Self-supervised and Interpretable Data Cleaning with Sequence Generative Adversarial Networks

Jinfeng Peng, Northeastern Univ., China, jinfengpeng0@gmail.com

Derong Shen*, Northeastern Univ., China, shenderong@cse.neu.edu.cn

Nan Tang, QCRI, HKBU, Qatar, ntang@hbku.edu.qa

Tieying Liu, Northeastern Univ., China, liutieying0@gmail.com

Yue Kou, Northeastern Univ., China, kouyue@cse.neu.edu.cn

Tiezheng Nie, Northeastern Univ., China, nietiezheng@cse.neu.edu.cn

Hang Cui, UIUC, USA, hangcui2@illinois.edu

Ge Yu, Northeastern Univ., China, yuge@cse.neu.edu.cn

ABSTRACT

We study the problem of self-supervised and interpretable data cleaning, which automatically extracts interpretable data repair rules from dirty data. In this paper, we propose a novel framework, namely Garf, based on sequence generative adversarial networks (SeqGAN). One key information Garf tries to capture is data repair rules (for example, if the city is "Dothan", then the county should be "Houston"). Garf employs a SeqGAN consisting of a generator G and a discriminator D that trains G to learn the dependency relationships (*e.g.,* given a city value "Dothan" as input, the county can be determined as "Houston"). After training, the generator G can be used to generate data repair rules, but may contain both trusted and untrusted rules, especially when learning from dirty data. To mitigate this problem, Garf further updates the learned relationships with another discriminator D' to iteratively improve the quality of both rules and data. Garf takes advantages of both logical and learning-based methods, which allow cleaning dirty data with high interpretability and have no requirements for prior knowledge and training data. Extensive experiments on real-world and synthetic datasets demonstrate the effectiveness of Garf. Garf achieves new state-of-the-art data cleaning result with high accuracy, through learning from dirty datasets without human supervision.

PVLDB Reference Format:
Jinfeng Peng, Derong Shen, Nan Tang, Tieying Liu, Yue Kou, Tiezheng Nie, Hang Cui, and Ge Yu. Self-supervised and Interpretable Data Cleaning with Sequence Generative Adversarial Networks. PVLDB, 16(3): XXX-XXX, 2022.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/PJinfeng/Garf-master.

1 INTRODUCTION

Data is a critical asset for decision-making across organizations, but data is useful only if it is of high-quality. Unfortunately, real-life data is often dirty [1, 23] and data cleaning is undeniably a laborious and time-consuming task [14, 15]. Broadly speaking, there are two types of data cleaning tasks: qualitative data cleaning and quantitative (or statistical) data cleaning [35]. Qualitative cleaning mainly focuses on repairing categorical values (*e.g.,* gender, city) that violate data dependency, while quantitative data cleaning relies on statistical methods to identify and repair erroneous data that often come in the form of numeric values. Both types are common in practice. Our focus in this work is on **qualitative data cleaning**.

Example 1. [*Qualitative data errors*] *Consider the Hospital dataset in Table 1 including seven tuples: t_1–t_7, with the schema (Tuple ID, Provider ID, City, State, Zip, County, Condition, Measure ID). Data errors are red-colored, and their corresponding ground truth values are green-colored.*

Logical data cleaning. One line of work on qualitative data cleaning uses *integrity constraints*, such as functional dependencies (FDs) [3], conditional functional dependencies (CFDs) [4], and pattern functional dependencies [36]. Another line of works employs *data repair rules*, such as editing rules [22], fixing rules [43], Sherlock rules [31], Detective rules [28], and so on. Integrity constraints or data repair rules methods are considered as *logical data cleaning* methods, which repair data by logical inference.

Example 2. [*Integrity constraints and data repair rules.*] *Consider the following FDs (f_1, f_2) and data repair rules (r_1, r_2, r_3):*

- *FD f_1: City $\rightarrow$ State*
- *FD f_2: State, Zip $\rightarrow$ County*
- *Rule r_1: [City = "Monticello"] $\rightarrow$ [State = "GA"]*
- *Rule r_2: [City = "Dothan"] $\rightarrow$ [State = "AL"]*
- *Rule r_3: [State = "AL", Zip=36301] $\rightarrow$ [County = "Houston"]*

where f_1 is an FD which means that a city uniquely determines a state, similarly to f_2; and r_1 is a data repair rule that if the value of the City in a tuple is "Monticello", then its State should be "GA", similarly to rules r_2 and r_3.

Integrity constraints are typically defined over more than one tuple (*e.g.,* FDs are defined over two tuples) and they only specify which attribute values putting together violate a given FD (*a.k.a.* static semantics [22]). For example, the four values (t_1[City], t_1[State], t_2[City], t_2[State]) violate f_1 in Example 2 but f_1 does not specify which value is wrong. In contrast, data repair rules are often defined on one tuple and clearly specify which attribute value should be updated (*a.k.a.* dynamic semantics [22, 24]). For example, rule r_2 says that t_2[State] should be repaired from "GAA" to "GA".

*Derong Shen is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 16, No. 3 ISSN 2150-8097.
doi:XX.XX/XXX.XX

Table 1: Sample tuples of a Hospital table.

TupleID	Provider ID	City	State	Zip	County	Condition	Measure ID
t_1	111303	Monticello	GA	31064	Jasper	Emergency Department	ED_1b
t_2	111303	Monticello	GAA (GA)	31064	Jasper	Emergency Department	ED_2b
t_3	40051	Monticello	AR	71655	Drew	Emergency Department	EDV
t_4	10001	Monticello (Dothan)	AL	36301	Houston	Preventive Care	IMM_2
t_5	10001	Dothan	AL	36301	Houston	Colonoscopy care	OP_29
t_6	10001	Dothan	AR (AL)	NULL (36301)	Houston	Colonoscopy care	OP_30
t_7	10001	Dothan	AL	36301	Houst (Houston)	Cancer care	OP_33

Manually specifying integrity constraints and data repair rules requires domain knowledge, which is time consuming and error-prone. Discovering them directly from dirty data is known to produce many false positives [10].

Learning-based data cleaning. Machine learning (ML) based methods have also been studied, such as Guided data repair [48] and SCAREd [47], which need a lot of training data. Unfortunately, the clean training data and the training data labels are usually unavailable or do not have enough capacity in real world scenarios.

Recently, the use of deep learning (DL), especially Transformer-based language models, for tabular data learning that can be used for data cleaning has also been investigated, such as RPT [41] and TURL [16]. RPT employs an encoder-decoder architecture and is pre-trained in a tuple-to-tuple fashion by corrupting the input tuple and then learning to reconstruct the original tuple. TURL employs an encoder-only architecture for learned representations, instead of generating an output, *e.g.,* a missing cell value. In fact, TURL has to link the learned presentation with a knowledge base (in the TURL paper, they used a collection of web tables, in total 4.6GB) in order to extract values for imputing the missing value. RPT is a positioning paper that does not provide a quantitatively evaluation for dare repair problems. TURL shows some preliminary result, but with low precision, *e.g.,* the reported precision of TURL on the data repair task (or "cell filling task" used by TURL) is only 54.8% using the top-1 result (see Table 13 in [16], column "P@1").

Limitations of existing solutions. *Logical data cleaning* methods that use constraints and rules are highly interpretable, but their main limitation is that obtaining high-quality data repair rules is very difficult in practice, regardless of whether they are provided by humans or are automatically discovered [33]. Moreover, these data repair rules are often used together with high-fidelity master data [22] or knowledge bases [12], which are often unavailable. *Learning-based data cleaning* methods detect and repair dirty data by learning the real data distribution, with two main limitations: (1) ML-based methods usually require clean and well-annotated training data, because learning on dirty datasets cannot guarantee correctness and may lead to new errors; (2) DL-based methods are typically uninterpretable, such as RPT [42] and TURL [16], which hinders their wide adoption in practice.

To tackle the above limitations that are often met for data cleaning, we seek to answer the following question: *whether we can use deep learning models to automatically (*e.g., *with self-supervision) generate interpretable data cleaning rules that can be used to clean data, by combining logical data cleaning and learning-based data cleaning methods in one framework?*

Our proposal: self-supervised and interpretable cleaning. We propose a generative adversarial repairing framework, namely GARF, a self-supervised framework that generates data repair rules and then repairs both inaccurate rules and dirty data based on generative adversarial training. GARF includes two parts: (1) a rule generation model and (2) a co-cleaning model.

The **rule generation model** introduces a deep learning model SeqGAN to learn the dependency relationships among attributes and values in datasets. Following the learned relationships, we generate fine-grained data repair rules to detect and repair dirty data. However, the automatically generated rules may not be fully trusted, especially when learning directly from dirty datasets.

EXAMPLE 3. *Consider Table 1 and the rules of Example 2. Tuples t_2, t_3, and t_4 violate r_1. However, it is hard to distinguish the false one and repair it in a trusted manner. Based on this rule, we may modify all the violated values in t_2, t_3 and t_4. Based on the data, we can change the value in the rule, such as* $[City = \text{"}Monticello\text{"}] \rightarrow [State = \text{"}AR\text{"}]$*. In addition, we can also add additional conditions to make the rule fit better to the data, such as* $[City = \text{"}Monticello\text{"}, Zip = 31064] \rightarrow [State = \text{"}GA\text{"}]$*. Although there are many optional operations, some are not trusted.*

The **co-cleaning model** simultaneously repairs dirty data and inaccurate rules, based on the relative confidence between data and rules. Intuitively, when the rules are relatively more trusted, we repair the wrong values using the rules. On the other hand, if data is relatively more trusted, we will update the learned relationships and repair the rules based on the data.

Contributions. Our contributions are summarized as follows:

(1) A novel framework. We introduce GARF, a self-supervised data repair framework, to automatically generate data repair rules and clean dirty data in a reliable and interpretable manner. This model takes advantages of both logical data cleaning and learning-based data cleaning methods. (Section 2)

(2) Rule generation with SeqGAN. To automatically generate data repair rules, we propose a novel rule generation model. We utilize a SeqGAN model to learn the relationships among data. Instead of updating values using SeqGAN directly, we convert the learned relationships into data repair rules, which are interpretable. The process is completely data-driven with no requirements for domain knowledge or training data. (Section 3)

(3) Repairing dirty data and inaccurate rules. Considering the practical case that both rules and data maybe inaccurate, we propose a co-cleaning model to repair both rules and data. We sign dynamic confidence measures to evaluate rules and data. Based on the confidence, we modify the inaccurate rules to fit the data and repair the dirty data using rules iteratively. (Section 4)

(4) Experiment. We conduct extensive experiments to evaluate the performance of our proposed framework, which verifies its effectiveness on both real-world and synthetic datasets. As our experiments show, GARF outperforms a variety of state-of-the-art data cleaning methods with greater accuracy, robustness, and interpretability. (Section 5)

2 PROBLEM AND SOLUTION OVERVIEW

2.1 Data Repair Rules

Consider a table $\mathbf{D}$ defined over the relational schema $R = (a_1, \ldots, a_n)$. The domain of an attribute $a_i \in R$ is denoted by $dom(a_i)$.

Data repair rules. We formalize a data repair rule r defined on R as the following syntax: $[A_L, v(A_L)] \rightarrow [A_R, v(A_R)]$, where A_L is a set of attributes, $v(A_L)$ is a set of attribute values relative to A_L, A_R is a single attribute, and $v(A_R)$ is a single attribute value relative to A_R. Also, A_L and A_R are often referred to as left-hand side attributes (LHS) and right-hand side attributes (RHS), respectively.

Intuitively, the LHS attributes/values can uniquely determine the RHS attribute/value, such as the rule r_3 [State = "AL", Zip = 36301] → [County = "Houston"] in Example 2. In other words, given any tuple, if its State value is "AL" and Zip value is 36301, then its County should be "Houston".

We also write $AV_L = [A_1, v(A_1), \ldots, A_m, v(A_m)]$ ($m = |AV_L|$ is the number of attributes in AV_L and A_i is the i-th attribute in AV_L), which denotes the set of (attribute name, attribute value) pairs for all attributes in A_L, and $AV_R = [A_R, v(A_R)]$, which denotes one (attribute name, attribute value) pair. Consider again rule r_3 in Example 2, we have AV_L = [State = "AL", Zip = 36301] and AV_R = [County = "Houston"].

Next, we describe the definitions of applying data repair rules.

Rule matching. Given a data repair rule $r : [A_L, v(A_L)] \rightarrow [A_R, v(A_R)]$ and a tuple t:

- tuple t **matches** rule r, denoted by $t \models r$, iff $t(A_L) = v(A_L)$ and $t(A_R) = v(A_R)$;
- tuple t **violates** rule r, denoted by $t \not\models r$, iff $t(A_L) = v(A_L)$ but $t(A_R) \neq v(A_R)$.

In addition, we say that a rule r is **applicable** to a tuple t if $t(A_L) = v(A_L)$, denoted by $t \vdash r$, *i.e.,* when either t matches r ($t \models r$) or t violates r ($t \not\models r$).

EXAMPLE 4. *Consider rule r_1 in Example 2 and Table 1. t_1 matches r_1 (*i.e., *$t_1 \models r_1$) but t_2 violates r_1 (*i.e., *$t_2 \not\models r_1$).*

We say that a table $\mathbf{D}$ matches rule r iff for any tuple t in $\mathbf{D}$, $t \models r$, and a table $\mathbf{D}$ violates rule r iff there exists at least one tuple t in $\mathbf{D}$ where $t \not\models r$.

Repair Instance. Consider a table $\mathbf{D}$ and a set Σ of data repair rules, we say that $\mathbf{D'}$ is a **repair instance** of $\mathbf{D}$ relative to Σ, denoted by $\mathbf{D'} \models \Sigma$, iff (1) $\mathbf{D'}$ is obtained from $\mathbf{D}$ by value updates, *i.e.,* without tuple/column insertions/deletions; and (2) $\mathbf{D'} \models r$ for each rule $r \in \Sigma$.

EXAMPLE 5. *Consider rules $\Sigma = \{r_1, r_2, r_3\}$ in Example 2 and table $\mathbf{D}$ (with erroneous values in red color) in Table 1. Apparently, $\mathbf{D} \not\models \Sigma$. Let $\mathbf{D'}$ be the updated instance of $\mathbf{D}$, where the erroneous values have been updated in green color. One can readily verify that $\mathbf{D'} \models \Sigma$; that is, $\mathbf{D'}$ is a repair instance of $\mathbf{D}$, relative to Σ.*

2.2 Self-supervised and Interpretable Cleaning

Self-supervised rule generation. Given a table $\mathbf{D}$, *self-supervised rule generation* is to learn from $\mathbf{D}$ without any human annotations about data errors and generate a set Σ of data repair rules.

Intuitively, when considering deep learning based solutions, generative models should be adopted.

Co-cleaning inaccurate rules and dirty data. Given a table $\mathbf{D}$ and a set of Σ data repair rules, the problem of *co-cleaning inaccurate rules and dirty data* is to repair both $\mathbf{D}$ to $\mathbf{D'}$ and Σ to Σ' such that $\mathbf{D'}$ is a repair instance of Σ', *i.e.,* $\mathbf{D'} \models \Sigma'$.

2.3 Solution Overview

An overview of GARF is shown in Figure 1. GARF takes as input a dirty dataset and returns a set of trusted rules and cleaned data. The whole cleaning process is data-driven without human supervision. GARF consists of two models "rule generation" and "co-cleaning":

Rule generation: Given a dirty dataset, GARF first generates data repair rules based on the rule generation model, which learns the dependency relationships among data. The core of rule generation is a SeqGAN which consists of a generator G_s and a discriminator D_s. Specifically, we embed the tuples in dirty dataset as the inputs to train the SeqGAN model in a self-supervised manner. After training, the generated data follows the same distribution as the real data, which means that the generator G_s has captured the dependency relationships among data. After this, we use an adaptor to convert the learned relationships from G_s to data repair rules, which can be used for detecting and repairing dirty data. We will provide the details of this model in Section 3.

Co-cleaning: After we have obtained rules from rule generation model, these rules may not be trusted enough. Therefore, we simultaneously clean inaccurate rules and dirty data in the co-cleaning model. We detect data using the generated rules to calculate the dynamic confidence for each rule and the tuples violating the rule. When the confidence of a rule is greater, we update the data by repairing the violated tuples according to the rule. Otherwise, if the confidence of the tuple is greater, we update the conflicting rule based on the data. Note that the rules are generated following the relationships learned from G_s, and we use another discriminator D_d to update G_s and generate new rules (see Section 4 for more details). Repeat the above operations until there is no tuple violating rules; we can obtain cleaned data and trusted rules.

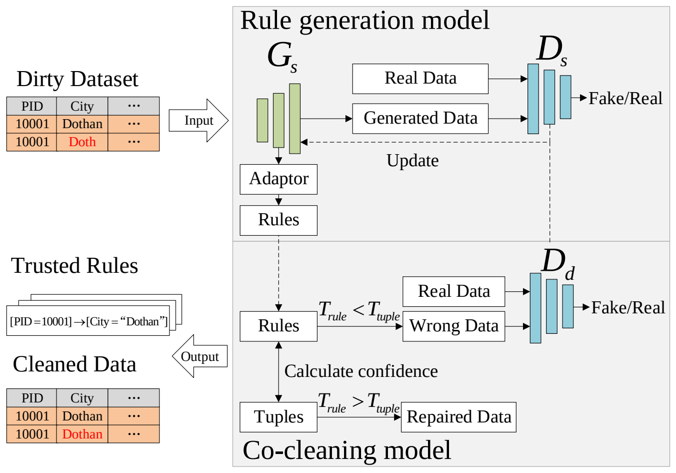


Figure 1: An overview of GARF.

3 SELF-SUPERVISED RULE GENERATION

3.1 Learning with SeqGAN

As noted earlier, manually providing data repair rules is always costly and error-prone, while ML-based methods usually require a lot of training data, which may be unavailable or not have enough capacity. Meanwhile, we notice that many deep learning models are widely used as they can capture data characteristics only based on the input data without labels. However, these models are usually not interpretable. To achieve high interpretability in data cleaning, we introduce a sequence GAN model to generate data repair rules by learning directly from dirty datasets.

Generative Adversarial Networks (GAN). GAN is a generative model that consists of a generator G and a discriminator D, which are competing in an adversarial process. The generator G takes as input random noise z and generates synthetic samples $G(z)$, while the discriminator D determines the probability that a given sample comes from real data rather than being generated by G. Intuitively, the optimal D could distinguish real samples from fake ones, and the optimal G could generate indistinguishable fake samples that make D random guesses.

GANs are adversarially trained iteratively using minibatch stochastic gradient descent algorithms. More specifically, during each iteration, it first freezes the generator G and only trains the discriminator D to be better; then it freezes the discriminator D and only trains the generator G. The training process terminates when the generator produces fake data as a perfect substitute for real data.

SeqGAN. Although GAN has been successful and widely used for computer vision tasks (*e.g.,* generating sample images [17] or image inpainting [51]), it is not easy to be used for generating tuples with many discrete tokens, which are common in real-world datasets. The fundamental reason is that the generator network in GAN is designed to be able to adjust the output continuously, which does not work well on discrete data generation [52].

We consider related attribute values as contexts and the dataset as a corpus where a tuple $(v_1, v_2, \ldots, v_n)$ can be seen as a value sequence under the order of attributes in dataset $\mathbf{D}$, which can be represented as a set of AV_i. The above problem can be addressed by training a SeqGAN [52], which benefits from reinforcement learning (RL) to bypass the problem in GAN generator that is difficult to deal with discrete tokens. RL framework maps scenarios to appropriate actions, with the goal of maximizing a cumulative reward, in which a learning agent interacts with a Markov Decision Process (MDP).

More specifically, we treat the generative model as the agent, and the state is the generated tokens so far while the action is the next token to be generated. Note that the reward in our model is different from the common cases in RL models that require a task-specific score (*e.g.,* BLEU in machine translation). We employ a discriminator to evaluate the sequence and then provide a reward to guide the learning of the generative model. We regard the generative model as a stochastic parametrized policy, in which we employ a Monte Carlo (MC) search to approximate the state-action value. MC search is a method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree based on the results. We iteratively build the search tree until the predefined length of sequence is reached, at which point the search is halted and the best-performing root action is returned. Each node in the search tree represents a current state, and the links directed to the child nodes represent actions leading to subsequent states. By training the generator via policy gradient, we avoid the difficulty updating for discrete tokens.

Training SeqGAN. We first initialize the value sequence generator G_s based on a random θ-parameterized policy network. Afterwards, we train G_s with the tuples of datasets at hand as training data, and produce a sequence $V_{1:n} = (v_1, ..., v_t, ..., v_n)$. For improving G_s, we train the discriminator D_s by providing positive examples from real datasets and negative examples from generated sequences. We denote $D_\phi\left(V_{1:n}\right)$ as the probability indicating how likely a sequence $V_{1:n}$ is real. The problem of updating discrete values can be addressed by reinforcement learning.

At time step t, the state s is the current value sequence $(v_1, ..., v_{t-1})$ and the action a is the next value v_t to select. Thus, the policy $G_\theta\left(v_t|V_{1:t-1}\right)$ is stochastic, whereas the state transition is deterministic after an action has been chosen. Then, G_s can be updated based on a policy gradient and Monte Carlo search on the basis of the expected final reward that is estimated by the likelihood that G_s would fool D_s. The objective of G_θ is to generate a sequence from the start state to maximize the expected final reward: $J\left(\theta\right) = \sum G_\theta\left(v_1|s_0\right) \cdot Q_{D_\phi}^{G_\theta}\left(s_0, v_1\right)$ and the action-value function Q can be denoted as:

$$Q_{D_\phi}^{G_\theta}\left(s = V_{1:t-1}, a = v_t\right) = \begin{cases} \frac{1}{N}\sum\limits_{k=1}^{N} D_\phi\left(Y_{1:n}^k\right) & for\ t < n \\ D_\phi\left(Y_{1:t}\right) & for\ t = n \end{cases} \tag{1}$$

where $Y_{1:n}^k$ is the sampled result of an N-time Monte Carlo search. Finally, the generator's parameters θ update following: $\theta \longleftarrow \theta + \alpha_h \nabla J\left(\theta\right)$, where α_h is the learning rate at h-th step.

When the SeqGAN converges, it gains the ability of generating synthetic tuples where the attributes and values adhere to the same relationships in real data. For the case that whenever the input has the same attribute value pairs, the output always holds the same result, we believe that the input can functionally determine the output, which can be seen as a dependency relationship.

Example 6. *Consider the dataset in Table 1. The tuple t_1 can be seen as a sequence of values "111303, Monticello, GA, 31064, ..." under the attribute order "Provider ID, City, State, Zip, ...". For a well-trained SeqGAN, given the value sequence "Monticello, GA" as the input of LSTM in generator, it may return "31064" as a predicted result with a very large probability. Then we could say that the generator G_s has learned the dependency relationship that the attribute value pairs $(City, \text{"}Monticello\text{"})$ and $(State, \text{"}GA\text{"})$ can functionally determine the attribute value pair $(Zip, 31064)$.*

3.2 Generating Data repair Rules

After the SeqGAN is trained, we next discuss how to generate data repair rules.

To utilize the knowledge captured by SeqGAN in an interpretable manner, we use an adaptor to convert the learned relationships to fine-grained rules. As mentioned earlier, when SeqGAN converges, the generated sequences adhere to the real relationships, which are controlled by G_s. Thus, we consider the input of G_s as the AV_L while the predicted result is the AV_R. If the value in AV_R is equal to the real one, we say AV_L can functionally determine AV_R. Following the syntax in Section 2.1, candidate repair rules can be generated.

Specifically, to generate a rule, we need to define the attributes and values in AV_L and AV_R. We consider that, for a tuple with n attributes, the number of latent AV_L is $n!$, while the number of AV_R is only n. Consequently, we build rules beginning from AV_R. As shown in Figure 2, we take the i-th attribute value pair $AV_i = (a_i, v_i)$ as the result part AV_R, where i ranges from 2 to n. Given the attribute value pairs of AV_L as input, we predict AV_R based on G_s. At first, $AV_L = \{AV_{i-1}\}$. If the predicted value is not equal to v_i, $AV_L \rightarrow AV_R$ is obviously false. Then we supplement AV_L with the previous attribute value as $AV_L = \{AV_{i-2}, AV_{i-1}\}$. We repeat the above process until the predicted value is equal to the real value or the first attribute value pair AV_1 has been included in AV_L.

Note that because relational data is order irrelevant, parsing the tuples only from left-to-right may cause many rules undiscovered. Therefore, we train the model twice with different attribute orders: forward and backward, respectively. By doing this, we can obtain all rules with an attribute in the precondition (*e.g.*, $[Zip = 36301] \rightarrow [County = \text{"}Houston\text{"}]$), and most rules with two or more attributes in the precondition (*e.g.*, $[City = \text{"}Dothan\text{"}, State = \text{"}AL\text{"}] \rightarrow [County = \text{"}Houston\text{"}]$). Although there are still some complex rules that cannot be discovered, the available rules can already achieve good cleaning results. In fact, complex rules are not common in practice.

3.3 Optimization

Although a large number of data repair rules have been generated, they cannot be used for cleaning data directly. A key point is that the generated rules may be (1) **inaccurate** or (2) **contain redundant attributes**. Next, we will describe optimization techniques to tackle the above two problems.

Example 7. *[Inaccurate rules and redundant attributes.] Consider the following rules.*

- $r_1 : [PID = 10001, City = \text{"}Monticello\text{"}] \rightarrow [State = \text{"}AL\text{"}]$
- $r_2 : [City = \text{"}Dothan\text{"}, State = \text{"}AL\text{"}] \rightarrow [County = \text{"}Houston\text{"}]$

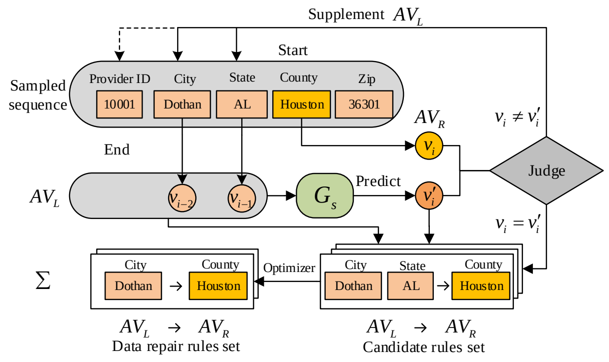


Figure 2: Rule generation based on G_s.

Algorithm 1 Rule generation model

```
Require: real dataset D
1:  Initialize generator policy $G_\theta$ and discriminator $D_s$ with random weights, rules set $\Sigma \leftarrow \emptyset$.
2:  repeat
3:    for g-steps of generator do
4:      Generate a sequence $V_{1:n} \sim G_\theta$ and compute $Q$ by Eq. (2)
5:      Update $G_\theta$ via policy gradient
6:    end for
7:    for d-steps of discriminator do
8:      Train discriminator $D_s$ using generated sequences and real tuples
9:    end for
10: until SeqGAN converges
11: Generate attribute value sequences set $S_g$
12: for each sequence $s$ in $S_g$ do
13:   for each attribute value $AV_i$ of sequence $s$ do
14:     $AV_L \leftarrow AV_{i-1}, AV_R \leftarrow AV_i, k = i$
15:     repeat
16:       Predict $v_i'$ using $AV_L$ based on $G_s$
17:       if $v_i \neq v_i'$ then
18:         $AV_L \leftarrow AV_L \cup AV_{k-1}, k = k - 1$
19:       else
20:         Detect the violated values as $AV_e$
21:         $\Sigma \leftarrow (AV_L \rightarrow AV_R)$
22:       end if
23:     until $v_i = v_i'$ or $k < 1$
24:   end for
25: end for
26: for each candidate rule $r$ in $\Sigma$ do
27:   $N \leftarrow$ Count the number of tuples $t \models r$
28:   Filter the rules which $N < 2$
29:   for each subset $AV_n$ in $AV_L$ of rule $r$ do
30:     Make $r$ minimal by masking $AV_n$ and predicting $AV_R$ again
31:   end for
32: end for
```

where r_1 is an inaccurate rule with only one erroneous tuple matching it and r_2 is a correct rule but contains redundant attributes.

We consider that inaccurate rules may be generated because the predicted values are based on the probability in the policy network, which can be impacted by dirty data or just coincidence. Here, we address this problem in an easy but useful filter. For each candidate rule r, we count the number of tuples $t \models r$ as N.

We notice that few errors are completely the same, which lead to many inaccurate rules have a commonality that they cannot match real tuples ($N = 0$) or only can match one erroneous tuple ($N = 1$). On the other hand, a rule that $N = 1$ has no contribution to repair data sets even if it is correct, as it cannot be used for other tuples and the only tuple matching it has been correct. As a result, we remove all the rules that $N < 2$. Note that the filter cannot delete all incorrect rules, and further repair of the rules will be discussed in Section 4.

Moreover, we notice that the information of $State$ in rule r_2 is redundant because the information that $City$ = "$Dothan$" is enough to determine $County$ = "$Houston$". We reduce the redundant attributes by utilizing the benefits of LSTM, which has a long short term memory. For the well-trained generator G_s in Figure 2, it can predict the value of $County$ as "$Houston$" even if we mask the value of $State$. As a result, we could optimize the rules by masking the subset of AV_L and predict the result again.

Formally, for the rule $r : \{AV_k, \ldots, AV_{i-1}\} \rightarrow AV_i$, we mask AV_{i-1} and predict v_i once again, if the predicted value is still equal to v_i, we believe that AV_{i-1} is redundant for the rule and should be deleted. Otherwise, AV_{i-1} is useful and we keep it unchanged. Then, we mask AV_{i-2} and repeat the process until AV_R is not functionally dependent on any proper subset of AV_L and finally the rule r can be minimal. We train the generator in g-steps while train the discriminator in d-steps, and the whole details of the rule generation can be seen in Algorithm 1.

Clearly, learning the relationships based on SeqGAN, we can generate many fine-grained data repair rules. The whole process is completely automatic, without humans or labels. Furthermore, by self-training on the target dataset, which can be considered as a corpus, the generated rules can break the limitations of domains and languages.

4 CO-CLEANING INACCURATE RULES AND DIRTY DATA

We consider the practical cases that dirty datasets include errors, and the rules learned from such datasets are also untrusted. As a result, when a tuple violates a rule, we cannot distinguish which one is the real error and how to repair it. In this section, we introduce a co-cleaning model to detect and repair both rules and data in a reliable manner.

4.1 Confidence of Rules and Data

To distinguish between rules and the data violating the rules, which are the real errors, the confidence for evaluating the dependability is necessary. Existing confidence metrics mainly rely either on the count of co-occurrences or the support rate, both of which require manually setting a rigid threshold. However, a rigid value may have trouble to distinguish some inaccurate rules in practical cases.

Example 8. *Consider the rules in Example 2 and Tabel 1. Following the rule r_1, we should fix "GAA" in t_2 and "AR" in t_3 to "GA" respectively. Nevertheless, in reality, cities in different states are allowed to have the same name; e.g. "Monticello" belongs to "Georgia"(GA) and also belongs to "Arkansas"(AR).*

Obviously, although r_1 maybe hold for most tuples with a high support rate, it is not always correct for the whole dataset. As a result, traditional metrics that use rigid thresholds can hardly tackle such cases, and repairing data that use such rules may lead to incorrect results. To address this problem, we sign dynamic confidence measures for evaluating rules and data.

We denote the confidence measure of the rules and the tuples by T_{rule} and T_{tuple}, respectively. Intuitively, for a rule r, (1) the more tuples match r, the larger likelihood that r is correct; (2) the rule r is usually inaccurate if a lot of tuples violate it. In other words, T_r should be increased when tuples match r while it would be reduced when r violates tuples. As a result, we initially set T_{rule} of all rules in set Σ as T_0 (default by 1), and adjust the confidence of each r as T_r by detecting tuples using rule r, as shown below:

$$T_r = T_r + \left|t^+\right| - w\left|t^-\right| \tag{2}$$

where $|t^+|$ is the number of tuples that match r while $|t^-|$ is the number of tuples that violate r and w is a punish weight (default by 1). However, we notice that T_{rule} is still not credible because it is correct that r violates error tuples, where T_{tuple} may be a negative number. Clearly, to further evaluate T_{rule} in a reliable manner, we need the confidence T_{tuple} of the rule being applied to tuples.

For a tuple $t = (AV_1, \ldots, AV_i, \ldots, AV_n)$ that $t \vdash r$, we firstly evaluate the confidence of AV_i in t as $T_t(AV_i)$. And we evaluate T_{tuple} according to the minimum of $T_t(AV_i)$, where AV_i is in AV_L or A_i is in AV_R of r. The reason is that, for the tuple t, once there are errors in the values, whose attributes are in AV_L or AV_R of r, it cannot be trusted to evaluate r. And $T_t(AV_i)$ is calculated on the basis of the other rules r' that can also be applicable to t. If a AV_i in t can be verified by r', we increase $T_t(AV_i)$ with the confidence of r'. In contrast, we reduce $T_t(AV_i)$ when AV_i in t violates AV_R of r'.

Formally, we initially set all the $T_t(AV_i)$ default as T_0, and then calculate $T_t(AV_i)$ and the T_{tuple} of t that $t \vdash r$ as:

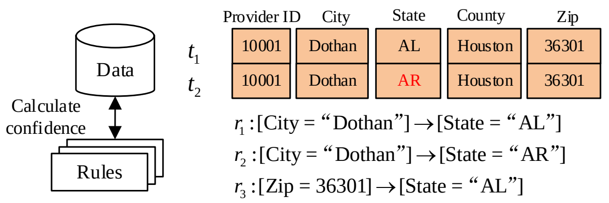
$$T_t(AV_i) = \begin{cases} T_t(AV_i) + T_{r'} & if\ A_i \in AV_R\ of\ r'\ \&\&\ t \models r' \\ T_t(AV_i) - T_{r'} & if\ A_i \in AV_R\ of\ r'\ \&\&\ t \not\models r' \end{cases} \tag{3}$$

$$T_{t \vdash r} = \min\left(T_t(AV_i)\right), AV_i \subseteq AV_L || A_i \in AV_R \tag{4}$$

After this, we reset each T_r to T_0 and update T_r based on $T_{t\vdash r}$. We further consider that a rule cannot be verified as fully trusted even if it matches a trusted tuple, while the rule has a large likelihood of being inaccurate if it violates the trusted tuples. Similarly, a rule cannot be verified as fully trusted when it violates other error tuples. As a result, we calculate T_{rule} for each rule r as follows:

$$T_r = \begin{cases} T_r + \log\left(1 + T_{t\vdash r}\right) & if\ t \models r\ \&\&\ T_{t\vdash r} \geq 0 \\ T_r + T_{t\vdash r} & if\ t \models r\ \&\&\ T_{t\vdash r} < 0 \\ T_r - wT_{t\vdash r} & if\ t \not\models r\ \&\&\ T_{t\vdash r} \geq 0 \\ T_r + \log\left(1 + |T_{t\vdash r}|\right) & if\ t \not\models r\ \&\&\ T_{t\vdash r} < 0 \end{cases} \tag{5}$$

Example 9. *We discuss an example as shown in Figure 3. At first, we calculate T_{rule} based on the detecting results as: $T_{r_1} = T_{r_3} = 1 + 1 - 1 = 1$, $T_{r_2} = 1 - 1 + 1 = 1$. As the value in City cannot be*



Data
Calculate confidence
Rules
Provider ID
City
State
County
Zip
t_1
t_2
10001
Dothan
AL
Houston
36301
10001
Dothan
AR
Houston
36301
r_1 : [City = "Dothan"] → [State = "AL"]
r_2 : [City = "Dothan"] → [State = "AR"]
r_3 : [Zip = 36301] → [State = "AL"]

Figure 3: Confidence calculation for data and rules.

determined, $T_1(City, \text{"}Dothan\text{"}) = T_2(City, \text{"}Dothan\text{"}) = 1$. *While based on* r_3, *we evaluate that* $T_1(State, \text{"}AL\text{"}) = 1 + T_{r_1} = 1 + 1 = 2$ *and* $T_2(State, \text{"}AL\text{"}) = 1 - T_{r_3} = 1 - 1 = 0$. *Hence, we could calculate* T_{tuple} *as:* $T_{t_1 \vdash r_1} = min(T_1(City, \text{"}Dothan\text{"}), T_1(State, \text{"}AL\text{"})) = 1$ *and* $T_{t_2 \vdash r_1} = min(T_2(City, \text{"}Dothan\text{"}), T_2(State, \text{"}AR\text{"})) = 0$. *After this, we update* T_{rule} *once again:* $T_{r_1} = 1 + \log(1 + T_{t_1 \vdash r_1}) - T_{t_2 \vdash r_1} = 1 + \log 2$. *Similar for* $T_{r_2} = 1 - 1 + 0 = 0$. *Based on these confidence, we could repair* t_2 *following* r_1 *and update* r_2 *to fit* t_1 *in subsequent operations.*

4.2 Cleaning Data or Rules based on Confidence

After calculating the confidence of rules and data, we clean both of them, which can be seen as a co-cleaning problem. We consider that the co-cleaning problem that data and rules clean each other while both of them cannot be trusted are similar as the case in the actor-critic (AC) algorithm. That is, both the actor and the critic are green hands (the agents without insufficient ability to give appropriate actions and rewards) at first, and they are trained and updated iteratively to be asymptotically accurate. Note that the GAN can also be seen as an actor-critic algorithm in stateless MDPs. As a result, we train and update the rules and data using the generator and discriminator while determining which one to repair based on confidence. Specifically, for each rule r in Σ, we detect the tuples violating it and determine whether to repair the rule or the tuples according to T_{rule} and T_{tuple}. We discuss the details below and illustrate the repairing process in Figure 4. The whole process of the co-cleaning model can be seen in Algorithm 2.

Rule is more confident. When the rule r has a higher confidence than the tuples violating it, we repair the wrong values in tuples based on r. Generally, we should modify the errors based on AV_R. However, we consider the values that match AV_L also maybe errors, such as "*Dothan*" of t_3 in Figure 4 (a). Here, we further to determinate where the error is. We denote the confidence of AV_i that matches AV_L and violates AV_R as T_{LHS} and T_{RHS}, respectively. Intuitively, $T_{LHS} = \min(T_t(AV_i)), AV_i \subseteq AV_L$ and $T_{RHS} = T_t(AV_i), A_i \in AV_R$. If $T_{LHS} \geq T_{RHS}$, we repair t according to r while if $T_{LHS} < T_{RHS}$, we repair t by modifying AV_i with the minimum confidence based on other rules.

As we can see in Figure 4 (a), the tuples t_1 and t_3 violate the rule r_1. We firstly judge which values are the real errors. For the tuple t_1, $T_{LHS} = min(T_1(City, \text{"}Dothan\text{"}))$ and $T_{RHS} = min(T_1(State, \text{"}AR\text{"}))$. As both $(City, \text{"}Dothan\text{"})$ and $(State, \text{"}AR\text{"})$ cannot be determined by other rules, $T_{LHS} = T_{RHS} = 1$. As a result, we update $(State, \text{"}AR\text{"})$ to $(State, \text{"}AL\text{"})$ based on r_1. For the tuple t_3, $T_{LHS} = min(T_3(City, \text{"}Dothan\text{"}))$ and $T_{RHS} =$

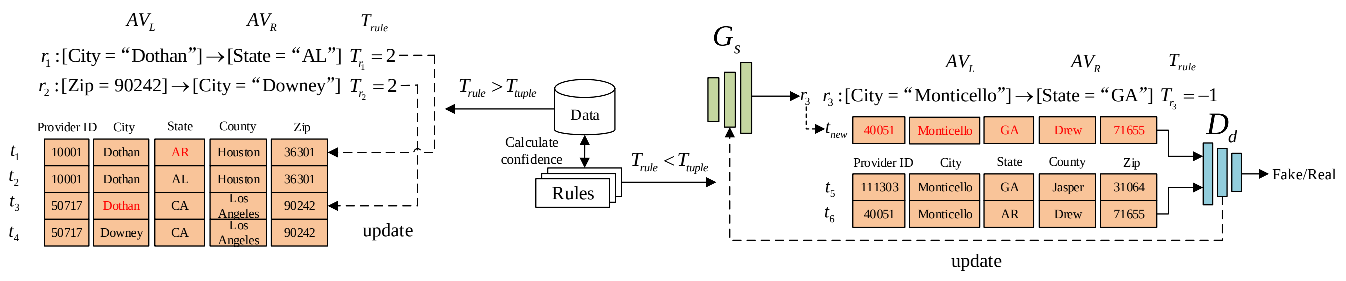
Algorithm 2 Co-cleaning model

```
Require: dirty dataset D; rules set $\Sigma$
 1: for each tuple $t$ in D with its applicable rules in $\Sigma$ do
 2:     Detect $t$ using the rules to calculate $T_{tuple}$, $T_{rule}$
 3:     for each rule $r$ that violates $t$ do
 4:         if $T_{t \vdash r} > T_r$ then
 5:             for g-steps of generator do
 6:                 Generate new data by using $r$ to change the values
 7:                 Update $G_d$ by modifying $r$ based on Algorithm 1
 8:             end for
 9:             for d-steps of discriminator do
10:                 Train $D_d$ using real data and the generated data
11:             end for
12:         else
13:             Calculate $T_{LHS}$ and $T_{RHS}$ of $t$
14:             if $T_{LHS} \geq T_{RHS}$ then
15:                 Repair $t$ by modifying the wrong value based on $r$
16:             else
17:                 Modify the $AV_i$ in $t$ with the minimum confidence
18:             end if
19:         end if
20:     end for
21: end for
```

$min(T_3(State, \text{"}CA\text{"}))$. Similar, $T_{RHS} = 1$. Meanwhile, we notice that $(Zip, 90242)$ matches AV_L while $(City, \text{"}Dothan\text{"})$ violates AV_R of r_2. As a result, $T_{LHS} = 1 - T_{r_2} = -1 < T_{RHS}$. And hence, we update $(City, \text{"}Dothan\text{"})$ to $(City, \text{"}Downey\text{"})$ based on r_2.

Data is more confident. When the confidence of rule is less than the violated tuples, we update the rule to fit data. However, unlike repairing data, where a wrong value can be modified directly based on a ground rule, inaccurate rules cannot be deterministically repaired based on data. Although data is confident, it is not enough that directly fit the rule to data as it is common that modify an inaccurate rule to another inaccurate rule. For instance, the tuple t_5 in Figure 4 (b) is correct and r_3 can be modified to r_3' : $[City = \text{"}monticello\text{"}] \rightarrow [State = \text{"}AR\text{"}]$ by fitting it to t_5. However, r_3' is still untrusted as it violates t_6. We consider that rules are defined following the relationships among data. We use another discriminator D_d to update the generator G_s and modify the rules, which is a process of relearning the relationships to adjust the generated rules. In details, we initialize discriminator D_d with the parameters of D_s and then update it by providing real data and new fake data that is generated based on the untrusted rule. Note that the real data remains unchanged in this step. Then, given real data and generated data as input, D_d calculates the cross entropy to update G_s to $\min E[1 - \log D(x)]$. Specifically, we repair r in two aspects: AV_L and AV_R. In each step, we predict a new AV_R using the same AV_L based on G_s and generate new data with AV_R'. When $E[1 - \log D(x)] < \varepsilon$, where ε is a small constant (default by 0.01) for robustness, we believe r has been repaired. Otherwise, we supplement AV_L of r as the same in Algorithm 1.

See Figure 4 (b), the rule r_3 violates the tuple t_5. To repair it in a reliable manner, we train D_d to update G_s to learn the relationships once again. In each step, r_3 generates a new tuple t_{new} as the fake



AV_L
AV_R
T_{rule}
r_1: [City = "Dothan"] → [State = "AL"] $T_{r_1} = 2$
r_2: [Zip = 90242] → [City = "Downey"] $T_{r_2} = 2$
Provider ID
City
State
County
Zip
t_1 10001 Dothan AR Houston 36301
t_2 10001 Dothan AL Houston 36301
t_3 50717 Dothan CA Los Angeles 90242
t_4 50717 Downey CA Los Angeles 90242
update
$T_{rule} > T_{tuple}$
Data
Calculate confidence
Rules
$T_{rule} < T_{tuple}$
G_s
r_3: [City = "Monticello"] → [State = "GA"] $T_{r_3} = -1$
t_{new} 40051 Monticello GA Drew 71655
t_5 111303 Monticello GA Jasper 31064
t_6 40051 Monticello AR Drew 71655
D_d
Fake/Real
update

(a) Repair data based on rules (b) Repair rules based on data

Figure 4: Cleaning inaccurate rules and dirty data.

Table 2: Effectiveness of rule discovering on different datasets.

Method	HOSP			FOOD			FLIGHT			UIS		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Tane*	0.47	0.42	0.44	0.41	0.36	0.38	0.55	0.49	0.52	0.87	0.46	0.60
FastFD*	0.62	0.57	0.60	0.61	0.49	0.54	0.47	0.41	0.44	0.89	0.66	0.76
FD_Mine*	0.65	0.31	0.42	0.75	0.46	0.57	0.73	0.51	0.60	0.94	0.41	0.57
Garf	**0.99**	**0.69**	**0.81**	**0.96**	**0.65**	**0.78**	**0.97**	**0.74**	**0.84**	**1.00**	**0.77**	**0.87**

data. D_d receives the fake data and real data as inputs to update G_s. Based on the updated G_s, r_3 can be modified and a new tuple can be generated. Repeat the above operations until D_d and G_s converge, the rule can be repaired by modifying the value in AV_R or adding the condition information in AV_L, *e.g.*, r_3 can be repaired as $[ProviderID = 40051, City = \text{“}Monticello\text{”}] \rightarrow [State = \text{“}AR\text{”}]$ and $[ProviderID = 111303, City = \text{“}Monticello\text{”}] \rightarrow [State = \text{“}GA\text{”}]$.

5 EXPERIMENT

The key questions we answer with our evaluations are: (1) Since Garf is a novel data cleaning framework that generates repair rules to clean dirty data, how do our generated rules compare in effectiveness and accuracy to existing widely-used rule discovering methods? (2) How does the choice of the threshold for generating rules affect the methods performance? (3) How does Garf compare in quality to state-of-the-art data cleaning methods with different number of given constraints? (4) How do dirty data affect the method performance for datasets with different error rate? (5) Although Garf is self-supervised and there is no demand for labels or other information, can we employ Garf to repair the existing rules for other methods and achieve better performance? (6) What are the advantages of Garf compared with other deep learning-based data cleaning methods? (7) What are the potential sources of gains for Garf? (8) What about the efficiency of Garf?

5.1 Experiment Settings

Datasets. We perform our experiments on both real-life and synthetic datasets: (1) Hospital information dataset (HOSP): This is a benchmark dataset used in many literatures [18, 39], which is taken from the US Department of Health Services. It consists of 100K tuples with 10 attributes. (2) Food dataset (FOOD): This is a real dataset from the City of Chicago with information on inspections of food establishments [38]. It consists of 200K tuples with 13 attributes. (3) Flight dataset (FLIGHT): This dataset contains data on the departure and arrival time of flights from different data sources [32]. It consists of 50K tuples with 6 attributes. (4) Address dataset (UIS): This is a synthetic dataset generated by the UIS dataset generator [2]. It consists of 100K tuples with 11 attributes.

Following many state-of-the-art methods [27, 43], we treat the original datasets as clean data, and dirty data is generated by adding noise with a certain rate e%, i.e., the percentage of dirty tuples on all data tuples (10% by default). For instance, a error rate of 30% means that 70% of the tuples are clean and the remaining 30% of the tuples have errors. We introduced three types of noise: missing values, typos, and errors from the active domain.

Baselines. For comparison, we implement three kinds of state-of-the-art methods for cleaning dirty data.

(1) We firstly implement different combinations of typical constraints discovery methods [8] with a well-studied data repair method FD-DR [43] as follows:

(i) *Tane*∗: Tane [30] + CTane [21] + FR-DR;

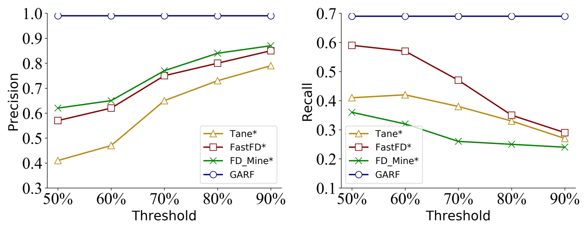
(ii) *FastFD*∗: FastFD [46] + FastCFD [21] + FR-DR;

(iii) *FD_Mine*∗: FD_Mine [49] + CFD_Miner [21] + FR-DR;

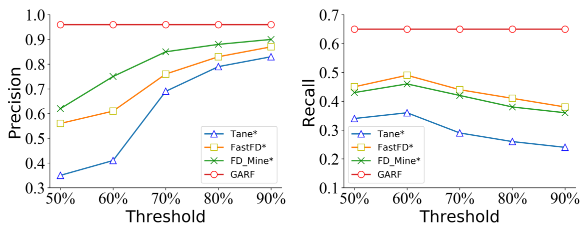
where Tane, FastFD, FD_Mine are the state-of-the-art algorithms for discovering FDs [34] while CTane, FastCFD, CFD_Miner are the common algorithms for discovering CFDs [36].

(2) Next, we compare Garf with two state-of-the-art automated data cleaning methods, including (i) WMRR-DR [2]: a logical data cleaning method, which can discover repair rules only based on constraints with dirty data, and (ii) Holoclean [38]: a data cleaning system that uses learning and probabilistic inference to produce repairs.

(3) Finally, we compare our approach with another deep learning-based method: (i) Baran [32]: a novel error correction system that fixes data errors with respect to their value, vicinity, and domain contexts; (ii) language models, including BERT, RoBERTa and GPT-1: modelling data distribution based on self-attention and repair



(a) Threshold vs. Precision (HOSP) (b) Threshold vs. Recall (HOSP)



(c) Threshold vs. Precision (FOOD) (d) Threshold vs. Recall (FOOD)

Figure 5: Evaluation results at different thresholds.

wrong values by predicting the correct ones.
Metrics. To evaluate the performance of different methods, we report the following metrics: (1) Precision (P): The number of correctly repaired tuples over the total number of repaired tuples in data, which assesses the correctness of repairing; (2) Recall (R): the number of correctly repaired tuples over the total number of dirty tuples, which assesses the completeness; (3) F1−score: the harmonic mean of precision and recall, which computed as $2(P \times R)/(P + R)$.
Implementation. GARF is implemented in Python. We run the experiments on the machine with 12-core 3.20GHz i7-8700 and 16GB main memory.

5.2 Effectiveness of Rule Discovering

Let's first focus on the effectiveness of rule discovery. We report the precision, recall, and F1-score obtained by GARF and competing baselines in Table 2. As we can see, GARF outperforms all baselines in terms of the F1-score in all the datasets as our approach in formulating data cleaning achieves both high precision and recall. In fact, since GARF generates rules based on the learned relationships among data rather than statistical information, such as the co-occurrences frequency and support rate, the generated rules can be much more comprehensive than other baselines. In addition, the rules are fine-grained, making the repair results more accurate. As a result, GARF can repair more dirty tuples with greater accuracy.

In addition, we observe that the results on FLIGHT and UIS are generally better than that on HOSP and FOOD. This is due to the fact that FLIGHT and UIS have more frequent patterns for each constraint. And when the ratio of frequent patterns is less than the threshold (defaulted by 0.6), some constraints and rules may not be discovered. Thus, we further examine how the performance results change at different thresholds.

As shown in Figure 5, we vary the thresholds for each method and compare precision and recall on the HOSP and FOOD datasets, respectively. As expected, as the threshold increases, the filter to distinguish constraints in baselines becomes stricter, which generally leads to a larger precision with a smaller recall. Noted that, when the threshold is at a low degree, there are many spurious constraints that may lead to new errors. And with the threshold increase, the number of spurious constraints reduces, which also improves the recall. In contrast to traditional methods, GARF does not adapt a fixed threshold to discover rules based on frequent patterns. GARF generates repair rules by learning the dependency relationships in datasets based on a SeqGAN model. Furthermore, to make the generated rules more trusted, GARF detects and repairs inaccurate rules by competing with dirty data based on the dynamic confidence measure, which is self-adaptive. The whole process is self-supervised and data driven, rather than requiring experts to set constant thresholds for different datasets. So, GARF always maintains stable results and outperforms other baselines.

5.3 Effectiveness of Repairing Quality

After demonstrating the effectiveness of GARF in rule discovery, we further compare repair quality with other state-of-the-art data repair methods. Note that although these methods are usually capable of obtaining good repair performance, most of them require existing constraints as prior knowledge. To this end, we compete the evaluation results of GARF with baselines by giving different number constraints. We randomly select different numbers of constraints (# Constraints) from all constraints discovered in Section 5.2 and compare precision, recall, and F1-score in different datasets in Table 3.

As we can see, when # Constraints is small, both precision and recall of WMRR-DR and Holoclean are at a low degree. And their results improve with the increase of # Constraints. Specifically, their recall increases with # Constraints nearly linearly at first and then slows down until constraints cover most attributes. Moreover, we observe that when constraints are not enough, many inaccurate rules can be kept, leading to a low precision. And with increasing constraints, many incorrect rules can be deleted by inconsistency resolution. Furthermore, it can be seen that WMRR-DR performs better on FLIGHT and UIS, while it performs worse on HOSP and FOOD. The reason is that WMRR-DR repairs data based on fixed rules, which benefits from frequent patterns, while Holoclean repairs data based on learning and probabilistic inference. Clearly, GARF owns the advantages of both of them and can always achieve better performance on different datasets without any prior knowledge given.

5.4 Robustness and Flexible Analysis

As mentioned above, dirty data can lead to inaccurate rules, which can also affect the quality of repair results. Consequently, to demonstrate the robustness of our approach, we examine the impact of errors on repair quality and report the results in Table 4, where we vary the error rate from 10% up to 90% with a step of 20%. The results show that as the error rate increases, precision and recall of all the methods naturally drops. In fact, we observe that missing values can lead to many useful rules cannot be found, which reduces the recall of baselines. Meanwhile, the typos in datasets diminish the trustworthy evidence, and there are not only

Table 3: Repairing performance on different datasets.

Method	# Constraints	HOSP			FOOD			FLIGHT			UIS		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1
WMRR-DR	10	0.53	0.30	0.38	0.41	0.29	0.34	0.38	0.32	0.35	0.45	0.26	0.33
	20	0.64	0.43	0.51	0.56	0.40	0.47	0.54	0.49	0.51	0.53	0.43	0.47
	30	0.72	0.52	0.60	0.62	0.48	0.54	0.72	0.62	0.67	0.72	0.57	0.64
	40	0.78	0.57	0.66	0.68	0.56	0.61	0.85	0.65	0.74	0.89	0.66	0.76
	50	0.83	0.59	0.69	0.74	0.58	0.65	0.87	0.66	0.75	0.91	0.71	0.80
Holoclean	10	0.59	0.23	0.33	0.44	0.21	0.28	0.53	0.29	0.37	0.44	0.18	0.26
	20	0.64	0.39	0.48	0.56	0.38	0.45	0.64	0.43	0.51	0.59	0.32	0.41
	30	0.72	0.53	0.61	0.62	0.51	0.56	0.72	0.59	0.65	0.68	0.45	0.54
	40	0.79	0.65	0.71	0.87	0.63	0.73	0.84	0.63	0.72	0.75	0.58	0.65
	50	0.85	**0.69**	0.76	0.91	**0.67**	0.77	0.85	0.63	0.72	0.88	0.68	0.77
GARF	0	**0.99**	**0.69**	**0.81**	**0.96**	0.65	**0.78**	**0.97**	**0.73**	**0.83**	**1.00**	**0.77**	**0.87**

Table 4: Precision comparison by varying error rates.

Dataset	Error-rate	WMRR-DR		Holoclean		GARF	
		P	R	P	R	P	R
HOSP	10%	0.83	0.59	0.85	0.69	0.99	0.69
	30%	0.68	0.54	0.71	0.59	0.98	0.63
	50%	0.43	0.39	0.65	0.42	0.98	0.55
	70%	0.37	0.20	0.51	0.37	0.97	0.51
	90%	0.25	0.17	0.27	0.33	0.95	0.36
FOOD	10%	0.85	0.59	0.91	0.67	0.96	0.65
	30%	0.64	0.51	0.88	0.59	0.96	0.64
	50%	0.40	0.35	0.67	0.52	0.95	0.60
	70%	0.31	0.26	0.43	0.41	0.94	0.58
	90%	0.19	0.17	0.26	0.25	0.94	0.47

Table 5: Hybrid methods.

Method	Dataset	P	R	F1
GARF +WMRR-DR	HOSP	0.99	0.73	0.84
	FOOD	0.98	0.71	0.82
	FLIGHT	1.0	0.79	0.88
	UIS	1.0	0.82	0.90
GARF +Holoclean	HOSP	0.98	0.72	0.83
	FOOD	0.98	0.69	0.81
	FLIGHT	1.0	0.75	0.86
	UIS	0.99	0.74	0.85

rules missing but also some inaccurate rules generated, which reduces both precision and recall. Furthermore, errors from the active domain can lead to new errors even if the rules are correct. For example, $[City = \text{“}Dothan\text{”}] \rightarrow [State = \text{“}AL\text{”}]$ is a correct rule, while $(Dothan, CA, LosAngeles, \dots)$ is an error tuple where “*Dothan*” should actually be “*Downey*”. Clearly, it is still inaccurate to repair the tuple using the rule. Unlike baselines, GARF repairs data following learned relationships, which are continuously updated with repair. Incorrect rules can be detected and repaired in a reliable manner. Moreover, we evaluate the confidence of each attribute value pair used in the rule to guarantee that the repaired one is a real error. As a result, GARF always keeps an excellent performance with high robustness.

Furthermore, we consider that $City \longrightarrow State$ is inaccurate for the whole HOSP dataset, as discussed in Example 8. Similarly, $Facility \longrightarrow Risk$ is also inaccurate although the *Risk* of more than half *Facility* with “*restaurant*” is “*high*” in FOOD dataset. Obviously, there are always many incorrect constraints in practical cases, can we repair them based on our approach for other methods and achieve better performance? Actually, GARF is flexible that, although it does not have prior knowledge requirements, it can help to improve existing constraints and take advantage of such constraints to achieve better performance. Thus, we implement combinations of our framework with WMRR-DR and Holoclean as GARF +WMRR-DR and GARF +Holoclean, respectively. The results are reported in Table 5. Clearly, the rules repaired by GARF can be trusted enough and the hybrid methods based on these rules can achieve high accuracy.

Table 6: Comparison with language models.

Method	HOSP	FOOD	FLIGHT	UIS
BERT	0.41	0.39	0.52	0.42
RoBERTa	0.42	0.39	0.53	0.42
GPT-1	0.32	0.28	0.45	0.34
GARF	0.99	0.96	0.97	1.00

5.5 Comparison with Deep Learning Methods

Since GARF considers the tuples in datasets as sequences and learns the dependency relationships in the sequences, which is similar as a Natural Language Processing (NLP) task, we then evaluate the results that repair data directly using other language models. For comparison, we use BERT, RoBERTa and GPT-1 to impute the data values which are detected as errors in our method because those models cannot detect errors in raw data.

Table 6 shows that GARF has the highest precision in all datasets. The reason is that GARF repairs data based on the rules that have been verified and repaired while other models repair data directly based on the prediction for values, which are probabilistic. In the NLP tasks, some approximate values may be accepted, however, it is unreasonable that repair data in datasets using such values.

Interpretability. GARF has a unique advantage on interpretability, while most DL-based methods can repair the wrong value but the repaired value is hard to interpret, which leads to the wrong repairs that can hardly be found and repaired by humans. Instead of

Table 7: Sample learned rules on different datasets.

Datasets	Data repair Rules ($AV_L \rightarrow AV_R$)	Confidence
HOSP	$[City = \text{“Dothan”}, State = \text{“AL”}] \rightarrow [Country = \text{“Houston”}]$	37
	$[City = \text{“Monticello”}] \rightarrow [State = \text{“GA”}]$	-15
	$[MeasureID = \text{“OP\_33”}] \rightarrow [Condition = \text{“Caner care”}]$	20
FOOD	$[AKA\ Name = \text{“THE EDGE”}] \rightarrow [BDA\ Name = \text{“GOLDEN NUGGET”}]$	6
	$[Address = \text{“525 E 130TH ST”}, AKA\ Name = \text{“ROSEBUD FARM INC”}] \rightarrow [Facility = \text{“Grocery Store”}]$	7
	$[License = 14616, Facility = \text{“Restaurant”}] \rightarrow [Risk = \text{“High”}]$	-12
FLIGHT	$[Flight = \text{“AA-4307-ORD-DTW”}] \rightarrow [Sched\_dep\_time = \text{“ 6 : 45p.m.”}]$	8
	$[Sched\_dep\_time = \text{“7 : 10 a.m.”}, Act\_dep\_time = \text{“7 : 16 a.m.”}] \rightarrow [Sched\_arr\_time = \text{“9 : 40 a.m.”}]$	9
	$[Src = \text{“Boston”}, Flight = \text{“AA-1434-DFW-MCO”}] \rightarrow [Act\_dep\_time = \text{“7 : 21 a.m.”}]$	7
UIS	$[CUID = 4719] \rightarrow [City = \text{“Cedar Bluff”}]$	47
	$[APMT = \text{“2h6”}] \rightarrow [Zip = 68653]$	66
	$[City = \text{“Platte Center”}, Lname = \text{“Orleman”}] \rightarrow [State = \text{“NE”}]$	52

modifying values directly, Garf repairs data based on the learned repair rules with confidence measure as shown in Table 7, which is easy to be understood. And even there are still some inaccurate rules, they can be found and repaired easily by humans.

5.6 Ablation Study

Since Garf has been demonstrated to be effectiveness with high accuracy, we finally evaluate what the potential sources of gains for Garf are: the rule generation model (including the generator and discriminator) and the co-cleaning model. In order to understand how each of them affects the performance of Garf, we exclude some of them and compare the precision of the resulting architectures against the full Garf architecture: (1) Garf w/o co-cleaning: we exclude the co-cleaning model and repair data using the generated rules directly. (2) Garf w/o discriminator: we exclude the discriminator and generate rules only based on the generator, which can be seen as converting GAN structure to LSTM. (3) Garf w/o Gates & discriminator: we exclude the gates that are used to transmit or forget memories and the discriminator, which can be seen as converting the GAN structure to RNN. (4) Garf w/o Gates & discriminator & co-cleaning: we only keep the generator with a RNN structure and exclude all other components. Table 8 shows the performance of Garf with different components.

As we can see from Table 8, there is a significant drop in both precision and recall without the co-cleaning model because the wrong rules cannot be detected and repaired. When we exclude the discriminator, the relationships are learned only based on the generator (LSTM) while a simple LSTM cannot capture the relationships accurately, which leads to many rules being generated and may not be repaired in co-cleaning. Besides, we also observe that the reasoning ability of rules reduces when excluding the gates in generator as it cannot transmit or forget long memories. Finally, when all the gates, the discriminator and the co-cleaning model are excluded, the generated rules include many errors and clean data based on such rules can lead to disastrous results.

In fact, comparing with a vanilla neural network model (e.g, LSTM), the GAN model benefits form the discriminator to achieve better performance. Although the discriminator may lead to the generator "overfit" to the real ones, and "overfit" is a disadvantage for deep learning models in most cases, it is an advantage for repairing data because the repaired values are expected to be identical as the real correct ones. By learning using the generator, updating using the discriminator, and repairing based on the co-cleaning, we can finally obtain clean data and trusted data repair rules.

5.7 Runtime Results

We report the running times results of our framework with baselines in Table 9. As we can see, for dirty datasets with all the methods have not been trained, WMRR-DR outperforms the running time as it repairs data only based on logical rules, and the learning-based methods require more time on statistical learning, inference, or modeling data distribution. However, we believe that optimizing machine runtime efficiency is not the main goal of data cleaning methods as improving effectiveness and reducing human involvement are more important objectives.

Although efficiency is not the main concern of Garf, it also has an unique advantage in running time. Most existing learning-based methods have the limitation that models need to be executed integrally for each repair task. Unlike such methods, Garf takes advantage of both logical data cleaning and learning-based data cleaning. Garf converts the dependency relationships learned by the machine to interpretable rules, which can be seen as general knowledge in the related domain (such as medical rules in the medical domain and address rules in the address domain). As a result, our approach can be seen as "off-line", which is independent of repair tasks. Once the training is completed, we can deal with subsequent cleaning tasks at a low runtime cost.

6 RELATED WORK

Logical Data Cleaning. A number of articles have investigated the logical data cleaning problem based on constraints and rules [3, 13]. They explore consistent databases by computing repairs that minimally change the data instance to satisfy a set of constraints [39]. To clean dirty data, different constraints are employed such as FDs [5, 45], CFDs [20], MDs [19], denial constraints [11] and neighborhood constraints [40]. As a result, many constraints discovery methods have been proposed, e.g., Tane [30], FastFD [46], and FD_Mine [49]. However, these constraints mainly focus on the static analyses for satisfiability and implication problems [6, 7], leading that they can determine whether a tuple is dirty or not, but they cannot explicitly specify how to repair the errors [43].

Table 8: Ablation Study.

Method	HOSP			FOOD			FLIGHT			UIS		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Garf	0.99	0.69	0.81	0.96	0.65	0.78	0.97	0.73	0.81	1.00	0.77	0.87
Garf w/o Co-cleaning	0.76	0.53	0.62	0.58	0.49	0.53	0.51	0.49	0.50	0.79	0.62	0.69
Garf w/o Discriminator (convert GAN to LSTM)	0.83	0.56	0.67	0.74	0.55	0.63	0.85	0.52	0.65	0.84	0.59	0.69
Garf w/o Gates & Discriminator (convert GAN to RNN)	0.81	0.35	0.49	0.72	0.28	0.40	0.81	0.48	0.60	0.81	0.40	0.54
Garf w/o Gates & Discriminator & Co-cleaning	0.12	0.11	0.11	0.09	0.07	0.08	0.25	0.18	0.21	0.15	0.12	0.13

Table 9: Runtime on different datasets (minutes).

Method	Datasets			
	HOSP	FOOD	FLIGHT	UIS
WMRR-DR	5	13	2	5
Holoclean	26	75	6	22
Baran	42	102	15	38
Garf	73	185	12	51
Garf (Training finished)	4	10	2	4

To this end, various repairing models are proposed to generate ground rules and repair data in a determinate manner. [44] proposes fixing rules that encode constant values in the rules. Chu et al. [11] tackle the problem in a unified framework that can express rules involving numerical values with predicates. Nevertheless, to obtain high-quality rules, these methods often require external information about the golden standard, which normally comes from domain experts, or knowledge bases. Fixing rules [43] require experts to examine the rules that are in conflict. Fan et al. [22] define editing rules depending on master data and user verification to perform reliable repairing. Hao et al. [29] use knowledge bases to generate deductive rules to fix errors, and the links between dirty databases and knowledge bases are still user guided. Unfortunately, involving users is typically costly and error-prone, and the required domain experts and knowledge base may not be available or without enough capacity. To this end, WMRR-DR [2] is proposed as the first automatic rules discovery method for data repairing, but a set of given FDs over target dataset is still necessary.

Furthermore, there are an increasing amount of literature for data cleaning on noisy datasets. Golab et al. [25] adopt a set of CFDs to modify the embedded FD for better fitting the data. However, modifying the data and relative trust were not discussed. Chiang et al. [9] address the problem based on a cost model that quantifies the trade-off of when an inconsistency is a true data error vs. an update to the model.

Different from the previous works, Garf targets at generating repair rules in an automated and sufficient manner. Garf provides a self-supervised rule generation model that learns dependency relationships from data and generates fine-grained data repair rules. The whole process is only based on the data in hand without human and prior knowledge, and the generated rules are sufficient, which can break the limitations of domains and languages.

Learning-based Data Cleaning. ML-based methods are also always employed for data cleaning. In contrast to logical data cleaning methods that require a lot of efforts to build rules, ML-based methods perform statistical repairing by applying probabilistic to infer the correct data. However, these methods are typically supervised since they heavily require training data and rely on the chosen features, such as Guided data repair [48] and SCAREd [47].

Recently, with the great success of deep learning in machine learning, artificial neural networks have gradually been introduced to data repairing [26, 42]. These methods can exploit potential characteristics of data, which are benefit for comprehensive repairing and self-supervised learning. However, due to the inexplicability of neural networks, the result of repairing can hardly be interpreted to humans. As a result, existing explainable data repair methods can hardly deal with the limitation caused by deep learning. [37] extracts the explanation by cleaning a small set of tuples by users and inferring the constraint underlying those repairs. Unfortunately, users cannot understand the reason for repairing operations and distinguish which operations are wrong only based on the parameter networks. Furthermore, many deep learning models are designed for images or continuous data, and these methods have limitations with discrete relational data. GAIN [50] is a data repair method for missing data imputation, which builds a GAN model to impute the missing values. Although it performs well on numerical data, it is not fit for categorical values.

To tackle the challenges above, we take advantages of both logical data cleaning methods and learning-based data cleaning methods. We utilize SeqGAN to learn the dependency relationships among data rather than updating values directly. We convert the learned relationships to data repair rules, which are deterministic and have high interpretability. The whole process is completely data-driven without human and external information.

7 CONCLUSION

In this paper, we propose Garf, a self-supervised framework to generate trusted data repair rules and to clean dirty data. Our approach can be completely data-driven without human supervision and priori knowledge. By learning the dependency relationships among data based on SeqGAN, we could generate lots of fine-grained data repair rules with no requirements of domain knowledge or labeled training data. Extensive experiment results demonstrate that our approach is able to effectively clean dirty data and outperforms state-of-the-art methods. And the whole framework is robustness, flexible with high interpretability.

8 ACKNOWEDGMENTS

This work is supported by National Natural Science Foundation of China (62172082, 62072084, 62072086), and the Fundamental Research Funds for the Central Universities (N2116008).

REFERENCES

[1] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting Data Errors: Where are we and what needs to be done? *Proc. VLDB Endow.* 9, 12 (2016), 993–1004. https://doi.org/10.14778/2994509.2994518

[2] Hiba Abu Ahmad and Hongzhi Wang. 2020. Automatic weighted matching rectifying rule discovery for data repairing. *VLDB J.* 29, 6 (2020), 1433–1447.

[3] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. 2005. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. 143–154.

[4] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional functional dependencies for data cleaning. In *2007 IEEE 23rd international conference on data engineering*. IEEE, 746–755.

[5] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. 2005. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 143–154.

[6] Loreto Bravo, Wenfei Fan, Floris Geerts, and Shuai Ma. 2008. Increasing the expressivity of conditional functional dependencies without extra complexity. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 516–525.

[7] Loreto Bravo, Wenfei Fan, and Shuai Ma. 2007. Extending Dependencies with Conditions.. In *VLDB*, Vol. 7. Citeseer, 243–254.

[8] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2020. Mining relaxed functional dependencies from data. *Data Min. Knowl. Discov.* 34, 2 (2020), 443–477.

[9] Fei Chiang and Renée J. Miller. 2011. A unified model for data and constraint repair. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan (Eds.). IEEE Computer Society, 446–457.

[10] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *Proc. VLDB Endow.* 6, 13 (2013), 1498–1509.

[11] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *29th IEEE International Conference on Data Engineering, ICDE*. 458–469.

[12] Xu Chu, Mourad Ouzzani, John Morcos, Ihab F. Ilyas, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: Reliable Data Cleaning with Knowledge Bases and Crowdsourcing. *Proc. VLDB Endow.* 8, 12 (2015), 1952–1955.

[13] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving Data Quality: Consistency and Accuracy.. In *VLDB*, Vol. 7. 315–326.

[14] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias (Eds.). ACM, 541–552. https://doi.org/10.1145/2463676.2465327

[15] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2017/papers/p44-deng-cidr17.pdf

[16] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *Proc. VLDB Endow.* 14, 3 (2020), 307–319. https://doi.org/10.5555/3430915.3442430

[17] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. 2015. Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv:1506.05751* (2015).

[18] Amr Ebaid, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2013. NADEEF: A Generalized Data Cleaning System. *Proc. VLDB Endow.* 6, 12 (2013), 1218–1221.

[19] Wenfei Fan, Hong Gao, Xibei Jia, Jianzhong Li, and Shuai Ma. 2011. Dynamic constraints for record matching. *VLDB J.* 20, 4 (2011), 495–520.

[20] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008), 6:1–6:48.

[21] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2011. Discovering Conditional Functional Dependencies. *IEEE Transactions on Knowledge and Data Engineering* 23, 5 (2011), 683–698.

[22] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2012. Towards certain fixes with editing rules and master data. *VLDB J.* 21, 2 (2012), 213–238.

[23] Wenfei Fan, Jianzhong Li, Nan Tang, and Wenyuan Yu. 2012. Incremental Detection of Inconsistencies in Distributed Data. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, Anastasios Kementsietsidis and Marcos Antonio Vaz Salles (Eds.). IEEE Computer Society, 318–329. https://doi.org/10.1109/ICDE.2012.82

[24] Wenfei Fan, Shuai Ma, Nan Tang, and Wenyuan Yu. 2014. Interaction between Record Matching and Data Repairing. *ACM J. Data Inf. Qual.* 4, 4 (2014), 16:1–16:38. https://doi.org/10.1145/2567657

[25] Lukasz Golab, Howard J. Karloff, Flip Korn, and Divesh Srivastava. 2010. Data Auditor: Exploring Data Quality and Semantics using Pattern Tableaux. *Proc. VLDB Endow.* 3, 2 (2010), 1641–1644.

[26] Xu Han, Xiaohui Chen, and Li-Ping Liu. 2021. GAN Ensemble for Anomaly Detection. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence*. 4090–4097.

[27] Shuang Hao, Nan Tang, Guoliang Li, Jian He, Na Ta, and Jianhua Feng. 2017. A Novel Cost-Based Model for Data Repairing. In *33rd IEEE International Conference on Data Engineering, ICDE*. 49–50.

[28] Shuang Hao, Nan Tang, Guoliang Li, and Jian Li. 2017. Cleaning Relations Using Knowledge Bases. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*. IEEE Computer Society, 933–944. https://doi.org/10.1109/ICDE.2017.141

[29] Shuang Hao, Nan Tang, Guoliang Li, Jian Li, and Jianhua Feng. 2018. Distilling relations using knowledge bases. *The VLDB Journal* 27, 4 (2018), 497–519.

[30] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. 1999. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Comput. J.* 42, 2 (1999), 100–111.

[31] Matteo Interlandi and Nan Tang. 2015. Proof positive and negative in data cleaning. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 18–29.

[32] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *Proc. VLDB Endow.* 13, 11 (2020), 1948–1961.

[33] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference*. 865–882.

[34] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. 2015. Functional Dependency Discovery: An Experimental Evaluation of Seven Algorithms. *Proc. VLDB Endow.* 8, 10 (2015), 1082–1093.

[35] Nataliya Prokoshyna, Jaroslaw Szlichta, Fei Chiang, Renée J. Miller, and Divesh Srivastava. 2015. Combining Quantitative and Logical Data Cleaning. *Proc. VLDB Endow.* 9, 4 (2015), 300–311. https://doi.org/10.14778/2856318.2856325

[36] Abdulhakim Ali Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern Functional Dependencies for Data Cleaning. *Proc. VLDB Endow.* 13, 5 (2020), 684–697.

[37] Joeri Rammelaere and Floris Geerts. 2018. Explaining Repaired Data with CFDs. *Proc. VLDB Endow.* 11, 11 (2018), 1387–1399.

[38] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proc. VLDB Endow.* 10, 11 (2017), 1190–1201.

[39] El Kindi Rezig, Mourad Ouzzani, Walid G. Aref, Ahmed K. Elmagarmid, Ahmed R. Mahmood, and Michael Stonebraker. 2021. Horizon: Scalable Dependency-driven Data Cleaning. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2546–2554.

[40] Shaoxu Song, Hong Cheng, Jeffrey Xu Yu, and Lei Chen. 2014. Repairing Vertex Labels under Neighborhood Constraints. *Proc. VLDB Endow.* 7, 11 (2014), 987–998.

[41] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Samuel Madden, and Mourad Ouzzani. 2021. RPT: Relational Pre-trained Transformer Is Almost All You Need towards Democratizing Data Preparation. *Proc. VLDB Endow.* 14, 8 (2021), 1254–1261. https://doi.org/10.14778/3457390.3457391

[42] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Samuel Madden, and Mourad Ouzzani. 2021. RPT: Relational Pre-trained Transformer Is Almost All You Need towards Democratizing Data Preparation. *Proc. VLDB Endow.* 14, 8 (2021), 1254–1261.

[43] Jiannan Wang and Nan Tang. 2014. Towards dependable data repairing with fixing rules. In *International Conference on Management of Data, SIGMOD*. 457–468.

[44] Jiannan Wang and Nan Tang. 2017. Dependable Data Repairing with Fixing Rules. *ACM J. Data Inf. Qual.* 8, 3-4 (2017), 16:1–16:34.

[45] Jef Wijsen. 2005. Database repairing using updates. *ACM Trans. Database Syst.* 30, 3 (2005), 722–768.

[46] Catharine M. Wyss, Chris Giannella, and Edward L. Robertson. 2001. FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances - Extended Abstract. In *Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001, Munich, Germany, September 5-7, 2001, Proceedings*. 101–110.

[47] Mohamed Yakout, Laure Berti-Équille, and Ahmed K. Elmagarmid. 2013. Don't be SCAREd: use SCalable Automatic REpairing with maximal likelihood and bounded changes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias (Eds.). ACM, 553–564.

https://doi.org/10.1145/2463676.2463706

[48] Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. 2011. Guided data repair. *Proc. VLDB Endow.* 4, 5 (2011), 279–289. https://doi.org/10.14778/1952376.1952378

[49] Hong Yao, Howard J. Hamilton, and Cory J. Butz. 2002. FD_Mine: Discovering Functional Dependencies in a Database Using Equivalences. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan.* 729–732.

[50] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. 2018. GAIN: Missing Data Imputation using Generative Adversarial Nets. In *Proceedings of the 35th International Conference on Machine Learning, ICML.* 5675–5684.

[51] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. 2019. Free-form image inpainting with gated convolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 4471–4480.

[52] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence.* 2852–2858.