

# GIDCL: A Graph-Enhanced Interpretable Data Cleaning Framework with Large Language Models

MENGYI YAN, School of Computer Science and Engineering, Beihang University, China

YAOSHU WANG\*, Shenzhen Institute of Computing Sciences, China, and Shenzhen University, China

YUE WANG, Shenzhen Institute of Computing Sciences, China

XIAOYE MIAO, Center for Data Science, Zhejiang University, China

JIANXIN LI, School of Computer Science and Engineering, Beihang University, China

Data quality is critical across many applications. The utility of data is undermined by various errors, making rigorous data cleaning a necessity. Traditional data cleaning systems depend heavily on predefined rules and constraints, which necessitate significant domain knowledge and manual effort. Moreover, while configuration-free approaches and deep learning methods have been explored, they struggle with complex error patterns, lacking interpretability, requiring extensive feature engineering or labeled data. This paper introduces GIDCL (Graph-enhanced Interpretable Data Cleaning with Large language models), a pioneering framework that harnesses the capabilities of Large Language Models (LLMs) alongside Graph Neural Network (GNN) to address the challenges of traditional and machine learning-based data cleaning methods. By converting relational tables into graph structures, GIDCL utilizes GNN to effectively capture and leverage structural correlations among data, enhancing the model's ability to understand and rectify complex dependencies and errors. The framework's creator-critic workflow innovatively employs LLMs to automatically generate interpretable data cleaning rules and tailor feature engineering with minimal labeled data. This process includes the iterative refinement of error detection and correction models through few-shot learning, significantly reducing the need for extensive manual configuration. GIDCL not only improves the precision and efficiency of data cleaning but also enhances its interpretability, making it accessible and practical for non-expert users. Our extensive experiments demonstrate that GIDCL significantly outperforms existing methods, improving F1-scores by 10% on average while requiring only 20 labeled tuples. The codes, datasets and full version of the paper are available [2].

CCS Concepts: • **Information systems** → **Data cleaning**.

Additional Key Words and Phrases: Data quality, Large Language Models, Graph Neural Network, Interpretable

## 1 Introduction

Data serves as a cornerstone in multiple applications, *e.g.*, data analysis, fault detection, recommendation systems, and so on, but its utility is contingent upon its quality. Real-life data often has various errors, rendering it 'dirty' and necessitating rigorous cleaning processes. Data cleaning (DC), despite its critical importance, is a time-intensive and labor-intensive task for data scientists. DC

\*Corresponding author

---

Authors' Contact Information: Mengyi Yan, School of Computer Science and Engineering, Beihang University, China, yanmy@act.buaa.edu.cn; Yaoshu Wang, Shenzhen Institute of Computing Sciences, China, and Shenzhen University, China, yaoshuw@sics.ac.cn; Yue Wang, Shenzhen Institute of Computing Sciences, China, ywangby@connect.ust.hk; Xiaoye Miao, Center for Data Science, Zhejiang University, China, miaoxy@zju.edu.cn; Jianxin Li, School of Computer Science and Engineering, Beihang University, China, lijx@buaa.edu.cn.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM 2836-6573/2024/12-ART236

<https://doi.org/10.1145/3698811>

task often consists of two major parts, error detection and error correction, where error detection identifies dirty cells [4], and error correction fixes these dirty ones to correct values [89].

Traditional DC systems follow the pre-configuration paradigm [89], where users have to provide different types of rules or constraints, such as functional dependencies [71], denial constraints [19], conditional functional dependencies [35], and so forth. For most non-expert users, this presents a significant barrier, as they must have prior knowledge of both the dataset and the DC system to configure the rules properly [4].

Meanwhile, multiple configuration-free DC approaches, *e.g.*, [50, 70, 104], have been proposed. The non-expert users only need to provide a few labeled examples, which contain errors and corresponding error corrections, and then these methods could automatically learn to generalize these error detection and correction strategies to unseen data. Configuration-free approaches suffer from finding correct values for error data in large search space. To tackle it, several works refine the search space, such as retrieving and ranking suitable values within the dataset itself [89, 103] or searching external data sources [20]. However, these configuration-free DC methods still heavily rely on predefined feature engineering and often struggle with undefined error patterns [68, 70]. As a result, they are unable to adapt the fixed feature engineering to address various datasets and error types. Furthermore, they are ineffective against complex textual errors when the correct values do not exist within the dataset [79, 103].

Recently, the use of deep learning models, especially Transformer based language models, shed lights on DC task, which detect and repair dirty data by learning the real data distribution. RPT [96] and TURL [25] employ encoder-decoder architectures and are pre-trained in a tuple-to-tuple fashion by corrupting the input tuple and then learning to reconstruct the original tuple. However, deep learning-based methods typically lack interpretability that are data cleaning patterns, rules and dependencies that can be explicitly verified by human beings [79] for both error detection and correction, and usually require large amount of clean and well-annotated training data. For example, TURL used a collection of web tables of 4.6GB in total for pre-training, in order to extract values for imputing the missing value. This is because learning on dirty datasets cannot guarantee correctness and may lead to new errors [81].

To summarize current works, pre-configuration DC methods that adopt data quality rules are highly interpretable, but these rules are difficult to generate or handcraft without domain knowledge. Configuration-free methods usually apply pruning strategies or dedicate feature engineering along with traditional ML models for error detection and correction, but they are difficult to handle complex scenarios such as undefined error pattern and complex textual errors, and heavily rely on pre-defined feature engineering. Deep learning-based DC methods can learn data distribution automatically, but they require a lot of high-quality labelling data, and typically lack of interpretability, which is crucial in the data systems of healthcare, finance and banking, and the government and public sectors.

Large language models (LLMs) [14, 97], which typically contain billions (or more) of parameters, and pre-trained on massive text data [94], have demonstrated surprising emergent behaviors and good zero-shot generalization to new tasks. Such effectiveness can be largely attributed to several inherent characteristics, including (1) **follow natural language instructions**; (2) **utilize few-shot prompting**, which involves providing LLM with a small number of example tasks to effectively adopt to similar new tasks; (3) leverage its **rich prior knowledge**, which is encoded into its parameters.

Given the considerable potential benefits of LLMs, this raises a fundamental question: Can we effectively integrate LLMs into a data cleaning framework that can address previously mentioned challenges systematically? This question remains unanswered. Besides, directly applying LLMs to the data cleaning task, without a carefully designed framework or mechanism, introduces several

substantial obstacles (details in Section 2.2): (1) LLMs face challenges comprehending data quality rules and dependencies across relational tables due to token limitations, resulting in inconsistent data cleaning outcomes. (2) The difficulty in understanding complex dependencies can lead LLMs to generate plausible yet incorrect or fictional data repairs, a phenomenon known as hallucination. (3) When fine-tuned on limited labeling data, LLMs tend to overfit, performing well on familiar data but poorly on unseen datasets. (4) The substantial size and complexity of LLMs compromise efficiency, making it impractical to apply data cleaning to all tuples sequentially. Currently, there remains a notable gap in systematic research dedicated to integrating LLMs into a data cleaning framework capable of concurrently and accurately solving error detection and correction issues. This integration seeks to overcome the previously mentioned challenges and limitations inherent in existing data cleaning solutions.

In this paper, we propose a Graph-enhanced Interpretable Data Cleaning with Large language models, denoted by GIDCL for short, that achieves both high precision and recall. GIDCL is a self-supervised learning method that unifies graph neural networks (GNN), pre-trained language models (PLMs) and large language models (LLMs) into an end-to-end workflow (shown in Figure 2) that could process the overall data cleaning procedure, as well as generating interpretable DC patterns.

GIDCL provides a systematic mechanism to incorporate LLMs in DC workflow, which contains several components working together to address the challenge of applying LLM for DC and limitations of existing works. First, we transform relational tables into graph structures, and capture the structural correlations among tuples and attributes with GNN. Second, we introduce a creator-critic workflow that involves prompting LLMs and fine-tuning PLMs for error detection, allowing iterative refinement of the detection model using few-shot labeled samples. Third, we utilize a graph-enhanced error correction approach that leverages both LLMs and GNN to generate reliable corrections efficiently. The integration of GNN and PLMs effectively assists LLMs in perceiving structural information and retrieving relevant context, thereby enhancing efficiency and suppressing hallucination in GIDCL.

By leveraging the rich prior knowledge of LLMs, GIDCL can automatically extract and generate various data cleaning rules using natural language for different scenarios, effectively overcoming the challenges posed by traditional DC methods that rely on handcrafted rules. GIDCL achieves robust error detection and correction in complex scenarios. Its capacity to follow natural language instructions and utilize few-shot prompting to automatically customize feature engineering is friendly to non-expert users, and directly addresses the significant limitations of configuration-free methods, which only rely on the predefined feature engineering in their algorithms to solve different problems. Furthermore, GIDCL requires only a minimal amount of labeled data (*e.g.*, 20 tuples) compared to existing deep learning-based methods. This benefit arises from the few-shot prompting ability of LLMs, enabling the model to quickly adapt to new tasks with limited demonstrations. By implicitly mastering the semantics and structural dependency through in-context learning, GIDCL achieves both highly effectiveness and efficiency, while extracting and generating interpretable DC patterns and dependencies with LLM. Our contributions are:

- (1) **An end-to-end framework for data cleaning.** We introduce GIDCL, an end-to-end data cleaning framework that automatically handles user labeling, error detection and correction in a reliable manner with high recall and precision. To the best of our knowledge, this is the first systematic effort to develop a data cleaning (DC) framework specialized in integrating LLMs. (Section 3)
- (2) **A creator-critic workflow for error detection.** To automatically extract reliable error detection patterns and optimize error detection model from few-shot labeled samples, we propose a LLM-enhanced creator-critic workflow for error detection, which iteratively refines an error

Table 1. Instances of a Hospital dirty table. Erroneous cells are marked in red color and the correction value is marked in blue.

TupleID	ProviderID	City	State	Zip	County
$t_1$	111303	Monticello	VA, Jasper → VA	31064	Jasper
$t_2$	111303	Monticello	VAA → VA	31064	Jasper
$t_3$	1x1303 → 111303	Monticello	AR	71655	Drew
$t_4$	10001	Monticello → Dothan	AL	36301	Houston
$t_5$	10001	Dothan	AL, Houston → AL	36301	NULL → Houston
$t_6$	10001	Dothan	AR → AL	NULL → 36301	Houston
$t_7$	10001	Dothan	AL	36301	Houst → Houston

detection model and a LLMs-generated pattern set. (Section 4, Section 5)

- (3) **Error correction based on LLMs.** We propose a retrieval-augmented paradigm for fine-tuning local LLMs in order to generate reliable corrections with both high effectiveness and efficiency. Furthermore, considering that LLMs are not very sensitive to dependency errors, we designed a graph structure learning method that learns the structural information of datasets. (Section 6)

## 2 Background

In this section we formally present the data cleaning (DC) task.

### 2.1 Data Cleaning

The data cleaning over a relational table is a process that identifies and repairs erroneous data with the correct values with a few annotated labels. Denote a relational table by  $\mathcal{T} = \{t_1, t_2, \dots, t_{|\mathcal{T}|}\}$ , where  $|\mathcal{T}|$  represents its size. The relational schema of this table is given by  $\mathcal{A} = \{A_1, A_2, \dots, A_n\}$ . Each element  $t_i$  represents a  $n$ -attribute tuple in  $\mathcal{T}$ , and  $t_{i,j}$  stands for the value of attribute  $A_j$  in  $t_i$ . Correspondingly,  $t_{i,j}^+$  represents the clean value of  $t_{i,j}$ , and  $\mathcal{T}^+$  is the ground truth of table  $\mathcal{T}$ . An error occurs when a cell value  $t_{i,j}$  in  $\mathcal{T}$  deviates from its ground truth  $t_{i,j}^+$ , i.e.,  $t_{i,j}^+ \neq t_{i,j}$ . In alignment with previous studies [68, 79, 86], our objective encompasses the detection and correction of both syntactic and semantic errors. These errors encompass missing values, typographical errors, formatting issues, violations of functional dependencies, and so on.

**Problem statement.** Given a dirty relational table  $\mathcal{T}$  and a limited labeling budget  $\theta$ , where users are only able to label at most  $\theta$  tuples, our objective is to cleanse the table  $\mathcal{T}$ , aiming to identify and rectify all errors in  $\mathcal{T}$ . Here we denote the set of labeled tuples by  $\mathcal{T}_{\text{label}} = \{(t_i, t_i^+) | t_i \in \mathcal{T}\}$ , where  $t_i^+$  is a tuple of all clean attributes.

**Example 1:** Consider a relational table in Table 1 that consists of 7 tuples with 6 attributes. There are many erroneous cells (marked with blue) in the table, e.g., Provided ID of  $t_3$ , City of  $t_4$  and State of  $t_2$ . The data cleaning task is to identify these cells and replace them with correct values (marked in red), e.g., revising City of  $t_4$  by Dothan, and imputing Zip of  $t_6$  by 36301.  $\square$

The data cleaning task is non-trivial [68, 70] for the following reasons. First, the labeling budget  $\theta$  is typically very limited due to the expensive user labeling cost, thereby posing a challenging in generalizing them across all instances, e.g., ML-based methods might overfit to the training data. Second, errors detected in the error detection phase might propagate to the error correction phase. If an error remains unidentified, it will never be repaired. Lastly, data cleaning involves addressing various types of errors, such as inconsistencies, missing values, and typos. Consequently accurately repairing erroneous cells with correct values is non-trivial.

### 2.2 Challenges of applying LLMs for DC

In this subsection, we first outline the challenges associated with applying LLMs to data cleaning, and then demonstrate why direct adoption is impractical. The summarized challenges include:

Table 2. General notations with corresponding descriptions.

Symbol	Description
$\mathcal{T}$	a relational table provided as input for cleaning
$\mathcal{G}$	the directed graph constructed from $\mathcal{T}$ for graph structure learning
$t_i, t_{i,j}$	the $i$ -th tuple in $\mathcal{T}$ / the $j$ -th attribute in $t_i$
$h_i$	the center node in $\mathcal{G}$ , corresponding to $t_i$
$A_i \in \mathcal{A}$	the $i$ -th attribute in $\mathcal{T}$ /all attribute in $\mathcal{T}$
$\mathcal{T}^+, t_i^+, t_{i,j}^+$	the clean versions of table $\mathcal{T}$ , tuple $t_i$ and cell $t_{i,j}$
$f_{A_j}^{\text{det}} \in \mathcal{F}^{\text{det}}$	the error detection function for the $j$ -th attribute
$f_{A_j}^{\text{gen}} \in \mathcal{F}^{\text{gen}}$	the error generation function for the $j$ -th attribute
$f_{A_j}^{\text{corr}} \in \mathcal{F}^{\text{corr}}$	the error correction function for the $j$ -th attribute
$\mathcal{F}$	$\mathcal{F} = \mathcal{F}^{\text{det}} \cup \mathcal{F}^{\text{corr}} \cup \mathcal{F}^{\text{gen}}$
$C_i$	$i$ -th cluster in $\mathcal{G}$ , divided by GSL
$\mathcal{M}_{\text{det}}$	error detection model
$\mathcal{M}_{\text{corr}}$	error correction model
$\mathcal{T}_{\text{label}}$	labelled tuples in $\mathcal{T}$ by users, s.t., $\mathcal{T}_{\text{label}} = \{(t_i, t_i^+)   t_i \in \mathcal{T}\}$
$\mathcal{T}_{\text{pseudo}}$	self-generated tuples in $\mathcal{T}$ , labelling by GIDCL
$\mathcal{T}_{\text{coreset}}$	coreset in $\mathcal{T}$ , generated by error detection model
$\mathcal{T}_{\text{err}}$	detected erroneous cells in $\mathcal{T}$
$D^{\text{train}}$	training data for $\mathcal{M}_{\text{det}}, \mathcal{M}_{\text{corr}}$
$\mathcal{O}$	the domain of all objects in $\mathcal{T}$

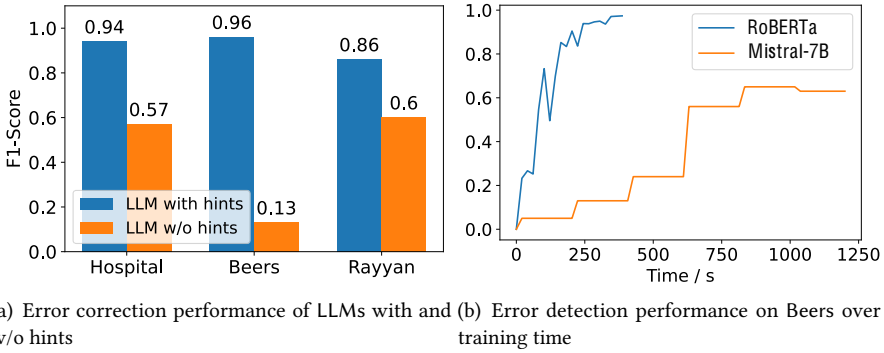


Fig. 1. Observations of LLMs for data cleaning

(1) **Understanding dependencies:** due to token limitation (the maximum input length, e.g., 4k tokens for GPT-3.5), it is impractical to feed entire relational tables into LLMs. This restriction severely impedes the models' ability to grasp comprehensive data quality rules and inherent dependencies, leading to inconsistent data cleaning outcomes across different instances. (2) **Hallucination:** The absence of necessary contextual information and adequate demonstrations may cause LLMs to generate plausible yet incorrect or fictional data repairs. This phenomenon occurs as the models fill gaps in their understanding with erroneous or fabricated information. (3) **Overfitting with limited labeled data:** When fine-tuned on limited labeled data, LLMs are susceptible to overfitting. This issue manifests as the model performing well on the training data but poorly on unseen data, limiting its generalizability. (4) **Efficiency issues:** The considerable size and complexity of LLMs pose efficiency challenges, making it impractical to apply data cleaning processes to all tuples individually and sequentially. This inefficiency is exacerbated in large-scale data environments.

A straightforward approach to implementing LLMs for the data cleaning task is as follows.

(a) **Error detection  $\mathcal{M}_{\text{det}}$ .** By comparing each pair of labeled cells  $(t_{i,j}, t_{i,j}^+) \in \mathcal{T}_{\text{label}}$ , one could design

a training pair  $(x_{i,j}, y_{i,j})$  for detecting attribute  $A_j$  for a tuple  $t_i$ , and fine-tune a LLM-based error detection model  $\mathcal{M}_{\text{det}}$ . Here  $x_{i,j} = (p_{\text{det}}^{A_j}, \text{serial}(t_i), t_{i,j})$ ,  $p_{\text{det}}^{A_j}$  is an instruction to identify errors in  $A_j$ , and  $\text{serial}(t_i)$  represents the serialization of  $t_i$  as row context of cell  $t_{i,j}$ . The label  $y_{i,j}$  is True if  $t_{i,j} = t_{i,j}^+$ , indicating no error, and False otherwise. During training, all user-labeled pairs  $(x_{i,j}, y_{i,j})$  are provided to fine-tune  $\mathcal{M}_{\text{det}}$  employing the supervised fine-tuning strategy. In the inference procedure, a cell  $t_{i,j}$  is first transformed into  $x_{i,j}$  and input to  $\mathcal{M}_{\text{det}}$  for prediction.  $\mathcal{M}_{\text{det}}$  operates as a binary classification model.

*(b) Error correction  $\mathcal{M}_{\text{corr}}$ .* Analogous to error detection, one constructs  $(x_{i,j}, y_{i,j}) \in \mathcal{T}_{\text{label}}$  to repair  $t_{i,j}$  using a LLM-based correction model  $\mathcal{M}_{\text{corr}}$ . Here  $x_{i,j} = (p_{\text{corr}}^{A_j}, \text{serial}(t_i), t_{i,j})$ ,  $y_{i,j} = t_{i,j}^+$ ,  $p_{\text{corr}}^{A_j}$  represents an instruction for error correction to generate a recommended fix for  $t_{i,j}$ . The training and inference procedure of  $\mathcal{M}_{\text{corr}}$  are similar with above  $\mathcal{M}_{\text{det}}$ .  $\mathcal{M}_{\text{corr}}$  operates as a generative model.

Following this paradigm, we elucidate our findings regarding efficiency and effectiveness.

*Efficiency.* Fine-tuning and inference with LLM are inherently time-consuming. Consequently, it is impractical to utilize LLM to identify and rectify a large number of cells in tables, as these models require making predictions for each cell individually. We conduct experiments and found error detection and correction using LLM for 10k cells take up to 10,330 and 24,114 seconds on consumer-level GPUs.

*Effectiveness.* We have made the following observations about applying LLM directly for DC: (1) When acting as error detection model, due to the decoder-only generative manner and huge number of parameters, LLMs **exhibit significantly longer convergence time and overfitting issue when trained with limited labelling data, compared to non-LLM models**. See Figure 1(b), LLMs (Mistral-7B) require a minimum of 50 epochs to converge, yet their F-measure remains inferior to that of a Transformer-based error detection model (RoBERTa) [109]. (2) **As a generative model for error correction, it is hard for LLMs to repair data without any context or dependencies**. As depicted in Figure 1(a), we repair  $t[A_1]$  using  $\mathcal{M}_{\text{corr}}$  without additional dependencies and context information, and the F-measure of LLMs is as low as 0.13. This indicates that LLMs may not accurately identify the correct value for  $A_1$ , primarily due to the hallucination issue. Besides, the limitation on the input length of LLMs makes it difficult for them to read the entire content of a relational table, and find a high-fidelity coreset to guide the correction of dependency violations.

### 3 A Data Cleaning Framework

In this section, we introduce our end-to-end data cleaning framework GIDCL for detecting and repairing all errors in a relational table with three main components, including (a) a graph structure learning model that captures the correlation among tuples and attributes for manual labeling and clustering; (b) a creator-critic workflow that involves prompting LLMs and fine-tuning PLMs for error detection, and (c) a graph-enhanced error correction based on LLMs and graph neural network.

We define  $\mathcal{T}_{\text{coreset}}$  as a set of cells in  $\mathcal{T}$  identified as clean ones with high confidences, and  $f_i^{\text{det}}$  as an error detection function for the  $i$ -th attributes to detect whether their values are erroneous.

**Architecture.** The ultimate goal of GIDCL is to identify and repair erroneous cells in  $\mathcal{T}$ . As depicted in Figure 2, GIDCL initializes the process by transforming  $\mathcal{T}$  into a directed attribute graph  $\mathcal{G}$ , facilitating the clustering of semantic and structural similar tuples through a graph structure learning strategy. Subsequently, GIDCL recommends a maximum of  $\theta$  representative tuples to users for manual labeling. Following this, a creator-critic workflow is invoked, which harmonizes

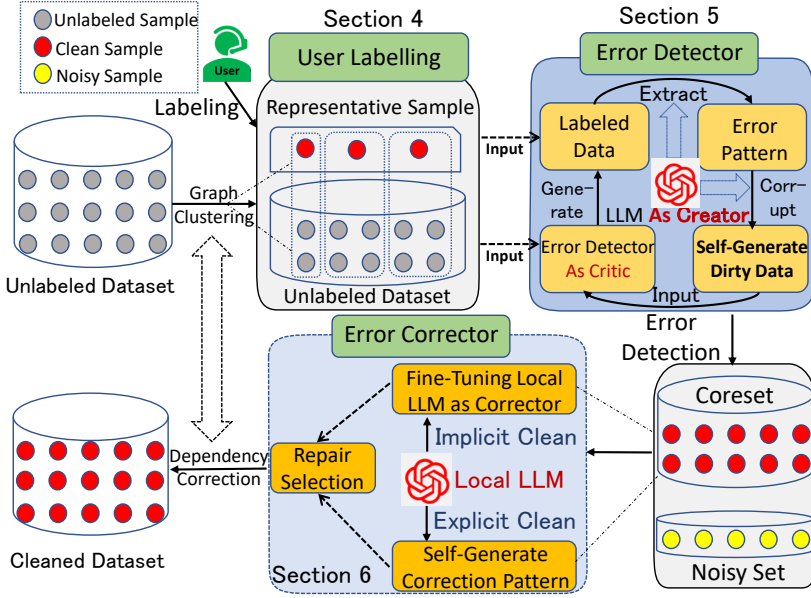


Fig. 2. Overview of GIDCL

the capabilities of LLMs and PLMs to enhance the performance for error detection while ensuring relatively fast execution time. Finally, leveraging the structural information gleaned by GNNs, GIDCL augments LLMs to correct errors in a retrieval-augmented generation manner, utilizing representative examples as in-context demonstrations.

More specifically, GIDCL consists of three phases, including (1) graph structure learning for manual labeling; (2) a creator-critic workflow for error detection, and (3) a fine-tuned graph-enhanced LLMs for error correction.

(1) Graph structure learning GSL. In this phase, GSL trains a graph neural network GNN to learn the structural information of  $\mathcal{T}$ . Formally, the input and output of GSL are as follows.

- *Input:*  $\mathcal{T}$ , the number of clusters  $k$ , and the labeling budgets  $\theta$ .
- *Output:*  $\theta$  representative tuples that need to be labeled by users, a directed attribute graph  $\mathcal{G}$  constructed from  $\mathcal{T}$ , cluster division  $\mathcal{C}$  learned from  $\mathcal{G}$ .

By taking  $\mathcal{T}$  as the input, GSL partitions tuples  $t \in \mathcal{T}$  into  $k$  clusters according to their similarities in the Euclidean space and picks up  $\theta$  tuples for user labelling. Notably, we first transform  $\mathcal{T}$  to a graph  $\mathcal{G}$  and apply GSL to obtain an embedding for each tuple, ensuring similar tuples are grouped into the same clusters; then a novel selection strategy is designed to pick up  $\theta$  representative tuples that requires labelling by users.

(2) Error detection GIDCL<sub>det</sub>. The error detection component in GIDCL named GIDCL<sub>det</sub> employs a creator-critic workflow to identify potential errors by combining the capabilities of PLMs and LLMs. The formal input and output are as follows.

- *Input:*  $\mathcal{T}$ , a set  $\mathcal{T}_{\text{label}}$  of  $\theta$  user-labeled tuples that are clean.
- *Output:* A set of cells  $\mathcal{T}_{\text{err}}$  identified as erroneous ones, the coreset  $\mathcal{T}_{\text{coreset}}$  identified as clean cells with high confidences, and self-generated training data  $\mathcal{T}_{\text{pseudo}}$ .

(i) Critic of GIDCL<sub>det</sub>. GIDCL incrementally fine-tunes a small PLM model to predict whether each

cell in  $\mathcal{T}$  is erroneous or not, acting as critic. By utilizing  $\mathcal{T}_{\text{label}}$  as training instances and set of *pseudo clean* tuples  $\mathcal{T}_{\text{pseudo}}$  generated by LLMs as inputs, we fine-tune a classifier  $\mathcal{M}_{\text{det}}$  over a few epochs. Following fine-tuning,  $\mathcal{M}_{\text{det}}$  scrutinizes all cells in  $\mathcal{T}$  and outputs the result  $(\mathcal{T}_{\text{err}}, \text{Conf})$ , where  $\mathcal{T}_{\text{err}} = \{t_{i,j} | \mathcal{M}_{\text{det}}(t_{i,j}) = \text{True}\}$  is the set of all erroneous cells in  $\mathcal{T}$ , and  $\text{Conf}$  records confidence scores of all cells as inferred by  $\mathcal{M}_{\text{det}}$ , e.g.,  $\text{Conf}(t_{i,j})$  represents the confidence level of decision inferred by  $\mathcal{M}_{\text{det}}(t_{i,j})$ , determined as the maximum value of the Softmax layer. Then the critic will gradually provide dirty and clean samples to LLM-based creator. The input and output of critic are as follows.

- *Input:*  $\mathcal{T}_{\text{label}}, \mathcal{T}_{\text{pseudo}}$  and  $\mathcal{T}$ .
- *Output:* a set of prediction results of  $(\mathcal{T}_{\text{err}}, \text{Conf}) \subset \mathcal{T}$  and  $\mathcal{M}_{\text{det}}$ .

(ii) Creator of  $\text{GIDCL}_{\text{det}}$ . Because the detection model  $\mathcal{M}_{\text{det}}$  is susceptible to overfitting when fine-tuned with few-shot  $\mathcal{T}_{\text{label}}$ , the creator is applied to generate(create) additional training data. This is achieved through a two-step data augmentation strategy involving iterative utilization LLMs with various prompts.

In the first step, we gather both dirty and clean cells  $\{t_{i,j}, t_{i,j}^+\}$  from  $\mathcal{T}_{\text{label}}$  and the prediction results of  $\mathcal{M}_{\text{det}}$ , then prompt a LLM to generate a transformation function  $f_{A_i}^{\text{det}}$  for each attribute  $A_i \in \mathcal{A}$ . The creator accepts  $f_{A_i}^{\text{det}}$  if the F-measure of  $\mathcal{T}_{\text{label}}$  after executing it for error detection surpasses a predefined threshold  $\tau$ . we denote  $\mathcal{F}^{\text{det}} = \{f_{A_1}^{\text{det}}, \dots, f_{A_n}^{\text{det}}\}$  as the detection function set.

In the second step, utilizing the accepted  $f_{A_i}^{\text{det}}$ , a generation function  $f_{A_i}^{\text{gen}}$  is generated by LLM, corrupting clean cells to dirty ones according to  $f_{A_i}^{\text{det}}$ . Specifically, given a cell  $t_{j,i}^+$  predicted as correct by  $\mathcal{M}_{\text{det}}$ , its dirty value is generated as  $t_{j,i} = f_{A_i}^{\text{gen}}(t_{j,i}^+)$ . Finally the creator samples a few correct cells in attribute  $A_i$ , and corrupt them to the dirty ones via  $f_{A_i}^{\text{gen}}$ , thereby providing  $\mathcal{M}_{\text{det}}$  with additional training data  $\mathcal{T}_{\text{pseudo}}$  to mitigate potential overfitting issue.

We have the formal input and output of the critic as follows.

- *Input:*  $\mathcal{T}_{\text{label}}$ , prediction result  $(\mathcal{T}_{\text{err}}, \text{Conf})$  by  $\mathcal{M}_{\text{det}}$ .
- *Output:* augmented data  $\mathcal{T}_{\text{pseudo}}$ , detection function set  $\mathcal{F}^{\text{det}}$ .

The interaction of the creator and critic iteratively processes until the creator cannot refine  $f_{A_i}^{\text{det}}$  for all attributes. In other word, the termination condition is either (1) no  $f_{A_i}^{\text{det}}$  would correctly predict  $\mathcal{T}_{\text{label}}$  with at least  $\tau$  F-measure, or (2) the set of transformation function is the same as the previous one. When the creator-critic iteration is terminated, we apply  $\mathcal{M}_{\text{det}}$  over the whole table  $\mathcal{T}$ , and predict the error cells  $\mathcal{T}_{\text{err}}$ , while the reliable coreset  $\mathcal{T}_{\text{coreset}} \in \mathcal{T}$  is also returned. We denote the labeled data  $D^{\text{train}} = \mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}$ .

(3) Error correction  $\text{GIDCL}_{\text{cor}}$ . In this phase, regarding the detected error cells  $\mathcal{T}_{\text{err}}$ , the error correction component  $\text{GIDCL}_{\text{cor}}$  implicitly fine-tune a LLM-based correction model  $\mathcal{M}_{\text{corr}}$ , to directly generate correction values; and explicitly prompting LLM to extract the correction pattern for repairing errors. Assembling the above corrections,  $\text{GIDCL}_{\text{cor}}$  further updates GSL for repairing dependency violations. The formal input and output are as follows.

- *Input:* labeled data  $\mathcal{T}_{\text{label}}$ , error cells  $\mathcal{T}_{\text{err}}$  to be repaired, clean cells  $\mathcal{T}_{\text{coreset}}$ , error detection model  $\mathcal{M}_{\text{det}}$  and function set  $\mathcal{F}^{\text{det}}$ .
- *Output:* the cleaned table  $\mathcal{T}_{\text{clean}}$  by repairing all errors in  $\mathcal{T}_{\text{err}}$ .

(i) LLM-based correction model. We enhance the LLM-based correction model  $\mathcal{M}_{\text{corr}}$  through a combination of in-context learning and supervised fine-tuning. This approach refines the model to accurately generate correct data cells when provided with their erroneous versions and relevant



contextual information. The contextual information includes labeled examples from  $\mathcal{T}_{\text{label}}$  and high-quality ones from  $\mathcal{T}_{\text{pseudo}}$  generated by GIDCL<sub>det</sub>. The formal input and output are as follows.

- *Input*: labeled data  $\mathcal{T}_{\text{label}}$ , clean cells  $\mathcal{T}_{\text{coreset}}$ ,  $\mathcal{T}_{\text{pseudo}}$ .
- *Output*: the generative correction model  $\mathcal{M}_{\text{corr}}$ .

(ii) LLM-generated correction function  $f_{A_i}^{\text{corr}}$ . We also design explicit function for error correction. By taking  $\mathcal{T}_{\text{label}}$  as input, we query LLM to return a correction function  $f_{A_i}^{\text{corr}}$  for  $A_i$ , referencing the accepted detection function  $f_{A_i}^{\text{det}}$ . Considering that one cell could be repaired by both of explicit and implicit functions, we treat  $\mathcal{M}_{\text{det}}$  as a ranking model to select the most suitable one.

- *Input*: labeled data  $\mathcal{T}_{\text{label}}$ , error cells  $\mathcal{T}_{\text{err}}$  to be repaired, detection module GIDCL<sub>det</sub>, correction model  $\mathcal{M}_{\text{corr}}$ .
- *Output*: repaired table  $\mathcal{T}_{\text{clean}}$ , correction function set  $\mathcal{F}^{\text{corr}}$ .

(iii) Considering inconsistencies might exist in data, we further re-train the embeddings of tuples in  $\mathcal{T}$  after  $\mathcal{T}$  is cleaned by LLMs as  $\mathcal{T}_{\text{clean}}$ , by updating  $\mathcal{G}$  and GSL. According to the correlations among detected clean data  $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{coreset}}$ , we discover a few functional dependencies FDs, and apply them as the final step to clean the remaining dependency errors in  $\mathcal{T}_{\text{clean}}$ .

- *Input*: repaired table  $\mathcal{T}_{\text{clean}}$ , graph  $\mathcal{G}$
- *Output*: cleaned table  $\mathcal{T}'_{\text{clean}}$ , discovered dependencies FDs.

In a word, GIDCL takes a relational table  $\mathcal{T}$  and up to  $\theta$  user-labeled data  $\mathcal{T}_{\text{label}}$  as input, and output the cleaned table  $\mathcal{T}'_{\text{clean}}$ , with a set of interpretable patterns, containing error detection functions  $\mathcal{F}^{\text{det}}$ , error correction functions  $\mathcal{F}^{\text{corr}}$  and functional dependencies FDs for further user verification.

#### 4 Graph Structure Learning for labeling

In this section, we first develop a graph structure learning approach to learn representations of tuples in  $\mathcal{T}$  in an unsupervised manner and cluster them into  $k$  groups. Next we propose a simple but effective tuple selection strategy to select  $\theta$  representative tuples for users to manually label.

**Graph construction.** We transform  $\mathcal{T}$  into a directed attribute graph  $\mathcal{G} = (V, E, L)$ , such that each edge  $e \in E$  is represented as a triplet  $e = (t_i, A_j, t_{i,j})$ , where  $t_i$ ,  $A_j$  and  $t_{i,j}$  is the  $i$ -th tuple of  $\mathcal{T}$ , the  $j$ -th attribute of  $\mathcal{A}$  and the value of the cell in  $\mathcal{T}[i, j]$ , respectively. Each vertices  $v \in V$  and edge  $e \in E$  contains attributes, denoted as  $v.a = t_i$ ,  $e.a = A_j$  respectively. We treat  $\mathcal{G}$  as multi-relational graph[92, 99], which structure is similar with knowledge graph.

**Example 2:** Consider tuples  $t_1, t_2$  with ProviderID, City, County in Table 1 as an example. 5 vertices and 6 edges are created to construct a graph, such that  $V = \{t_1, t_2, \text{Monticello}, 111303, \text{Jasper}\}$  and  $E = \{(t_1, \text{City}, \text{Monticello}), (t_2, \text{City}, \text{Monticello}), (t_1, \text{ProviderID}, 111303), (t_2, \text{ProviderID}, 111303), (t_1, \text{County}, \text{Jasper}), (t_2, \text{County}, \text{Jasper})\}$ . □

Inspired by [98], we design a value function  $y$  to measure whether each possible triple  $e$  exists in  $\mathcal{G}$ , i.e.,  $y(e) = 1$ , or not, i.e.,  $y(e) = -1$ . Our main goal is to learn  $f$  such that higher  $f(e)$  indicates higher probability that  $e$  exists.

**Graph representation learning.** To learn the scoring function  $f(e)$ , we adopt the Knowledge Graph Embedding approach (KGE), which maps the vertices in  $V$  into continuous low-dimensional vectors while preserving their semantic meanings. If two tuples in  $\mathcal{T}$  has similar structural information, their embeddings in the hidden space should be close with each other. We conduct the following pipeline to learn these vectors.

(a) *Training data construction.* Besides the real triples in  $E$ , we automatically generate  $m$  fake triples by corrupting  $h$  and  $t$  in each triple  $e = (h, r, t)$  by using some heuristic corruption strategies [92, 98] for each real triple. After corrupting all triplets in  $E$ , we generate fake tuples and create the training set  $\mathcal{G}_{\text{train}}$ , such that  $y(e) = 1$  for true triples  $e$  and  $-1$  for fake ones. We set  $m$  as 1 by default, following the setting of existing work [92].

(b) *Initialization of embeddings.* We adopt a PLM SentenceBert [88] to generate initial embeddings of all vertices and edges in  $\mathcal{G}$ , such that the latent semantic correlations are more likely to be maintained and noises in  $\mathcal{T}$  are tolerated. More specifically, for each vertex or edge, denoted by  $u$ , we first serialize it to a sequence  $\text{serial}(u)$  and then adopt SentenceBert with pre-trained model bge-large-en to transform it to a high-dimensional embedding, s.t.,  $\mathbf{u} = \text{SentenceBert}(\text{serial}(u))$ , where  $\mathbf{u}$  is the initial embedding of  $u$ .

(c) *Self-supervised learning.* We apply ComplEx [98] as scoring function  $f(e)$ , with GNN-style method CompGCN [99] to learn embedded vectors for all vertices in  $V$ . Loss function is defined as:

$$f(t) = \Re(\langle \mathbf{w}_r, \mathbf{e}_h, \overline{\mathbf{e}_t} \rangle) \\ \text{Loss}(\Theta) = -\frac{1}{(1+m)|\Theta|} \sum_{(h,r,t) \in E} y \log l(f(h, r, t)) + (1-y) \log(1-l(f(h, r, t))) \quad (1)$$

where  $\mathbf{e}_h$  and  $\mathbf{e}_t$  are embedding vectors of vertices  $h$  and  $t$ ,  $\mathbf{w}_r$  is a relation parameter,  $\Re$  is the real part of a complex vector,  $\langle \cdot \rangle$  is the trilinear product of  $(h, r, t)$  embedding vectors, and  $\overline{\mathbf{e}_t}$  is the complex conjugate of vector  $\mathbf{e}_t$ .  $\Theta$  is a set of embeddings of all vertices and edges in  $G$ , and  $l$  is the logistic sigmoid function. We use CompGCN because it can incorporate multi-relational information with GNN via node aggregation, and leverage the advantage of KGE techniques for better understanding of high-dimensional dependencies in  $\mathcal{T}$  across different tuples and attributes.

**Representative tuple selection.** We first formalize our objective function and then give our solution to select representation tuples.

*Objective function.* We aim to find  $\theta$  tuples that are inclined to be erroneous and their errors are representative. We formulate the representative tuple selection as the bilevel optimization problem.

$$\min_{T \subseteq D, |T|=\theta} \left( \sum_{t \in T} \sum_{p \in \mathcal{B}_t} \text{dist}(t, p) \right) \text{ s.t. } \min_{\mathcal{B}} \left( \sum_{k=1}^K \sum_{t_i, t_j \in \mathcal{B}_k} \text{dist}(t_i, t_j) \right) \quad (2)$$

where  $\mathcal{B} = \{\mathcal{B}_1, \dots, \mathcal{B}_K\}$  are  $K$  non-overlapped partitions.

*Solution.* Following the acquisition of embeddings for all tuples, we first adopt the clustering strategy to partition  $\mathcal{T}$  into  $K$  partitions, and then separately select the outlier tuples from these partitions. More specifically, we employ the straightforward  $k$ -means clustering technique to partition  $\mathcal{T}$  into  $k$  groups  $C = \{C_1, C_2, \dots, C_k\}$ . To generate training data for error detection and correction, GIDCL presents the top- $\theta$  most *ambiguous* tuples to users as these tuples exhibit significant uncertainty and learning them is likely to yield substantial information gain compared to others.

To identify the most ambiguous tuples, we compute the average cosine similarity within each cluster, i.e.,  $\text{Dist}(C_i) = \sum (\{\cos(e_i, e_j) | e_i \in C_i, e_j \in C_i, i \neq j\}) / |C_i|$ , and select  $\theta$  clusters with the lowest average cosine similarity. Subsequently, within each cluster  $C_i$ , we identify the most anomalous vertex  $v_i$  to construct the labeled training data  $\mathcal{T}_{\text{label}}$  for subsequent error detection and correction tasks.

**Example 3:** In Table 1, a well-trained graph  $\mathcal{G}$  will assign tuples  $t_4, t_5, t_6, t_7$  into the same cluster  $C$ . In  $C$ ,  $t_5$  is the outlier vertex, since  $t_5$  has little correlations with other vertices compared to

other ones. Thus we add  $t_5$  into  $\mathcal{T}_{\text{label}}$ .  $\square$

**Remark.** The most ambiguous tuples are erroneous ones in most cases. Based on the assumption that  $\mathcal{T}$  contains small number of errors and erroneous tuples have explicit or implicit inconsistencies with others, error detection is equivalent to anomaly detection of tuples in a learned latent space. Thus, the most ambiguous tuples detected by graph representation learning are more likely to be erroneous ones. By integrating other strategies to enhance the training data (Section 5 and 6), the class-imbalance issues could be reduced. As evaluated in Section 7, the ratio of erroneous cells in the training data to fine-tune GIDCL<sub>cor</sub> is 45.87% on average, which verifies that enough erroneous cells are selected to learn.

## 5 A Creator-Critic Workflow for Error Detection

In this section, we combine PLMs and LLMs for error detection, given a limited labelled examples  $\mathcal{T}_{\text{label}}$ . In particular, we propose a creator-critic workflow that jointly fine-tunes a PLM-based classifier  $\mathcal{M}_{\text{det}}$  as detector, and refine an interpretable LLM-generation function set  $\mathcal{F}^{\text{det}}$  for error detection. The proposed workflow could effectively prevent  $\mathcal{M}_{\text{det}}$  from over-fitting to the limited labelling examples, and incorporate LLMs to generate interpretable patterns for error detection, over all attribute in  $\mathcal{T}$ .

**Error detection model  $\mathcal{M}_{\text{det}}$ .** We develop an error detection model  $\mathcal{M}_{\text{det}}$  using a set of labeled  $\mathcal{T}_{\text{label}}$  and pseudo-labeled  $\mathcal{T}_{\text{pseudo}}$  tuples as training data, denoted as  $D^{\text{train}}$ . The model, which incorporates a pre-trained PLM with bi-level context attention [25], identifies each cell  $t_{i,j}$  in  $\mathcal{T}$  as clean or dirty. This is achieved by mapping a serialized sequence of the cell to a binary label 0, 1, where 0 (resp. 1) denotes a clean (resp. dirty) cell, framing the task as sequence classification [72]. Notably, the detection of a potentially erroneous cell relies on two key contextual dependencies: (1) *row-contextual dependency* that errors are correlated to values of other attributes within the same row, and (2) *column-contextual dependency* that errors are associated with other values in the same column within the same clusters.

Given a cell  $t_{i,j}$ , we incorporate the aforementioned dependencies into the input for  $\mathcal{M}_{\text{det}}$ . The main idea is that we serialize  $t_{i,j}$  with contexts along both the  $i$ -th row and  $j$ -column. (1) For the row-contextual dependency, we serialize the entire row  $t_i$  to aid  $\mathcal{M}_{\text{det}}$  in identifying attribute correlations within the same row. (2) For the column-contextual dependency, we select distinct values from attribute  $A_j$  of all tuples in the same cluster as  $t_i$ , denoted by  $C(t_i)[A_j]$ . Thus, the final input  $\text{serial}(t_{i,j})$  for  $\mathcal{M}_{\text{det}}$  is represented as follows.

$$\text{serial}(t_{i,j}) = \begin{cases} \text{serial}(t_i)[\text{SEP}]\text{serial}(t_i[A_j]) & (\text{row-context}) \\ \text{serial}(C(t_i)[A_j])[\text{SEP}]\text{serial}(t_i[A_j]) & (\text{column-context}) \end{cases}$$

where  $\text{serial}(t_i[A_j])$  only serializes the value of  $t_{i,j}$  itself;  $\text{serial}(t_i)$  and  $\text{serial}(C(t_i)[A_j])$  are serializations of  $t_i$  and  $C(t_i)[A_j]$  following the format in [110] based on tags  $\langle \text{COL} \rangle$  and  $\langle \text{VAL} \rangle$ .

**Example 4:** Consider  $(t_3, t_3^+) \in \mathcal{T}_{\text{label}}$  in Table 1 for the 2nd attribute ProviderID. We generate the sequence  $\text{serial}(t_{3,2})$  and  $\text{serial}(t_{3,2}^+)$  w.r.t. row and column-contextual dependencies as follows.

(1) For cell  $t_{3,2}$ , its row-context serialization is:

“ $\langle \text{COL} \rangle \text{ProviderID} \langle \text{VAL} \rangle 1x1303 \langle \text{COL} \rangle \text{City} \langle \text{VAL} \rangle \text{Monticello} \langle \text{COL} \rangle \text{State} \langle \text{VAL} \rangle \text{AR} \langle \text{COL} \rangle \text{Zip} \langle \text{VAL} \rangle 71655 \langle \text{COL} \rangle \text{County} \langle \text{VAL} \rangle \text{Drew} [\text{SEP}] \langle \text{COL} \rangle \text{ProviderID} \langle \text{VAL} \rangle 1x1303$ ”.

Also its column-context dependency is

“ $\langle \text{COL} \rangle \text{ProviderID} \langle \text{VAL} \rangle 111303 \langle \text{VAL} \rangle 1x1303 \langle \text{VAL} \rangle 10001 [\text{SEP}] \langle \text{VAL} \rangle 1x1303$ ”.

The labels of the two serializations are 0, indicating  $t_{3,2}$  is a negative instance.

(2) For clean cell  $t_{3,2}^+$ , we also serialize it as:

“ $\langle \text{COL} \rangle \text{ProviderID} \langle \text{VAL} \rangle 111303 \langle \text{COL} \rangle \text{City} \langle \text{VAL} \rangle \text{Monticello} \langle \text{COL} \rangle \text{State} \langle \text{VAL} \rangle \text{AR} \langle \text{COL} \rangle \text{Zip} \langle \text{VAL} \rangle 71655 \langle \text{COL} \rangle \text{County} \langle \text{VAL} \rangle \text{Drew} [\text{SEP}] \langle \text{COL} \rangle \text{ProviderID} \langle \text{VAL} \rangle 111303$ ”, and  
 “ $\langle \text{COL} \rangle \text{ProviderID} \langle \text{VAL} \rangle 111303 \langle \text{VAL} \rangle 1x1303 \langle \text{VAL} \rangle 10001 [\text{SEP}] \langle \text{VAL} \rangle 111303$ ”.

We label them as 1 as positive ones.  $\square$

After serialization, we fine-tune  $\mathcal{M}_{\text{det}}(\text{serial}(t_{i,j}))$  using the Cross-Entropy as the loss function. In the inference process, we serialize all cells in  $\mathcal{T}$  and identify erroneous ones using  $\mathcal{M}_{\text{det}}$ .

**LLMs-generated interpretable function for error detection.** Existing error detection approaches, e.g., Raha [70] and activeClean [58], are hard to learn or discover generalized and interpretable patterns only based on the observation from few-shot examples without prior knowledge injected by human experts. However recently [15] found that LLMs are few-shot learners and could extract the generalized patterns to distinguish clean and dirty values with limited labeled examples, acting like a data scientists [17, 75]. We follow its idea to generate a set of error detection functions with delicately handcrafted prompts in a multi-turn interaction.

In detail, for the first cycle of interacting with the LLM, we denote the set of unique values in  $A_j$  of  $\mathcal{T}$  by  $v(A_j)$  as contextual information, and query LLM by serializing all the dirty and clean cell pairs  $\mathcal{T}_{\text{label}}$ , as  $\text{LLM}(p_1, \mathcal{T}_{\text{label}}, v(A_j))$ , and the returned result is an interpretable function  $f_{A_j}^{\text{det}}$ , which can detect whether a given cell  $t_{i,j}$  in  $A_j$  is dirty or not.  $p_1$  is a handcrafted prompt of generating a detection function. We restrict LLM to generate function with regular expression, which is proved to be effective in DC [84].

To evaluate the quality of generated  $f_{A_j}^{\text{det}}$ , we apply  $f_{A_j}^{\text{det}}$  over the labelling set  $\mathcal{T}_{\text{label}}$  to identify whether each cell is dirty or not. If the performance for  $f_{A_j}^{\text{det}}$  on  $\mathcal{T}_{\text{label}}$  is above the predefined threshold  $\tau$  in F-measure, we accept  $f_{A_j}^{\text{det}}$  as a reliable detection function; otherwise, for all the examples that are wrongly detected, denoted by  $\mathcal{T}_{\text{label}}^{\text{wrong}}$ , we start the next conversation such that  $\text{LLM}(p_2, \mathcal{T}_{\text{label}}^{\text{wrong}}, v(A_j))$ , and a new function  $f_{A_j}^{\text{det}}$  is generated and replaced with the old one. Here  $p_2$  is a new prompt that considers the wrongly predicted instances. The iteration of conversations continues until the number of rounds reaches the maximum iteration  $n'$ , or all dirty and clean instances in  $\mathcal{T}_{\text{label}}$  are evaluated by  $f_{A_j}^{\text{det}}$  such that the F-measure is at least  $\tau$ ; otherwise we do not accept  $f_{A_j}^{\text{det}}$  and only rely on  $\mathcal{M}_{\text{det}}$  for error detection.

**Example 5:** In Table 1, consider  $(t_1, t_1^+), (t_2, t_2^+) \in \mathcal{T}_{\text{label}}$ . To generate the interpretable  $f_{\text{State}}^{\text{det}}$  for the 4th attribute State, the input fed in LLMs in the first conversation is as follows.

- Instruction  $p_1$ : Please conclude a general pattern for dirty and clean cells, and write a general function with regular expression to detect whether a given cell is dirty or not.
- Demonstration  $\mathcal{T}_{\text{label}}$ : [VA, Jasper  $\rightarrow$  VA; VAA  $\rightarrow$  VA]
- Values  $v(A_j)$ : [VA; VAA; AL; ...]

The output function from LLM is  $f_{\text{State}}^{\text{det}} = \text{^[A-Z]+\$}$ , meaning clean values in State should be composed of upper letters. However it wrongly detects VAA of  $t_2$  as a clean one. Thus we continue with the second conversation for refinement. The input is as follows:

- Instruction  $p_2$ : Please conclude a general pattern for dirty and clean cells, regarding the provided wrongly detected cells.
- Demonstration  $\mathcal{T}_{\text{label}}^{\text{wrong}}$ : [VAA  $\rightarrow$  VA]
- Values  $v(A_j)$ : [VA; VAA; AL; ...]

The refined function from LLM is  $f_{\text{State}}^{\text{det}} = \text{^[A-Z]2\$}$ , meaning clean values in State should begin with the first two upper letters. Now the function is finalized because it satisfies all instances in  $\mathcal{T}_{\text{label}}$ .  $\square$

We could further adopt the LLM-generated functions to augment more training data. Once  $f_{A_j}^{\text{det}}$  is accepted, we query LLM to generate the interpretable corruption function  $f_{A_j}^{\text{gen}} = \text{LLM}(p_3, \mathcal{T}_{\text{label}}, f_{A_j}^{\text{det}}, v(A_j))$ , which is used to generate dirty values from clean ones, acting as a customized data augmentation operator. Next, we select all the clean values in  $A_j$  that matches the previous  $f_{A_j}^{\text{det}}$ , and use  $f_{A_j}^{\text{gen}}$  to generate similar error data  $\mathcal{T}_{\text{pseudo}}$ , such that for  $t_{i,j}^{\text{pseudo}} \in \mathcal{T}_{\text{pseudo}}$ , we have  $t_{i,j}^{\text{pseudo}} = f_{A_j}^{\text{gen}}(t_{i,j}^+)$ . These data will be added to  $D^{\text{train}}$  and are used to incrementally fine-tune  $\mathcal{M}_{\text{det}}$ .

**Example 6:** In Table 1, consider  $(t_3, t_3^+) \in \mathcal{T}_{\text{label}}$  and  $f_{\text{ProviderID}}^{\text{det}}$  for ProviderID is generated and accepted as a reliable one. To query LLM to generate  $f_{\text{ProviderID}}^{\text{gen}}$ , the input is as follows.

- Instruction  $p_3$ : Write a function, randomly transfer clean value to dirty.
- Demonstration  $\mathcal{T}_{\text{label}}$ : [1x1303  $\rightarrow$  111303, ...]
- $f_{\text{ProviderID}}^{\text{det}}$ : ^\d+\$ (ProviderID only contains numbers).
- Values  $v(A_j)$ : [1x1303; 111303; 10001; ...]

The output function from LLM is  $f^{\text{gen}} = \text{replace}([0, 9], \text{x})$ , meaning randomly replace a number of the clean value with letter x.  $\square$

**The creator-critic workflow.** We unify the above  $\mathcal{M}_{\text{det}}$  and  $\mathcal{F}^{\text{det}}$  and establish a creator-critic workflow for error detection, as depicted in Figure 3. Our process begins with the execution of a creator and critic in lines 2-23, with the creator, critic, and termination phases outlined as follows.

**Creator Phase.** We begin by leveraging LLMs to generate patterns for distinguishing between clean and dirty cells over each attribute  $A_j \in \mathcal{A}$ . We create a function  $f_{A_j}^{\text{det}}$  to detect whether a given cell is dirty or clean (line 6). This approach helps prevent LLMs from memorizing specific cases in  $\mathcal{T}_{\text{label}}$  and avoids hallucination issues. In line 7-11, the multi-turn conversations based on LLMs are iteratively invoked until the evaluation performance of  $f_{A_j}^{\text{det}}$  over  $\mathcal{T}_{\text{label}}$ , denoted by  $\text{Eval}(f_{A_j}^{\text{det}}, \mathcal{T}_{\text{label}})$ , such as F-measure, exceeds a threshold  $\tau$ . Otherwise, we discard the generated function.

Once  $f_{A_j}^{\text{det}}$  is accepted, we extend the conversation with LLM to further generate  $f_{A_j}^{\text{gen}}$ , which generates similar error data  $\mathcal{T}_{\text{pseudo}}$  (line 12-14). We then merge the augmented data with  $\mathcal{T}_{\text{label}}$  to form  $D^{\text{train}} = \mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}$ , which is used to train the critic  $\mathcal{M}_{\text{det}}$ .

**Critic Phase.** We serialize  $D^{\text{train}}$  into a set of bi-level context-aware instances, and fine-tune the relatively small PLM-based detection model  $\mathcal{M}_{\text{det}}$  until convergence (line 16-18). Finally,  $\mathcal{M}_{\text{det}}$  outputs a set  $\mathcal{T}_{\text{coreset}}$  of cells identified as correct ones with high confidences to the creator. We consider  $\mathcal{T}_{\text{coreset}}$  is as reliable as  $\mathcal{T}_{\text{label}}$ , and can be further used in in-context learning and discovering dependencies.

**Termination Phase.** Given that  $\mathcal{M}_{\text{det}}$  is trained on all attributes  $\mathcal{A} = \{A_1, A_2, \dots, A_n\}$ ,  $\mathcal{M}_{\text{det}}$  converges, and the creator updates the LLMs-generated functions for all  $n$  attributes. We execute the creator and the critic in multiple rounds until  $f_{A_j}^{\text{det}}$  no longer change (line 21-22). The output of the creator-critic workflow is a set of detected dirty cells  $\mathcal{T}_{\text{err}}$ , a set of self-generated data  $\mathcal{T}_{\text{pseudo}}$  and a set of reliable cells  $\mathcal{T}_{\text{coreset}}$ .

**Remark.** One advantage of our create-critic workflow is to incrementally augment the training data by adding  $\mathcal{T}_{\text{pseudo}}$  by LLMs so that  $\mathcal{M}_{\text{det}}$  is fine-tuned in enough training data. Such workflow

---

*Input:* the dirty relational table  $\mathcal{T}$  of  $\mathcal{A}$ , a set of labeled tuples  $\mathcal{T}_{\text{label}}$ .  
*Output:* A set  $\mathcal{T}_{\text{err}}$  of cells identified as erroneous ones, a set  $\mathcal{T}_{\text{pseudo}}$  of self-generated data, and a set  $\mathcal{T}_{\text{coreset}}$  identified as clean cells.

1.  $\mathcal{T}_{\text{pseudo}} := \emptyset, \mathcal{M}_{\text{det}} := \emptyset, \mathcal{T}_{\text{coreset}} := \emptyset;$
2. **while** true **do**
3.   */\* Creator \*/*
4.    $\mathcal{F}^{\text{det}} := \emptyset, \mathcal{F}^{\text{gen}} := \emptyset;$
5.   **for each**  $A_i \in \mathcal{A}$  **do**
6.      $\text{iter} := 0, v(A_i) := \mathcal{T}[A_i], f_{A_i}^{\text{det}} := \text{LLM}(p_1, \mathcal{T}_{\text{label}}, v(A_i));$
7.     **while**  $\text{Eval}(f_{A_i}^{\text{det}}, \mathcal{T}_{\text{label}}) \leq \tau$  and  $\text{iter} \leq \text{Iter}_{\text{max}}$  **do**
8.       Collect  $\mathcal{T}_{\text{label}}^{\text{wrong}} := \{(t_{j,k}, t_{j,k}^+) | 1 \leq j \leq |D^{\text{train}}|, 1 \leq k \leq |\mathcal{A}|, f_{A_i}^{\text{det}}(t_{j,k}) = 1, t_{j,k} \neq t_{j,k}^+\}$
9.        $f_{A_i}^{\text{det}} := \text{LLM}(p_2, \mathcal{T}_{\text{label}}^{\text{wrong}}, v(A_i));$
10.        $\text{iter} := \text{iter} + 1;$
11.       Add  $f_{A_i}^{\text{det}}$  into  $\mathcal{F}^{\text{det}}$  if  $\text{Eval}(f_{A_i}^{\text{det}}, \mathcal{T}_{\text{label}}) > \tau;$
12.        $f_{A_i}^{\text{gen}} := \text{LLM}(p_3, \mathcal{T}_{\text{label}}, f_{A_i}^{\text{det}}, v(A_i));$
13.        $\Delta \mathcal{T}_{\text{pseudo}} := \{t_{z,i}^{\text{pseudo}} | t_{z,i}^{\text{pseudo}} := f_{A_i}^{\text{gen}}(t_{z,i}^+), (t_z, t_z^+) \in \mathcal{T}_{\text{label}}\};$
14.        $\mathcal{T}_{\text{pseudo}} := \mathcal{T}_{\text{pseudo}} \cup \Delta \mathcal{T}_{\text{pseudo}};$
15.   */\* Critic \*/*
16.    $D^{\text{train}} := \mathcal{T}_{\text{pseudo}} \cup \mathcal{T}_{\text{label}};$
17.   Generate row/column-contextual dependency for each  $t \in D^{\text{train}};$
18.   Fine-tune  $\mathcal{M}_{\text{det}}$  using  $D^{\text{train}};$
19.   Select a subset  $\Delta \mathcal{T}_{\text{coreset}} \subseteq \mathcal{T}$  with high confidences of  $\mathcal{M}_{\text{det}};$
20.    $\mathcal{T}_{\text{label}} := \mathcal{T}_{\text{label}} \cup \Delta \mathcal{T}_{\text{coreset}}, \mathcal{T}_{\text{coreset}} := \mathcal{T}_{\text{coreset}} \cup \Delta \mathcal{T}_{\text{coreset}};$
21.   **if**  $\mathcal{F}^{\text{det}}$  does not change **do**
22.     **break**
23. Identify all errors  $\mathcal{T}_{\text{err}}$  in  $\mathcal{T}$  using  $\mathcal{M}_{\text{det}};$
24. **return**  $(\mathcal{T}_{\text{err}}, \mathcal{T}_{\text{pseudo}}, \mathcal{T}_{\text{coreset}});$

---

Fig. 3. The Creator-critic workflow of error detection

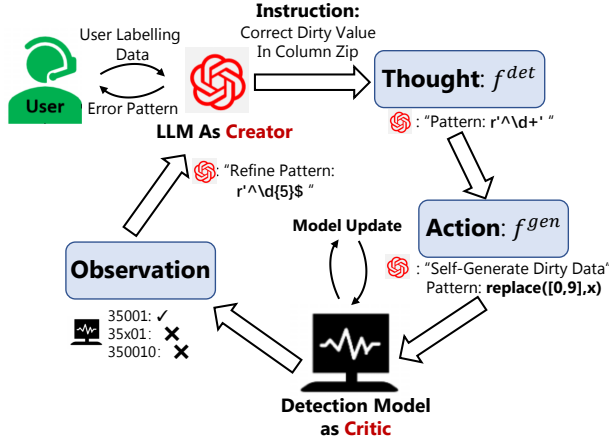


Fig. 4. Creator-critic workflow for error detection

also alleviates the impact of training data curation based on graph structure learning (Section 4), such that if the training data curation has the class-imbalance issue, GIDCL<sub>det</sub> could still be effective.

---

*Input:* a set of  $\mathcal{T}_{\text{err}}$  of dirty cells, a set  $\mathcal{T}_{\text{label}}$  of labeled data, a set  $\mathcal{T}_{\text{pseudo}}$  of self-generated tuples, a set  $\mathcal{T}_{\text{coreset}}$  of data identified by  $\mathcal{M}_{\text{det}}$  as correct ones, the fine-tuned  $\mathcal{M}_{\text{det}}$ , and a set  $\mathcal{C}$  of data groups.

*Output:* the clean relational table  $\mathcal{T}_{\text{clean}}$ .

1.  $D^{\text{train}} := \mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}$ ,  $\mathcal{M}_{\text{corr}} := \emptyset$ ,  $\mathcal{T}_{\text{clean}} := \mathcal{T}$ ;
2. Generate  $E^{\text{ICL}} := \{E_1^{\text{ICL}}, \dots, E_{|\mathcal{A}|}^{\text{ICL}}\}$  of ICL context from  $D^{\text{train}}$ ;
3. Generate  $E^{\text{RAG}} := \{E_1^{\text{RAG}}, \dots, E_{|\mathcal{A}|}^{\text{RAG}}\}$  of RAG examples from  $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{coreset}}$ ;
4. Generate the training set  $D_{\text{LLM}}^{\text{train}}$  using  $E^{\text{ICL}}$  and  $E^{\text{RAG}}$ , s.t.,  

$$D_{\text{LLM}}^{\text{train}} := \{[\text{context}(t_{i,j}), t_{i,j}^+](t_{i,j}, t_{i,j}^+) \in D^{\text{train}}\}$$
5. Fine-tune  $\mathcal{M}_{\text{corr}}$  using  $D_{\text{LLM}}^{\text{train}}$ , /\* implicit error correction \*/
6.  $\mathcal{F}^{\text{corr}} := \emptyset$ ;
7. **for each**  $A_i \in \mathcal{A}$  **do** /\* explicit error correction \*/
8.   iter := 0,  $v(A_i) := \mathcal{T}[A_i]$ ,  $f_{A_i}^{\text{corr}} := \text{LLM}(c, \mathcal{T}_{\text{label}}, v(A_i), f_{A_i}^{\text{det}})$ ;
9.   **while**  $\text{Eval}(f_{A_i}^{\text{corr}}, \mathcal{T}_{\text{label}}) \leq \tau$  and iter  $\leq \text{iter}_{\text{max}}$  **do**
10.     Collect  $\mathcal{T}_{\text{label}}^{\text{wrong}} := \{(t_{j,k}, t_{j,k}^+) | f_{A_k}^{\text{corr}} \neq t_{j,k}^+, (t_{j,k}, t_{j,k}^+) \in D^{\text{train}}\}$ ;
11.      $f_{A_i}^{\text{corr}} := \text{LLM}(c, \mathcal{T}_{\text{label}}^{\text{wrong}}, v(A_i), f_{A_i}^{\text{det}})$ ;
12.     iter := iter + 1;
13.     Add  $f_{A_i}^{\text{corr}}$  into  $\mathcal{F}^{\text{corr}}$  if  $\text{Eval}(f_{A_i}^{\text{corr}}, \mathcal{T}_{\text{label}}) > \tau$ ;
14. /\* Error detection in  $\mathcal{T}$  \*/
15. **for each**  $t_{i,j} \in \mathcal{T}_{\text{err}}$  **do**
16.    $t := \arg\max\{\mathcal{M}_{\text{det}}(\mathcal{M}_{\text{corr}}(t_{i,j})), \mathcal{M}_{\text{det}}(f_{A_j}^{\text{corr}}(t_{i,j}))\}$ ;
17.   Repair  $t_{i,j}$  of  $\mathcal{T}_{\text{clean}}$  using  $t$ ;
18.  $\Delta R := \text{GraphStructureRelearning}(\mathcal{T}_{\text{label}}, \mathcal{T}_{\text{coreset}}, f_{\text{GNN}}, \mathcal{T}_{\text{err}})$ ;
19. Repair  $\mathcal{T}_{\text{clean}}$  using  $\Delta R$ ;
20. **return**  $\mathcal{T}_{\text{clean}}$ ;

---

Fig. 5. The error correction algorithm

## 6 Error Correction

In this section, we propose an error correction algorithm for rectifying a dirty cell  $t_{i,j}$  to the clean value  $t_{i,j}^+$ . As depicted in Figure 5, we introduce two approaches: implicit error correction, which involves fine-tuning a local LLM as a generative model and the explicit error correction, which leverages an LLM as a few-shot learner to condense generated functions. Additionally, to address inconsistency errors, we refine the graph  $\mathcal{G}$  based on tuples predicted as clean in  $\mathcal{T}$  to discover high-quality functional dependencies (FDs).

**Implicit error correction.** We combine in-context learning (ICL) [29, 74], retrieval augmented generation (RAG) [61], and supervised fine-tuning (SFT) to learn an error correction model  $\mathcal{M}_{\text{corr}}$  that directly generates the true value by referencing its dirty one and some contextual information.

We leverage ICL and RAG to enhance the learning process. ICL uses correction pairs, e.g.,  $(t_{i,j}, t_{i,j}^+)$  from labeled data  $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}$  to direct LLMs in correcting cells accurately, preventing irrelevant outputs. Conversely, RAG utilizes clean examples from  $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{coreset}}$  to activate the emergent abilities of LLMs, ensuring effective contextualization and application of corrections. This dual strategy refines the model's accuracy and relevancy in error correction tasks.

Specifically, for each (dirty, clean) cell  $(t_{i,j}, t_{i,j}^+) \in D^{\text{train}}$ , we construct a sequence denoted by  $\text{context}(t_{i,j})$  for fine-tuning LLM with ICL and RAG context. In detail,  $\text{context}(t_{i,j})$  contains a handcrafted prompt  $q$ , the serial representation of the cell and its context serial  $(t_{i,j})$ , and relevant repair examples  $E_j^{\text{ICL}} = \{(t_{i,j}, t_{i,j}^+) | t_{i,j} \in \mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}, t_{i,j} \neq t_{i,j}^+\}$  for ICL demonstration, and a set  $E_i^{\text{RAG}}$  of tuples sampled within  $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{coreset}}$ , also from the same cluster  $C_i$  containing  $t_i$  as RAG.

The sequence format is :

$$\text{context}(t_{i,j}) = q \circ \text{serial}(t_{i,j}) \circ E_j^{\text{ICL}} \circ E_i^{\text{RAG}}$$

The training dataset for LLM is re-organized as  $D_{\text{LLM}}^{\text{train}} = \{[\text{context}(t_{i,j}), t_{i,j}^+]|t_{i,j} \in D^{\text{train}}\}$ , where  $t_{i,j}^+$  is the correct value as label. We further fine-tune a local LLM  $\mathcal{M}_{\text{corr}}$  using the LoRA technique [52]. During inference, we serialize the context of an erroneous cell  $t_{i,j} \in \mathcal{T}_{\text{err}}$  into  $\text{context}(t_{i,j})$  and obtain the corrected value via  $\mathcal{M}_{\text{corr}}(t_{i,j})$ .

**Example 7:** Consider  $t_2$  in Table 1 and  $\mathcal{M}_{\text{det}}$  detects VAA of the 3rd attribute State as a dirty cell. To repair it, we generate the sequence  $\text{context}(t_{2,3})$  that consists of the following components.

- Instruction  $q$  = Given the dirty row, you are required to correct the value in column State.
- Input  $\text{serial}(t_{2,3}) = \{\text{City} : \text{Monticello}, \text{State} : \text{VAA}, \dots\}$
- ICL Demonstration:  $[\text{AL}, \text{Houston} \rightarrow \text{AL}]$
- RAG Context:  $\text{serial}(t_1^+) = \langle \text{COL} \rangle \text{ProviderID} \langle \text{VAL} \rangle 111303 \langle \text{COL} \rangle \text{City} \langle \text{VAL} \rangle \text{Monticello} \langle \text{COL} \rangle \text{State} \langle \text{VAL} \rangle \text{VA} \langle \text{COL} \rangle \text{Zip} \langle \text{VAL} \rangle 31064 \langle \text{COL} \rangle \text{County} \langle \text{VAL} \rangle \text{Jasper}$ .

Here  $t_1^+$  and  $t_2$  are in the same cluster and  $t_1^+ \in \mathcal{T}_{\text{label}}$ . Finally  $\mathcal{M}_{\text{corr}}$  rectifies  $t_{2,3}$  to VA.  $\square$

*Remark.* Beside  $\mathcal{T}_{\text{label}}$ , we also adopt  $\mathcal{T}_{\text{coreset}}$ , the high-quality training data from  $\mathcal{M}_{\text{det}}$ , to construct RAG, and  $\mathcal{T}_{\text{pseudo}}$ , the training data by leveraging LLMs to fine-tune  $\mathcal{M}_{\text{cor}}$ . More specifically, we designed various data augmentation strategies to enhance the training data so that  $\mathcal{M}_{\text{cor}}$  is more accurate and robust.

**Explicit and interpretable error correction.** While the generative model  $\mathcal{M}_{\text{corr}}$  excels at handling intricate corrections, such as transforming bxxmxxgha to birmingham, it encounters simpler tasks where a more transparent approach can be beneficial. For example, tasks like converting 16.0 ounces to 16 or reformatting dates from yyyy/dd/mm to dd/mm/yy might not necessitate the full generative capabilities of LLMs. In such cases, querying LLMs to generate an interpretable function  $f_{A_j}^{\text{corr}}$  tailored for error correction specific to an attribute  $A_j \in \mathcal{A}$  presents a viable alternative. This method not only simplifies the correction process but also helps in mitigating the potential hallucination issues associated with LLMs, ensuring that corrections are both accurate and straightforward.

Similar with generating  $f_{A_j}^{\text{gen}}$  with LLM in Section 5, we employ the following query by extending existing conversation with LLM:  $f_{A_j}^{\text{corr}} = \text{LLM}\left(c, \mathcal{T}_{\text{label}}, v(A_j), f_{A_j}^{\text{det}}\right)$ , where  $c$  is a handcrafted prompt instructing the LLM to generate a function for error correction. The expected output  $f_{A_j}^{\text{corr}}$  is a function that transforms a dirty cell  $t_{i,j}$  to its clean value.  $f_{A_j}^{\text{corr}}$  is accepted only if it achieves a higher F-measure than a given threshold  $\tau$  over  $\mathcal{T}_{\text{label}}$ , similar to  $f_{A_j}^{\text{det}}$ .

**Example 8:** Consider  $t_5$  in Table 1 and  $\mathcal{M}_{\text{det}}$  detects AL,Houston of the 4-th attribute State is dirty.

We query  $f_{\text{State}}^{\text{corr}}$  using:

- Instruction  $c$ : Please write a function to correct the dirty value to clean.
- Demonstration  $[t_{i,j}, t_{i,j}^+]$ :  $[\text{VA}, \text{Jasper} \rightarrow \text{VA}]$
- Context  $v(A_j)$ : Clean:  $[\text{VA}, \text{AL}, \dots]$ ; Dirty:  $[\text{VA}, \text{Jasper}, \dots]$
- $f_{A_j}^{\text{det}}$ :  $\wedge[\text{A-Z}]\{2\}\$, (\text{State should be 2 upper letters})$

Finally  $f_{\text{State}}^{\text{corr}}(t_{i,j}) = t_{i,j}[:2].\text{upper}()$  for State, which is interpretable, and  $f_{\text{State}}^{\text{corr}}(t_{5,4}) = \text{AL}$ .  $\square$

**Repair selection.** Considering  $\mathcal{M}_{\text{corr}}(t_{i,j})$  and  $f_{A_j}^{\text{corr}}(t_{i,j})$  for correcting  $t_{i,j}$  might be different. We design a simple but effective ranking method based on the error detector  $\mathcal{M}_{\text{det}}$  to pick up the most suitable one. Recall  $\mathcal{M}_{\text{det}}$  is a binary classifier to identify whether  $t_{i,j}$  is dirty or not with



a confidence score  $\text{Conf}(t_{i,j})$ , i.e., the output of the Softmax layer. To select the suitable repair of  $t_{i,j}$ , we compute and compare the confidence scores of  $\text{Conf}(\mathcal{M}_{\text{corr}}(t_{i,j}))$  and  $\text{Conf}(f_{A_j}^{\text{corr}}(t_{i,j}))$ . The value with a larger confidence score is the final repair, denoted as  $\text{GIDCL}_{\text{cor}}(t_{i,j})$ . By applying  $\text{GIDCL}_{\text{cor}}$  over all erroneous cells  $\mathcal{T}_{\text{err}}$ , we can repair the dirty table  $\mathcal{T}$  to  $\mathcal{T}_{\text{clean}}$ , and update the constructed graph  $\mathcal{G}$  to  $\mathcal{G}'$  with  $\mathcal{T}_{\text{clean}}$  accordingly.

Although LLM-based error correction above is able to give the repair suggestions by referencing correlated tuples and contexts, it might not always give exact corrections for inconsistency errors, because LLM may not extract and understand violation of attribute dependencies (VAD) across the whole relational table. Considering this issue, we update the graph structure to discover a few functional dependencies (FDs) based on tuples that are more likely to be clean.

**Re-learning graph structure.** After cleaning  $\mathcal{T}$  using  $\text{GIDCL}_{\text{cor}}$ , we then apply  $\mathcal{M}_{\text{det}}$  to select a few tuples  $\mathcal{T}_{\text{coreset}}$  that have high confidence scores, i.e., high qualities. Then we focus on mining FDs in  $\mathcal{T}_{\text{coreset}} \cup \mathcal{T}_{\text{label}}$  and repair inconsistencies using graph structure learning based on the mined FDs.

FDs discovery. For simplicity, we only consider discovering FDs :  $X \rightarrow Y$ , where  $X$  and  $Y$  are single attributes. In the discovery process, we enumerate all pairs of attributes  $(A_i, A_j)$  to check whether  $A_i \rightarrow A_j$  and  $A_j \rightarrow A_i$  are valid FDs in  $\mathcal{T}_{\text{coreset}} \cup \mathcal{T}_{\text{label}}$ , where  $A_i, A_j \in \mathcal{A}$ . The process needs  $O(|\mathcal{T}||\mathcal{A}|^2)$  time complexity.

Graph Learning. For each valid FD :  $A_i \rightarrow A_j$ , we extract the sub-graph  $\mathcal{G}'_{\text{sub}}$  from  $\mathcal{G}'$  such that  $\mathcal{G}'_{\text{sub}} = \{(h, r, t) | r \in \{A_i, A_j\}\}$ . Then we re-cluster  $\mathcal{G}'$  into  $k$  clusters such that if tuples  $t_i$  and  $t_j$  reside in the same cluster, they share identical values of at least one attribute of  $A_i$  and  $A_j$ . Considering a central node  $h_i$  within cluster  $C_i$  of the sub-graph  $\mathcal{G}'_{\text{sub}}$ , we modulate the weight of each directed edge  $e$  in  $\mathcal{G}'_{\text{sub}}$  utilizing the message passing function  $\omega(h, r, t)$ , defined as:

$$\omega(h, r, t) = \begin{cases} 1 & \text{if } h_i \in \mathcal{T}_{\text{label}} \\ 1/|C_i| & \text{if } h_i \in \mathcal{T}_{\text{coreset}} \\ 1/\lambda|C_i| & \text{if } h_i \notin \mathcal{T}_{\text{coreset}} \end{cases} \quad (3)$$

Given that  $\mathcal{T}_{\text{label}}$  represents the ground truth, it is imperative to prioritize its aggregation within the cluster. While  $\mathcal{T}_{\text{coreset}}$  likely approximates the ground truth, its aggregation is accorded secondary priority, ensuring its cumulative weight does not surpass that of  $\mathcal{T}_{\text{label}}$ . Conversely, cleaned cells should aggregate with minimal priority. The hyper-parameter  $\lambda$  modulates this prioritization.

Error correction. Upon updating the edge weights in  $\mathcal{G}'_{\text{sub}}$  by clusters, we apply link prediction task[26] in trained  $\mathcal{G}'_{\text{sub}}$  for attribute  $A_j$  on a cluster basis. In detail, we utilize the modified  $\omega(h, r, t)$  to update edge weight for  $\mathcal{G}'_{\text{sub}}$ , then retrieving refined node embedding  $\Theta$  with Eq.1. Given node embedding  $\Theta(h_i)$  and edge embedding  $\Theta(A_j)$ , link prediction task aims to find the most similar embedding  $\Theta(t)$ , where cell  $t \in \mathcal{T}[A_j]$  also lies within cluster  $C_i$ .

This implies that if a node  $t_i \in \mathcal{T}_{\text{label}}$  exists within cluster  $C_i$ , the user-labeled ground truth, noted as triplet  $(h_i, A_j, t)$ , should be universally applicable to all central nodes  $h_k \in C_i$ . In the absence of labeled data within cluster  $C_i$ , the weighted majority value of attribute  $A_j$  is designated to all central nodes  $h_k \in C_i$ , via the node aggregation mechanism of GNN. Besides, such aggregation is limited into relations in FDs, and controlled by  $\omega$  in Eq.3, preventing the over-smoothing issues with GCN, which propagate noise information over the whole graph.

**Example 9:** Consider  $C_{\text{ProviderID}}$  in Table 1, where  $|C_{\text{ProviderID}}|=4$ , and the aggregated weight  $\omega(h, \text{City}, \text{Dothan}) = \frac{3}{4}$  ( $t_5, t_6, t_7 \in \mathcal{T}_{\text{coreset}}$ ),  $\omega(h, \text{City}, \text{Monticello}) = \frac{1}{4}\lambda$  ( $t_4 \notin \mathcal{T}_{\text{coreset}}$ ), so the value of City in  $t_4, t_5, t_6, t_7$  should be unified as Dothan.  $\square$

This correction procedure iterates across all elements with FDs to get the final correction result.

**Remark.** GIDCL<sub>cor</sub> is capable to solve data imputation on missing values by leveraging its implicit error correction and re-learning graph structure, such that the implicit error correction  $\mathcal{M}_{\text{corr}}$  imputes missing values in the generative way, and re-learning graph structure imputes values based on the dependencies of other values.

## 7 Experimental Study

Using standard datasets, we empirically evaluated our method GIDCL on (1) the effectiveness and efficiency of error detection and correction, (2) the robustness on the impact of error rate and labelling budgets  $\theta$ , (3) the effectiveness of the creator-critic framework and automatically generated function set  $\mathcal{F}$ , and (4) the impact of parameter size for LLMs.

**Experimental settings.** We start with our settings.

Datasets. We use 5 benchmark datasets following the settings in the existing literature Baran [68] and one real-life large dataset .

Hospital [89] and Flights [64] offer rich contextual information with a high degree of data redundancy. Notably, Hospital has scarce and randomly imposed noises, while Flights exhibit a very high error rate. Tax is a large synthetic dataset from the BART repository [7], featuring various data error types, thus resulting in a vast search space to identify true corrections. Beers [68] and Rayyan [77] are also real-life datasets but lack data redundancy, posing challenging for correction. IMDB [1] is a large real-life dataset encompassing millions of movies and TV series spanning from 1905 to 2022, and contains various error types that are nontrivial to repair.

Following [68, 79], we have the clean data as the ground truth and dirty data for evaluation. For Beers, Flights and Rayyan, the dirty data contains real-life noises and cleaned by data owners, while for the remaining datasets, the dirty data is generated by adding noise with a certain rate  $e\%$ , i.e., the percentage of dirty cells on all data cells. We introduce four types of noise, including missing value (MV), typo (T), formatting issue (FI), and violated attribute dependency (VAD). Detail is provided in [2].

Baselines. We implemented GIDCL in Python and used the following baselines. (1) Raha [70], an error detection method involving feature engineering, interactive labeling and ML models for detection; (2) Rotom [72], a meta-learning data augmentation framework that formulates tabular error detection as a seq2seq task; (3) Roberta<sub>det</sub> [65], a binary classifier adapted for error detection utilizing the pre-trained language model Roberta [65]; (4) Baran [68], a hybrid error correction approach based on feature engineering and traditional ML models; (5) Garf [79], a deep learning-based error correction approach that employs SeqGAN [106] to generate data repair rules in an unsupervised manner; (6) HoloClean [89], an error correction method that leverages data quality rules to construct factor graph for data repairs; (7) T5 [85], a generative PLM for error correction; (8) JellyFish-13B [107], an LLM-based method addressing error detection and data imputation separately, utilizing a 13B LLM to solve multiple data preprocessing tasks.

All error correction methods followed end-to-end data cleaning pipelines. This implies that for each method, the range of error correction relies on its error detection results. Among error correction methods lacking the support of error detection, we adopt Raha as the error detection for Baran as referenced in [69]. For the remaining error correction methods, including HoloClean and T5, we utilized our error detection method GIDCL<sub>det</sub> for fair comparison.

Measures. We report precision (P), recall (R), and the F1 (F) score to evaluate the effectiveness for error detection and error correction, the same with [70] and [68]. We also report the runtime

Table 3. Error detection performance in comparison to the baselines

System	Hospital			Flights			Beers			Rayyan			Tax			IMDB		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
GIDCL	<b>1.00</b>	0.95	<b>0.98</b>	<b>0.96</b>	<b>0.95</b>	<b>0.96</b>	0.99	0.99	0.99	0.84	<b>0.98</b>	<b>0.90</b>	0.99	<b>0.98</b>	<b>0.98</b>	0.88	0.89	<b>0.88</b>
GIDCL <sub>offline</sub>	0.96	0.90	0.93	0.96	0.95	0.96	0.99	0.97	0.98	0.76	0.94	0.84	0.99	0.98	0.98	0.97	0.68	0.80
Raha	0.96	0.67	0.79	0.85	0.93	0.89	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>0.88</b>	0.88	0.88	0.84	0.77	0.80	0.28	0.37	0.32
Rotom	0.95	0.94	0.95	0.47	0.89	0.61	0.91	0.95	0.93	0.26	0.88	0.40	<b>1.00</b>	0.60	0.75	0.17	<b>1.00</b>	0.29
Roberta <sub>det</sub>	0.87	<b>0.97</b>	0.92	0.99	0.47	0.64	0.99	0.97	0.98	0.66	0.94	0.77	0.66	0.87	0.75	<b>0.94</b>	0.18	0.30
JellyFish	0.87	0.91	0.89	0.55	0.85	0.67	0.89	0.75	0.81	0.66	0.72	0.69	0.66	0.87	0.75	0.25	0.85	0.38

for model training and inference and show the number of labeled tuples to evaluate the human involvement and impacts for baselines.

**Configuration.** We select RoBERTa as the backbone for  $\mathcal{M}_{\text{det}}$ , and Mistral-7B [55] as the backbone model for  $\mathcal{M}_{\text{corr}}$ . We adopt gpt-4-turbo as the online reference LLM to generate function set  $\mathcal{F}$  as default setting, denoted as GIDCL, and apply Mistral-7B as offline reference LLM, denoted as GIDCL<sub>offline</sub>. All prompts in GIDCL are handcrafted once and listed in [2]. The default  $\theta_{\text{label}} = 20$ ,  $\tau = 0.85$  and  $\lambda = 4$ . The cluster number for Hospital, Flights, Beers, Rayyan, Tax, and IMDB are 20, 20, 10, 10, 30, and 50, respectively. We conduct our experiment on a single machine powered by 256GB RAM and 32 processors with Intel(R) Xeon(R) Gold 5320 CPU @2.20GHz and 4 RTX3090 GPUs. Each experiment was conducted twice, averaging the results reported here.

**Experimental results.** We next report our findings.

**Exp-1: Effectiveness.** We evaluated GIDCL with other baselines in terms of error detection and correction. For fair comparisons, all baselines take the same input  $\mathcal{T}$  and the labelling budget  $\theta_{\text{label}}$ .

**Error Detection.** Table 3 shows the performance of all baselines. GIDCL surpasses all baselines in 5 out of 6 datasets, achieving an average accuracy improvement of 23.1% and up to 58% in F1. This verifies that the creator-critic framework in GIDCL is effective, and  $\mathcal{M}_{\text{det}}$  and  $\mathcal{F}$  could help with each other to boost the overall performance. Additionally, GIDCL enhances model interpretability, and effectively mitigates overfitting risk through its generation functions, thereby improving the robustness and reliability. However, GIDCL<sub>offline</sub> has a slight performance decrease compared to GIDCL, due to the instability of the offline LLM as generator of function set  $\mathcal{F}$ , potentially leading to failure in detecting and augmenting data on certain key attributes. Nevertheless, GIDCL<sub>offline</sub> still outperforms other baselines in most cases.

Raha demonstrates the comparable performance in Beers and Rayyan. However, in datasets with richer information, in-context learning of GIDCL significantly benefits both  $\mathcal{F}^{\text{det}}$  and  $\mathcal{M}_{\text{det}}$ . Compared to Raha, GIDCL is 19%, 7%, 18% and 56% higher F-measure in Hospital, Flights, Tax and IMDB, respectively. Rotom primarily generates random augmentations, limiting its ability to interpret dirty and clean data patterns, leading Rotom to inferior performance in scenarios with complex textual inconsistencies. Roberta<sub>det</sub> performs similarly to Rotom in most datasets. However, due to a small labelling budget, Roberta<sub>det</sub> suffers from the over-fitting problem.

It is worth noting that LLM-based error detection baseline JellyFish performs worse than non-LLM baselines in most cases, indicating that the naive adoption of LLM falls short in error detection, as discussed in the observation of Section 2.2.

**Error correction.** Table 4 shows the F1-score of error correction, and GIDCL outperforms all baselines with 20.5% higher F1-score on average, compared to the best of others. This verifies the effectiveness of unifying  $\mathcal{M}_{\text{corr}}$ ,  $\mathcal{F}^{\text{corr}}$  and graph structure learning. GIDCL<sub>offline</sub> exhibits a slight performance decline compared to GIDCL. This is attributed to the function set  $\mathcal{F}$  generated by the offline model, which may not match the quality of that produced by the online model GPT-4. Nonetheless, GIDCL<sub>offline</sub> still surpasses other baselines, e.g., 16% higher F1-score than the best of others on

Table 4. End-to-end error correction performance in comparison to the baselines

System	Hospital			Flights			Beers			Rayyan			Tax			IMDB		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
GIDCL	0.97	<b>0.96</b>	<b>0.97</b>	<b>0.94</b>	<b>0.92</b>	<b>0.93</b>	<b>0.97</b>	<b>0.97</b>	<b>0.97</b>	<b>0.80</b>	<b>0.93</b>	<b>0.86</b>	<b>0.95</b>	<b>0.94</b>	<b>0.95</b>	<b>0.87</b>	<b>0.89</b>	<b>0.87</b>
GIDCL <sub>offline</sub>	0.94	0.90	0.92	0.84	0.81	0.82	0.95	0.95	0.95	0.78	0.85	0.81	0.89	0.89	0.89	0.79	0.80	0.80
Raha + Baran	0.95	0.52	0.67	0.84	0.56	0.67	0.93	0.87	0.90	0.44	0.21	0.28	0.84	0.77	0.80	0.19	0.08	0.12
GIDCL <sub>det</sub> + HoloClean	<b>0.98</b>	0.71	0.82	0.89	0.67	0.76	0.01	0.01	0.01	0.00	0.00	0.00	0.11	0.11	0.11	0.22	0.18	0.20
Garf	0.68	0.56	0.61	0.57	0.25	0.35	0.40	0.03	0.04	0.34	0.40	0.37	0.55	0.58	0.56	0.30	0.25	0.27
GIDCL <sub>det</sub> + T5	0.54	0.39	0.45	0.39	0.27	0.32	0.73	0.97	0.83	0.55	0.62	0.58	0.72	0.59	0.65	0.45	0.35	0.39
JellyFish	0.84	0.71	0.77	0.75	0.71	0.73	0.73	0.66	0.69	0.65	0.52	0.58	0.85	0.65	0.74	0.50	0.41	0.45

average.

HoloClean demonstrates relatively good precision and recall in datasets with high redundancy, such as Hospital and Flights. However, its performance degrades in datasets with lower redundancy or fewer dependencies among attributes. In such scenarios, HoloClean is difficult to repair errors. Baran has a significant performance drop in most datasets. The primary issue lies in its generation of an excessive number of candidates, *e.g.*, 455,390 candidates in total for Hospital, making it difficult to select the most suitable one via learning traditional ML classifiers. The experimental results highlight the need for a robustly trained generative model to effectively address such issues, *e.g.*,  $\mathcal{M}_{\text{corr}}$  in GIDCL. Garf employs a SeqGAN model for unsupervised generation of data repair rules, subsequently refined through a co-training framework. The generated data quality rules might not be capable of handling unseen data. For instance, in Hospital, Garf cannot generate the correct value Birmingham from the dirty cell Bxrmxngham if Birmingham is absent in the dataset. The limitations of T5 are apparent in its inability to employ the retrieval-augmented generation paradigm and to repair inconsistency errors, leading to a noticeable decline in its performance.

The LLM baseline JellyFish demonstrates the substantial potential of generative models in data cleaning tasks. However, its performance is notably inferior to that of GIDCL<sub>offline</sub>. This highlights the efficacy of our in-context learning strategy and the use of self-annotated data, which effectively mitigate hallucination issues in LLMs and result in a significant performance boost.

*Varying different error detectors/correctors.* In Table 5, we further evaluate the effectiveness of the proposed correction method GIDCL<sub>cor</sub>, by varying different error detectors in Hospital and Rayyan. GIDCL<sub>cor</sub> shows its robustness and stability in dealing with false positive (FP) cases, *e.g.*, for dataset Rayyan, detector Rotom detects 1,376 correct values to be error (*w.r.t.* 1,376 FN cases, here we define positive sample as clean values, and vice versa), however GIDCL<sub>cor</sub> keeps most of the above FN clean data unchanged, only correcting real errors, decreasing FN cases to 100. Similarly, by introducing GIDCL<sub>det</sub> and varying different correction models, we also find the high performance of GIDCL<sub>det</sub> significantly boost the performance of baseline correction models, *e.g.*, Baran and JellyFish, by adding error values that are wrongly detected to be clean by original detectors (*w.r.t.* FP). For example, in dataset Rayyan, GIDCL<sub>det</sub> adds 148 FP cases that are neglected by Raha detector, which leads to non-trivial performance boosting of Baran from 0.28 to 0.57.

*Varying sampling methods.* We evaluate the effectiveness of GSL for GIDCL in Table 6, compared with three sampling baselines, including SBert that replaces the graph structure learning in Section 4 with SentenceBert [88], Raha that adopts the idea of Raha [69] to select  $\theta$  representative tuples and Random that randomly select  $\theta$  tuples. GSL consistently outperforms others, *e.g.*, at most 14% higher F1-scores. The above result verifies the effectiveness of GSL to select representative tuples, which can cover most of error attributes, as well as propagating labels within the largest  $k$  clusters.

Table 5. Error correction varying error detectors/correctors

Detector	Hospital			Rayyan		
	P	R	F	P	R	F
GIDCL	0.97	<b>0.96</b>	<b>0.97</b>	0.80	<b>0.93</b>	<b>0.86</b>
GIDCL <sub>offline</sub>	0.94	0.90	0.92	0.78	0.85	0.81
Raha +GIDCL <sub>cor</sub>	<b>0.98</b>	0.50	0.66	<b>0.85</b>	0.69	0.76
Rotom +GIDCL <sub>cor</sub>	0.90	0.80	0.85	0.74	0.67	0.71
GIDCL <sub>det</sub> + Baran	0.94	0.88	0.91	0.80	0.44	0.57
GIDCL <sub>det</sub> + JellyFish	0.88	0.75	0.81	0.78	0.91	0.84

Table 6. Error correction by varying sampling methods

Sample Method	Hospital			Rayyan		
	P	R	F	P	R	F
GSL	<b>0.94</b>	<b>0.90</b>	<b>0.92</b>	<b>0.78</b>	<b>0.85</b>	<b>0.81</b>
SBert	0.92	0.87	0.90	0.75	0.74	0.75
Raha	0.80	0.79	0.79	0.70	0.71	0.71
Random	0.78	0.76	0.77	0.71	0.72	0.72

Table 7. Ratio  $r_{err}$  in  $D^{train} = \mathcal{T}_{label} \cup \mathcal{T}_{pseudo}$  with sampling methods

Methods	Hospital		Rayyan	
	Cell $r_{err}$	Column $r_{err}$	Cell $r_{err}$	Column $r_{err}$
GIDCL <sub>offline</sub>	<b>0.4337</b>	<b>14</b>	<b>0.4837</b>	<b>7</b>
SBert	0.4232	8	0.4739	6
Raha	0.4002	9	0.4730	6
Random	0.3609	6	0.4697	5

Table 8. Error correction in Facilities and Inpatient

System	Facilities			Inpatient		
	P	R	F	P	R	F
GIDCL	0.77	<b>0.75</b>	<b>0.76</b>	0.90	<b>0.69</b>	<b>0.78</b>
HoloClean	<b>1.00</b>	0.61	0.75	0.96	0.22	0.36
Raha + Baran	0.50	0.31	0.38	0.64	0.44	0.52
Garf	0.96	0.28	0.44	<b>0.97</b>	0.09	0.17
GIDCL <sub>det</sub> + T5	0.21	0.22	0.21	0.16	0.20	0.18
JellyFish	0.25	0.28	0.27	0.56	0.54	0.55

**Class imbalance issues.** In Table 7 (same setting with Table 6 above), we demonstrate the ratio  $r_{err}$  of erroneous cells in the training data used for training error correction models, based on different sampling methods with the same labelling budget. The ratio is defined as  $r_{err} = \frac{\# \text{erroneous cells}}{\# \text{all cells}}$ . A ratio  $r_{err}$  close to 0.5 indicates a balanced training dataset.

Among all the datasets, GIDCL achieves the most reasonable ratios of erroneous cells and attributes (w.r.t. columns), e.g., 43.37% and 48.37% error cells, covering 14 and 7 error attributes in Hospital and Rayyan datasets respectively.

**More real-life data.** We involve two more real-life datasets Facilities and Inpatient in CMS [43] in Table 8. GIDCL also achieves the best F1-score among all error correction baselines, e.g., its F1-score is 0.86 in Inpatient, compared with 0.55, the best of others.

Table 9. System runtime (seconds, detection time + correction time)

System	Hospital	Beer	Flight	Rayyan	Tax	IMDB
HoloClean	148	96	39	112	25778	35995
Garf	186	160	120	171	11013	15233
Baran	750	114	22	26	11936	13120
GIDCL (Training)	388 + 3437	866 + 2769	212 + 2052	399 + 4256	496 + 5959	1425 + 5664
GIDCL (Inference)	16 + 1209	10 + 1325	10 + 1025	10 + 1745	477 + 878	535 + 2341

**Exp-2: Efficiency.** Table 9 reports the running time of all baselines, including both error detection and correction. In contrast to other LLM-based research [97], which often requires thousands of GPU hours and extensive GPU memory for training, GIDCL could be trained within approximately 120 minutes for most datasets, e.g., 55 minutes for error correction in Flights in consumer-level GPUs. Because GIDCL mainly adopts PLMs for error detection, it is fast for training and inference, e.g., 496s and 477s in Tax, respectively.

One notable aspect of GIDCL is its utilization of function set  $\mathcal{F}^{\text{corr}}$  and generative model  $\mathcal{M}_{\text{corr}}$  for joint data cleaning, resulting in a relatively small training and inference time that do not increase linearly with dataset size. For example, when the data sizes of Hospital and Tax increase from 1,000 to 200,000, most baselines like HoloClean and Baran exhibit a linear increase in running time. In contrast, GIDCL leverages the LLM-generated  $\mathcal{F}$  for quick detection of dirty cells over a large relational table  $\mathcal{T}$ , and then applies error correction model  $\mathcal{M}_{\text{corr}}$  only on identified dirty cells. Also, the size of training data  $D^{\text{train}}$  remains approximately the same in all datasets, such that GIDCL is insensitive with  $|\mathcal{T}|$  in runtime.

Furthermore, the inference speed of LLMs is always a bottleneck [16, 23, 97] because of the huge size of parameters. To solve it in the data cleaning task, GIDCL incorporates the vLLM technique [59] that utilizes PagedAttention to group similar queries with a KV cache, significantly speeding up inference time. However, when dealing with small datasets, e.g., Hospital, GIDCL is still not comparable with traditional data cleaning methods. Although GIDCL has various strategies to accelerate the process, it still needs to fine-tune LLM with huge number of parameters.

**Exp-3: Ablation study.** We show how error ratios for  $\mathcal{T}$  impact the performance of GIDCL and other baselines, as well as component analysis for GIDCL.

*Component analysis* In Table 10, we further analyze the impact of removing different components for GIDCL. The removal of the graph components results in the inability of the model to handle VAD errors, also prevents correction model  $\mathcal{M}_{\text{corr}}$  from obtaining high-quality demonstrations for ICL. Removing the creator component leads to a deficiency in generating additional training samples, causing both the detection model and correction model to suffer from severe overfitting issues. Without the critic component, the function set  $\mathcal{F}$  fails to extract complex semantic error patterns and contextual inconsistencies. The lack of representation learning capacity induces a drastic decline in F1. These results substantiate the indispensability of all three modules in the GIDCL framework. However, in extreme condition, not all components are effective. e.g., in Beers, the function set  $\mathcal{F}^{\text{det}}, \mathcal{F}^{\text{gen}}$  can detect and generate errors alone, and the critic component is not effective; similarly, for dataset Flights, which is dominated by VAD errors, critic component cannot generate useful  $\mathcal{T}_{\text{pseudo}}$  regarding VAD errors.

*Robustness analysis.* To evaluate the robustness of GIDCL, we investigate its performance under different error rates in the Hospital dataset by adjusting the error rate from 10% to 50% as presented in Table 11. As error rates increase, GIDCL generally demonstrates greater robustness, achieving an F-measure of 0.85 in the Hospital dataset at a 50% error rate. Despite some randomness in

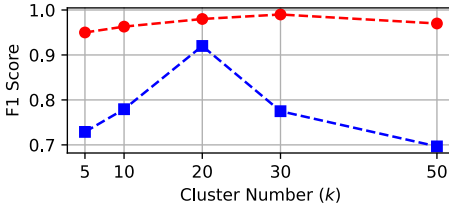
Table 10. Ablation study

System	Hospital			Tax			Rayyan			Beers			Flights		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
GIDCL	<b>0.97</b>	<b>0.96</b>	<b>0.97</b>	<b>0.95</b>	<b>0.94</b>	<b>0.95</b>	<b>0.80</b>	<b>0.93</b>	<b>0.86</b>	<b>0.97</b>	<b>0.97</b>	<b>0.97</b>	<b>0.94</b>	<b>0.92</b>	<b>0.93</b>
GIDCL <sub>offline</sub>	0.94	0.90	0.92	0.89	0.89	0.89	0.78	0.85	0.81	0.95	0.95	0.95	0.84	0.81	0.82
GIDCL w/o Graph	0.92	0.91	0.91	0.88	0.68	0.77	0.78	0.91	0.84	0.68	0.58	0.63	0.65	0.53	0.58
GIDCL w/o Creator	0.83	0.73	0.78	0.74	0.43	0.55	0.50	0.49	0.49	0.34	0.38	0.36	0.73	0.65	0.69
GIDCL w/o Critic	0.98	0.83	0.90	0.86	0.61	0.72	0.61	0.13	0.22	0.97	0.97	0.97	0.93	0.92	0.93

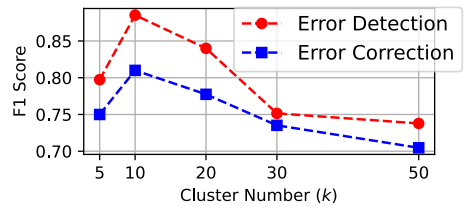
Raha+Baran, an increased error ratio means more error types are included in the 20 labeled tuples, allowing Raha+Baran to encounter a wider variety of cases. The features generated by Raha+Baran are also robust against noise. Therefore, its performance remains stable. The performance of JellyFish deteriorates sharply, highlighting that without the proposed creator-critic flow for data augmentation, LLM is prone to hallucination problems with limited training data.

Table 11. Error Correction Performance comparison by varying error rates

Dataset	Error-Rate	GIDCL			Garf			Raha + Baran			GIDCL + T5			JellyFish		
		P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
Hospital	10%	0.87	0.95	<b>0.91</b>	0.76	0.08	0.14	0.83	0.60	0.70	0.14	0.38	0.21	0.43	0.71	0.53
	20%	0.88	0.93	<b>0.90</b>	0.98	0.05	0.10	0.80	0.63	0.71	0.07	0.38	0.12	0.41	0.66	0.51
	30%	0.84	0.92	<b>0.88</b>	0.98	0.02	0.04	0.82	0.68	0.74	0.04	0.36	0.08	0.33	0.65	0.44
	40%	0.82	0.93	<b>0.87</b>	0.97	0.01	0.03	0.83	0.63	0.72	0.03	0.36	0.05	0.28	0.67	0.39
	50%	0.80	0.93	<b>0.85</b>	0.95	0.01	0.02	0.74	0.57	0.65	0.03	0.36	0.05	0.23	0.66	0.35



(a) Hospital Dataset



(b) Rayyan Dataset

Fig. 6. Performance w.r.t. the number of clusters  $k$ 

*Varying different  $k$ .* We vary the number of clusters  $k$  to evaluate error detection and correction of GIDCL in Hospital and Rayyan. The trend of F1-score increases first and then declines later when  $k$  increases. This shows that a medium value of  $k$  would lead to the best performance, e.g.,  $k = 20$  and 10 for Hospital and Rayyan, respectively. A small  $k$  leads to clusters with large sizes such that non-similar tuples are involved in the same clusters so that representative tuples are inclined to be less representative; a large  $k$  leads to skewed distribution in clusters, s.t., most clusters contain few tuples, e.g., fewer than 10, which deteriorate the quality of RAG of LLM in GIDCL<sub>cor</sub>.

## 8 Related Work

We categorize the data cleaning algorithms as follows.

**Error detection.** As the first step in the data cleaning pipeline, there are a host of error detection

algorithms that could be classified into two categories. (1) *Rule-based methods*. Rule-based methods usually adopt different types of rules to find violations in data using various detection methods, e.g., FDs [5, 11], DCs [19, 44, 47, 89], CFDs [13, 32, 33], PFDs [83], REEs [40, 42], user-defined rules [31] and manually-defined parameters [8, 82]. Considering the difficulty of handcrafting data quality rules, several rule discovery algorithms [12, 18, 34, 36, 37, 67, 78] are proposed to find hidden patterns in data. (2) *Data-driven methods*. There are many data-driven methods that detect errors based on statistical and ML models in the supervised and unsupervised learning manners. Supervised methods [50, 70, 72, 76, 80, 108] require users to provide a few labeled data and then design machine learning models to identify errors in data. Statistical hypothesis [102], co-occurrence dependency [54], ActiveClean [58], meta-data [100] and rule generation [79] aim to learn abnormal data in an unsupervised learning way. Different from previous works, GIDCL targets at generating detection rules and self-labeled data in an automated and sufficient manner, a data-driven method without prior knowledge, which can break the limitation of domains and languages.

**Error correction.** After identifying erroneous data, data cleaning needs the error correction part to repair. We mainly classify the error correction approaches as follows. (1) *Rule-based methods*. Similar with error detection, FDs, CFDs, denial constraints, PFDs and REEs are mainly used to correct errors using various repairing mechanism, e.g., heuristic fixes [6, 9, 10, 21, 22, 28, 45, 46, 48, 49, 87, 93, 101, 104], certain fixes [38–41, 90]. (2) *ML models*. ML-based methods adopt ML models with handcrafted features for data cleaning. SCARE [103] designs methods that combine ML models and likelihood approach for data repair. HoloClean [89] uses pre-defined denial constraints and MDs as features and designs a factor graph for error correction. There are also generative models [24, 30, 53, 60, 73, 91] that adopt probabilistic inference to iteratively clean data in the generative mode given some injected prior knowledge. Many ML models focus on imputing missing values, e.g., autoencoder [79] and GAIN [105]. Data cleaning are also employed to improve downstream ML models, e.g., [27, 62, 66]. (3) *Hybrid methods*. which unify logical rules and machine learning models for data cleaning, e.g., Baran [68] adopts rule-based features to find inconsistencies in data and train traditional ML models to find suitable repairs for dirty cells. There is also a host of work [51, 56, 57, 63, 95] to apply DC for improving downstream ML performance. (4) *external data based methods*. These methods refer to various external data to help data repair, e.g., master data [39], knowledge bases [20] and web table [3]. Different from previous works, GIDCL adopt LLMs-dominated model to jointly generate correction value and data cleaning rules, the whole process is completely data-driven without human and external information.

## 9 Conclusion

Data cleaning plays a pivotal role in numerous applications. GIDCL introduced an end-to-end data cleaning framework that consists of a user labeling mechanism based on graph neural network, a creator-critic workflow for error detection and a LLM-based error correction. Even with a limited number of label data, GIDCL maintains high accuracy and provides a degree of interpretability. The experimental results show that GIDCL outperforms the existing data cleaning approaches by at least 10% F-measure on average, verifying the proposed framework is effective in various scenarios.

## Acknowledgments

This work was supported by the National Science and Technology Major Project (No. 2022ZD0117800), China NSFC 62225202, Longhua Science and Technology Innovation Bureau 10162A20220720B12AB12, Beijing Life Science Academy (No.2024900CB0080) and the Fundamental Research Funds for the Central Universities (No.226-2024-00030).



## References

- [1] 2023. IMDB. <https://www.imdb.com/interfaces/>.
- [2] 2024. Code, datasets and full version. <https://github.com/SICS-Fundamental-Research-Center/GIDCL>.
- [3] Ziawasch Abedjan, Cuneyt G Akcora, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. 2015. Temporal rules discovery for web data cleaning. *Proceedings of the VLDB Endowment* 9, 4 (2015), 336–347.
- [4] 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? *Proceedings of the VLDB Endowment* 9, 12 (2016), 993–1004.
- [5] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Vol. 8. Addison-Wesley Reading.
- [6] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.
- [7] Patricia C Arocena, Boris Glavic, Giansalvatore Mecca, Renée J Miller, Paolo Papotti, and Donatello Santoro. 2015. Messing up with BART: error generation for evaluating data-cleaning algorithms. *Proceedings of the VLDB Endowment* 9, 2 (2015), 36–47.
- [8] Laure Berti-Equille, Tamraparni Dasu, and Divesh Srivastava. 2011. Discovery of complex glitch patterns: A novel approach to quantitative data cleaning. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 733–744.
- [9] Leopoldo Bertossi. 2011. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers.
- [10] Leopoldo E. Bertossi, Solmaz Kolahi, and Laks V. S. Lakshmanan. 2011. Data cleaning and query answering with matching dependencies and matching functions. In *ICDT*. 268–279. <https://doi.org/10.1145/1938551.1938585>
- [11] George Beskales, Ihab F Ilyas, Lukasz Golab, and Artur Galiullin. 2013. On the relative trust between inconsistent data and inaccurate constraints. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 541–552.
- [12] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient denial constraint discovery with hydra. *Proceedings of the VLDB Endowment* 11, 3 (2017), 311–323.
- [13] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional Functional Dependencies for Data Cleaning. In *ICDE*. 746–755. <https://doi.org/10.1109/ICDE.2007.367920>
- [14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.
- [15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs.CL]*
- [16] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023. LongLoRA: Efficient Fine-tuning of Long-Context Large Language Models. *arXiv preprint arXiv:2309.12307* (2023).
- [17] Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. 2023. Seed: Domain-specific data curation with large language models. *arXiv e-prints* (2023), arXiv–2310.
- [18] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *Proceedings of the VLDB Endowment* 6, 13 (2013), 1498–1509.
- [19] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. IEEE, 458–469.
- [20] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1247–1261.
- [21] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving data quality: Consistency and accuracy. In *VLDB*.
- [22] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving Data Quality: Consistency and Accuracy. In *VLDB*. 315–326.
- [23] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.
- [24] Sushovan De, Yuheng Hu, Venkata Vamsikrishna Meduri, Yi Chen, and Subbarao Kambhampati. 2016. BayesWipe: A Scalable Probabilistic Framework for Improving Data Quality. *Journal of Data and Information Quality* 8, 1 (2016), 5:1–5:30. <https://doi.org/10.1145/2992787>
- [25] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2022. Turl: Table understanding through representation learning. *ACM SIGMOD Record* 51, 1 (2022), 33–40.
- [26] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

- [27] Daniel Deutch, Nave Frost, Amir Gilad, and Oren Sheffer. 2021. Explanations for Data Repair Through Shapley Values. In *CIKM*. 362–371. <https://doi.org/10.1145/3459637.3482341>
- [28] Xiaou Ding, Hongzhi Wang, Jiaxuan Su, Muxian Wang, Jianzhong Li, and Hong Gao. 2020. Leveraging currency for repairing inconsistent and incomplete data. *TKDE* (2020).
- [29] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey for in-context learning. *arXiv preprint arXiv:2301.00234* (2022).
- [30] Prashant Doshi, Lloyd G Greenwald, and John R Clarke. 2003. Using Bayesian Networks for Cleansing Trauma Data. In *FLAIRS*. 72–76.
- [31] Amr Ebaid, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, Jorge-Arnulfo Quiane-Ruiz, Nan Tang, and Si Yin. 2013. NADEEF: A generalized data cleaning system. *Proceedings of the VLDB Endowment* 6, 12 (2013), 1218–1221.
- [32] Wenfei Fan, Floris Geerts, and Xibei Jia. 2008. Semandaq: a data quality system based on conditional functional dependencies. *PVLDB* 1, 2 (2008), 1460–1463. <https://doi.org/10.14778/1454159.1454200>
- [33] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Transactions on Database Systems* 33, 2 (2008), 6:1–6:48. <https://doi.org/10.1145/1366102.1366103>
- [34] Wenfei Fan, Floris Geerts, Laks V. S. Lakshmanan, and Ming Xiong. 2009. Discovering Conditional Functional Dependencies. In *ICDE*. 1231–1234. <https://doi.org/10.1109/ICDE.2009.208>
- [35] Wenfei Fan, Floris Geerts, Jianzhong Li, and Ming Xiong. 2010. Discovering conditional functional dependencies. *IEEE Transactions on Knowledge and Data Engineering* 23, 5 (2010), 683–698.
- [36] 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. <https://doi.org/10.1109/TKDE.2010.154>
- [37] Wenfei Fan, Ziyan Han, Yaoshu Wang, and Min Xie. 2022. Parallel Rule Discovery from Large Datasets by Sampling. In *SIGMOD*, Zachary Ives, Angela Bonifati, and Amr El Abbadi (Eds.). 384–398.
- [38] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2011. CerFix: A System for Cleaning Data with Certain Fixes. *PVLDB* 4, 12 (2011), 1375–1378.
- [39] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2012. Towards certain fixes with editing rules and master data. *The VLDB journal* 21 (2012), 213–238.
- [40] Wenfei Fan, Ping Lu, and Chao Tian. 2020. Unifying logic rules and machine learning for entity enhancing. *Science China Information Sciences* 63 (2020), 1–19.
- [41] Wenfei Fan, Ping Lu, Chao Tian, and Jingren Zhou. 2019. Deducing Certain Fixes to Graphs. *PVLDB* 12, 7 (2019), 752–765.
- [42] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel Discrepancy Detection and Incremental Detection. *PVLDB* 14, 8 (2021), 1351–1364.
- [43] Centers for Medicare and Medicaid Services. [n.d.]. Provider data catalog. <https://data.cms.gov/provider-data/>.
- [44] Congcong Ge, Yunjun Gao, Xiaoye Miao, Bin Yao, and Haobo Wang. 2020. A hybrid data cleaning framework using markov logic networks. *IEEE Transactions on Knowledge and Data Engineering* 34, 5 (2020), 2048–2062.
- [45] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2013. The LLUNATIC data-cleaning framework. *Proceedings of the VLDB Endowment* 6, 9 (2013), 625–636.
- [46] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2020. Cleaning data with llunatic. *The VLDB Journal* 29 (2020), 867–892.
- [47] Stella Giannakopoulou, Manos Karpathiotakis, and Anastasia Ailamaki. 2020. Cleaning denial constraint violations through relaxation. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 805–815.
- [48] Amir Gilad, Daniel Deutch, and Sudeepa Roy. 2020. On multiple semantics for declarative database repairs. In *SIGMOD*. 817–831.
- [49] Shuang Hao, Nan Tang, Guoliang Li, Jian He, Na Ta, and Jianhua Feng. 2016. A novel cost-based model for data repairing. *IEEE Transactions on Knowledge and Data Engineering* 29, 4 (2016), 727–742.
- [50] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data*. 829–846.
- [51] Benjamin Hilprecht, Christian Hammacher, Eduardo S Reis, Mohamed Abdelaal, and Carsten Binnig. 2023. Diffml: End-to-end differentiable ML pipelines. In *Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning*. 1–7.
- [52] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).
- [53] Yuheng Hu, Sushovan De, Yi Chen, and Subbarao Kambhampati. 2012. Bayesian Data Cleaning for Web Data. *CoRR* abs/1204.3677 (2012). [arXiv:1204.3677](https://arxiv.org/abs/1204.3677) <http://arxiv.org/abs/1204.3677>
- [54] Zhipeng Huang and Yeye He. 2018. Auto-detect: Data-driven error detection in tables. In *Proceedings of the 2018 International Conference on Management of Data*. 1377–1392.

- [55] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825* (2023).
- [56] Bojan Karlaš, Peng Li, Renzhi Wu, Nezihe Merve Gürel, Xu Chu, Wentao Wu, and Ce Zhang. 2020. Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *arXiv preprint arXiv:2005.05117* (2020).
- [57] Sanjay Krishnan, Michael J Franklin, Ken Goldberg, and Eugene Wu. 2017. Boostclean: Automated error detection and repair for machine learning. *arXiv preprint arXiv:1711.01299* (2017).
- [58] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment* 9, 12 (2016), 948–959.
- [59] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. *arXiv preprint arXiv:2309.06180* (2023).
- [60] Alexander K. Lew, Monica Agrawal, David A. Sontag, and Vikash Mansinghka. 2021. PClean: Bayesian Data Cleaning at Scale with Domain-Specific Probabilistic Programming. In *AISTATS*. 1927–1935. <http://proceedings.mlr.press/v130/lew21a.html>
- [61] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.
- [62] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. CleanML: A Study for Evaluating the Impact of Data Cleaning on ML Classification Tasks. In *ICDE*. 13–24.
- [63] Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. CleanML: A study for evaluating the impact of data cleaning on ml classification tasks. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 13–24.
- [64] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. 2015. Truth finding on the deep web: Is the problem solved? *arXiv preprint arXiv:1503.00303* (2015).
- [65] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
- [66] Zifan Liu, Zhechun Zhou, and Theodoros Rekatsinas. 2022. Picket: guarding against corrupted data in tabular data during learning and inference. *VLDB Journal* 31, 5 (2022), 927–955.
- [67] Ester Livshits, Alireza Heidari, Ihab F Ilyas, and Benny Kimelfeld. 2020. Approximate denial constraints. *arXiv preprint arXiv:2005.08540* (2020).
- [68] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective error correction via a unified context representation and transfer learning. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1948–1961.
- [69] Mohammad Mahdavi and Ziawasch Abedjan. 2021. Semi-Supervised Data Cleaning with Raha and Baran. In *11th Conference on Innovative Data Systems Research, CIDR 2021, Virtual Event, January 11-15, 2021, Online Proceedings*. www.cidrdb.org.
- [70] 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*. 865–882.
- [71] Heikki Mannila and Kari-Jouko Räihä. 1994. Algorithms for inferring functional dependencies from relations. *Data & Knowledge Engineering* 12, 1 (1994), 83–99.
- [72] Zhengjie Miao, Yuliang Li, and Xiaolan Wang. 2021. Rotom: A meta-learned data augmentation framework for entity matching, data cleaning, text classification, and beyond. In *Proceedings of the 2021 International Conference on Management of Data*. 1303–1316.
- [73] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L Ong, and Andrey Kolobov. 2007. 1 Blog: Probabilistic Models with Unknown Objects. *Statistical Relational Learning* (2007), 373.
- [74] Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2021. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943* (2021).
- [75] Avani Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911* (2022).
- [76] Felix Neutatz, Mohammad Mahdavi, and Ziawasch Abedjan. 2019. Ed2: A case for active learning in error detection. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 2249–2252.
- [77] Mourad Ouzzani, Hossam Hammady, Zbys Fedorowicz, and Ahmed Elmagarmid. 2016. Rayyan—a web and mobile app for systematic reviews. *Systematic reviews* 5 (2016), 1–10.
- [78] Eduardo HM Pena, Eduardo C De Almeida, and Felix Naumann. 2019. Discovery of approximate (and exact) denial

- constraints. *Proceedings of the VLDB Endowment* 13, 3 (2019), 266–278.
- [79] Jinfeng Peng, Derong Shen, Nan Tang, Tieying Liu, Yue Kou, Tiezheng Nie, Hang Cui, and Ge Yu. 2022. Self-supervised and Interpretable Data Cleaning with Sequence Generative Adversarial Networks. *Proceedings of the VLDB Endowment* 16, 3 (2022), 433–446.
  - [80] Minh Pham, Craig A Knoblock, Muhao Chen, Binh Vu, and Jay Pujara. 2021. SPADE: A Semi-supervised Probabilistic Approach for Detecting Errors in Tables. In *IJCAI*. 3543–3551.
  - [81] Jose Picado, John Davis, Arash Termehchy, and Ga Young Lee. 2020. Learning over dirty data without cleaning. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1301–1316.
  - [82] Clement Pit-Claudel, Zelda Mariet, Rachael Harding, and Sam Madden. 2016. Outlier detection in heterogeneous datasets using automatic tuple expansion. (2016).
  - [83] Abdulhakim Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern functional dependencies for data cleaning. *PVLDB* 13, 5 (2020), 684–697.
  - [84] Jianbin Qin, Sifan Huang, Yaoshu Wang, Jing Zhu, Yifan Zhang, Yukai Miao, Rui Mao, Makoto Onizuka, and Chuan Xiao. 2023. BClean: A Bayesian Data Cleaning System. *arXiv preprint arXiv:2311.06517* (2023).
  - [85] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.
  - [86] Erhard Rahm, Hong Hai Do, et al. 2000. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.* 23, 4 (2000), 3–13.
  - [87] Joeri Rammelaere and Floris Geerts. 2018. Explaining repaired data with CFDs. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1387–1399.
  - [88] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv:1908.10084* [cs.CL]
  - [89] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820* (2017).
  - [90] 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.
  - [91] Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. 2019. A Formal Framework for Probabilistic Unclean Databases. In *ICDT*. 6:1–6:18. <https://doi.org/10.4230/LIPIcs.ICDT.2019.6>
  - [92] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *ESWC*.
  - [93] Shaoxu Song, Han Zhu, and Jianmin Wang. 2016. Constraint-variance tolerant data repairing. In *Proceedings of the 2016 International Conference on Management of Data*. 877–892.
  - [94] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*. 645–654.
  - [95] Ki Hyun Tae, Yuji Roh, Young Hun Oh, Hyunsu Kim, and Steven Euijong Whang. 2019. Data cleaning for accurate, fair, and robust models: A big data-AI integration approach. In *Proceedings of the 3rd international workshop on data management for end-to-end machine learning*. 1–4.
  - [96] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Sam Madden, and Mourad Ouzzani. 2020. RPT: relational pre-trained transformer is almost all you need towards democratizing data preparation. *arXiv preprint arXiv:2012.02469* (2020).
  - [97] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
  - [98] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *International conference on machine learning*. PMLR, 2071–2080.
  - [99] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. 2019. Composition-based multi-relational graph convolutional networks. *arXiv preprint arXiv:1911.03082* (2019).
  - [100] Larysa Visengeriyeva and Ziawasch Abedjan. 2018. Metadata-driven error detection. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. 1–12.
  - [101] Jiannan Wang, Sanjay Krishnan, Michael J Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. 2014. A sample-and-clean framework for fast and accurate query processing on dirty data. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 469–480.
  - [102] Pei Wang and Yeye He. 2019. Uni-detect: A unified approach to automated error detection in tables. In *Proceedings of the 2019 International Conference on Management of Data*. 811–828.
  - [103] 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 2013 ACM SIGMOD International Conference on Management of Data*. 553–564.
- [104] Mohamed Yakout, Ahmed K Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F Ilyas. 2011. Guided data repair. *arXiv preprint arXiv:1103.3103* (2011).
  - [105] Jinsung Yoon, James Jordon, and Mihaela Schaar. 2018. Gain: Missing data imputation using generative adversarial nets. In *International conference on machine learning*. PMLR, 5689–5698.
  - [106] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.
  - [107] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2023. Jellyfish: A Large Language Model for Data Preprocessing. *arXiv preprint arXiv:2312.01678* (2023).
  - [108] Yaru Zhang, Jianbin Qin, Yaoshu Wang, Muhammad Asif Ali, Yan Ji, and Rui Mao. 2023. TabMentor: Detect Errors on Tabular Data with Noisy Labels. In *International Conference on Advanced Data Mining and Applications*. Springer, 167–182.
  - [109] Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. 2023. Large language models are not robust multiple choice selectors. In *The Twelfth International Conference on Learning Representations*.
  - [110] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang. 2021. Deep graph structure learning for robust representations: A survey. *arXiv preprint arXiv:2103.03036* 14 (2021).

Received April 2024; revised July 2024; accepted August 2024