RESPONSE TO REVIEWS PAPER

Dear Meta Reviewer and Referees:

We have substantially revised the paper following the Referees' suggestions. For the convenience of Referees, changes to the paper are color-coded in blue. We would like to thank the Referees for insightful comments and for supporting this work! Below please find our responses to the reviews. We have changed the name of our framework from GEIL to GIDCL based on R1.

Response to the comments of the Meta Reviewer.

[MetaO1] *The authors should present a plan to address all required revision points. Specifically, points O1, O2, O3 by Reviewer 1, and points O1, O2, O3, O6 and O7 by Reviewer 4.*

[A] We have addressed all required revision points, specifically points R1[O1], R1[O2], R1[O3], R4[O1], R4[O2], R4[O3], R4[O6] and R4[O7].

[MetaO3] *Finally, the authors should address the concerns on interpretability raised by Reviewer 2 (point O1) with an appropriate extension of their discussion and experimental evaluation (qualitative examples should also be presented).*

[A] We have enhanced the motivation by providing a formal definition of interpretability in data cleaning (Section 1, pp. 1) and Example 5 (pp. 7), 6 (pp. 7) and 8 (pp. 9) to show the interpretable patterns and rules generated by GIDCL (R2[O1]). Furthermore, we listed all discovered FDs, full rule set $\mathcal{F}$ generated by LLMs and selected tuples with interpretability in our full version [2].

We also provided more experiments in Table 7 (pp. 12) to show that the interpretable rules generated by GIDCL could boost the performance of data cleaning in a variety of error types.

Besides, we have added some theoretical discussion of GIDCL (R2[O2]) in Section 4 (pp. 6) and Section 6 (pp. 9).

Response to the comments of Referee #1.

[R1O1-A] *The experimental results can be significantly expanded. First the authors should clarify to which extent the benefits of their correction is stemming from the already high-performing detection. I suggest the authors picked GEIL as the detector for all competitors as they already show that it is finding more errors than the other detection approaches, (similar to GEIL-Holoclean).*

[A] Thanks! In Table 11 (pp.11) and corresponding analysis, we have added experiments by varying different error detectors and correction models (Here we define positive sample as clean values, and vice versa).

By varying different error detectors and unifying $\text{GIDCL}_{\text{cor}}$ as error correction model, we justify that even with low-performing error detection result, our correction model $\text{GIDCL}_{\text{cor}}$ still have robustness and stability in correcting true negatives (TN) and neglecting false negatives (FN). For detector Rotom in dataset Rayyan, it achieves 0.40 f1-score in error detection with 1,376 FN cases; based on above detection results, $\text{GIDCL}_{\text{cor}}$ still achieves 0.71 f1-score in error correction, by decreasing FN cases to 100 during correction procedure.

Similarly, by applying $\text{GIDCL}_{\text{det}}$ and varying different correction models, we also find the high performance of $\text{GIDCL}_{\text{det}}$ significantly boost the performance of downstream 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, $\text{GIDCL}_{\text{det}}$ adds 148 FP cases that are neglected by Raha detector, which leads to non-trivial performance boosting of correction model Baran from 0.28 to 0.57.

[R1O1-B] *Details on how k is chosen needs to be discussed. Do you choose large or small k?*

[A] Thanks! We adopted the elbow method [8] to select the values of k for the k-means clustering for each dataset, incorporating manual refinements. The specific k of each dataset is listed in Table 3. The experimental results indicate the effectiveness of this strategy, as demonstrated in Table 4, 5 (pp. 11) and Table 10 (pp. 13).

Furthermore, to evaluate which values of k indeed impact the performance, we analyzed the effects of varying k from 5 to 50 in Figure 7 (pp. 12). We observe that a smaller k leads to larger clusters, which involves non-related tuples into the same clusters. As a result,the procedure of representative tuple selection, as well as RAG in training correction model $\mathcal{M}_{\text{corr}}$, are becoming more difficult. On the contrary, a larger k leads to skewed distribution in clusters, s.t. most clusters contains few examples, which also deteriorate the quality of RAG and pseudo-labeled data $\mathcal{T}_{\text{pseudo}}$.

[R1O1-C] *Runtime discussion should be properly labeled as it suggests that only correction was considered.*

[A] Thanks! In Table 6 (pp. 11), we have separately provided the training and inference time for error detection $\text{GIDCL}_{\text{det}}$ and error correction $\text{GIDCL}_{\text{cor}}$ and gave detailed discussions (pp. 11-12).

[R1O1-D] *The authors describe some post processing with* FDs. *It comes across as ad-hoc and it is very hard for the reader to fully grasp the approach and its goals.*

[A] Thanks! In Section 6 (pp. 10), we expand the detailed method for the post processing with FDs, including the steps of re-learning graph structure and error correction, mainly tackling with inconsistency errors(*w.r.t.* VAD errors). Specifically, we provide the objective function in Eq.1 and Eq.3. By the above clarification, we unified the post processing as link prediction task with modified GCN. We also provide motivation for such design, which can effectively overcome the potential over-smoothing issue for GCN structure.

Besides, we also provide all the discovered FDs in full version[2] for better illustration. In Table 7, we extend the ablation study to 5 datasets, and GIDCL w/o Graph also indicated the effectiveness of our proposed GSL methods.

[R1O1-E] *The component analysis can be improved. Why were certain datasets selected and some not? Also a deeper discussion with reference to individual datasets would help to better understand the benefits of individual components.*

[A] Thanks! In Table 7 (pp. 12), we extend our ablation study to all public datasets in previous work Baran [76], which can cover all error types. Besides, we also add additional analysis for individual components, and discussed several extreme conditions where datasets are dominated by certain error types, and not all components are effective.

[R1O2] *The authors do not discuss the problem of class-imbalance. I am a bit surprised that their approach for active learning gives them insights on erroneous cells. The learning is only considering semantic and syntactic relationships between tuples. It is not intuitively clear*

why among those clusters erroneous cells/tuples could be selected. Without examples on errors among 20 selected labels it will be impossible for the system to find proper detectors/correctors. My impression is that the experiments reported are averaged over many runs? Or is there any sort of warmstarting to ensure that both classes are among the labeled tuples? If so the authors should clarify this influence.

[A] Thanks! To address the mentioned concerns, we conduct some theoretical discussions and experiments to evaluate the sampling strategy and its impact on the class imbalance issue.

First, in Section 4, we have added a Remark on pp. 6 to clarify our motivation for selecting potential erroneous tuples based on clustering principles. Based on the assumption that $\mathcal{T}$ contains small number of errors and that erroneous tuples have explicit or implicit inconsistencies with others, error detection can be conceptualized as an anomaly detection problem for tuples. To address this issue, we introduced a formal objective function in Eq.2.

Secondly, by incorporating GCN (one of a type of GNN structure) for graph structure learning (GSL), it could learn high-order relationships between tuples and attributes through the message passing mechanism of GCN, extending beyond the detection of semantic and syntactic errors.

Next, to further justify the aforementioned intuition, we presented a new experiment in Table 10 (pp. 12). By varying different sampling methods, we demonstrated that incorporating graph structure can consistently retrieve representative labelling tuples for $\mathcal{T}_{\text{label}}$, and achieved better performance across different datasets.

Finally, regarding the class imbalance problem, we also provide a new discussion in Table 8 (pp. 11), which shows that, even although the initial error rate in $\mathcal{T}_{\text{label}}$ is small, generating pseudo training data $\mathcal{T}_{\text{pseudo}}$ with the cooperation of the creator and critic components in $\text{GIDCL}_{\text{det}}$, effectively balances the class distribution of training data D^{train}.

For the setting of experiment, we have clarified that our experiment runs three times, and reported the average result.

[R1O3] *Similar to O2 it is suprising to me that the 20 labels lead to tuples that result in sufficiently different detection/correction rules by the* LLM*. The examples provided rather capture very simple rules. The authors should provide intuition why this works.*

[A] Thanks! As discussed in [R1O2], we added a Remark on pp. 6 to show GIDCL could select potential erroneous tuples, and two Remarks on pp. 8 to discuss our strategies for augmenting the 20 training instances. By integrating the strategies of GIDCL with the few shot capabilities of LLMs, GIDCL can generate detection and correction rules effectively. Table 8 (pp. 11) also demonstrated that our selection methods can cover most of error attributes among all baseline methods. Due to space limit, we provided all generated rules over all datasets in Appendix of full version[2].

[R1O4] *The authors mention hand-crafted prompts. Now this was surprising because I assumed that the prompts are automatically generated. The authors should clarify this further.*

[A] Thanks! Only three prompts needed to be hand-crafted and they only need to be written once. We have clarified the above setting in Section 7 (pp. 11), and have listed all used prompts in the appendix in our full version[2].

[R1D&I] *The acronym of the paper is GEIL which is a german expression for "horny". Now this does not strictly violate D&I compliance but it is not a proper term to be used in professional context. I suggest the authors to change it to something with less sexual connotation.*

[A] Thanks! We change the name to GIDCL .

[R1Q1] *The authors claim that prior work requires "Extensive manual feature engineering". This is misleading as the feature engineering is done during algorithm creation and not burdened on the user. Unless the authors want to also say that their approach requires extensive manual model design.*

[A] Thanks! In Section 1 (pp. 1 and pp. 2), we clarify that prior work involves extensive feature engineering during the algorithm creation phase, which is not necessarily a burden on the end-user, but for the designer. We also differentiate our approach by explaining how it minimizes the need for manual feature engineering or model design, highlighting any automated aspects that simplify the process for the user.

[R1Q2] *The example contains an odd color coding blue for error and red for cleaned version. This might be confusing for some readers.*

[A] Thanks! In Table 1, we marked the error cells in red and the cleaned version in blue.

[R1Q3] *Following the setting of existing data cleaning work, we treat the original datasets as clean data, and dirty data is generated by adding noise with a certain rate e%, i.e., the percentage of dirty cells on all data cells. This statement is not correct as at least* Flights *and* Rayyan *contain real errors as described by the references.*

[A] Thanks! In Section 7 (pp. 10), we have revised it.

[R1Q4] *typo: "And captures"*

[A] Thanks! We have fixed it.

Response to the comments of Referee #2.

[R2Q1] *I am not convinced about the interpretability aspect which is very crucial for data cleaning. I didn't see any convincing experiment that shows the approach provides interpretability.*

[A] Thanks! we have revised several key aspects in the revised version to address the above concern for interpretability.

In Section 1, we clarify the definition of interpretability, which contains data cleaning patterns, rules and dependencies that can be explicitly verified by human beings. We have also listed all the extracted patterns, rules and dependencies of all datasets (in Appendix for full version[2]), which shows the the interpretability of our approach.

To justify the importance of interpretability, in experiment aspect, we provide expanded ablation study in Table 7, among with additional component analysis in pp. 12, to illustrate the importance of extracted interpretable patterns (*w.r.t.* creator component, generating patterns with LLM) and dependencies (*w.r.t.* graph component, extracting and applying functional dependencies(FDs)). In most conditions, removing the above components will lead to drastically drop in error correction performance, which justified the importance of interpretability.

[R2Q2] *It's another adhoc method, and does not provide any theoretical justification for proposing the framework.*

[A] Thanks! In the revised version, we provide several theoretical motivation and experimental justification to support the design of

the proposed framework GIDCL .

Similar to the response for **[R1O2]**, we add Remark in pp. 6, to illustrate our theoretical motivation for selecting potential errorneous tuples via clustering basis. Based on the assumption that $\mathcal{T}$ contains small number of errors, while errorneous tuples have explicit or implicit inconsistencies with others, we treat error detection as the anomaly detection process. To solve such problem, we provide a formal objective function in Eq.2 for outlier detection. Table 8 prove that our proposed sampling methods can cover most error attributes among all baselines.

Secondly, by introducing GCN (one of GNN structure) for graph structure learning(GSL), it can learn high-order relations across tuples and attributes via message passing mechanism by GCN, and not limited to semantic and syntactic errors. In Section 6 (pp. 10), we also add theoretical motivation to illustrate why our proposed GSL method can propagate correct information through modified message passing mechanism.

To justify the above theoretical analysis and framework design, we introduce additional experiment in Table 10 (pp. 12). By varying different sampling methods, we prove that considering graph structure can consistently retrieve representative labelling tuples for $\mathcal{T}_{\text{label}}$, and achieve better performance for different datasets.

We also add deeper discussion, noted as Remark in the end of Section 4/5/6, to emphasize the correlation of each components in GIDCL . Expanded ablation study in Table 7 also justifies the necessity and functionality of each component of our framework, which is a systematic effort for data cleaning tasks.

[R2Q3] *The datasets used for experiments (especially given this is mostly an experimental paper) do not show all variability IMO. Also the experiments on interpretability is not convincing.*

[A] Thanks! We add 2 real-world dataset Inpatient and Facilities in Table 12 (pp. 13) from CMS [47], containing real-world errors. We have also clarified datasets Beers,Flights,Rayyan contain real-world errors and cleaned by data owners[75].

Response to the comments of Referee #4.

[R4O1] *The presentation should be improved. The overall architecture section leaves many questions, some of which are covered later. There are also quite a fewgrammar/syntax/typo errors that should be fixed to improve clarity and readability (see minor remarks). Even in the detailed sections (mostly in Section 4), there are still vaguepoints and unjustified decisions.*

[A] We have polished the paper and revised the technique part as follows. (1) We fixed a few grammar, syntax and typo errors; (2) We define a clearer notations in Table 2, illustrating several key definition, *e.g.,* $\mathcal{T}_{\text{coreset}}$ and $f^{\text{det}}_{A_i}$ in the beginning of Section 3; and (3) we address some vague points and unjustified decisions in Sections 3-6, all highlighted in blue color.

[R4O1.1] *How is SentenceBert used exactly for initialization?*
[R4O1.2] *Why was CompGCN used over other methods, like ComplEX with N3 (Lacroix et al. 2018, Canonical Tensor Decomposition for Knowledge Base Completion)? This should be more properly justified in Section 4 and/or Section 8.*

[A] Thanks! We have added details of SentenceBert and discussions why CompGCN is used in Section 4 (pp. 6).

[R4O1.3] *How were anomalous vertices detected exactly?*

[R4O1.4] *What is the formal objective (function) of selecting* θ_{label} *tuples for manual labeling? In other words, which theta tuples are considered better than theta other tuples and why? Would any theta tuples do the work? What if (all/most) those theta tuples are already clean? This is also related to O5 discussed later, about the lack of evaluation for this component.*

[A] Thanks! Anomalous vertices are extracted from the outlier node of each clusters in C by distance from centroid node of each cluster, then further filtered with $\text{Dist}(C_l)$. We have added a formal objective function and solution in Section 4 (pp. 6) to illustrate how we select θ_{label} representative tuples for manual labeling. We further conducted experiments and discussed different $\mathcal{T}_{\text{label}}$ selection methods for data cleaning performance in Table 10 (pp. 12).

[R4O1.5] *Why is the number of fake tuples the same as the number of correct tuples? Would other options offer better results? What is the intuition/justification behind this choice?*

[A] Thanks! Following RGCN [102], we generate m fake tuples for each correct tuple, with $m = 1$ being the default value [102] and proven to be effective. Our intuition is that the same number of fake and correct tuples result in class-balance data, which is beneficial for graph learning. We have clarified it in Section 4 (pp. 7).

[R4O1.6] $\text{Serial}(t_i)$ *is not properly described/defined.*
[R4O1.7] *How can both* (t_3, t_3^+) *and* $t_{3,2}$ *be elements of the same set,* $\mathcal{T}_{label}$*, as written in Example 4?*
[R4O1.8] *In Example 7, don't describe what RAG contains; write it explicitly. E.g.,* $\text{Serial}(t_1) = \cdots$ *. Then, after the last bullet about RAG, you can say that* t_1 *and* t_2 *are in the same cluster.*

[A] Thanks! We have revised Example 4 (Section 5, pp.7) and 7 (Section 6, pp. 9) to explain Serial, $\mathcal{T}_{\text{label}}$ and RAG more clearly.

[R4O1.9] *In problem statement, "as many errors with* $\mathcal{T}$ *as possible" is not a clear/formal description. What does this mean exactly? Why not all? Possible for whom/which method?*

[A] We made the formal problem definition clear in Section 2.

[R4O1.10] GNNs *are defined on* (V, E, L) *graphs... The next sentence mentions the attributes of* $\mathcal{G}$*. If you want to use attributes, then include them in the definition of the graph, or at least define what those attributes are.*

[A] We revised the definition of $\mathcal{G}$ with attributes in Section 4.

[R4O1.11] *In error correction* $\text{GIDCL}_{\text{cor}}$*, page 5:* D_{train} *is defined as* $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}$*. Then I am really confused with the following sentence: "The contextual information includes clean and representative examples as context from* D_{train}*, as well as labeled tuples for demonstration from* $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}$*." Isn't* D_{train} *the same as* $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}$*?*

[A] Thanks! We have remarked that D_{train} is equivalent to $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}$ and revised the statement in Section 3 (pp. 5).

[R4O2] *No discussion on the methodology about data imputation on missing values, although it's one of the supported types of noise and shown in the example. Based on the text, it is not straightforward to understand how the authors deal with that, so it should either be explicitly described, or it should be mentioned that it is not handled.*

[A] Thanks! We added a remark paragraph in Section 6 (pp. 10). In current experiment, if the cell $t_{i,j}$ is null, we treat the missing value

$t_{i,j}$ as error cell by default, and try to correct $t_{i,j}$ with $\text{GIDCL}_{\text{cor}}$. However, if $t^{+}_{i,j}$ is also null, we do not count the correction of $t_{i,j}$ as valid, which follows the existing data cleaning methods for evaluation [75, 87].

[R4O3] *The noise in the experimental evaluation is synthetically added. Adding noise and then removing it (knowing how you added it already) is very far from a realistic scenario. I understand that it's hard to evaluate the system without this. Also, most of the datasets, except one, were already used in another paper (Baran). This is only mentioned clearly in the supplementary material, but it should also be mentioned explicitly in the paper itself (benefiting the paper).*

[A] Thanks! We have clarified the setting of noise injection and the statements of datasets in Section 7 (pp. 10). Currently for fair comparison, the dirty table $\mathcal{T}$ and the clean table $\mathcal{T}^{+}$ is identically the same as the released version of Baran. More specifically, to verify the variability of GIDCL , we have added 2 new real-life datasets Inpatients and Facilities in Section 7 to evaluate the effectiveness in Table 12, and the experimental results show that GIDCL consistently outperforms other baselines. Additionally, we have explicitly mentioned about the sources of the datasets in Section 7 (pp. 11) to make the paper self-contained.

[R4O4] *Extremely slow for smaller datasets, compared to baseline methods (but faster in larger datasets and better in all datasets). Although the authors have made some efforts to improve efficiency, it's still probably preferable to use one of the much faster methods, even if they are worse in effectiveness, in some cases (e.g., when one cannot afford to spend hours, but only 1-2 minutes). I understand the difficulty with* LLMs*, but it should be clearly stated as a weakness.*

[A] Thanks! We have added more discussion about the inefficiency of LLMs in small datasets in Section 7 (pp. 12). We also added discussion among various LLM-based data cleaning research methods, and discussed how our approach strikes an effective balance between cost, runtime, and performance, and can be trained and deployed on consumer-level hardware.

[R4O5] *One of the three components,* GSL*, is not described clearly and it is not evaluated directly (apart from an ablation study). Following **[R4O1.4]** above, it's not clear what is a good/bad selection of θ_{label} tuples for labeling. What if they are randomly chosen? Evaluating other alternatives for selection of theta tuples should be included. How is the value of k selected for k-means?*

[A] Thanks! We added more experiments and compared the influence of different $\mathcal{T}_{\text{label}}$ selection methods for data cleaning performance in Table 10 (pp. 12), and showed that GSL is better than other baselines. Also we added the effectiveness evaluation by varying different k in Figure 7(a) and 7(b), which showed that a proper value of k is necessary for the performance of GIDCL .

[R4O6] *Experimentally evaluating your suggested approach is not a contribution, but a prerequisite for having a paper published. Contribution (4) should be removed.*

[A] Thanks! We have removed it from our contributions.

[R4O7] *The table of notation (Table 2) is helpful, but it should have clearer descriptions (this is also related to O1). In more detail:*
[R4O7.1] *instead of "the relational table", you could provide more information,* e.g.*, "a relational table provided as input for cleaning"*
[R4O7.2] *instead of "detection/generation/correction", you could say "error detection", "noise/error generation", etc.*
[R4O7.3] *it's much easier to understand "$\mathcal{F} = \mathcal{F}^{\text{det}} \cup \mathcal{F}^{\text{gen}} \cup \mathcal{F}^{\text{corr}}$" (if my understanding is correct) than the current description.*
[R4O7.4] *(related to **[R4O1.7]**): it would be better to also write and explain the format of $\mathcal{T}_{\text{label}}$*

[A] Thanks! We have revised Table 2 to illustrate notations more clearly, and used suggested phases and terms accordingly.

[R4M1] *The use of the words "divided", "transferred", and "summarized" are quite confusing and probably not what the authors intended to express. E.g., a graph is not "transferred from" a table, but "constructed/generated" based on/from a table;* LLMs *do not summarize a transformation function (implying that a transformation function already existed and the* LLMs *describe it in natural language), but instead, they "return"/"generate" a transformation function; $\mathcal{T}_{\text{coreset}}$ is not divided (into subsets), but it is a subset itself; C_i is not divided (into subclusters) by* GSL*, but it is a cluster itself, suggested/returned/provided by* GSL*.)*

[A] Thanks! We have revised them marked in blue.

[R4M2] *What is $\mathcal{T}_{coreset}$ exactly? What is $f^{det}_{A_i}$ exactly? Those are central points not clear at all in Section 3. They are explained later (not ideally, but somehow explained), but the reader is lost with such lack of information when reading Section 3*

[A] Thanks! We have added the definition of coreset $\mathcal{T}_{\text{coreset}}$ and $f^{det}_{A_i}$ in the start of Section 3 (pp.4) in the revision version. We have also clarified the definition of mentioned variables in Table 2.

[R4M3] *Footnote 2 should be part of the text, as it's very important for the paper.*

[A] Thanks! We have moved it into Section 1.

[R4M4,M5,M7] *(1) "if an error remain unidentified, it will likely remain unaddressed" –> why "likely" and not "certainly"? also change the first "remain" to "remains". (2) Example 2: "PrivoderID" –> "ProviderID". (3) Page 7: "Existing approaches [...] are hard to learn [...]" –> Existing approaches for what? same paragraph: "sheds light on it" is not used properly.*

[A] Thanks! We have revised them marked in blue,

[R4M6] *The usage of notation O is probably different in Section 5 (serialized sequence) than in Table 2 (the domain of all objects).*

[A] Thanks! We have added a formal definition of O in Table 2.

[R4M8] *I am not sure what M_G is in page 10, configuration (but perhaps I missed it).*

[A] M_G is $\mathcal{M}_{\text{corr}}$ as the correction model. We have revised it.

[R4M9] *Any idea why* Raha + Baran *is so stable, even improving when the error-rate increases, in Table 8?*

[A] Thanks! Despite some randomness in 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 is likely to remain stable. We have added more detailed discussion about such phenomenon in the experiment section.(pp.13)

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

ABSTRACT

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].

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 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 [97].

Traditional DC systems follow the pre-configuration paradigm [97], where users have to provide different types of rules or constraints, such as functional dependencies [78], denial constraints [22], conditional functional dependencies [39], 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.,* [54, 77, 117], 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 [97, 116] or searching external data sources [23]. However, these configuration-free DC methods still heavily rely on predefined feature engineering and often struggle with undefined error patterns[75, 77]. 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[87, 116].

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 [106] and TURL [28] 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 [87] 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[89].

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) [16, 107],which typically contain billions (or more) of parameters, and pre-trained on massive text data[104], 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 incorporates 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 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 $\mathcal{T} = \{t_1, t_2, \cdots, t_{|\mathcal{T}|}\}$, where $|\mathcal{T}|$ represents its size. The relational schema of this table is given by $\mathcal{A} = \{A_1, A_2, \cdots, A_n\}$. 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 [75, 87, 94], 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 θ, where users are only able to label at most θ 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. □

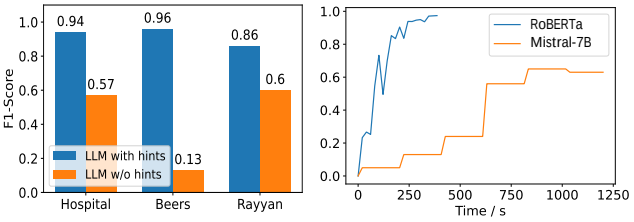
The data cleaning task is non-trivial [75, 77] for the following reasons. First, the labeling budget θ is typically very limited due to the expensive user labeling cost, thereby posing a challenging in generalizing them across all instances, *e.g.,* ML-based methods might overfit to the training data. 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.

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

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., $\mathcal{T}_{\text{label}} = \{(t_i, t_i^+) \mid t_i \in \mathcal{T}\}$
$\mathcal{T}_{\text{pseudo}}$	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^{train}	training data for $\mathcal{M}_{\text{det}}$, $\mathcal{M}_{\text{corr}}$
$\mathcal{O}$	the domain of all objects in $\mathcal{T}$



(a) Error correction performance of LLMs with and w/o hints

(b) Error detection performance on Beers over training time

Figure 1: Observations of LLMs for data cleaning

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: (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) [122]. (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.

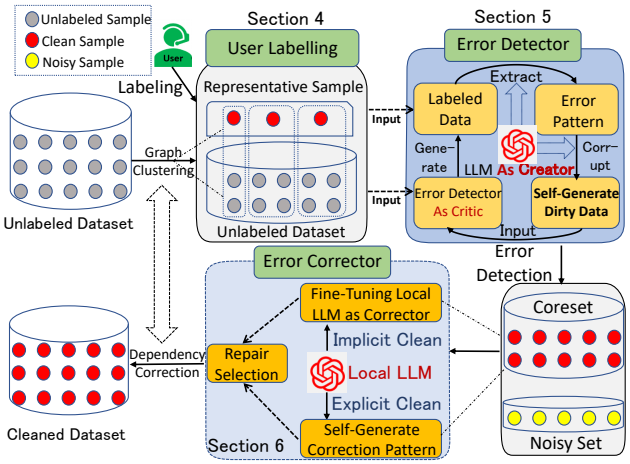


Figure 2: Overview of GIDCL

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^{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 θ representative tuples to users for manual labeling. Following this, a creator-critic workflow is invoked, which harmonizes 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 θ.
- *Output*: θ representative tuples that need to be labeled by users, a directed attribute graph $\mathcal{G}$ constructed from $\mathcal{T}$, cluster division 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 θ 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 θ representative tuples that requires labelling by users.

(2) Error detection $\text{GIDCL}_{\text{det}}$. The error detection component in GIDCL named $\text{GIDCL}_{\text{det}}$ 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 θ 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 $\text{GIDCL}_{\text{det}}$. 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 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 τ. we denote $\mathcal{F}^{\text{det}} = \{f_{A_1}^{\text{det}}, \ldots, f_{A_n}^{\text{det}}\}$ 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 τ 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 GIDCL$_{\text{cor}}$. In this phase, regarding the detected error cells $\mathcal{T}_{\text{err}}$, the error correction component 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, 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 $\mathcal{T}^{'}_{clean}$ 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$_{\text{det}}$. 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 $f^{\text{corr}}_{A_i}$ .We also design explicit function for error correction. By taking $\mathcal{T}_{\text{label}}$ as input, we query LLM to return a correction function $f^{\text{corr}}_{A_i}$ for A_i, referencing the accepted detection function $f^{\text{det}}_{A_i}$. 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$_{\text{det}}$, 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 retrain 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 θ 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 θ 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[102, 109], 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 $E = \{(t_1, \text{City}, \text{Monticello}), (t_2, \text{City}, \text{Monticello}), (t_1, \text{PrivoderID}, 111303), (t_2, \text{ProviderID}, 111303), (t_1, \text{County}, \text{Jasper}), (t_2, \text{County}, \text{Jasper})\}$. □

Inspired by [108], 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 [102, 108] 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[102].

(b) Initialization of embeddings. We adopt a PLM SentenceBert [96] 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 serial(u) and then adopt SentenceBert with pre-trained model bge-large-en to transform it to a high-dimensional embedding, s.t., $\mathbf{u}$ = SentenceBert(serial(u)), where $\mathbf{u}$ is the initial embedding of u.

(c) Self-supervised learning. We apply ComplEx [108] as scoring function $f(e)$, with GNN-style method CompGCN [109] to learn embedded vectors for all vertices in V. Loss function is defined as:

$$
\begin{aligned}
f(t) &= \Re\left(\langle w_r, e_h, \overline{e_t}\rangle\right) \\
Loss(\Theta) &= -\frac{1}{(1+m)|\Theta|} \sum_{(h,r,t)\in E} y \log l(f(h,r,t)) \\
&\quad + (1-y)\log(1 - l(f(h,r,t)))
\end{aligned} \tag{1}
$$

where $\mathbf{e}_h$ and $\mathbf{e}_t$ are embedding vectors of vertices h and t, 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 $\mathbf{e}_t$. Θ 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 θ tuples that are inclined to be erroneous and their errors are representative. We formulate the representative tuple selection as the bilevel optimization problem.

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

where $\mathcal{B}=\{\mathcal{B}_1,\ldots,\mathcal{B}_K\}$ 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 $C=\{C_1,C_2,\ldots,C_k\}$. To generate training data for error detection and correction, GIDCL presents the top-θ 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.,* $\text{Dist}(C_l)=\sum(\{\cos(e_i,e_j)|e_i\in C_l,e_j\in C_l,i\neq j\})/|C_l|$. and select θ clusters with the lowest average cosine similarity. Subsequently, within each cluster C_l, 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 the outlier vertex, since t_5 has little correlations with other vertices compared to other ones. Thus we add t_5 into $\mathcal{T}_{\text{label}}$. □

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 $\text{GIDCL}_{\text{cor}}$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^{train}. The model, which incorporates a pre-trained PLM with bi-level context attention [28], 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 [79]. 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.

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

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 [123] based on tags ⟨COL⟩ and ⟨VAL⟩.

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 "⟨COL⟩ProviderID ⟨VAL⟩1x1303⟨COL⟩City⟨VAL⟩Monticello⟨COL⟩State⟨VAL⟩AR⟨COL⟩ Zip⟨VAL⟩71655⟨COL⟩County⟨VAL⟩Drew[SEP]⟨COL⟩ProviderID⟨VAL⟩ 1x1303". Also its column-context dependency is "⟨COL⟩ProviderID ⟨VAL⟩111303⟨VAL⟩1x1303⟨VAL⟩10001[SEP]⟨VAL⟩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 (1) "⟨COL⟩ProviderID ⟨VAL⟩111303⟨COL⟩City⟨VAL⟩Monticello⟨COL⟩State⟨VAL⟩AR⟨COL⟩ Zip⟨VAL⟩71655⟨COL⟩County⟨VAL⟩Drew[SEP]⟨COL⟩ProviderID⟨VAL⟩ 111303", and (2) "⟨COL⟩ProviderID⟨VAL⟩111303⟨VAL⟩1x1303⟨VAL⟩10001 [SEP]⟨VAL⟩111303". We label them as 1 as positive ones. □

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 [77] and activeClean [63], 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 [17] 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 [19, 82]. 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 [92].

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 τ 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 τ; 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→VA; VAA→VA]
- Values $v(A_j)$: [VA; VAA; AL; $\cdots$]

The output function from LLM is $f_{\text{State}}^{\text{det}}$ = ^[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→VA]
- Values $v(A_j)$: [VA; VAA; AL; $\cdots$]

The refined function from LLM is $f_{\text{State}}^{\text{det}}$=^[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 $\mathcal{T}_{\text{label}}$. □

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^{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^{gen}, the input is as follows.

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

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. 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 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. iter := 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^{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** $(\mathcal{T}_{\text{err}}, \mathcal{T}_{\text{presudo}}, \mathcal{T}_{\text{coreset}})$;

Figure 3: The Creator-critic workflow of error detection

- Values $v(A_j)$: [1x1303; 111303; 10001; $\cdots$]

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

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 τ. 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^{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 $\mathcal{A} = \{A_1, A_2, \cdots, A_n\}$, $\mathcal{M}_{\text{det}}$ converges, and the creator updates

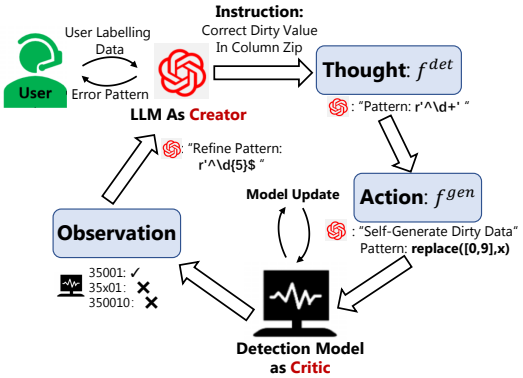


Figure 4: Creator-critic workflow for error detection

the LLMs-generated functions for all n attributes. We execute the creator and the critic in multiple rounds until $f^{\text{det}}_{A_j}$ 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 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$_{\text{det}}$could still be effective.

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) [32, 81], retrieval augmented generation (RAG) [66], 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 context$(t_{i,j})$ for fine-tuning LLM with ICL and RAG context. In detail, 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 $E^{\text{ICL}}_j = \{(t_{i,j}, t^+_{i,j}) | t_{i,j} \in \mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}, t_{i,j} \neq t^+_{i,j}\}$ for ICL demonstration, and a set E^{RAG}_i of tuples sampled within $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{coreset}}$, also from the same cluster

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 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 $E^{\text{ICL}} := \{E^{\text{ICL}}_1, \ldots, E^{\text{ICL}}_{|\mathcal{A}|}\}$ of ICL context from D^{train};
3. Generate $E^{\text{RAG}} := \{E^{\text{RAG}}_1, \ldots, E^{\text{RAG}}_{|\mathcal{A}|}\}$ of RAG examples from $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{coreset}}$;
4. Generate the training set $D^{\text{train}}_{\text{LLM}}$ using using E^{ICL} and E^{RAG}, s.t., $D^{\text{train}}_{\text{LLM}} := \{[\text{context}(t_{i,j}), t^+_{i,j}] | (t_{i,j}, t^+_{i,j}) \in D^{\text{train}}\}$
5. Fine-tune $\mathcal{M}_{\text{corr}}$ using $D^{\text{train}}_{\text{LLM}}$; /* implicit error correction */
6. $\mathcal{F}^{\text{corr}} := \emptyset$;
7. **for each** $A_i \in \mathcal{A}$ **do** /* explicit error correction */
8. iter $:= 0$, $v(A_i) := \mathcal{T}[A_i]$, $f^{\text{corr}}_{A_i} := \text{LLM}(c, \mathcal{T}_{\text{labe}}, v(A_i), f^{\text{det}}_{A_i})$;
9. **while** Eval$(f^{\text{corr}}_{A_i}, \mathcal{T}_{\text{label}}) \leq \tau$ and iter $\leq$ iter$_{\text{max}}$ **do**
10. Collect $\mathcal{T}^{\text{wrong}}_{\text{label}} := \{(t_{j,k}, t^+_{j,k}) | f^{\text{corr}}_{A_k} \neq t^+_{j,k}, (t_{j,k}, t^+_{j,k}) \in D^{\text{train}}\}$;
11. $f^{\text{corr}}_{A_i} := \text{LLM}(c, \mathcal{T}^{\text{wrong}}_{\text{label}}, v(A_i), f^{\text{det}} A_i)$;
12. iter $:=$ iter $+ 1$;
13. Add $f^{\text{corr}}_{A_i}$ into $\mathcal{F}^{\text{corr}}$ if Eval$(f^{\text{corr}}_{A_i}, \mathcal{T}_{\text{label}}) > \tau$;
14. /* Error detection in $\mathcal{T}$ */
15. **for each** $t_{i,j} \in \mathcal{T}_{\text{err}}$ **do**
16. $t := \text{argmax}\{\mathcal{M}_{\text{det}}(\mathcal{M}_{\text{corr}}(t_{i,j})), \mathcal{M}_{\text{det}}(f^{\text{corr}}_{A_j}(t_{i,j}))\}$;
17. Repair $t_{i,j}$ of $\mathcal{T}_{\text{clean}}$ using t;
18. $\Delta R :=$ GraphStructureRelearning$(\mathcal{T}_{\text{label}}, \mathcal{T}_{\text{coreset}}, f_{\text{GNN}}, \mathcal{T}_{\text{err}})$;
19. Repair $\mathcal{T}_{\text{clean}}$ using ΔR;
20. **return** $\mathcal{T}_{\text{clean}}$;

Figure 5: The error correction algorithm

C_i containing t_i as RAG. The sequence format is :

$$\text{context}(t_{i,j}) = q \circ \text{serial}(t_{i,j}) \circ E^{\text{ICL}}_j \circ E^{\text{RAG}}_i$$

The training dataset for LLM is re-organized as $D^{\text{train}}_{\text{LLM}} = \{[\text{context}(t_{i,j}), t^+_{i,j}] | t_{i,j} \in D^{\text{train}}\}$, 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 [56]. During inference, we serialize the context of an erroneous cell $t_{i,j} \in \mathcal{T}_{\text{err}}$ into 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 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 serial$(t_{2,3})$ = {City : Monticello, State : VAA, $\cdots$}
- ICL Demonstration: [AL,Houston $\rightarrow$ AL]
- RAG Context: serial(t^+_1) = ⟨COL⟩ProviderID⟨VAL⟩111303⟨COL⟩City⟨VAL⟩Monticello⟨COL⟩State⟨VAL⟩VA⟨COL⟩Zip⟨VAL⟩31064⟨COL⟩County⟨VAL⟩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. □

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

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 $f^{\text{corr}}_{\text{State}}$ using:

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

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

Repair selection. Considering $\mathcal{M}_{\text{corr}}(t_{i,j})$ and $f^{\text{corr}}_{A_j}(t_{i,j})$ 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 $\text{Conf}(f^{\text{corr}}_{A_j}(t_{i,j}))$. 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:

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

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 λ modulates this prioritization.

Error correction. Upon updating the edge weights in $\mathcal{G}'_{\text{sub}}$ by clusters, we apply link prediction task[29] 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 Θ 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 ω 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 $\omega(h, \text{City, Dothan}) = \frac{3}{4}$ ($t_5, t_6, t_7 \in \mathcal{T}_{\text{coreset}}$), $\omega(h, \text{City, Monticello}) = \frac{1}{4}\lambda$ ($t_4 \notin \mathcal{T}_{coreset}$), so the value of City in t_4, t_5, t_6, t_7 should be unified as Dothan. □

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

Remark. $\text{GIDCL}_{\text{cor}}$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 θ, (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 [75] and one real-life large dataset in Table 3. Hospital [97] and Flights [70] 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

Table 3: Datasets. The error types are missing value (MV), typo (T), formatting issue (FI), and violated attribute dependency (VAD).

Name	$\lvert\mathcal{T}\rvert \times \lvert\mathcal{A}\rvert$	Error Rate	Error Types	Cluster Number
Hospital	1000×20	3%	T, VAD	20
Flights	2376×7	30%	MV, FI, VAD	20
Beers	2410×11	16%	MV, FI, VAD	10
Rayyan	1000×11	9%	MV, T, FI, VAD	10
Tax	200000×15	4%	T, FI, VAD	30
IMDB	1000000×6	1%	T, FI, VAD	50

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 [75] and Rayyan [85] 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 [75, 87], 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).

Baselines. We implemented GIDCL in Python and used the following baselines. (1) Raha [77], an error detection method involving feature engineering, interactive labeling and ML models for detection; (2) Rotom [79], a meta-learning data augmentation framework that formulates tabular error detection as a seq2seq task; (3) $\text{Roberta}_{\text{det}}$ [72], a binary classifier adapted for error detection utilizing the pre-trained language model Roberta [72]; (4) Baran [75], a hybrid error correction approach based on feature engineering and traditional ML models; (5) Garf [87], a deep learning-based error correction approach that employs SeqGAN [119] to generate data repair rules in an unsupervised manner; (6) HoloClean [97], an error correction method that leverages data quality rules to construct factor graph for data repairs; (7) T5 [93], a generative PLM for error correction; (8) JellyFish-13B [120], an LLM-based method addressing error detection and data imputation seperately, 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 [76]. For the remaining error correction methods, including HoloClean and T5, we utilized our error detection method $\text{GIDCL}_{\text{det}}$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 [77] and [75]. We also report the runtime 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 [59] 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 $\text{GIDCL}_{\text{offline}}$. All prompts in GIDCL are handcrafted once and listed in [2]. The default $\theta_{\text{label}} = 20$, $\tau = 0.85$ and $\lambda = 4$. 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 θ_{label}.

Error Detection. Table 4 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, $\text{GIDCL}_{\text{offline}}$ 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, $\text{GIDCL}_{\text{offline}}$ 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. $\text{Roberta}_{\text{det}}$ performs similarly to Rotom in most datasets. However, due to a small labelling budget, $\text{Roberta}_{\text{det}}$ 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 5 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. $\text{GIDCL}_{\text{offline}}$ 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, $\text{GIDCL}_{\text{offline}}$ still surpasses other baselines, *e.g.,* 16% higher F1-score than the best of others on 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 re-

Table 4: 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	**1.00**	0.95	**0.98**	**0.96**	**0.95**	**0.96**	0.99	0.99	0.99	0.84	**0.98**	**0.90**	0.99	**0.98**	**0.98**	0.88	0.89	**0.88**
$\text{GIDCL}_{\text{offline}}$	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	**1.00**	**1.00**	**1.00**	**0.88**	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	**1.00**	0.60	0.75	0.17	**1.00**	0.29
$\text{Roberta}_{\text{det}}$	0.87	**0.97**	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	**0.94**	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

Table 5: 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	**0.96**	**0.97**	**0.94**	**0.92**	**0.93**	**0.97**	**0.97**	**0.97**	**0.80**	**0.93**	**0.86**	**0.95**	**0.94**	**0.95**	**0.87**	**0.89**	**0.87**
$\text{GIDCL}_{\text{offline}}$	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
$\text{GIDCL}_{\text{det}}$ + Holoclean	**0.98**	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
$\text{GIDCL}_{\text{det}}$ + 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

Table 6: 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

fined 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 $\text{GIDCL}_{\text{offline}}$. 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 11, we further evaluate the effectiveness of the proposed correction method $\text{GIDCL}_{\text{cor}}$, by varying different error detectors in Hospital and Rayyan. $\text{GIDCL}_{\text{cor}}$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 $\text{GIDCL}_{\text{cor}}$keeps most of the above FN clean data unchanged, only correcting real errors, decreasing FN cases to 100. Similarly, by introducing $\text{GIDCL}_{\text{det}}$and varying different correction models, we also find the high performance of $\text{GIDCL}_{\text{det}}$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, $\text{GIDCL}_{\text{det}}$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 10, compared with three sampling baselines, including SBert that replaces the graph structure learning in Section 4 with SentenceBert [96], Raha that adopts the idea of Raha [76] to select θ representative tuples and Random that randomly select θ 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.

Class imbalance issues. In Table 8 (same setting with Table 10 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 $r_{\text{err}} = \frac{\#\text{erroneous cells}}{\#\text{all cells}}$. 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 [47] in Table 12. 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.

Exp-2: Efficiency. Table 6 reports the running time of all baselines, including both error detection and correction. In contrast to other LLM-based research [107], 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,

Table 7: Ablation study

System	Hospital			Tax			Rayyan			Beers			Flights		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
GIDCL	**0.97**	**0.96**	**0.97**	**0.95**	**0.94**	**0.95**	**0.80**	**0.93**	**0.86**	**0.97**	**0.97**	**0.97**	**0.94**	**0.92**	**0.93**
$\text{GIDCL}_{\text{offline}}$	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

Table 8: Ratio r_{err} in $D^{\text{train}} = \mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}$ with sampling methods

Methods	Hospital		Rayyan	
	Cell r_{err}	Column r_{err}	Cell r_{err}	Column r_{err}
$\text{GIDCL}_{\text{offline}}$	**0.4337**	**14**	**0.4837**	**7**
SBert	0.4232	8	0.4739	6
Raha	0.4002	9	0.4730	6
Random	0.3609	6	0.4697	5

the size of training data D^{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 [18, 26, 107] because of the huge size of parameters, To solve it in the data cleaning task, GIDCL incorporates the vLLM technique [64] 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.

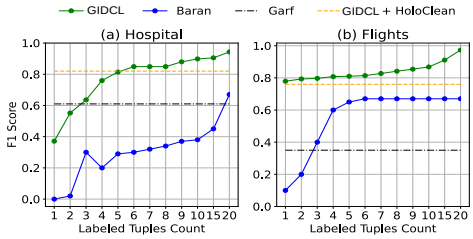


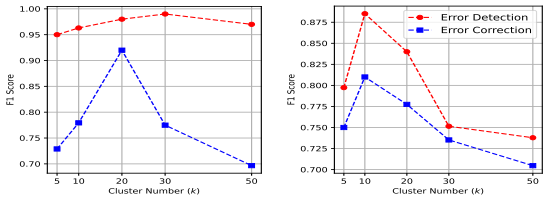
Figure 6: Performance w.r.t. the number of labelling budgets

Exp-3: Ablation study. We show how labeling budges and error ratios for $\mathcal{T}$ impact the performance of GIDCL and other baselines.

Labelling Budget. Figure 6 shows the performance of baselines in error correction by varying the number of labelled tuples. GIDCL shows a more rapid convergence in the F1 score than others when a very small labeling budget is provided, *e.g.*, 19.5% higher on average when the budget is 10. This demonstrates the few-shot learning capabilities of LLMs, and a small number of user-provided labels is enough for LLM-based $\mathcal{M}_{\text{corr}}$ to learn well.

Component analysis In Table 7, 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 9. 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 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.



(a) Hospital Dataset (b) Rayyan Dataset

Figure 7: 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 $\text{GIDCL}_{\text{cor}}$.

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, 12], DCs [22, 48, 51, 97], CFDs [14, 36, 37], PFDs [91], REEs [44, 46], user-defined rules [35]

Table 9: 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	**0.91**	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	**0.90**	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	**0.88**	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	**0.87**	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	**0.85**	0.95	0.01	0.02	0.74	0.57	0.65	0.03	0.36	0.05	0.23	0.66	0.35

Table 10: Error correction by varying sampling methods

Sample Method	Hospital			Rayyan		
	P	R	F	P	R	F
GSL	**0.94**	**0.90**	**0.92**	**0.78**	**0.85**	**0.81**
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 11: Error correction by varying error detectors/correctors

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

Table 12: Error correction in Facilities and Inpatient

System	Facilities			Inpatient		
	P	R	F	P	R	F
GIDCL	0.77	**0.75**	**0.76**	0.80	**0.93**	**0.86**
HoloClean	**1.00**	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	**0.97**	0.09	0.17
$GIDCL_{det}$+ 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

and manually-defined parameters [9, 90]. Considering the difficulty of handcrafting data quality rules, several rule discovery algorithms [13, 21, 38, 40, 41, 74, 86] 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 [54, 77, 79, 83, 88, 121] require users to provide a few labeled data and then design machine learning models to identify errors in data. Statistical hypothesis [112], co-occurrence dependency [58], ActiveClean [63], meta-data [110] and rule generation [87] 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, 10, 11, 24, 25, 31, 49, 50, 52, 53, 95, 103, 111, 117], certain fixes [42–45, 98]. *(2) ML models.* ML-based methods adopt ML models with handcrafted features for data cleaning. SCARE [116] designs methods that combine ML models and likehood approach for data repair. HoloClean [97] uses pre-defined denial constraints and MDs as features and designs a factor graph for error correction. There are also generative models [27, 33, 57, 65, 80, 99] 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 [87] and GAIN [118]. Data cleaning are also employed to improve downstream ML models, *e.g.,* [30, 68, 73]. *(3) Hybrid methods.* which unify logical rules and machine learning models for data cleaning,*e.g.,*Baran [75] 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 [55, 61, 62, 69, 105] 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 [43], knowledge bases [23] 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.

Large language models. Recently researchers found that scaling PLM (e.g., model size or data size) often leads to an improved model capacity on various downstream tasks, following scaling laws [60], *e.g.,* the 175B-parameter GPT-3 [16] and the 540B-parameter PaLM [20]. Compared with PLMs, LLMs show the emergent abilities [114], in-context learning [32, 81], instruction following [84, 100, 113] and step-by-step reasoning [115]. These abilities guarantee LLMs to generate function set correctly with only a few examples for demonstration. Recently, a host of pioneering works focus on transforming DC tasks into generation tasks and apply LLM to solve them, containing error detection[67, 82, 120] and data imputation[15, 19, 67, 82, 120], but there is no work that integrates LLM for an end-to-end DC system.

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.

REFERENCES

[1] 2023. IMDB. https://www.imdb.com/interfaces/.

[2] 2024. Code, datasets and full version. https://anonymous.4open.science/r/GEIL-0D20.

[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] Benjamin Bengfort, Rebecca Bilbro, Nathan Danielsen, Larry Gray, Kristen McIntyre, Prema Roman, Zijie Poh, et al. 2018. *Yellowbrick*.

[9] 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.

[10] Leopoldo Bertossi. 2011. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers.

[11] 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

[12] 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.

[13] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. 2017. Efficient denial constraint discovery with hydra. *Proceedings of the VLDB Endowment* 11, 3 (2017), 311–323.

[14] 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

[15] Alexander Brinkmann, Roee Shraga, and Christian Bizer. 2023. Product Attribute Value Extraction using Large Language Models. *arXiv preprint arXiv:2310.12537* (2023).

[16] 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.

[17] 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]

[18] 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).

[19] 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.

[20] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311* (2022).

[21] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *Proceedings of the VLDB Endowment* 6, 13 (2013), 1498–1509.

[22] 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.

[23] 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.

[24] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. 2007. Improving data quality: Consistency and accuracy. In *VLDB*.

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

[26] 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.

[27] 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

[28] 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.

[29] 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.

[30] 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

[31] Xiaoou Ding, Hongzhi Wang, Jiaxuan Su, Muxian Wang, Jianzhong Li, and Hong Gao. 2020. Leveraging currency for repairing inconsistent and incomplete data. *TKDE* (2020).

[32] 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).

[33] Prashant Doshi, Lloyd G Greenwald, and John R Clarke. 2003. Using Bayesian Networks for Cleansing Trauma Data. In *FLAIRS*. 72–76.

[34] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2021. Glm: General language model pretraining with autoregressive blank infilling. *arXiv preprint arXiv:2103.10360* (2021).

[35] 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.

[36] 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

[37] 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

[38] 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

[39] 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.

[40] 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

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] Wenfei Fan, Ping Lu, Chao Tian, and Jingren Zhou. 2019. Deducing Certain Fixes to Graphs. *PVLDB* 12, 7 (2019), 752–765.

[46] Wenfei Fan, Chao Tian, Yanghao Wang, and Qiang Yin. 2021. Parallel Discrepancy Detection and Incremental Detection. *PVLDB* 14, 8 (2021), 1351–1364.

[47] Centers for Medicare Medicaid Services. [n.d.]. Provider data catalog. https://data.cms.gov/provider-data/.

[48] 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.

[49] 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.

[50] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2020. Cleaning data with llunatic. *The VLDB Journal* 29 (2020), 867–892.

[51] 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.

[52] Amir Gilad, Daniel Deutch, and Sudeepa Roy. 2020. On multiple semantics for declarative database repairs. In *SIGMOD*. 817–831.

[53] 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.

[54] 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.

[55] 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.

[56] 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).
[57] Yuheng Hu, Sushovan De, Yi Chen, and Subbarao Kambhampati. 2012. Bayesian Data Cleaning for Web Data. *CoRR* abs/1204.3677 (2012). arXiv:1204.3677 http://arxiv.org/abs/1204.3677
[58] 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.
[59] 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).
[60] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).
[61] 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).
[62] 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).
[63] 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.
[64] 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).
[65] 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
[66] 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.
[67] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2023. Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv preprint arXiv:2310.09263* (2023).
[68] 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.
[69] 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.
[70] 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).
[71] Xiaoran Liu, Hang Yan, Shuo Zhang, Chenxin An, Xipeng Qiu, and Dahua Lin. 2023. Scaling Laws of RoPE-based Extrapolation. *arXiv preprint arXiv:2310.05209* (2023).
[72] 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).
[73] 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.
[74] Ester Livshits, Alireza Heidari, Ihab F Ilyas, and Benny Kimelfeld. 2020. Approximate denial constraints. *arXiv preprint arXiv:2005.08540* (2020).
[75] 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.
[76] 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.
[77] 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.
[78] Heikki Mannila and Kari-Jouko Räihä. 1994. Algorithms for inferring functional dependencies from relations. *Data & Knowledge Engineering* 12, 1 (1994), 83–99.
[79] 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.
[80] 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.
[81] Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2021. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943* (2021).
[82] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911* (2022).
[83] 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.
[84] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems* 35 (2022), 27730–27744.
[85] 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.
[86] 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.
[87] 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.
[88] 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.
[89] 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.
[90] Clement Pit-Claudel, Zelda Mariet, Rachael Harding, and Sam Madden. 2016. Outlier detection in heterogeneous datasets using automatic tuple expansion. (2016).
[91] Abdulhakim Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern functional dependencies for data cleaning. *PVLDB* 13, 5 (2020), 684–697.
[92] 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).
[93] 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.
[94] Erhard Rahm, Hong Hai Do, et al. 2000. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.* 23, 4 (2000), 3–13.
[95] Joeri Rammelaere and Floris Geerts. 2018. Explaining repaired data with CFDs. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1387–1399.
[96] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv:1908.10084 [cs.CL]
[97] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820* (2017).
[98] 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.
[99] 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
[100] Victor Sanh, Albert Webson, Colin Raffel, Stephen H Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, et al. 2022. Multitask Prompted Training Enables Zero-Shot Task Generalization. In *The Tenth International Conference on Learning Representations, ICLR*.
[101] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100* (2022).
[102] 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*.
[103] 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.
[104] 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.
[105] 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.
[106] 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).

[107] 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).

[108] 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.

[109] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. 2019. Composition-based multi-relational graph convolutional networks. *arXiv preprint arXiv:1911.03082* (2019).

[110] 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.

[111] 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.

[112] 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.

[113] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652* (2021).

[114] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* (2022).

[115] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.

[116] 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.

[117] Mohamed Yakout, Ahmed K Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F Ilyas. 2011. Guided data repair. *arXiv preprint arXiv:1103.3103* (2011).

[118] 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.

[119] 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.

[120] Haochen Zhang, Yuyang Dong, Chuan Xiao, and Masafumi Oyamada. 2023. Jellyfish: A Large Language Model for Data Preprocessing. *arXiv preprint arXiv:2312.01678* (2023).

[121] 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.

[122] 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*.

[123] 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).

A MODEL ARCHITECTURE OVERVIEW

In Figure 9, we present the model architecture overview of the proposed GIDCL in detail.

B PROMPTS

As the prompts in GIDCL are dynamically composed, we use <> to represent placeholders for simplicity.

B.1 Straightforward Prompts for Error Detection

This example refers to tuple t_3 in Table 1 as an illustration of straightforward error detection with LLM in Section 2.2.

Instruction Data – Error Detection

(system message) *You are an AI assistant that follows instruction extremely well. User will give you a question. Your task is to answer as faithfully as you can.*
(task description) *Your task is to determine if there is an error in the value of a specific attribute within the whole record provided.*
(instruction) *Errors may include, but are not limited to, spelling errors, inconsistencies, or values that don't make sense given the context of the whole record.*
(input) *Record [ProviderID: "1x1303", City: "Monticello", State: "AR", Zip: "71655", County: "Drew"]*
Attribute for Verification: [ProviderID: "1x1303"]
(question) *Is there an error in the value of the "ProviderID" attribute?*
(output format) *Choose your answer from: [Yes, No]*

B.2 Straightforward Prompts for Error Correction

This example refers to tuple t_3 in Table 1 of straightforward error correction with LLM in Section 2.2..

Instruction Data – Error Correction

(system message) *You are an AI assistant that follows instruction extremely well. User will give you a question. Your task is to answer as faithfully as you can.*
(task description) *Your task is to correct the dirty value of a specific attribute within the whole record provided to clean one.*
(instruction) *Errors may include, but are not limited to, spelling errors, inconsistencies, or values that don't make sense given the context of the whole record.*
(input) *Record [ProviderID: "1x1303", City: "Monticello", State: "AR", Zip: "71655", County: "Drew"]*
Attribute for Correction: [ProviderID: "1x1303"]
(question) *Correct the dirty value to clean value.*
(output format) *ProviderID:" "*

B.3 Serialize Context for Error Detection Model $\mathcal{M}_{\text{det}}$

Consider $t_3 \in \mathcal{T}_{label}$ in Table 1 for the 2nd attribute ProviderID. We generate the sequence serial$(t_{3,2})$ with aspects of row-contextual and column-contextual dependencies as follows.

$$\underbrace{\langle\text{COL}\rangle A_1 \langle\text{VAL}\rangle \cdots \langle\text{VAL}\rangle t_i[A_n]}_{\text{serialization of } t_i}\ [\text{SEP}]\ \underbrace{\langle\text{COL}\rangle A_j \langle\text{VAL}\rangle C(t_i)[A_j]}_{\text{values in } A_j}\ [\text{SEP}]\ \underbrace{\langle\text{COL}\rangle A_j}_{\text{target attribute } A_j}$$

Serialize Context

- row-contextual dependency*:*
  ⟨COL⟩*ProviderID*⟨VAL⟩*1x1303*⟨COL⟩*State*⟨VAL⟩*AR*[SEP]
  ⟨COL⟩*ProviderID*⟨VAL⟩*1x1303:* dirty*;*
  ⟨COL⟩*ProviderID*⟨VAL⟩*111303*⟨COL⟩*State*⟨VAL⟩*AR*[SEP]
  ⟨COL⟩*ProviderID*⟨VAL⟩*111303:* clean*;*
- column-contextual dependency*:*
  ⟨COL⟩*ProviderID*⟨VAL⟩*1x1303*⟨VAL⟩
  111303[SEP]⟨COL⟩*ProviderID*⟨SEP⟩*1x1303:* dirty*;*
  ⟨COL⟩*ProviderID*⟨VAL⟩*1x1303*⟨VAL⟩
  111303[SEP]⟨COL⟩*ProviderID*⟨SEP⟩*111303:* clean.

B.4 Prompts Related to Generate Detection Function $\mathcal{F}^{\text{det}}$ 1st round

Please check Example 5 for detail, here we only list the instruction/input/output for querying LLM with multi-turn conversation.

Instruction : Generate Error Detection Function 1st round

(system message) *You are an AI assistant that follows instruction extremely well. User will give you a question. Your task is to answer as faithfully as you can.*
(task description) *Please conclude a general pattern for dirty and clean cells, and write a general function with simple and precise regular expression to detect whether a given cell is dirty or not.*
(input) *The input*

<VA,Jasper⟶VA>

are [dirty,clean] cell pairs from table Tax *column* State*, and*

<VA,VAA,AL>

are examples of all cells from this columns.
(Output)*:*
Based on the given examples, the pattern of clean cells seems to contain upper letters. Therefore, a simple regular expression to check if a given cell is dirty or not can be:

```
def is_dirty(input_str):
    pattern = r'^[A-Z]+$'
    if re.match(pattern, input_str):
        return False
    else:
        return True
```

B.5 Prompts Related to Generate Detection Function $\mathcal{F}^{\text{det}}$ 2nd round

Please check Example 5 for detail, here we only list the instruction/input/output for querying LLM with multi-turn conversation.

Instruction : Generate Error Detection Function 2nd round

(system message) *You are an AI assistant that follows instruction extremely well. User will give you a question. Your task is to answer as faithfully as you can.*
(task description) *Please conclude a general pattern for dirty and clean cells,* ***regarding the wrongly generated patterns and counterexamples****, and revise a general function with simple and precise regular expression to detect whether a given cell is dirty or not.*
(input) *The input*
<VAA⟶VA>

are counterexample of wrong pattern ^[A-Z]+$, and

<VA,VAA,AL>

are examples of all cells from this columns.
(Output)*:*
Sorry for the misleading result. Based on the given examples, the pattern of clean cells seems to contain 2 upper letters. Therefore, a simple regular expression to check if a given cell is dirty or not can be:

```python
def is_dirty(input_str):
    pattern = r'^[A-Z]2$'
    if re.match(pattern, input_str):
        return False
    else:
        return True
```

B.6 Prompts Related to Generate Generation Function $\mathcal{F}^{\text{gen}}$

Please check Example 6 for detail, here we only list the instruction/input/output for querying LLM with multi-turn conversation.

Instruction : Generate Generation Function

(system message) *You are an AI assistant that follows instruction extremely well. User will give you a question. Your task is to answer as faithfully as you can.*
(task description) *Please conclude a general pattern for dirty and clean cells, and write a general function with python module* re *and* random*, with simple and precise regular expression, to randomly transfer clean value to dirty value similarly.*
(input) *The input*
<1x1303⟶111303>

are [dirty,clean] cell pairs from table Hospital *column* ProviderID*, and*

Pattern ^\d+$

are error patterns. And

<1x1303,111303,10001>

are examples of all cells from this columns.
(Output)*:*
Based on the given pattern and example, the pattern of clean cells seems to contain digits. Therefore, a simple regular expression to transfer clean value to dirty can be:

```python
def Generate(input_string):
    def is_dirty(input_str):
        pattern = r'^\d+$'
        if re.match(pattern, input_str):
            return False
        else:
            return True
    clean_cell = input_string
    if not is_dirty(clean_cell): ##Corrupt clean value
        random_index = random.randint(0, len(clean_cell) -
            1) # generate random index
        dirty_cell = clean_cell[:random_index] + "x" +
            clean_cell[random_index:] # corrupt cell
        return dirty_cell
    else
        return clean_cell
```

B.7 Prompts Related to Generate Correction Function $\mathcal{F}^{\text{corr}}$

Please check Example 8 for detail, here we only list the instruction/input/output for querying LLM with multi-turn conversation.

Instruction : Generate Correction Function

(system message) *You are an AI assistant that follows instruction extremely well. User will give you a question. Your task is to answer as faithfully as you can.*
(task description) *Please conclude a general pattern for dirty and clean cells, and write a correction function with python module* re*, with simple and precise regular expression, to correct a dirty value to corresponding clean one.*
(input) *The input*

<VA,Jasper⟶VA>

are [dirty,clean] cell pairs from table Tax *column* State

Pattern ^[A-Z]2$

are clean value patterns. And

<VA;AL;· · · >

are examples of clean cells from this columns. And

<VA,Jasper;VAA;AL,Houston;· · · >

are examples of dirty cells from this column.
(Output)*:*
Based on the given pattern and example, the pattern of clean cells seems to contain 2 upper letters. Therefore, a simple regular expression to transfer dirty value to clean can be:

```
def Correction(input_string):
    def is_dirty(input_str):
        pattern = r'^\d+$'
        if re.match(pattern, input_str):
            return False
        else:
            return True
    clean_cell = input_string
    if is_dirty(clean_cell): ##Correct dirty cell
        clean_cell = clean_cell[:2].upper()
    else
        return clean_cell
```

C ADDITIONAL EXPERIMENT RESULT

We evaluate the quality of generated function set $\mathcal{F}$ of online model GPT-4 and offline model Mistral in Table 13/14/15.

We observe that GPT-4 performs significantly better than offline model Mistral, both in the quality of generated function, as well as the frequency of regeneration. However, with the design of multi-turn conversation, offline model can also extract correct patterns over 75% of all features.

Such result suggests that the design of GIDCL can minimize the performance gap between online and offline LLM model, thus keep data privacy while holding relative high performance.

C.1 More Ablation Study

Parameter Size of LLMs. According to recent research [71, 107], the ability of ICL and few-shot learning for LLMs are highly correlated to the parameter sizes. To analyze its impact on the data cleaning task, we vary the parameter size of $\mathcal{M}_G$ to show the performance of $\mathcal{M}_G$ and $\mathcal{F}$.

(1) The error correction $\mathcal{M}_G$. we conduct our experiments on Hospital, Beers and Rayyan in Table 16 by using LLMs of different parameter sizes from 560M(Million) to 7B(Billion), including BLOOM-560M [101], ChatGLM-6B [34], Mistral-7B[59]. As the parameter size increases, there is a notable improvement in precision, recall, and F1 score of GIDCL across all datasets, *e.g.,* In Hospital, the F1-measurement increases from 0.76 to 0.97. Similar trends are shown in Beers and Rayyan, and the largest model, *i.e.,*Mistral-7B, achieves the highest F1 scores. This underscores the impact of larger parameter sizes in enhancing the model's ability to accurately correct errors, especially in datasets with complex error patterns, *e.g.,*Rayyan. Notably, ChatGLM3-6B with 6.2 billion parameters demonstrates a significant leap in accuracy, indicating that GIDCL can leverage the potential abilities of LLMs whose parameter sizes is as small as 6B, balancing computational resources and model performance.

(2) Function set $\mathcal{F}$. We further explore the effects of varying different LLMs for generating the function set $\mathcal{F}$. Specifically, we employ a local model Mistral-7B and an online model gpt-4-turbo to conduct experiments on Hospital and Beers datasets. Table 7 reports the experimental results. While gpt-4-turbo demonstrates strong capabilities in function generation, the locally hosted Mistral-7B model also achieves comparable performance. This finding suggests that GIDCL offers considerable flexibility, allowing users to select from a range of LLMs according to their specific requirements and constraints, no matter whether they are local or online models.

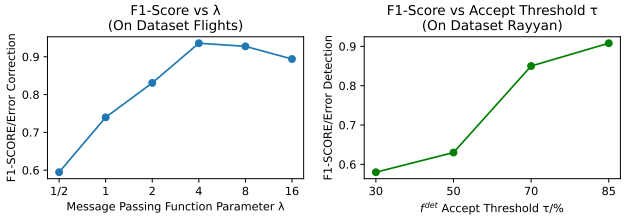


Figure 8: Performance evaluation with respect to the hyper-parameter λ on error correction for Flights**, and τ for f^{det} on error detection for** Rayyan

Hyper-parameter analysis. In Figure 8, we analyze the impact of hyper-parameters λ in message passing function ω for error correction, and threshold τ for detection function f^{det} for error detection.

By varying λ, we can see that a very large λ (meaning that we do not consider any information outside $\mathcal{T}_{\text{coreset}}$) or a very small λ (meaning that we equally consider all information in $\mathcal{T}$, regardless the division of $\mathcal{T}_{\text{coreset}}$) will lead to the drop of correction performance, and $\lambda = 4$ is a proper threshold.

By varying τ, which is the overall acceptance threshold for f^{det}, we can see a low threshold is inclined to lead LLM to generate naive patterns without refinement, and do harm to the detection performance. Although a high threshold might require multi-turn refinements with LLM, it could significantly improve the overall performance. However if τ is too high, no valid functions will be accepted within the given conversation turns budget n'. Thus we

Dataset	Total Error Feature Number	Function Number Above 85%(Mistral)	Function Number Above 85%(GPT-4)	Average Number of Regenerations(Mistral)	Average Number of Regenerations(GPT-4)
Hospital	16	14	16	7.4	3.2
Beer	4	3	4	6.6	1.1
IMDB	5	5	5	4.2	1.1
Tax	9	5	7	6.5	1.3
Rayyan	7	4	6	9.5	2.4

Table 13: Head-to-head comparison for the quality of generated detection function set $\mathcal{F}^{\text{det}}$, we use F-measure(F1) to evaluate the quality of generated function over $\mathcal{T}_{\text{label}}$

Dataset	Total Error Feature Number	Function Number Above 85% F1(Mistral)	Function Number Above 85% F1(GPT-4)	Average Number of Regenerations(Mistral)	Average Number of Regenerations(GPT-4)
Hospital	16	16	16	8.5	2.2
Beer	4	4	4	3.6	1.1
IMDB	5	5	5	1.2	1.1
Tax	9	7	7	3.4	1.4
Rayyan	7	4	5	11.4	3.4

Table 14: Head-to-head comparison for the quality of generated augmentation function set $\mathcal{F}^{\text{gen}}$. Since clean cell $t_{i,j}$ is randomly corrupted by $\mathcal{F}^{\text{gen}}$, we use F-measure(F1) to evaluate the quality of generated augmentation function over $\mathcal{T}_{\text{label}}$ with detector $\mathcal{F}^{\text{det}}$.

Dataset	Total Error Feature Number	Function Number Above 85% Acc(Mistral)	Function Number Above 85% Acc(GPT-4)	Average Number of Regenerations(Mistral)	Average Number of Regenerations(GPT-4)
Hospital	16	0	0	/	/
Beer	4	3	3	5.7	1
IMDB	5	2	4	7.5	1.2
Tax	9	7	7	3.4	1.4
Rayyan	7	4	5	11.4	3.4

Table 15: Head-to-head comparison for the quality of generated correction function set $\mathcal{F}^{\text{cor}}$, we use accuracy(Acc) to evaluate the quality of generated correction function over $\mathcal{T}_{\text{label}}$. Due to information incompleteness, *e.g.,* dataset Hospital, not all features can be successfully repaired with regular expression or rules.

System	Hospital			Beers			Rayyan		
	P	R	F	P	R	F	P	R	F
BLOOM-560M	0.79	0.74	0.76	0.92	0.85	0.88	0.59	0.60	0.59
ChatGLM3-6B	0.90	0.89	0.90	0.96	0.96	0.96	0.69	0.80	0.74
Mistral-7B	**0.97**	**0.96**	**0.97**	**0.97**	**0.97**	**0.97**	**0.80**	**0.93**	**0.86**

Table 16: Parameter Size Impact for $\mathcal{M}_G$

set $\tau = 85\%$ by default.

C.2 Discovered Functional Dependencies(FDs)

Here we list all FDs we discovered for each dataset from coreset $\mathcal{T}_{\text{coreset}}$ in Table 17. Not all FDs in clean data $\mathcal{T}_{\text{clean}}$ are listed, since some of them cannot be discovered in $\mathcal{T}_{\text{coreset}}$ during our experiment.

C.3 Generated Data Cleaning Pattern $\mathcal{F}$ by LLM

Here we list all data cleaning patterns $\mathcal{F} = \mathcal{F}^{\text{det}}, \mathcal{F}^{\text{gen}}, \mathcal{F}^{\text{corr}}$ generated by LLM.

The function set $\mathcal{F}$ are generated by gpt-4-turbo by default, queried with the prompt template listed in Section B.

We do not change the content of each pattern(*w.r.t.* function in regular expression for each attribute), and only unify the function name, *e.g., Rayyan-Detect-atitle*. We also keep the comments after # symbol that are generated by LLM, for better understanding of interpretability. We remove the comments that are not proper or not generated in English.

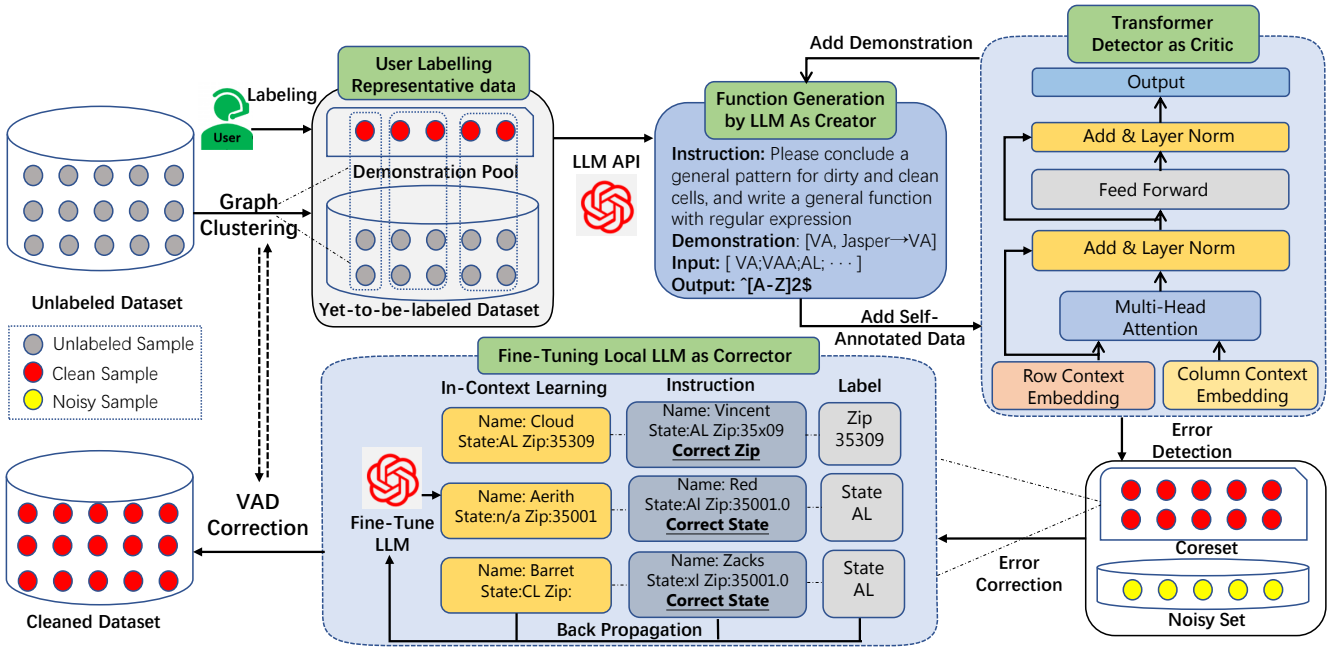


Figure 9: Overview of model architecture for GIDCL

Table 17: Datasets and discovered FDs from coreset $\mathcal{T}_{\text{coreset}}$

Name	$\lvert\mathcal{T}\rvert \times \lvert\mathcal{A}\rvert$	Error Rate	Error Types	FDs
Hospital	1000×20	3%	T, VAD	city → zip, city → county, zip → city, zip →state, zip → county, county → state, MeasureCode→Stateavg
Flights	2376×7	30%	MV, FI, VAD	flight →state actual departure time, flight →state actual arrival time, flight →state scheduled departure time, flight →state scheduled arrival time
Beers	2410×11	16%	MV, FI, VAD	city → state
Rayyan	1000×11	9%	MV, T, FI, VAD	journal abbreviation → journal title, journal abbreviation → journal issn, journal issn → journal title
Tax	200000×15	4%	T, FI, VAD	zip → city, zip → state, (married exemp,single exemp)→ marital status, child exemp → has child
IMDB	1000000×6	1%	T, FI, VAD	director → genres
Inpatients	4017×11	10%	MV, T, FI, VAD	provider-id → NPI, address-id → address line, address-id → city, address-id → state, address-id → zip, (state,zip)→ address-id
Facilities	7992×10	10%	MV, T, FI, VAD	ZipCode → City, ZipCode → State, ZipCode→ County , CCN → Facility, CCN → Address, CCN → PhoneNumber

Discovered Detection Pattern $\mathcal{F}^{\text{det}}$ for Dataset Rayyan

(article-title)

```python
def Rayyan_Detect_atitle(cell):
    # Regular expression to detect if the cell contains special characters or combining diacritical marks
    pattern = re.compile(r'[\u0300-\u036F]')
    return bool(pattern.search(cell))
```

journal-title

```python
def Rayyan_Detect_jtitle(cell):
    # Regular expression to detect if the cell contains special characters or combining diacritical marks
    pattern = re.compile(r'[\u0300-\u036F]')
```

journal-issn

```python
def Rayyan_Detect_issn(cell):
    # Regular expression to detect if the cell is in the format Mon-DD
    date_pattern = re.compile(r'^[A-Za-z]{3}-\d{1,2}$')

    # Regular expression to detect if the ISSN starts with a number other than 9
    issn_pattern = re.compile(r'^[^9]\d{12}$')

    return bool(date_pattern.match(cell) or issn_pattern.match(cell))
```

article-jvolumn/article-jissue

```python
def Rayyan_Detect_jissue(cell):
    pattern = re.compile(r'^\s*$')
    return bool(pattern.match(cell))
```

article-jcreated-at

```python
def Rayyan_Detect_jcreate(cell):
    try:
        match = re.match(r'(\d{2})/(\d{2})/(\d{2})', cell)
        if match:
            YY, MM, DD = match.groups()
            YY = int(YY)
            MM = int(MM)
            DD = int(DD)
            if (1 <= MM <= 12) and (1 <= DD <= 31):
                return True
        return False
    except:
        return False
```

article-pagination

```python
def Rayyan_Detect_pagination(cell):
    clean_pattern = re.compile(r'^(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)-\d{2}$')

    # Dirty pattern
    dirty_pattern1 = re.compile(r'^\d{2}-(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)$')
    dirty_pattern2 = re.compile(r'^\d{2}-\d$')

    if clean_pattern.match(cell):
        return False
    elif dirty_pattern1.match(cell) or dirty_pattern2.match(cell):
        return True
    else:
        return False
```

author-list

```python
def Rayyan_Detect_author(cell):
    # Regular expression to detect if the cell contains special characters or combining diacritical marks
    pattern = re.compile(r'[\u0300-\u036F]')
    return bool(pattern.search(cell))
```

Discovered Generation Pattern $\mathcal{F}^{\text{gen}}$ for Dataset Rayyan 1/2

(article-title)

```python
def Rayyan_Generate_atitle(cell):
    # List of special characters and combining diacritical marks
    special_chars = ['', '\u0301', '\u0300', '\u0302', '\u0303', '\u0304']

    # Randomly choose a special character
    char = random.choice(special_chars)

    # Randomly choose a position to insert the special character
    position = random.randint(0, len(cell))

    # Insert the special character at the chosen position
    dirty_cell = cell[:position] + char + cell[position:]

    return dirty_cell
```

journal-title

```python
def Rayyan_Generate_jtitle(cell):
    # List of special characters and combining diacritical marks
    special_chars = ['', '\u0301', '\u0300', '\u0302', '\u0303', '\u0304']

    # Randomly choose a special character
    char = random.choice(special_chars)

    # Randomly choose a position to insert the special character
    position = random.randint(0, len(cell))

    # Insert the special character at the chosen position
    dirty_cell = cell[:position] + char + cell[position:]

    return dirty_cell
```

journal-issn

```python
def Rayyan_Generate_issn(cell):
    # If the cell matches the format DD-Mon, reverse it to Mon-DD
    date_match = re.match(r'(\d{1,2})-([A-Za-z]{3})', cell)
    if date_match:
        return f"{date_match.group(2)}-{date_match.group(1)}"

    # If the cell matches the ISSN format starting with '9', change the leading '9' to another number
    issn_match = re.match(r'^9\d{12}$', cell)
    if issn_match:
        return f"{random.choice(['0', '1', '2', '3', '4', '5', '6', '7', '8'])}{cell[1:]}"

    return cell
```

article-jvolumn/article-jissue

```python
def Rayyan_Generate_jissue(cell):
    return ""
```

article-jcreated-at

```python
def Rayyan_Generate_jcreate(cell): ## Clean -> Dirty
    if cell:
        match = re.match(r'(\d{2})/(\d{2})/(\d{2})', cell)
        if match:
            MM, DD, YY = match.groups()
            DD = '{:02}'.format(int(DD))
            MM = '{:02}'.format(int(MM))
            return f'{YY}/{MM}/{DD}'
    return ''
```

Discovered Generation Pattern $\mathcal{F}^{\text{gen}}$ for Dataset Rayyan 2/2

article-pagination

```python
def Rayyan_Generate_pagination(cell):
    # Extract month and year from the clean format
    match = re.match(r'([A-Za-z]{3})-(\d{2})', cell)
    if not match:
        return cell # Return the original cell if it doesn't match the clean format

    month, year = match.groups()

    # Convert month to its corresponding number
    month_to_num = {
        'Jan': '1', 'Feb': '2', 'Mar': '3', 'Apr': '4', 'May': '5', 'Jun': '6',
        'Jul': '7', 'Aug': '8', 'Sep': '9', 'Oct': '10', 'Nov': '11', 'Dec': '12'
    }
    month_num = month_to_num.get(month, '')

    # Randomly decide to add extra numbers or not
    if random.choice([True, False]):
        year = str(random.randint(1000, 9999)) + year

    # Return the dirty format
    return f"{year}-{month_num}"
```

author-list

```python
def Rayyan_Generate_author(cell):
    # List of special characters and combining diacritical marks
    special_chars = ['', '\u0301', '\u0300', '\u0302', '\u0303', '\u0304']

    # Randomly choose a special character
    char = random.choice(special_chars)

    # Randomly choose a position to insert the special character
    position = random.randint(0, len(cell))

    # Insert the special character at the chosen position
    dirty_cell = cell[:position] + char + cell[position:]

    return dirty_cell
```

Discovered Correction Pattern $\mathcal{F}^{\text{corr}}$ for Dataset Rayyan 1/2

journal-issn

```python
def Rayyan_Correct_issn(cell):
    date_match = re.match(r'([A-Za-z]{3})-(\d{1,2})', cell)
    if date_match:
        return f"{date_match.group(2)}-{date_match.group(1)}"

    # If the cell matches the ISSN format not starting with '9', replace the leading digit with '9'
    issn_match = re.match(r'^[^9]\d{12}$', cell)
    if issn_match:
        return f"9{cell[1:]}"

    return cell # If no patterns match, return the original cel
```

article-jvolumn/article-jissue

```python
def Rayyan_Correct_jissue(cell):
    return "-1"
```

Discovered Correction Pattern $\mathcal{F}^{\text{corr}}$ for Dataset Rayyan 2/2

article-jcreated-at

```python
def Rayyan_Correct_jcreate(cell): ## Dirty -> Clean
    if(cell!=''):
        match = re.match(r'(\d{2})/(\d{2})/(\d{2})', cell)
        if match:
            YY, MM, DD = match.groups()
            YY = '{:02}'.format(int(YY))
            return f'{MM}/{DD}/{YY}'
    return ''
```

article-pagination

```python
def Rayyan_Correct_pagination(cell):
    clean_pattern = re.compile(r'^(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)-\d{2}$')

    # Dirty pattern
    dirty_pattern1 = re.compile(r'^(\d{2})-(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)$')
    dirty_pattern2 = re.compile(r'^(\d{2})-(\d)$')

    # If the cell matches the clean pattern, return it as is
    if clean_pattern.match(cell):
        return cell

    # If the cell matches the first dirty pattern, reverse month and year
    match = dirty_pattern1.match(cell)
    if match:
        return f"{match.group(2)}-{match.group(1)}"

    # If the cell matches the second dirty pattern, reverse month and year
    match = dirty_pattern2.match(cell)
    if match:
        month_map = {
            '1': 'Jan', '2': 'Feb', '3': 'Mar', '4': 'Apr', '5': 'May', '6': 'Jun',
            '7': 'Jul', '8': 'Aug', '9': 'Sep', '10': 'Oct', '11': 'Nov', '12': 'Dec'
        }
        return f"{month_map[match.group(2)]}-{match.group(1)}"

    # If the cell doesn't match any pattern, return it as is
    return cel
```

article-title/journal-title/author-list

```python
def Rayyan_Clean_atitle(cell):
    # Remove special characters like
    cleaned = re.sub(r'', '', cell)

    # Remove combining diacritical marks (from Unicode range U+0300 to U+036F)
    cleaned = re.sub(r'[\u0300-\u036F]', '', cleaned)

    return cleaned
```

Discovered Detection Pattern $\mathcal{F}^{\text{det}}$ for Dataset Beer

ounces

```python
def Beer_Detection_Ounces(cell):
    pattern = re.compile(r'^\d+$')
    return not bool(pattern.match(cell))
```

abv

```python
def Beer_Detection_abv(cell):
    # Regular expression to detect if the cell contains "%" or has more than three decimal places
    pattern = re.compile(r'%|^\d+\.\d{4,}$')
    return bool(pattern.search(cell))
```

city

```python
def Beer_Detection_city(cell):
    pattern = re.compile(r'\b[A-Z]{2}$')
    return bool(pattern.search(cell))
```

state

```python
def Beer_Detection_state(cell):
    return cell == ""
```

Discovered Detection Pattern $\mathcal{F}^{\text{det}}$ for Dataset IMDB

titleType/title

```python
def imdb_title_detection(cell):
    return cell.__contains__('x')
```

startYear

```python
def imdb_year_detection(cell):
    """
    Check if the given cell is dirty or not for the startYear column.

    :param cell: A string representing a cell from the startYear column.
    :return: Boolean indicating whether the cell is dirty.
    """
    # Regular expression for a clean cell: exactly four digits
    clean_pattern = r'^\d{4}$'

    # Return True (dirty) if cell does not match the clean pattern
    return not re.match(clean_pattern, cell)
```

runtimeMinutes

```python
def imdb_runtime_detection(cell):
    pattern = r'^\d+$'
    return not re.match(pattern, cell)
```

director

```python
def imdb_director_detection(cell):
    # Regular expression to detect if the cell contains special characters or combining diacritical marks
    pattern = re.compile(r'[\u0300-\u036F]')
    return bool(pattern.search(cell))
```

Discovered Generation Pattern $\mathcal{F}^{\text{gen}}$ for Dataset Beer

ounces

```python
def Beer_Generation_Ounces(cell):
    descriptors = [" oz.", " ounce", " OZ.", " oz. Alumi-Tek", " oz. Silo Can"]
    value = correct_dirty_cell(cell)
    descriptor = random.choice(descriptors)

    # Append the descriptor to the cell to make it dirty
    dirty_cell = value + descriptor

    return dirty_cell
```

abv

```python
def Beer_Generation_abv(cell):
    choices = ["append_percent"]
    action = random.choice(choices)

    if action == "append_percent":
        return cell + "%"
    elif action == "alter_float" and "." in cell:
        # Introduce a small random change to the floating point
        parts = cell.split(".")
        if len(parts[1]) == 3:
            last_digit = str(int(parts[1][2]) + random.choice([-1, 1]) % 10) # Increment or decrement the last digit
            return parts[0] + "." + parts[1][:2] + last_digit + "%"
```

city

```python
def Beer_Generation_city(cell):
    state_abbreviations = ['AL', 'AK', 'AZ', 'AR', 'CA', 'CO', 'CT', 'DE', 'FL', 'GA', 'HI', 'ID', 'IL', 'IN', 'IA', 'KS', 'KY', 'LA', 'ME', 'MD', 'MA', 'MI', 'MN', 'MS', 'MO', 'MT', 'NE', 'NV', 'NH', 'NJ', 'NM', 'NY', 'NC', 'ND', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VT', 'VA', 'WA', 'WV', 'WI', 'WY']

    # Randomly choose a state abbreviation
    state = random.choice(state_abbreviations)

    # Append the state abbreviation to the cell to make it dirty
    dirty_cell = cell + " " + state

    return dirty_cell
```

state

```python
def Beer_Generation_state(cell):
    return ""
```

Discovered Generation Pattern $\mathcal{F}^{\text{gen}}$ for Dataset IMDB

titleType/title

```python
def imdb_title_generation(input_string):
    if not input_string:
        return input_string
    char_to_replace = random.choice(input_string)
    result_string = input_string.replace(char_to_replace, 'x')
    return result_string
```

startYear

```python
def imdb_year_generation(clean_cell):
    """
    Generate a dirty cell from a clean one. The clean cell is assumed to be
    a four-digit year. The dirty cell will be the last two digits of the year.

    :param clean_cell: A string representing a clean cell (four-digit year).
    :return: A string representing a dirty cell (last two digits of the year).
    """
    # Extract the last two digits of the year
    dirty_cell = clean_cell[-2:]

    return dirty_cell
```

runtimeMinutes

```python
def imdb_runtime_generation(clean_cell):
    minutes = int(clean_cell)

    # Convert minutes to hours and add a random level of decimal precision
    hours = minutes / 60
    precision = random.choice([1, 2, 3]) # Random precision level
    formatted_hours = round(hours, precision)

    # Format the dirty cell
    dirty_cell = f"{formatted_hours} h"

    return dirty_cell
```

director

```python
def imdb_director_generation(cell):
    special_chars = ['', '\u0301', '\u0300', '\u0302', '\u0303', '\u0304']

    # Randomly choose a special character
    char = random.choice(special_chars)

    # Randomly choose a position to insert the special character
    position = random.randint(0, len(cell))

    # Insert the special character at the chosen position
    dirty_cell = cell[:position] + char + cell[position:]

    return dirty_cell
```

Discovered Detection Pattern $\mathcal{F}^{\text{det}}$ for Dataset Tax

f-name/l-name

```python
def Tax_Detect_fname(cell):
    return bool(re.search(r"'", cell))
```

city/state/single-exemp/child-exemp

```python
def Tax_Detect_city(cell):
    return bool(re.search(r'-\*$', cell))
```

zip

```python
def Tax_Detect_zip(cell):
    return cell == '1907'
```

Discovered Generation Pattern $\mathcal{F}^{\text{den}}$ for Dataset Tax

f-name/l-name

```python
def Tax_Generate_fname(cell):
    # Find all positions of single quotes in the cell
    positions = [i for i, char in enumerate(cell) if char == "'"]

    # If there's no single quote, return the original cell
    if not positions:
        return cell

    # Randomly choose a position from the positions of single quotes
    chosen_position = random.choice(positions)

    # Insert an additional single quote at the chosen position
    dirty_cell = cell[:chosen_position] + "'" + cell[chosen_position:]

    return dirty_cell
```

city/state/single-exemp/child-exemp

```python
def Tax_Generate_city(cell,num_samples=5):
    dirty_cell = cell + '-*'
    return dirty_cell
```

zip

```python
def Tax_Generate_zip(cell):
    return '1907'
```

Discovered Correction Pattern $\mathcal{F}^{\text{corr}}$ for Dataset Tax

f-name/l-name

```python
def Tax_Correct_fname(dirty_cell):
    return dirty_cell.replace("'", "'")
```

city/state/single-exemp/child-exemp

```python
def Tax_Correct_city(cell):
    # Use regex to remove the '-*' pattern from the end of the string
    corrected_value = re.sub('-\*$', '', cell)
    return corrected_value
```

Discovered Detection Pattern $\mathcal{F}^{\text{det}}$ for Dataset Hospital 1/3

ProviderNumber

```python
def Hospital_Detect_ProviderNum(cell):
    # This regex pattern searches for one or more consecutive 'x' characters in a given cell. If a match is found, the function returns `True`, indicating that the cell is dirty. Otherwise, it returns `False`, indicating that the cell is clean.
    return bool(re.search(r'x{1,}', cell))
```

HospitalName

```python
def Hospital_Detect_HospitalName(cell):
    # This function uses the `re.search()` function from the `re` module to search for a match of the regular expression `r'[xX]\w+'` in the given cell. If a match is found, the function returns `True`, indicating that the cell is dirty. Otherwise, it returns `False`, indicating that the cell is clean.
    return bool(re.search(r'[xX]\w+', cell))
```

Address1

```python
def Hospital_Detect_Address1(cell):
    # This function uses the regular expression `r'x\w+'` to search for one or more "x" characters followed by one or more word characters in the given cell. If a match is found, the function returns `True`, indicating that the cell is dirty. Otherwise, it returns `False`.
    return bool(re.search(r'x\w+', cell))
```

City

```python
def Hospital_Detect_City(cell):
    # This regular expression looks for a single occurrence of the letter "x" or "X" followed by one or more word characters. If a match is found, the function returns `True`, indicating that the cell is dirty. If no match is found, the function returns `False`, indicating that the cell is clean
    return bool(re.search(r'[xX]\w+', cell))
```

State

```python
def Hospital_Detect_State(cell):
    # This function uses the `re.search()` function to search for a match in the given cell using the specified regular expression pattern. If a match is found, the function returns `True`, otherwise it returns `False`.
    return bool(re.search(r'[xX][\w\d]*', cell))
```

ZipCode

```python
def Hospital_Detect_ZipCode(cell):
    # This regular expression looks for one or more occurrences of the letter "x" in a cell. If it finds any, the cell is considered dirty. If not, the cell is considered clean.
    return bool(re.search(r'x{1,}', cell))
```

CountyName

```python
def Hospital_Detect_CountyName(cell):
    # This regular expression looks for the letter "x" followed by one or more word characters, which matches the pattern observed in the dirty cells. The `is_dirty` function returns `True` if the regular expression matches the input cell, and `False` otherwise.
    return bool(re.search(r'x\w+', cell))
```

Discovered Detection Pattern $\mathcal{F}^{\text{det}}$ for Dataset Hospital 2/3

PhoneNumber

```python
def Hospital_Detect_PhoneNumber (cell):
    # This function uses the regular expression `r'[xX]{1,}'` to search for one or more consecutive "x" or "X" characters in the given cell. If the search returns a match, the cell is considered dirty, and the function returns `True`. Otherwise, the cell is clean, and the function returns `False`.
    return bool(re.search(r'[xX]{1,}', cell))
```

HospitalType

```python
def Hospital_Detect_HospitalType (cell):
    # This regular expression uses the `re` module in Python to search for one or more consecutive "x" characters in the given cell. If a match is found, the function returns `True`, indicating that the cell is dirty. Otherwise, it returns `False`, indicating that the cell is clean.
    return bool(re.search(r'x{1,}', cell))
```

HospitalOwner

```python
def Hospital_Detect_PhoneNumber (cell):
    # This regular expression will match any sequence of one or more "x" characters followed by one or more word characters. If such a match is found in a cell, the function returns `True`, indicating that the cell is dirty. Otherwise, it returns `False`.
    # This regular expression should work for the given examples as well as any similar variations where "x" characters appear in the wrong places in the cell text. However, it's important to note that this regular expression might not cover all possible variations of dirty cells, and additional checks or modifications might be necessary depending on the specific requirements of the data.
    return bool(re.search(r'[xX]\w+', cell))
```

EmergencyService

```python
def Hospital_Detect_EmergencyService (cell):
    # This regular expression `[xX]{1,}` matches one or more consecutive occurrences of "x" or "X" characters, and the `bool(re.search(...))` function returns `True` if a match is found and `False` otherwise.
    return bool(re.search(r'[xX]{1,}', cell))
```

Condition

```python
def Hospital_Detect_Condition (cell):
    # This regular expression `r'x\w+'` matches one or more occurrences of the letter "x" followed by one or more word characters. The `bool()` function returns `True` if a match is found and `False` otherwise..
    return bool(re.search(r'x\w+', cell))
```

MeasureName

```python
def Hospital_Detect_MeasureName (cell):
    # Explanation of the regular expression:

    # 1. `[^A-Za-z0-9_ ]+` matches one or more non-alphanumeric or underscore characters.
    # 2. `|` is a pipe character that separates alternative patterns.
    # 3. `[xX]\w+` matches one or more word characters (letters or digits) following an "x" or "X".
    # 4. `[^ ]+` matches one or more non-space characters.
    # 5. `+` indicates that the preceding pattern should match one or more times
    return bool(re.search(r'[^A-Za-z0-9_ ]+|[xX]\w+|[^ ]+ [xX]\w+', cell))
```

Discovered Detection Pattern $\mathcal{F}^{\text{det}}$ for Dataset Hospital 3/3

Score

```python
def Hospital_Detect_Score (cell):
    # This regular expression uses the `{1,}` quantifier to
          match one or more consecutive occurrences of the
          character "x" or "X". The function returns `True`
          if a match is found, indicating that the cell is
          dirty. Otherwise, it returns `False`.
    return return bool(re.search(r'[xX]{1,}', cell))
```

Sample

```python
def Hospital_Detect_Sample (cell):
    # This regular expression `r'[xX]{1,}'` will match any
          sequence of one or more "x" or "X" characters in a
           cell, making it suitable for detecting dirty
          cells as defined in the examples. The `is_dirty()`
           function returns `True` if a match is found and `
          False` otherwise.
    return bool(re.search(r'[xX]{1,}', cell))
```

Stateavg

```python
def Hospital_Detect_Stateavg(cell):
    # This regular expression matches the pattern of "x"
          followed by one or more word characters ("\w+"),
          which appears in the dirty cell examples provided.
           The function returns `True` if the regular
          expression matches the cell and `False` otherwise.
    return bool(re.search(r'x\w+', cell))
```

Discovered Generation Pattern $\mathcal{F}^{\text{gen}}$ for Dataset Hospital

All Attributes in Dataset Hospital

```python
def Hospital_Generate(input_string):
    if not input_string:
        return input_string

    char_to_replace = random.choice(input_string)

    result_string = input_string.replace(char_to_replace, 'x
          ')

    return result_string
```

Discovered Detection Pattern $\mathcal{F}^{\text{det}}$ for Dataset Flights

actual departure time, actual arrival time, scheduled departure time, scheduled arrival time

```python
def Flights_Detection(time_str):
    # Define a regular expression pattern for a clean time
          format (HH:MM a.m./p.m.)
    time_pattern = r'^\d{1,2}:\d{2} (a\.m\.|p\.m\.)$'

    # Use regex to check if the time string matches the
          expected pattern
    if re.match(time_pattern, time_str):
        return False
    else:
        return True
```

Discovered Generation Pattern $\mathcal{F}^{\text{gen}}$ for Dataset Flights

actual departure time, actual arrival time, scheduled departure time, scheduled arrival time

```python
def Flight_Row_Generation(cell):
    task = [0,1,2,3,4,5,6]
    task_select = np.random.choice(task)
    if(task_select==0):
        cell_output = cell.replace(' ','').replace('.','')
        cell_output = cell_output[:-1] + 'Dec 1'
    elif(task_select==1):
        cell_output = '11/30 ' + cell
    elif(task_select==2):
        cell_output = 'Not Available'
    elif(task_select==3):
        cell_output = cell.replace(' ','').replace('.','')
        cell_output = cell_output[:-2] + 'noon'
    elif(task_select==4):
        cell_output = cell + ' (-00:00)'
    elif(task_select==5):
        cell_output = '12/02/2011 ' + cell
    else:
        cell_output = 'Dec 02 ' + cell
    return cell_output
```