcal{T}$. We next show that $\mathcal{T}$ is indeed the set of influential tuples desired. Suppose by contradiction that $\mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k^{\mathcal{T}}) < \mathsf{acc}(\hat{\mathcal{M}}, \mathcal{D}_k) + 1$. Then $\hat{\mathcal{M}}$ fails to predict all $\mathcal{C}_k$ labels correctly. Indeed, when predicting labels of a batch d' of n tuples in $\mathcal{C}_k$, since $f(d') = e + w_t$, the predicated label of each tuple $s \in \mathcal{C}_k$ is $f(d') + s[\mathsf{id}] = e + w_t + s[\mathsf{id}]$. Meanwhile, the truth label of tuple s is $s[\mathsf{id}] + e$, indicating $w_t \neq 0$. However, if ϕ is satisfied by a truth assignment, consider the batch d that consists of all tuples $t \in \mathcal{B}_k^{\mathcal{T}}$; then $\hat{\mathcal{M}}$ predicts $t[\mathsf{label}]$ as $f(d) + t[\mathsf{id}] = m + w_t + t[\mathsf{id}]$, and $t[\mathsf{label}]$ is $t[\mathsf{id}] + m$; as a result, during the training phase, the optimal w_t must be adjusted to 0; this contradicts that $w_t \neq 0$. $\square$