ReClean: Reinforcement Learning for Automated Data Cleaning in ML Pipelines*

Mohamed Abdelaal*, Anil Bora Yayak†, Kai Klede†, Harald Schöning*
* *Software AG, Darmstadt, Germany*
† *University of Erlangen-Nuremberg, Erlangen, Germany*
* First.Last@softwareag.com, Abora.Yayak@gmail.com, kai.klede@fau.de

***Abstract*—Addressing data quality issues is a challenging task due to the labor-intensive nature of manual data cleaning processes and the inadequacy of automated tools that lack effective repair strategies. In this paper, we introduce ReClean, a novel automated data-cleaning method, dedicated to ML pipelines, that employs reinforcement learning (RL) to optimize data-cleaning tasks. ReClean treats data cleaning as a sequential decision process, where RL agents learn to choose optimal data repair operations that improve ML model convergence and predictive performance. Our extensive experimental evaluation shows that ReClean surpasses existing baseline methods, successfully determining and applying data repair tools to enhance downstream predictive tasks automatically and without supervision.**

I. Introduction

Data Quality Problems. Recently, Machine learning (ML) algorithms have enabled diverse applications across multiple domains, e.g., autonomous driving, healthcare, finance, and robotics. Data quality plays a crucial role in such ML applications, as the performance of ML models is directly dependent on the data used to train those models. Poor or low-quality data with issues such as missing values, duplicates, inconsistencies, or inaccuracies can negatively impact the ability of ML algorithms to learn meaningful patterns from the training data [1]. This data discrepancy, in turn, affects the generalization capabilities of the resulting ML models on unseen data.

Challenges. Data wrangling, including data cleaning, transformation, and enrichment, is the most time-consuming phase of data science projects. Notably, the manual undertaking of data cleaning can be fraught with challenges, consuming significant time and being susceptible to errors. Particularly, the identification and rectification of data quality issues demand extensive human intervention, rendering it a labor-intensive strategy, ill-suited for managing the scale, diversity, and pace of data ubiquitously encountered. Therefore, multiple academic and commercial (semi-) automated tools for detecting and repairing data errors have been developed, e.g., RAHA [13], SAGED [3], OpenRefine and Trifacta [17], and AutoCure [2]. While these tools demonstrate proficient performance in identifying instances of erroneous data, their deficiency lies in the absence of repair strategies for such errors.

The indiscriminate removal of entire tuples as a remedy may prove unsatisfactory, as these tuples might encapsulate pivotal information. An alternative approach involves the generation of appropriate repair values to replace the erroneous instances. However, existing repair tools, e.g., BARAN [13] and HoloClean [19], typically operate independently of the downstream ML tasks, leading to negative impacts on the predictive performance. Moreover, the identification of well-suited repair values is contingent upon multiple factors, encompassing dataset integrity and interdependencies among tuples and columns. Conceiving the pursuit of generating well-suited repair values as a supervised problem introduces challenges in the labeling process, marked by its protracted temporal demands or even infeasibility.

Recently, several ML-oriented tools have emerged, focusing on data quality improvement for ML models, e.g., DiffML [8], DiffPrep [11], Learn2Clean [4], and CPClean [10]. For instance, DiffML formulates data engineering steps in a differentiable way such that the entire pipeline can be trained using backpropagation. However, these tools usually have limitations. They can be computationally complex, may only work with specific ML models, or lack a mechanism for selecting the best data-cleaning tools for deployment in production environments. Addressing these issues is crucial for the practical application of ML, as it would enable more automated and effective data quality management in diverse settings.

Proposed Solution. To address these challenges, we present, in this paper, a novel automated data cleaning method for tabular data based on reinforcement learning (RL), denoted as ReClean. Specifically, ReClean formulates data cleaning as a sequential decision process whereby RL agents learn to select optimal repair operations based on their effects on ML model convergence. Specifically, RL agents choose among available data repair tools or tool combinations that maximize predictive utility for the target application, as measured by evaluation metrics such as AUC or R^2. An RL agent is trained to sequentially select actions that maximize cumulative reward over an episode of cleaning/prediction steps. A variety of common data repair techniques are implemented as the action space, like imputation of missing values, smoothing of outliers, encoding of textual fields, etc. The agent combines experience replay with the Reinforce algorithm [24] to learn stochastic repair policies in an off-policy manner without exact environment definitions. Through iterative interactions with cleaned data, RL implicitly discovers repair strategies tailored to specific error patterns without direct supervision. This joint optimization of data cleaning and ML prediction distinguishes

This work was supported (in part) by the Federal Ministry of Education and Research through grants 02L19C155, 01IS21021A (ITEA project number 20219).

ReClean from prior unidirectional methods.

To sum up, the paper provides the following contributions: (1) We introduce a novel data-cleaning method for ML pipelines that automatically determines the well-suited repair tools without human intervention. (2) We formulate the problem of selecting the most suited data cleaning tools as a sequential decision process which can be solved by the Reinforce algorithm. (3) We conduct extensive experimental evaluation, comparing the performance of ReClean against a broad range of baseline methods on standardized benchmarks. The evaluation demonstrates ReClean can appropriately determine suited data repair techniques to improve downstream predictive task outcomes in an automatic, unsupervised manner. To the best of our knowledge, ReClean is the first method that effectively leverages RL agents to select and combine data repair tools, to enhance downstream ML task performance.

II. Overview

In this section, we introduce the architecture of ReClean, depicted in Figure 1. Let's define the dirty dataset as $\mathcal{D}^d = (x_i, y_i) \mid i \in \{1, 2, ..., N\} \sim \mathcal{P}$, where x_i symbolizes a feature vector residing in the feature space, denoted as $\mathcal{X}$, while y_i signifies the corresponding label to x_i within the label space, represented as $\mathcal{Y}$. When an ML-based error detection method (e.g., [16] and [7]), denoted as $\mathcal{E}$, is applied to the dirty dataset $\mathcal{D}^d$, it generates a detection dictionary, dt. This dictionary dt includes tuples of row and column indices s, t that pinpoint the locations of data errors within the dataset. Subsequently, ReClean generates a set of feature vectors, $\mathcal{V}_i^d \mid i \in \mathbb{R}^N$, each of which corresponds to an individual sample in $\mathcal{D}^d$. To this end, ReClean leverages a TF-IDF (Term Frequency-Inverse Document Frequency) weighting method [21] at the character level. This character-level analysis broadly captures the relative importance of characters within the context of the dataset, thereby enhancing the repair tool selection process.

The repair process begins with randomly sampling a data batch $\mathcal{D}^b = \left(x_i^b, y_i^b\right) \mid i \in \{1, 2, ..., M\}$ from the dirty dataset $\mathcal{D}^d$, accompanied by its associated feature vectors $\mathcal{V}_i^b \mid i \in \mathbb{R}^M$. Utilizing the indices in the detection dictionary dt, the data errors are assigned to null values. ReClean embodies a meta-learning framework that enables the concurrent training of two distinct, yet learnable functions: (1) the target task predictor model, denoted as f_θ, and (2) the cleaner selector model, represented as h_ϕ. The predictor model, formulated as $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$, is designed to minimize a specific loss function, $\mathcal{L}_f$. This could assume the form of cross-entropy for classification tasks, or Mean Squared Error (MSE) for regression tasks, operating on the cleaned batch $\mathcal{D}^c$. As depicted in Equation 1, the function f_θ constitutes any trainable function with parameters θ, e.g., a neural network.

$$f_\theta = \arg\min_{\hat{f} \in \mathcal{F}} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_f \left(\hat{f}(\mathbf{x}_{h_\phi(\mathcal{V}_i)}), y_{h_\phi(\mathcal{V}_i)} \right) \tag{1}$$

The cleaner selector function h_ϕ, modeled as a deep neural network, is optimized to output weights that can be leveraged to automatically select a cleaner for the dirty samples within $\mathcal{D}_i^b$. Accordingly, the cleaner selection module can choose one or multiple cleaners, from the cleaner inventory, denoted as $\mathcal{C}$, to repair the data batch $\mathcal{D}^b$. In this case, the notation $(\mathbf{x}_{h_\phi(\mathcal{V}_i)}, y_{h_\phi(\mathcal{V}_i)})$ symbolizes the cleaner selected for sample i, pertinent to the feature vector $\mathcal{V}_i$. This cleaner is then applied to the training sample $((\mathbf{x_i}, y_i))$, resulting in a cleaned version of the sample. After the cleaned batch $\mathcal{D}^c$ is obtained by applying the selected cleaners on erroneous samples, it is used to train the predictor model f_θ. The corresponding optimization problem can be formulated as:

$$\begin{aligned} &\min_{h_\phi} \mathbb{E}_{(\mathbf{x}^v, y^v) \sim P^t} [\mathcal{L}_h(f_\theta(\mathbf{x}^v), y^v)] \\ &\text{s.t. } f_\theta = \arg\min_{\hat{f} \in \mathcal{F}} \mathbb{E}_{(\mathbf{x}, y) \sim P} \left[\mathcal{L}_f \left(\hat{f}(\mathbf{x}_{h_\phi(\mathcal{V})}), y_{h_\phi(\mathcal{V})} \right) \right]. \end{aligned} \tag{2}$$

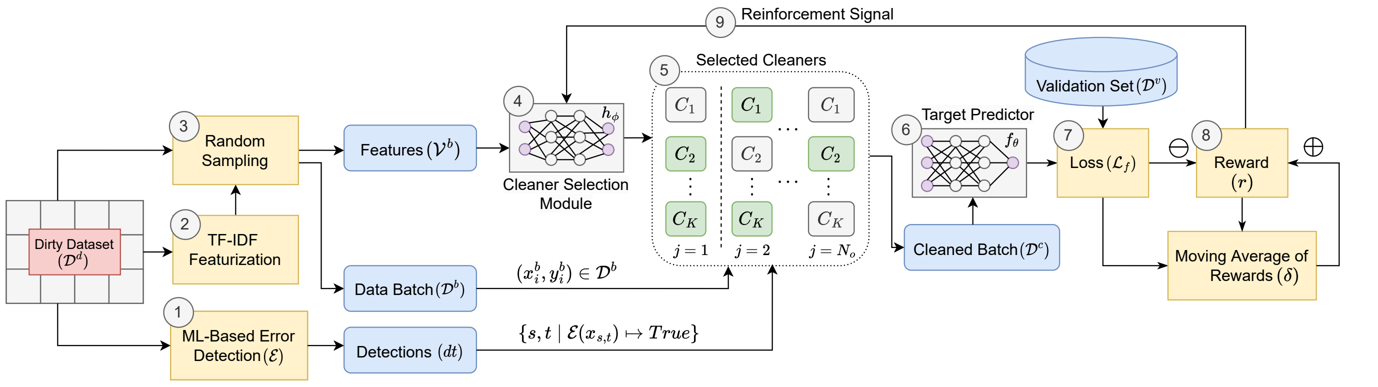
The cleaner selector model is also a trainable function, e.g. neural network. $\mathcal{L}_h$ can also be a loss function such as MSE or cross-entropy depending on the target task. Lastly, it is assumed that a clean validation dataset, designated as $\mathcal{D}^v = (\mathbf{x}_i^v, y_i^v) \mid i \in \{1, 2, ..., K\} \sim \mathcal{P}^t$, is available and is derived from a target distribution $\mathcal{P}^t$ (cf. Section III-C). In the next section, we elaborate on the process of selecting the best repair tools, which enhance the performance of the target models.

III. Automated Repair Tool Selection

While the process of selecting sample-wise cleaners can be theoretically framed as a supervised classification task, the practical execution of this task is fraught with complications. Specifically, it proves hardly possible to ascertain the definitive ground truth labels (i.e., cleaners) for the erroneous samples. The lack of such labels results in a non-differentiable loss function, rendering gradient descent-based optimization methods unsuitable for these challenges. Despite the existence of several strategies designed to circumvent the obstacle of non-differentiable optimization [9], [20], the scope of ReClean is focused on the application of RL for the selection of well-suited cleaners in the presence of dirty samples within datasets. The utilization of policy gradients allows the execution of gradient-based optimization on non-differentiable losses. As a result, policy gradients can be effectively applied to supervised learning tasks, provided they are appropriately framed as RL problems. To optimize these policy gradients, the Reinforce algorithm [25] is employed in conjunction with the rewards garnered from a validation dataset indicative of performance on the target task. Therefore, the policy gradients corresponding to the selected actions are weighted by the rewards that these actions generate.

A. MDP Formulation

In ReClean, we define a "state" S_i as a batch of feature vectors $(\mathcal{V}_i^d) \mid i \in \mathbb{R}^M$. These vectors correspond to the instances in a dirty batch $\mathcal{D}^b$, which has been identified by an error detection algorithm $\mathcal{E}$. Consequently, a state encompasses multiple dirty samples that need to be cleaned simultaneously. The policy, denoted by π and represented by an agent, is responsible for selecting an action (i.e., a cleaner) A_i for a given feature vector. The reward R_i is derived from the predictor model's performance on a clean validation dataset,



Reinforcement Signal
Selected Cleaners
Validation Set ($\mathcal{D}^v$)
Target Predictor
f_θ
h_ϕ
Random Sampling
Features ($\mathcal{V}^b$)
Cleaner Selection Module
C_1
C_2
C_K
$j=1$
$j=2$
$j=N_o$
Loss ($\mathcal{L}_f$)
Reward (r)
Dirty Dataset ($\mathcal{D}^d$)
TF-IDF Featurization
Cleaned Batch ($\mathcal{D}^c$)
Moving Average of Rewards (δ)
Data Batch ($\mathcal{D}^b$)
$(x_i^b, y_i^b) \in \mathcal{D}^b$
ML-Based Error Detection ($\mathcal{E}$)
Detections (dt)
$\{s, t \mid \mathcal{E}(x_{s,t}) \mapsto True\}$

Fig. 1: Architecture of ReClean. In each iteration $j \in \{i, \cdots, N_o\}$, the cleaner selection module chooses a different set of cleaners (shown in green) to increase the reward and optimize the target model. The threshold β_{exp} is defined to achieve a balance between exploration and exploitation.

symbolized by $\mathcal{L}_h$. The reward is equivalent to the validation loss or a loss-related evaluation metric. Our model operates in episodes, with each episode concluding when all samples in a dirty batch have been cleaned by the cleaners selected by the cleaner selector network. In this particular context, each episode lasts merely a one-time step, making the process efficient and expedient. ReClean actively promotes the exploration of the cleaner selector (i.e., the policy) in the direction of the optimal solution for Equation 3. Such an equation defines the loss function $\hat{l}(\phi)$ of the cleaner selector network under the target distribution P^t and the policy $\pi_\phi(\mathcal{V})$:

$$\begin{aligned}\hat{l}(\phi) &= \mathbb{E}_{(\mathbf{x}^v, y^v)\sim P^t}\left[\mathbb{E}_{\pi_\phi(\mathcal{V})}[\mathcal{L}_h(f_\theta(\mathbf{x}^v), y^v)]\right] \\ &= \int P^t(x^v)\left[\sum_{\mathcal{V}} \pi_\phi(\mathcal{V}) \cdot [\mathcal{L}_h(f_\theta(\mathbf{x}^v), y^v)]\right] d\mathbf{x}^v \end{aligned} \tag{3}$$

We can then compute the gradient $\nabla_\phi \hat{l}(\phi)$ as:

$$\begin{aligned}\nabla_\phi \hat{l}(\phi) &= \int P^t(x^v)\left[\sum_{\mathcal{V}} \nabla_\phi \pi_\phi(\mathcal{V}) \cdot [\mathcal{L}_h(f_\theta(\mathbf{x}^v), y^v)]\right] d\mathbf{x}^v \\ &= \int P^t(x^v)\left[\sum_{\mathcal{V}} \nabla_\phi \log(\pi_\phi(\mathcal{V}) \cdot \pi_\phi(\mathcal{V}) \cdot [\mathcal{L}_h(f_\theta(\mathbf{x}^v), y^v)]\right] d\mathbf{x}^v \\ &= \mathbb{E}_{(\mathbf{x}^v, y^v)\sim P^t}\left[\mathbb{E}_{\pi_\phi(\mathcal{V})}[\mathcal{L}_h(f_\theta(\mathbf{x}^v), y^v)] \cdot \nabla_\phi \log(\pi_\phi(\mathcal{V}))\right]\end{aligned} \tag{4}$$

where $\nabla_\phi \log(\pi_\phi(\mathcal{V}))$ is

$$\begin{aligned}\nabla_\phi \log(\pi_\phi(\mathcal{V})) &= -\nabla_\phi \sum_{i=1}^{N} \log\left(h_\phi(\mathcal{V}_i)\right) \\ &= -\sum_{i=1}^{N} \frac{\log\left(h_\phi(\mathcal{V}_i)\right)}{h_\phi(\mathcal{V}_i)}\end{aligned} \tag{5}$$

B. Cleaner Selection Module

The cleaner selection module is implemented as a deep neural network, specifically tailored to yield the index of an appropriate repair tool for each instance of dirty data. This system is constructed as a four-layer feed-forward neural network, incorporating the Rectified Linear Unit (ReLU) activation function for non-linearity. The quantity of hidden units within this network is modulated according to the dimensionality of the feature vectors, thereby maintaining an optimal balance between complexity and performance. The network ingests a collection of feature vectors as input, and generates, as output, the index of a suitable repair tool for each corresponding dirty data sample. For instance, assume a batch of dirty data. The first row of this batch may be repaired utilizing the repair tool designated by the index R1, while the second row's anomalies might be more suitably addressed by the tool indexed as R2.

The next essential step is to present the mechanism by which the reward and loss functions are employed to update the cleaner selection module. In particular, the loss function serves the purpose of modulating the weights within the cleaner selection module. This module's output is represented as $y_{pred} = [p_{i,j}] \mid i \in \mathbb{R}^M, j \in \mathbb{R}^K$, in which M and K indicate the number of data samples per batch and the number of available repair tools, respectively. In this context, the notation p_{ij} denotes the probability that a specific data sample (x_i, y_i) is processed using the repair tool c_j. Considering that the ground truth labels corresponding to the predictions produced by the cleaner selection module are not readily available, ReClean leverages the Reinforce algorithm to facilitate an optimization process for the cleaner selection module.

In detail, the reward signal, which originates from the target predictor, is utilized to compute the loss estimate for the cleaner selection module. Subsequently, this estimated loss and its corresponding gradient are employed to adjust the weights within the cleaner selection module. In our implementation, the reward, denoted as R, is calculated as the difference between the current loss $\mathcal{L}_h$ $(f_\theta(x^v), y^v)$–derived from a validation set–and the moving average of previous losses, designated L_{movAvg}. In this context, x^v and y^v represent the validation set, whereas $f(.)$ embodies the prediction function of the target predictor. During the training process, we use a moving average of previous loss with a window size T as the baseline (δ) to stabilize the reward function. Hence, actions yielding an improvement over the average of the preceding T losses will be more likely selected by the selection module.

Following the reward estimation, the loss for the cleaner selection module is computed. During this estimation process, striking an equilibrium between exploitation and exploration emerges as a critical aspect. In general, exploration encapsulates the strategy of experimenting with novel actions to gain additional information about the environment, whilst exploitation pertains to the approach of implementing actions deemed beneficial based on the agent's existing knowledge. In ReClean, the cleaner selection module can facilitate exploration by selecting random repair tools or experimenting with previously untested tools. In contrast, the module can engage in exploitation by choosing the action, in this case, a repair tool, that is associated with the highest expected reward. To maintain a delicate balance between exploration and exploitation, we introduce an exploration threshold, denoted as β_{exp}. This threshold represents a hyperparameter which determines how much exploration is encouraged.

More specifically, ReClean adds a regularization term to Equation 3 to encourage the model to explore different actions. If the average probability of the selected actions is too close to 0 (i.e., underconfidence) or 1 (i.e., overconfidence), determined by the threshold β_{exp}, the loss of the cleaner selection module is intentionally augmented. This increase in loss inclines the cleaner selection module towards the exploration of new (i.e., unexplored) repair tools. Figure 2 presents the pseudocode encapsulating the core mechanics of ReClean. The inputs to ReClean encompass the batch size of the input data, the cycle count for both the cleaner selection module and the target model, the dirty dataset, the validation dataset, the feature vectors, and the window size for the moving average computation. During each iteration of the cleaner selection module, a batch of dirty data is sampled (cf. line 4), followed by the application of the selected repair tools to clean the sampled data (cf. lines 5 and 6). After the data cleaning process, the weights of the target predictor are updated (cf. lines 7 and 8). The final step in the iteration involves updating the weights of the cleaner selection module (cf. line 10).

C. Validation Set Extraction

ReClean incorporates a pivotal component, namely the validation dataset, which serves a critical role in the estimation of the reward signal. The first step in this process involves the identification and isolation of erroneous data samples. Once these errors have been detected, the input dataset $\mathcal{D}^d$ is partitioned into two distinct subsets. The first of these is referred to as the clean fraction, which is devoid of the detected errors. The second component retains the label of the dirty fraction, containing the previously identified errors. Once the data has been appropriately segregated, ReClean employs a random sampling strategy designed to extract data samples from the clean fraction. These randomly sampled data points collectively constitute the validation dataset.

The utility of this approach lies in its capacity to generate a validation dataset irrespective of the nature or size of the input dataset. Further, it achieves this without necessitating any form of user intervention. This autonomous generation of the validation set adds a layer of efficiency and convenience to the data validation process. In essence, the creation of the validation dataset involves a systematic sequence of steps, starting with error identification in the input data, followed by the segregation of clean and dirty fractions, and finally, the random sampling from the clean fraction. This robust technique ensures the generation of a high-quality validation dataset that can effectively gauge the reward signal.

```
Require: Mini-batch size $B_s$, number of iterations for RL
    agent $N_O$, number of iterations for predictor $N_I$, dirty
    training dataset $\mathcal{D}^d$, validation dataset $\mathcal{D}^v$, feature vectors
    $\mathcal{V}^d$, moving average window $T > 0$
1: Initialize parameters $\phi$, $\theta$, moving average $\delta = 0$
2: for j = 1, ..., $N_O$ do
3:     Sample a mini-batch of samples from the dirty training
       dataset and their corresponding feature vectors: $\mathcal{D}^b = (\mathbf{x}_i, y_i)_{i=1}^{B_s}$ and $V^b = (\mathcal{V}_i)_{i=1}^{B_s}$
4:     Output cleaners $C_i = h_\phi(\mathcal{V})$
5:     Apply cleaners on the samples of $\mathcal{D}^b$: $\mathcal{D}^c = (\tilde{\mathbf{x}}_i, \tilde{y}_i)_{i=1}^{B_s}$
6:     for j = 1, ..., $N_I$ do
7:         Update the parameters of the predictor network
               $$\theta \leftarrow \theta - \alpha \frac{1}{B_s} \sum_{i=1}^{B_s} \nabla_\theta \mathcal{L}_f(f_\theta(\tilde{\mathbf{x}}_i, \tilde{y}_i))$$
8:     Update the parameters of the cleaner selector
           $$\phi \leftarrow \phi - \beta \frac{1}{B_s} \left[ \sum_{i=1}^{B_s} [\mathcal{L}_h(f_\theta(\mathbf{x}_i^v, y_i^v)) - \delta \right] \nabla_\theta \log \pi_\phi(\mathcal{V}^b)$$
9:     Update the moving average baseline: $\delta \leftarrow \frac{T-1}{T}\delta + \frac{1}{LT}\sum_{j=1}^{K}[\mathcal{L}_h(f_\theta(\mathbf{x}_j), y_j)]$
```

Fig. 2: Training algorithm of ReClean

D. Runtime Optimization

In this section, we shed light on an important implementation trick that has been incorporated into ReClean. This trick has a profound impact on computational efficiency, reducing the runtime drastically, from a several-hour operation to a succinct process that can be completed within minutes, contributing to the practicality and scalability of ReClean. In machine learning, the term "epoch" typically refers to a single complete traversal through the entire training dataset. To compute the loss of our target model, it is necessary to train the target predictor using data that has undergone the repair process. This requirement suggests the need to run all repair tools on every batch of data during each epoch. However, the number of epochs can span anywhere from 2000 to 5000, contingent on the characteristics of the dataset in question. Therefore, the execution of all repair tools during each epoch can significantly escalate the runtime of ReClean.

To tackle this challenge, the ReClean implementation transforms the resource-intensive repair operation into a more efficient assignment operation. Before initiating the training phase, each repair tool generates a list of repaired datasets. As the training proceeds, based on the repair tools selected for each data batch, we substitute the dirty data samples with their corresponding repaired versions, sourced from the pre-generated repaired datasets. This strategic maneuver allows us to bypass the need for executing the repair tools on every batch during each epoch, thereby drastically reducing the computational load and the runtime. Moreover, it provides an efficient pathway to handle the repair operation without compromising the accuracy of the cleaning process.

IV. Performance Evaluation

In this section, we present our evaluation of ReClean in comparison to a set of baseline methods. Through a series of experiments, we aim to address the following key questions: (1) What is the number of repair tools employed by ReClean while cleaning a dirty dataset? and (2) what is the accuracy of ReClean compared to the baseline tools? (3) what is the impact of increasing the error rate on ReClean and the compared baselines? By addressing these questions, we shed light on the effectiveness and potential advantages of ReClean over the baseline tools in the context of error repair. We first describe the setup of our evaluations, before discussing the results and the lessons learned throughout this study.

A. Experimental Setup

We conducted multiple experiments employing a diverse set of six real-world datasets that encompassed varying data sizes and exhibited distinct error rates. These datasets are Smart Factory [5], Breast Cancer Wisconsin Diagnostic (WDBC) [23], Airfoil Self-Noise dataset (NASA) [22], Wine Quality dataset [6], Combined Cycle Power Plant dataset (CCPP) [18] and Retail Sales dataset [15]. Table I provides a summary of the key characteristics of each dataset. It is worth noting that such datasets are commonly encountered in the domain of data cleaning and have been extensively used in related literature. In ReClean, we synthesized errors on the datasets with different error rates. To this end, we utilize a Python library called *error-generator*.

TABLE I: Datasets used in the evaluation where the error types are outliers (OT), missing values (MV), Gaussian noise (GN), white noise (WN), and typos (TP); and the ML tasks are binary classification (BC) and regression (R)

Data Set	**Rows**	**Columns**	**Error Types**	**ML Task**
Smart Factory	23645	19	MV, OT, GN	BC
Nasa	1504	6	MV, OT, TP	R
WDBC	460	11	MV, OT, WN	BC
Wine	1300	11	MV, OT, GN	R
CCPP	1600	5	MV, OT, TP	R
Retail	1140	10	MV, OT, TP	R

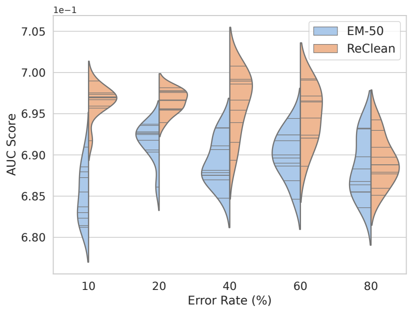
In our search for effective error detection, we evaluated several state-of-the-art tools, including RAHA [13], ED2 [16], Picket [12], and HoloClean [19]. Our analysis revealed that ED2 consistently provided high detection recall and precision—averaging 99%—across various datasets. Consequently, we integrated ED2 into our pipeline for accurate identification of dirty samples. For error correction, we utilized a combination of statistical and ML-based imputation tools. The statistical methods included Mean and Median imputation, which provide simple yet effective measures for handling missing data. In terms of ML-based imputation, we employed Expectation Maximization (EM), k-Nearest Neighbors (KNN), Bayesian Ridge, and MissForest. These robust imputation methods not only served to generate repair candidates in our system but also served as baseline tools for comparison. To further enhance the comparative study, we utilize different versions of the baseline methods, each with distinct configurations of given parameters. For instance, The KNN imputer is implemented in three configurations, differing in the number of neighbors. The EM imputer is utilized in two configurations with varying numbers of iterations signifying 50 and 100 iterations, respectively. The MissForest imputer is implemented in three configurations, differing in the number of trees in the forest parameter, indicating 50, 100, and 200 trees, respectively. The Mean, Median, and Bayesian Ridge imputers do not have any configured parameters.

In our experimental setup, we configured the outer iteration count (N_O) to be 2000, and the moving average window (T) was set to 10. The mini-batch size was chosen to be 80% of the total dataset size, a common practice in RL problems to enhance the stability of model training, as highlighted in [14]. The remaining 20% of the dataset was allocated for validation purposes. For the evaluation of our target model's performance, we utilize two distinct metrics: the Area Under the Curve (AUC) and the Root Mean Square Error (RMSE). All experiments have been repeated ten times, where the means of the ten runs are reported. We run all the experiments on an Ubuntu 20.04 LTS machine with 16 2.60 GHz cores and 64 GB memory.

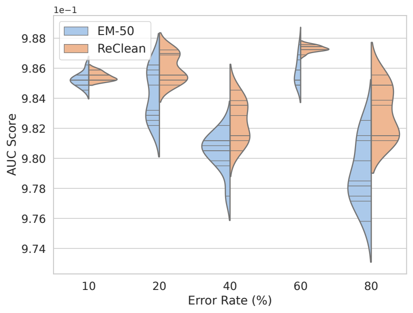
B. Results

Figure 3 presents a comparison between the performance of ReClean and the top-performing baseline tools across various datasets. To ensure clarity, we have chosen to feature only the leading baseline tool for each dataset within the figure[1]. A closer examination of Figures 3a-3f reveals that the superior baseline tool varies depending on the dataset in question. This variability underscores the rationale behind ReClean, which automatically selects and combines the most effective repair tools tailored to the dataset at hand. Figure 3a shows the distribution of the AUC scores of the target predictor, detailing its performance at varying error rates within the Smart Factory dataset. The figure shows that ReClean consistently outperforms the leading repair tool, i.e., EM-50, in terms of AUC scores derived from the validation set (on average by circa 1%), signaling superior performance. This pattern suggests that the repair tools, either selected or integrated by ReClean, yield higher-quality data. Consequently, models trained on the refined data demonstrate improved predictive capabilities when compared to those trained on datasets processed by the best single repair tool available.

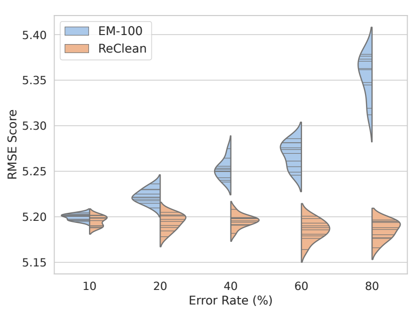
[1]All results have been made publicly available in the project's repository.



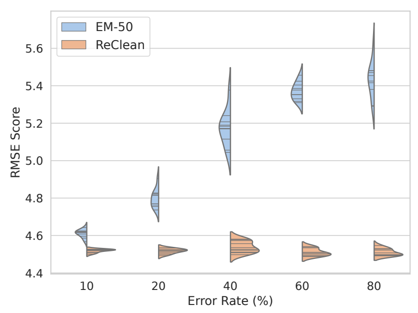
(a) Smart Factory



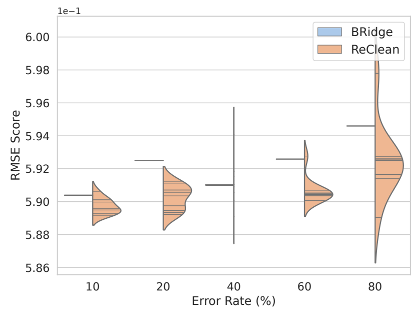
(b) WDBC



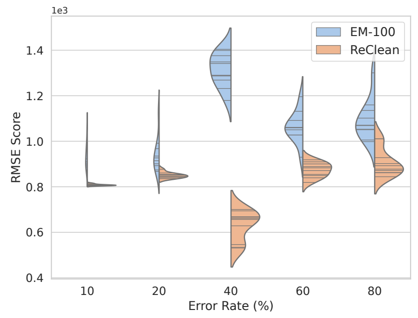
(c) Nasa



(d) CCPP



(e) Wine



(f) Retail

Fig. 3: Performance of ReClean and the best baseline tools

Figure 3b compares the performance of the target predictor trained on the WDBC dataset after repairs have been conducted using both ReClean and the EM-50 repair tool, which is identified as the best tool for this particular dataset. The results demonstrate that ReClean's repaired dataset enables the predictor to achieve performance that is either comparable with or slightly superior to that of the baseline, with an average improvement of 0.16%. In Figures 3c and 3d, the performance of ReClean on the NASA and CCPP datasets is quantified by the reduction in Root Mean Square Error (RMSE) values. On average, ReClean decreases the RMSE by 0.3% for the NASA dataset and by 0.5% for the CCPP dataset. This improvement is particularly notable as the error rate increases. Finally, for the Wine and Retail datasets, Figures 3e and 3f demonstrate the impact of ReClean on the accuracy of the predictor when applied to the Wine and Retail datasets. The results show an improvement in accuracy by 0.36% for the Wine dataset and a substantial increase of 25.6% for the Retail dataset. These figures underscore the efficacy of ReClean in refining the prediction accuracy across different types of data.

To understand the reasons behind the superior performance of ReClean compared to the best standalone repair tools, Table II provides a summary of the number of repair tools utilized by ReClean for various datasets at different error rates. The table highlights that, depending on the error rate, ReClean typically selects multiple tools, with the mean ranging from 1.9 to 4.4 across different datasets and error rates, suggesting a tailored approach to error correction for each scenario. For instance, in the Smart Factory dataset with a 10% error rate, ReClean uses an average (mean) of 3.2 repair tools with a standard deviation of 1.53. The tendency to employ a variety of tools correlates with the enhanced performance metrics observed in Figure 3. The bold numbers indicate the highest average number of tools used for each dataset, which tend to occur at the lowest error rate (10% for Smart Factory and NASA, 20% for WDBC and Retail), suggesting a possible trend where more tools may be applied at lower error rates for certain datasets.

TABLE II: Number of repair tools employed by ReClean, where "M" and "Std" denote the mean and standard deviation of ten experiments and γ represents the error rate.

	Smart Factory		WDBC		Nasa		Wine		CCPP		Retail	
γ(%)	M	Std	M	Std	M	Std	M	Std	M	Std	M	Std
10	**3.2**	1.53	3.1	1.51	**2.9**	1.49	**4**	1.94	**3.2**	1.3	4.1	1.3
20	2.6	1.01	**3.5**	1.20	2.7	1.26	2.8	0.6	2.7	1.00	**4.4**	1.11
40	2.5	0.92	2.7	1.00	2.7	0.9	2.4	0.91	3	1.18	3.6	1.35
60	2.3	0.78	2.3	0.9	2.6	1.04	1.9	0.83	3.1	1.60	4.3	0.91
80	2.7	0.9	2.1	0.53	2.6	0.74	1.9	0.94	3	1.18	3.5	0.80

V. Conclusion & Future Work

In this paper, we present ReClean, an innovative automated data cleaning method for tabular datasets, eliminating the need for manual configuration. ReClean leverages model-free reinforcement learning to automatically select and integrate appropriate repair tools for any given dataset. By formulating data cleaning as a label-free, sequential decision-making task, ReClean showcases superior performance, consistently surpassing traditional baseline methods in diverse scenarios. Future directions involve the inclusion of other data engineering steps, such as feature selection and data augmentation.

References

[1] Mohamed Abdelaal, Christian Hammacher, and Harald Schoening. REIN: A Comprehensive Benchmark Framework for Data Cleaning Methods in ML Pipelines. In *26th International Conference on Extending Database Technology (EDBT)*, March 2023.

[2] Mohamed Abdelaal, Rashmi Koparde, and Harald Schoening. Autocure: Automated tabular data curation technique for ml pipelines. In *Proceedings of the Sixth International Workshop on Exploiting Artificial Intelligence Techniques for Data Management in conjunction with SIGMOD 2023*, pages 1–11, 2023.

[3] Mohamed Abdelaal, Tim Ktitarev, Daniel Städtler, and Harald Schöning. SAGED: Few-shot Meta Learning for Tabular Data Error Detection. In *27th International Conference on Extending Database Technology (EDBT)*, March 2024.

[4] Laure Berti-Equille. Learn2clean: Optimizing the sequence of tasks for web data preparation. In *The world wide web conference*, pages 2580–2586, 2019.

[5] Oliver Birgelen, Alexander; Niggemann. Smart factory: High storage system data for energy optimization, 2018. accessed on January 2022.

[6] Dheeru "Dua and Casey Graff. UCI machine learning repository, 2017.

[7] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data*, pages 829–846, 2019.

[8] Benjamin Hilprecht, Christian Hammacher, Eduardo S Reis, Mohamed Abdelaal, and Carsten Binnig. Diffml: End-to-end differentiable ml pipelines. In *Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning*, DEEM '23, New York, NY, USA, 2023. Association for Computing Machinery.

[9] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2016.

[10] Bojan Karlaš, Peng Li, Renzhi Wu, Nezihe Merve Gürel, Xu Chu, and Ce Zhang. Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *arXiv preprint arXiv:2005.05117*, 2020.

[11] Peng Li, Zhiyi Chen, Xu Chu, and Kexin Rong. Diffprep: Differentiable data preprocessing pipeline search for learning over tabular data. *Proc. ACM Manag. Data*, 1(2), jun 2023.

[12] Zifan Liu, Zhechun Zhou, and Theodoros Rekatsinas. Picket: Guarding against corrupted data in tabular data during learning and inference. *arXiv preprint arXiv:2006.04730*, 2020.

[13] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data*, pages 865–882, 2019.

[14] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *CoRR*, abs/1812.06162, 2018.

[15] Tyler Morse. Online business sales 2017-2019, 2019.

[16] Felix Neutatz, Mohammad Mahdavi, and Ziawasch Abedjan. Ed2: A case for active learning in error detection. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 2249–2252, 2019.

[17] Dessislava Petrova-Antonova and Rumyana Tancheva. Data cleaning: A case study with openrefine and trifacta wrangler. In *Quality of Information and Communications Technology: 13th International Conference, QUATIC 2020, Faro, Portugal, September 9–11, 2020, Proceedings 13*, pages 32–40. Springer, 2020.

[18] Heysem Kaya Pınar Tüfekci. Wine quality data set, 2014.

[19] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820*, 2017.

[20] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models, 2014.

[21] Thomas Roelleke and Jun Wang. TF-IDF Uncovered: A Study of Theories and Probabilities. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 435–442, 2008.

[22] D. Stuart Pope Thomas F. Brooks and Michael A. Marcolini. Nasa airfoil self-noise dataset, 2014. accessed on January 2022.

[23] Olvi L. Mangasarian William H. Wolberg, W. Nick Street. Breast cancer wisconsin (diagnostic) data set, 1992.

[24] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[25] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, may 1992.