GEIL: 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 GEIL (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, GEIL 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. GEIL 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 GEIL significantly outperforms existing methods, improving F1-scores by 10% on average while requiring only 20 labeled tuples.

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 [3], 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 [77], denial constraints [20], conditional functional dependencies [37], 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 [3].

Meanwhile, multiple configuration-free DC approaches, *e.g.,* [52, 76, 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 [21]. However, these configuration-free DC methods still heavily rely on predefined feature engineering and often struggle with undefined error patterns[74, 76]. Furthermore, they are ineffective against complex textual errors when the correct values do not exist within the dataset[86, 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 [26] 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 [1], 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[88].

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

Large language models (LLMs) [14, 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

[1]Here interpretability means DC patterns, rules and dependencies that can be explicitly verified by human beings[86], for both error detection and correction.

substantial obstacles (details in Section 2.3):

- **Understanding Dependencies:** Due to token limitations, LLMs struggle to comprehend data quality rules and the inherent dependencies across entire relational tables, which may result in inconsistent data cleaning outcomes across different instances.
- **Hallucination:** The difficulty in understanding complex dependencies predisposes LLMs to the hallucination issue, where they might generate plausible but incorrect or fictional data repairs.
- **Overfitting with Limited labelling data:** Fine-tuning LLMs on limited labelling data can lead to overfitting, where the model performs well on training data but poorly on unseen data.
- **Efficiency Issues:** The size and complexity of LLMs can lead to efficiency problems, since it is unrealistic to apply DC over all tuples solely and sequentially with LLMs.

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 method with Large language models, denoted by GEIL for short, that achieves both high precision and recall. GEIL 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.

GEIL 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 captures 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 GEIL.

GEIL effectively addresses the challenges of applying LLMs to data cleaning. It uses graph neural networks to enhance dependency understanding, overcoming LLMs' token constraints and ensuring consistent results. The creator-critic workflow mitigates hallucination by refining models iteratively for reliable repairs. GEIL's few-shot learning capabilities reduce the risk of overfitting despite limited labeled data. Overall efficiency is improved by optimizing the processing pipeline, allowing GEIL to manage the large size and complexity of LLMs efficiently. These innovations provide a robust, interpretable framework for integrating LLMs into data cleaning processes.

By leveraging the rich prior knowledge of LLMs, GEIL can automatically extract and generate data cleaning rules using natural language, effectively overcoming the challenges posed by traditional DC methods that rely on handcrafted rules. GEIL 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 often rely on extensive manual feature engineering. Furthermore, GEIL requires only a minimal amount of labeled data (*e.g.,* 20 tuples) compared to exist 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, GEIL achieves both highly effectiveness and efficiency, while extracting and generating interpretable DC patterns and dependencies with LLM. Our contributions are summarized as follows:

(1) **An end-to-end framework for data cleaning.** We introduce GEIL, 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)

(4) **Extensive experiments.** We conduct thorough experiments to evaluate the performance of GEIL under various error types. GEIL outperforms a variety of state-of-the-art data cleaning baselines w.r.t. efficiency, effectiveness and robustness within local consumer-level GPUs. (Section 7)

2 BACKGROUND

In this section we first formally present the main concepts behind the data cleaning (DC) and provide background on graph neural networks (GNNs), pre-trained language models (PLMs) and large language models (LLMs) before we describe how we unify and fine-tune them to train specific DC models.

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 [74, 86, 94], our objective encompasses the de-

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

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

tection 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 $\mathcal{T}$, ensuring that as many errors with $\mathcal{T}$ as possible are identified and accurately rectified. 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 whose all attributes are clean.

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 [74, 76] for the following reasons. Firstly, 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. Secondly, errors detected in the error detection phase might propagate to the error correction phase. In essence, if an error remain unidentified, it will likely remain unaddressed. Lastly, data cleaning involces addressing various types of errors, such as inconsistencies, missing values, and typos. Consequently accurately learning and repairing erroneous cells with correct values is a non-trivial task.

Table 2: **General notations with corresponding descriptions.**

Symbol	Description
$\mathcal{T}$	the relational table
$\mathcal{G}$	the directed graph, transferred from $\mathcal{T}$
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 $\mathcal{T}$
$\mathcal{T}^+, t_{i,j}^+$	the clean version of table $\mathcal{T}$ (resp. cell $t_{i,j}$)
$f_{A_j}^{\text{det}} \in \mathcal{F}^{\text{det}}$	detection function for the j-th attribute
$f_{A_j}^{\text{gen}} \in \mathcal{F}^{\text{gen}}$	generation function for the j-th attribute
$f_{A_j}^{\text{corr}} \in \mathcal{F}^{\text{corr}}$	correction function for the j-th attribute
$\mathcal{F}$	function set containing $\mathcal{F}^{\text{det}}, \mathcal{F}^{\text{gen}}, \mathcal{F}^{\text{corr}}$
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
$\mathcal{T}_{\text{pseudo}}$	self-generated tuples in $\mathcal{T}$, labelling by GEIL
$\mathcal{T}_{\text{coreset}}$	coreset in $\mathcal{T}$, divided by error detection model
$\mathcal{T}_{\text{err}}$	detected erroneous cells in $\mathcal{T}$
D^{train}	training data for $\mathcal{M}_{\text{det}}$, $\mathcal{M}_{\text{corr}}$
O	the domain of all objects in $\mathcal{T}$

2.2 Preliminary

We delves into the technical backgrounds GNNs, PLMs and LLMs, which are critical to the GEIL framework. While these technologies have individually advanced their respective fields, their combination within our data cleaning framework presents a novel and challenging endeavor.

Graph Neural Networks. Consider a graph $\mathcal{G} = (V, E, L)$, where V, E, and L denote the sets of vertices, edges and labels in $\mathcal{G}$, respectively. GNNs generates embeddings for each vertex $v \in V$ by leveraging its attributes and recursively aggregating messages from neighboring vertices, *i.e.,* $v' \in V$ and $(v, v') \in E$ [50]. The mechanism of GNNs enables the modeling of feature interactions and associations among vertices from a more unified and generalized perspective.

Pre-Trained Language Models. Traditional pre-trained language models (PLMs), *e.g.,*Bert [28], GPT-2[92], RoBERTa [71] and T5 [93], have demonstrated remarkable performance on a wide range of NLP tasks by using Transformer architecture [109] with the encoder-decoder structure. To achieve more prior knowledge, PLMs are usually pre-trained on a large amount of text corpora in a self-supervised learning manner, and further adopted to multiple downstream tasks later, *e.g.,* classification and regression. When using them for a specific task, one should add a task-specific layer after PLMs and fine-tune the parameters so that they would adapt to it.

Large Language Models Typically, large language models (LLMs) refer to Transformer language models that contain billions (or more) of parameters[122], which are trained on massive text data[102]. Representative LLMs contain OpenAI's online GPT series (in particular, GPT-3, 3.5, and 4) and open-source LLaMa [107], and exhibit strong capacities to understand natural language and solve complex tasks via text generation. However, LLMs also exhibit hallucination behaviors when presented with query that surpasses their prior knowledge or capacities, resulting in generating factual errors or unrelated answers[122]. To address this issue, the following methods are commonly employed to constrain the response of LLMs:

- Prompt: Serving the same purpose as instructions, prompts are utilized to interact with LLMs for accomplishing specific tasks. Prompt plays a critical role in customizing LLMs to ensure that the responses align with the requirements of particular use.
- In-Context Learning(ICL): This method involves with prompt engineering, where demonstrations of the task are included as part of the prompt[30].
- Retrieval Augmented Generation(RAG): RAG is a technique used to retrieve relevant contextual data entries and provide them to the model as references, thereby enhancing the quality of LLM responses without directly modifying the parameters [65].

2.3 Challenges of applying LLMs for DC

Applying LLMs directly to DC introduces several challenges: **(1) Understanding Dependencies** [104], Due to token limitation (the maximum input length, e.g., 4k tokens for GPT-3.5), LLMs often struggle to fully comprehend data quality rules and the complex dependencies inherent over whole relational tables, and the DC results may exhibit inconsistency across different instances; **(2) Hallucination** [57], as the lack of necessary context information and demonstrations can lead LLMs to generate plausible but incorrect or fictional data repairs; **(3) Overfitting with Limited Labeling Data** [59], where fine-tuning LLMs on a small dataset can lead to models performing well on training data but poorly on unseen data; and **(4) Efficiency Issues** [107], as the size and complexity of LLMs can lead to problems when attempting to apply DC across all tuples using only LLM. Below, we will verify the aforementioned challenges through experimentation and demonstration. A straightforward approach to implementing LLMs for the data cleaning task is as follows.

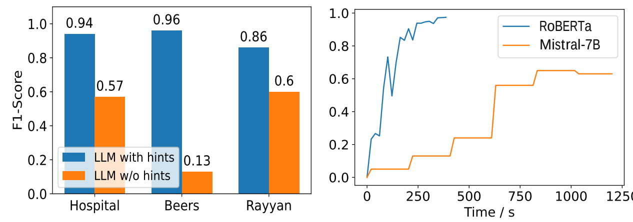
(a) Error detection $\mathcal{M}_{\text{det}}$. By comparing each pair of labeled cells $(t_{i,j}, t^+_{i,j}) \in \mathcal{T}_{\text{label}}$, one could design a training pair $(x_{i,j}, y_{i,j})$ for detecting attribute A_j for a tuple t_i, and fine-tune a LLM-based error detection model $\mathcal{M}_{\text{det}}$. Here $x_{i,j} = (p^{A_j}_{\text{det}}, \text{serial}(t_i), t_{i,j})$, $p^{A_j}_{\text{det}}$ is an instruction to identify errors in A_j, and 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 $x_{i,j} = (p^{A_j}_{\text{corr}}, \text{serial}(t_i), t_{i,j})$, $y_{i,j} = t^+_{i,j}$, $p^{A_j}_{\text{corr}}$ 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.

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) [123]. (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.



(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

3 A DATA CLEANING FRAMEWORK

In this section, we introduce our end-to-end data cleaning framework GEIL 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.

Architecture. The ultimate goal of GEIL is to identify and repair erroneous cells in $\mathcal{T}$. As depicted in Figure 2, GEIL initializes the process by transforming $\mathcal{T}$ into a graph $\mathcal{G}$, facilitating the clustering of semantic and structural similar tuples through a graph structure learning strategy. Subsequently, GEIL 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, GEIL augments LLMs to correct errors in a retrieval-augmented generation manner, utilizing a few representative examples as in-context demonstrations.

More specifically, GEIL 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, graph $\mathcal{G}$ transferred 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

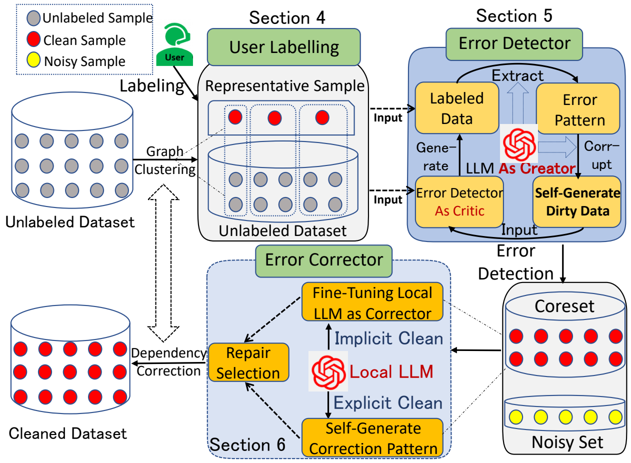


Figure 2: Overview of GEIL

tuples that requires labelling by users.

(2) Error detection GEIL_{det}. The error detection component in GEIL named GEIL_{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 divided coreset $\mathcal{T}_{\text{coreset}}$ identified as clean cells, and self-generated training data $\mathcal{T}_{\text{pseudo}}$.

(i) Critic of GEIL_{det}. GEIL 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. The input and output of critic are as follows. Then the critic will gradually provide dirty and clean samples to LLM-based creator.

- *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 GEIL_{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 summarize a transformation function $f^{\text{det}}_{A_i}$ for each attribute $A_i \in \mathcal{A}$. The creator accepts $f^{\text{det}}_{A_i}$ if the F-measure of $\mathcal{T}_{\text{label}}$ after executing it for error detection surpasses a predefined threshold τ. we denote $\mathcal{F}^{\text{det}} = \{f^{\text{det}}_{A_1}, \ldots, f^{\text{det}}_{A_n}\}$ as the detection function set.

In the second step, utilizing the accepted $f^{\text{det}}_{A_i}$, a generation function $f^{\text{gen}}_{A_i}$ is generated by LLM, corrupting clean cells to dirty ones according to $f^{\text{det}}_{A_i}$. Specifically, given a cell $t^{+}_{j,i}$ predicted as correct by $\mathcal{M}_{\text{det}}$, its dirty value is generated as $t_{j,i} = f^{\text{gen}}_{A_i}(t^{+}_{j,i})$. Finally the creator samples a few correct cells in attribute A_i, and corrupt them to the dirty ones via $f^{\text{gen}}_{A_i}$, 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 $f^{\text{det}}_{A_i}$ for all attributes. In other word, the termination condition is either (1) no $f^{\text{det}}_{A_i}$ 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 divided. We denote the labeled data $D^{\text{train}} = \mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}$.

(3) Error correction GEIL_{cor}. In this phase, regarding the detected error cells $\mathcal{T}_{\text{err}}$, the error correction component GEIL_{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, GEIL_{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 clean and representative examples as context from D^{train}, as well as labeled tuples for demonstration from $\mathcal{T}_{\text{label}} \cup \mathcal{T}_{\text{pseudo}}$. The formal input and output are as follows.

- *Input*: labeled data $\mathcal{T}_{\text{label}}$, clean cells $\mathcal{T}_{\text{coreset}}$, self-generate data $\mathcal{T}_{\text{pseudo}}$.
- *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 summarize 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 GEIL_{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, GEIL 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. Then we propose a simple but effective tuple selection strategy to select θ representative tuples to users for manual labeling.

Graph construction. We transform $\mathcal{T}$ into a directed 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.

Example 2: Consider tuples t_1, t_2 with PrivoderID, 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 $E = \{(t_1, \text{City}, \text{Monticello}), (t_2, \text{City}, \text{Monticello}), (t_1, \text{PrivoderID}, 111303), (t_2, \text{PrivoderID}, 111303), (t_1, \text{County}, \text{Jasper}), (t_2, \text{County}, \text{Jasper})\}$. □

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 a few fake triples by corrupting h and t in a triple $e = (h, r, t)$ by using some heuristic corruption strategies [108]. After corrupting all triplets in E, we generate the same number of fake tuples and create the training set $\mathcal{G}_{\text{train}}$, such that $y(e) = 1$ for true triples e and 0 for fake ones.

(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.

(c) Self-supervised learning. We apply ComplEx [108] to learn embedded vectors for all vertices in V. The loss function is defined as follows:

$$f(t) = \Re\left(\langle w_r, e_h, \overline{e_t}\rangle\right)$$
$$Loss(\Theta) = \sum_{(h,r,t)\in E} \log\left(1 + e^{-y(h,r,t)f(h,r,t)}\right) + \lambda\|\Theta\|_2^2$$

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 $\|\cdot\|_2^2$ is the Euclidian norm.

Representative tuple selection. Following the acquisition of embeddings for all tuples, 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, GEIL 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 divide 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 $\mathcal{T}_{\text{label}}$. □

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 [26], identifies each cell $t_{i,j}$ in $\mathcal{T}$ as clean or dirty. This is achieved by mapping a serialized sequence O 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 [78]. 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 $\mathcal{M}_{\text{det}}$. Specifically, 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. For the column-contextual dependency, we select 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:

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

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

(1) For $t_{3,2} \in \mathcal{T}_{\text{label}}$, we have the serialization of row-contextual dependency as ⟨COL⟩ *ProviderID* ⟨VAL⟩ *1x1303* ⋯ ⟨COL⟩ *State* ⟨VAL⟩ *AR* [SEP] ⟨COL⟩ *ProviderID* ⟨VAL⟩ *1x1303*. Also, its serialization of column-contextual dependency is ⟨COL⟩ *ProviderID* ⟨VAL⟩ *1x1303* ⋯ ⟨VAL⟩ *111303* [SEP] ⟨COL⟩ *ProviderID* ⟨VAL⟩ *1x1303*. The label of the two serializations is 0, indicating $t_{3,2}$ is a negative instance.

(2) For clean cell $t_{3,2}^+ \in \mathcal{T}_{\text{label}}$, we also serialize it as (i) the row-contextual dependency ⟨COL⟩ *ProviderID* ⟨VAL⟩ *111303* ⋯ ⟨COL⟩ *State* ⟨VAL⟩ *AR* [SEP] ⟨COL⟩ *ProviderID* ⟨VAL⟩ *111303* and (ii) the column-contextual dependency ⟨COL⟩ *ProviderID* ⟨VAL⟩ *1x1303* ⋯ ⟨VAL⟩ *111303* [SEP] ⟨COL⟩ *ProviderID* ⟨SEP⟩ *111303*. We label them as 1, indicating positive instances. □

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 function for error detection.** Existing approaches, *e.g.*,Raha [76] and activeClean [62], 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 the recent arise of LLMs sheds light on it. [15] found that LLMs are few-shot learners and could extract the generalized patterns to distinguish clean and dirty values with limited labeled examples, acting like a data scientists [17, 81]. 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 [91].

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 $f_{\text{State}}^{\text{det}}$ 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; ⋯]

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; ⋯]

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 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, ⋯]
- $f_{\text{ProviderID}}^{\text{det}}$: ^\d+$ (ProviderID only contains numbers).
- Values $v(A_j)$: [1x1303; 111303; 10001; ⋯]

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

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.

<u>Creator Phase.</u> 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. This approach helps prevent LLMs from memorizing specific cases in $\mathcal{T}_{\text{label}}$ and avoids hallucination issues. The multi-turn conversations based on LLMs are iteratively invoked until the evaluation performance of $f_{A_j}^{\text{det}}$ over $\mathcal{T}_{\text{label}}$, 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 $\mathcal{T}_{\text{pseudo}}$. 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 creator $\mathcal{M}_{\text{det}}$.

<u>Critic Phase.</u> We serialize D^{train} into a set of bi-level context-aware

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.

$\mathcal{T}_{\text{pseudo}} := \emptyset$, $\mathcal{M}_{\text{det}} := \emptyset$, $\mathcal{T}_{\text{coreset}} := \emptyset$;
while true **do**
  /* Creator */
  $\mathcal{F}^{\text{det}} := \emptyset$, $\mathcal{F}^{\text{gen}} := \emptyset$;
  for each $A_i \in \mathcal{A}$ **do**
    iter := 0, $v(A_i) := \mathcal{T}[A_i]$, $f^{\text{det}}_{A_i} := \text{LLM}(p_1, \mathcal{T}_{\text{label}}, v(A_i))$;
    while $\text{Eval}(f^{\text{det}}_{A_i}, \mathcal{T}_{\text{label}}) \leq \tau$ and iter $\leq \text{Iter}_{\max}$ **do**
      Collect $\mathcal{T}^{\text{wrong}}_{\text{label}} := \{(t_{j,k}, t^+_{j,k}) | 1 \leq j \leq |D^{\text{train}}|, 1 \leq k \leq |\mathcal{A}|, f^{\text{det}}_{A_i}(t_{j,k}) = 1, t_{j,k} \neq t^+_{j,k}\}$
      $f^{\text{det}}_{A_i} := \text{LLM}(p_2, \mathcal{T}^{\text{wrong}}_{\text{label}}, v(A_i))$;
      iter := iter + 1;
    Add $f^{\text{det}}_{A_i}$ into $\mathcal{F}^{\text{det}}$ if $\text{Eval}(f^{\text{det}}_{A_i}, \mathcal{T}_{\text{label}}) > \tau$;
    $f^{\text{gen}}_{A_i} := \text{LLM}(p_3, \mathcal{T}_{\text{label}}, f^{\text{det}}_{A_i}, v(A_i))$;
    $\Delta\mathcal{T}_{\text{pseudo}} := \{t^{\text{pseudo}}_{z,i} | t^{\text{pseudo}}_{z,i} := f^{\text{gen}}_{A_i}(t^+_{z,i}), (t_z, t^+_z) \in \mathcal{T}_{\text{label}}\}$;
    $\mathcal{T}_{\text{pseudo}} := \mathcal{T}_{\text{pseudo}} \cup \Delta\mathcal{T}_{\text{pseudo}}$;
  /* Critic */
  $D^{\text{train}} := \mathcal{T}_{\text{pseudo}} \cup \mathcal{T}_{\text{label}}$;
  Generate row/column-contextual dependency for each $t \in D^{\text{train}}$;
  Fine-tune $\mathcal{M}_{\text{det}}$ using D^{train};
  Select a subset $\Delta\mathcal{T}_{\text{coreset}} \subseteq \mathcal{T}$ with high confidences of $\mathcal{M}_{\text{det}}$;
  $\mathcal{T}_{\text{label}} := \mathcal{T}_{\text{label}} \cup \Delta\mathcal{T}_{\text{coreset}}$, $\mathcal{T}_{\text{coreset}} := \mathcal{T}_{\text{coreset}} \cup \Delta\mathcal{T}_{\text{coreset}}$;
  if $\mathcal{F}^{\text{det}}$ does not change **do**
    break
Identify all errors $\mathcal{T}_{\text{err}}$ in $\mathcal{T}$ using $\mathcal{M}_{\text{det}}$;
return $(\mathcal{T}_{\text{err}}, \mathcal{T}_{\text{presudo}}, \mathcal{T}_{\text{coreset}})$;

Figure 3: The Creator-Critic workflow of error detection

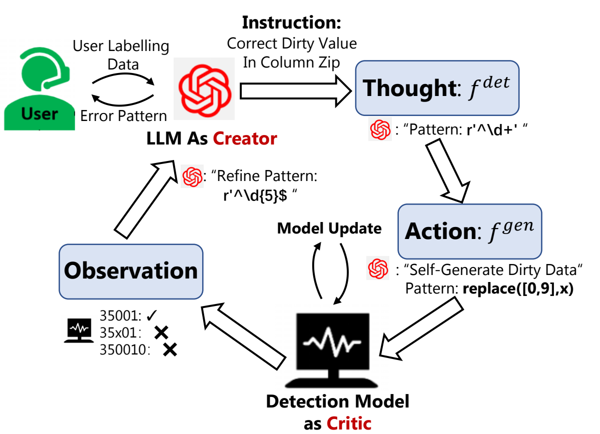


Figure 4: Illustration of iterative workflow of Creator-Critic for error detection

instances, and fine-tune the relatively small PLM-based detection model $\mathcal{M}_{\text{det}}$ until convergence. 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 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. 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 $\mathcal{T}_{\text{coreset}}$.

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 $\mathcal{T}_{\text{clean}}$.

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

Figure 5: The error correction algorithm

6 ERROR CORRECTION

In this section, we show how to correct a dirty cell $t_{i,j}$ to the clean value $t^+_{i,j}$. In particular, 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) [30, 80], retrieval augmented generation (RAG) [65], and supervised fine-tuning (SFT) to learn an error correction model $\mathcal{M}_{\text{corr}}$ that directly generates the true value by referencing its dirty one and some contextual information.

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

Specifically, for each (dirty, clean) cell $(t_{i,j}, t^+_{i,j}) \in D^{\text{train}}$, we construct a sequence denoted by $\text{context}(t_{i,j})$ for fine-tuning LLM with ICL and RAG context. In detail, $\text{context}(t_{i,j})$ contains a hand-crafted prompt q, the serial representation of the cell and its context $\text{serial}(t_{i,j})$, 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 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 [54]. 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 $\longrightarrow$ AL]
- RAG Context: serial(t_1), where t_1 and t_2 are in the same cluster.

Finally $\mathcal{M}_{\text{corr}}$ rectifies $t_{2,3}$ to VA. □

Explicit 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 $\longrightarrow$ 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, $f^{\text{corr}}_{\text{State}}(t_{5,4})$ = 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 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 Conf$(\mathcal{M}_{\text{corr}}(t_{i,j}))$ and Conf$(f^{\text{corr}}_{A_j}(t_{i,j}))$. The value with a larger confidence score is the final repair, denoted as GEIL$_{\text{cor}}(t_{i,j})$. By applying GEIL$_{\text{cor}}$ over all erroneous cells $\mathcal{T}_{\text{err}}$, we can repair the dirty table $\mathcal{T}$ to $\mathcal{T}_{\text{clean}}$, and update the transferred 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 GEIL$_{\text{cor}}$, we then apply $\mathcal{M}_{\text{det}}$ to select a few tuples $\mathcal{T}_{\text{coreset}}$ that have high confidence scores. Then we focus on mining FDs in $\mathcal{T}_{\text{coreset}} \cup \mathcal{T}_{\text{label}}$.

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, where $A_i, A_j \in \mathcal{A}$. 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:

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

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.

Upon updating the edge weights in $\mathcal{G}'_{\text{sub}}$ by clusters, we deploy the trained $\mathcal{G}'_{\text{sub}}$ to predict links for attribute A_j on a cluster basis. This implies that if a node $t_i \in \mathcal{T}_{\text{label}}$ exists within cluster C_i, the triplet (h_i, A_j, t) should be universally applicable to all central nodes $h_k \in C_i$. In the absence of such a node within cluster C_i, the weighted majority value of attribute A_j is designated to all central nodes $h_k \in C_i$.

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

7 EXPERIMENTAL STUDY

Using standard datasets, we empirically evaluated our method GEIL 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 [74] and one real-life large dataset in Table 3. Hospital [97] and Flights [69] offer rich contextual information with

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

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

a high degree of data redundancy. Notably, Hospital has scarce and randomly imposed noises, while Flights exhibit a very high error rate. Tax is a large synthetic dataset from the BART repository [6], featuring various data error types, thus resulting in a vast search space to identify true corrections. Beers [74] and Rayyan [84] 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 [74, 86], 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. We introduce four types of noise, including missing value (MV), typo (T), formatting issue (FI), and violated attribute dependency (VAD).

Baselines. We implemented GEIL in Python and used the following baselines. (1) Raha [76], an error detection method involving feature engineering, interactive labeling and ML models for detection; (2) Rotom [78], a meta-learning data augmentation framework that formulates tabular error detection as a seq2seq task; (3) Roberta$_{\text{det}}$ [71], a binary classifier adapted for error detection utilizing the pre-trained language model Roberta [71]; (4) Baran [74], a hybrid error correction approach based on feature engineering and traditional ML models; (5) Garf [86], 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 both error detection and correction, utilizing a 13B LLM to solve multiple data pre-processing 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 [75]. For the remaining error correction methods, including HoloClean and T5, we utilized our error detection method GEIL$_{\text{det}}$ 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 [76] and [74]. 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 [58] as the backbone model for $\mathcal{M}_G$ by default. We adopt gpt-4-turbo as the online reference LLM to generate function set $\mathcal{F}$ as default setting, denoted as GEIL; and apply Mistral-7B as offline reference LLM, denoted as GEIL$_{\text{offline}}$. 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 thrice, averaging the results reported here.

Experimental results. We next report our findings.

Exp-1: Effectiveness. We evaluated GEIL 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. GEIL 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 GEIL is effective, and $\mathcal{M}_{\text{det}}$ and $\mathcal{F}$ could help with each other to boost the overall performance. Additionally, GEIL enhances model interpretability, and effectively mitigates overfitting risk through its generation functions, thereby improving the robustness and reliability. However, GEIL$_{\text{offline}}$ has a slight performance decrease compared to GEIL, 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, GEIL$_{\text{offline}}$ 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 GEIL significantly benefits both $\mathcal{F}^{\text{det}}$ and $\mathcal{M}_{\text{det}}$. Compared to Raha, GEIL is 19%, 9%, 18% and 56% higher F-measure in Hospital, Flights, Tax and IMDB, respectively. Rotom primarily generates random augmentations, limiting its ability to interpret dirty and clean data patterns, leading Rotom to inferior performance in scenarios with complex textual inconsistencies. Roberta$_{\text{det}}$ performs similarly to Rotom in most datasets. However, due to a small labelling budget, 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 2.3.

Error correction. Table 5 shows the F1-score of error correction, and GEIL outperforms all baselines with 20.5% higher F1-score on average, compared to the best of others. This verifies the effectiveness of unifying $\mathcal{M}_G$, $\mathcal{F}^{\text{corr}}$ and graph structure learning. GEIL$_{\text{offline}}$ exhibits a slight performance decline compared to GEIL. 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, GEIL$_{\text{offline}}$ 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.*, $\mathcal{M}_G$ in GEIL. Garf employs a SeqGAN model

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
GEIL	**1.00**	0.95	**0.98**	**0.99**	**0.97**	**0.98**	0.99	0.99	0.99	0.84	**0.98**	**0.90**	0.99	**0.98**	**0.98**	0.88	0.89	**0.88**
$\text{GEIL}_{\text{offline}}$	0.96	0.90	0.93	0.99	0.97	0.98	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
GEIL	0.97	**0.96**	**0.97**	**0.96**	**0.96**	**0.96**	**0.97**	**0.97**	**0.97**	**0.80**	**0.93**	**0.86**	**0.95**	**0.94**	**0.95**	**0.87**	**0.89**	**0.87**
$\text{GEIL}_{\text{offline}}$	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
GEIL_{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
GEIL_{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)

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
GEIL(Training)	3825	3645	2265	4655	6455	7089
GEIL(Inference)	1225	1335	1035	1755	1355	2876

Table 7: Ablation study

System	Hospital			Tax			Rayyan		
	P	R	F	P	R	F	P	R	F
GEIL	**0.97**	**0.96**	**0.97**	**0.95**	**0.94**	**0.95**	**0.80**	**0.93**	**0.86**
$\text{GEIL}_{\text{offline}}$	0.94	0.90	0.92	0.89	0.89	0.89	0.78	0.85	0.81
GEIL w/o Graph	0.92	0.91	0.91	0.88	0.68	0.77	0.78	0.91	0.84
GEIL w/o Creator	0.83	0.73	0.78	0.74	0.43	0.55	0.50	0.49	0.49
GEIL w/o Critic	0.98	0.83	0.90	0.86	0.61	0.72	0.61	0.13	0.22

for unsupervised generation of data repair rules, subsequently refined through a co-training framework. The generated data quality rules might not be capable of handling unseen data. For instance, in Hospital, Garf cannot generate the correct value Birmingham from the dirty cell Bxrmxngham if Birmingham is absent in the dataset. The limitations of T5 are apparent in its inability to employ the retrieval-augmented generation paradigm and to repair inconsistency errors, leading to a noticeable decline in its performance.

The LLM baseline JellyFish demonstrates the substantial potential of generative models in data cleaning tasks. However, its performance is notably inferior to that of $\text{GEIL}_{\text{offline}}$. 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.

Exp-2: Efficiency. Table 6 reports the running time of all baselines, encompassing 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, GEIL could be trained within approximately 120 minutes for most datasets, *e.g.*, 55 minutes in Flights in consumer-level GPUs.

One notable aspect of GEIL is its utilization of function set $\mathcal{F}^{\text{corr}}$ and generative model $\mathcal{M}_G$ 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, GEIL 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 $\mathcal{M}_G$ only on identified dirty cells. Also, the size of training data $\mathcal{D}_{\text{correct}}$ remains approximately the same in all datasets, such that GEIL is insensitive with $|\mathcal{T}|$ in training time.

Furthermore, the inference speed of LLMs is always a bottleneck [16, 24, 107] because of the huge size of parameters, To solve it in the data cleaning task, GEIL incorporates the vLLM technique [63] that utilizes PagedAttention to group similar queries with a KV cache, significantly speeding up inference time.

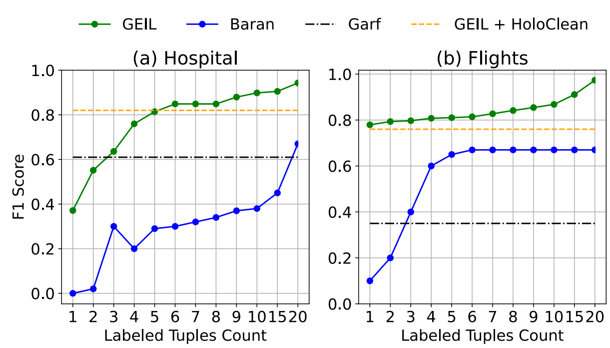


Figure 6: Performance evaluation with respect to the number of labelling budgets

Exp-3: Ablation study. We show how labeling budges and error ratios impact the performance of GEIL and other baselines.

Labelling Budget. Figure 6 shows the performance of baselines in error correction by varying the number of labelled tuples. GEIL shows a more rapid convergence in the F1 score than others when a

Table 8: Error Correction Performance comparison by varying error rates

Dataset	Error-Rate	GEIL			GARF			Raha + Baran			GEIL + 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

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 $\mathcal{M}_G$ to learn well.

Component analysis In Table 7, we further analyze the impact of removing different components for GEIL. The removal of the graph components results in the inability of the model to handle VAD errors, also prevents correction model $\mathcal{M}_G$ 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 GEIL framework.

Robustness analysis. To evaluate the robustness of GEIL, 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 8. As error rates increase, GEIL generally demonstrates greater robustness, achieving an F-measure of 0.85 in the Hospital dataset at a 50% error rate. 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.

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 [4, 10], DCs [20, 45, 48, 97], CFDs [12, 34, 35], PFDs [90], REEs [42, 44], user-defined rules [33] and manually-defined parameters [7, 89]. Considering the difficulty of handcrafting data quality rules, several rule discovery algorithms [11, 19, 36, 38, 39, 73, 85] 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 [52, 76, 78, 82, 87, 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 [56], ActiveClean [62], meta-data [110] and rule generation [86] aim to learn abnormal data in an unsupervised learning way.

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, DCs, PFDs and REEs are mainly used to correct errors using various repairing mechanism, *e.g.,* heuristic fixes [5, 8, 9, 22, 23, 29, 46, 47, 49, 51, 95, 103, 111, 117], certain fixes [40–43, 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 DCs and MDs as features and designs a factor graph for error correction. There are also generative models [25, 31, 55, 64, 79, 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 [86] and GAIN [118]. Data cleaning are also employed to improve downstream ML models, *e.g.,* [27, 67, 72]. *(3) Hybrid methods.* which unify logical rules and machine learning models for data cleaning,*e.g.,*Baran [74] 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 [53, 60, 61, 68, 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 [41], knowledge bases [21] and web table [2].

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 [59], *e.g.,* the 175B-parameter GPT-3 [14] and the 540B-parameter PaLM [18]. Compared with PLMs, LLMs show the emergent abilities [114], in-context learning [30, 80], instruction following [83, 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[66, 81, 120] and data imputation[13, 17, 66, 81, 120], however no exist work integrates LLM for an end-to-end DC system.

9 CONCLUSION

Data cleaning is a key component in multiple applications. GEIL provided 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. The experimental results show that GEIL outperforms the existing data cleaning approaches by at least 10% F-measure on average, verifying that the proposed framework is effective in various scenarios.

REFERENCES

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

[2] 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.

[3] 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.

[4] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Vol. 8. Addison-Wesley Reading.

[5] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. 68–79.

[6] 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.

[7] 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.

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

[9] 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

[10] 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.

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

[12] 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

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

[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

[16] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023. LongLoRA: Efficient Fine-tuning of Long-Context Large Language Models. *arXiv preprint arXiv:2309.12307* (2023).

[17] Zui Chen, Lei Cao, Sam Madden, Tim Kraska, Zeyuan Shang, Ju Fan, Nan Tang, Zihui Gu, Chunwei Liu, and Michael Cafarella. 2023. Seed: Domain-specific data curation with large language models. *arXiv e-prints* (2023), arXiv–2310.

[18] 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).

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

[20] 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.

[21] 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.

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

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

[24] 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.

[25] 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

[26] 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.

[27] Daniel Deutch, Nave Frost, Amir Gilad, and Oren Sheffer. 2021. Explanations for Data Repair Through Shapley Values. In *CIKM*. 362–371. https://doi.org/10.1145/3459637.3482341

[28] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

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

[30] 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).

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

[32] 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).

[33] 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.

[34] 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

[35] 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

[36] 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

[37] 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.

[38] 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

[39] 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.

[40] 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.

[41] 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.

[42] 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.

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

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

[45] 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.

[46] 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.

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

[48] 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.

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

[50] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*. PMLR, 1263–1272.

[51] 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.

[52] 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.

[53] 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.

[54] 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).
[55] 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
[56] 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.
[57] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *Comput. Surveys* 55, 12 (2023), 1–38.
[58] 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).
[59] 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).
[60] 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).
[61] 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).
[62] 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.
[63] 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).
[64] 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
[65] 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.
[66] 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).
[67] 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.
[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 *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 13–24.
[69] 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).
[70] 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).
[71] 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).
[72] 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.
[73] Ester Livshits, Alireza Heidari, Ihab F Ilyas, and Benny Kimelfeld. 2020. Approximate denial constraints. *arXiv preprint arXiv:2005.08540* (2020).
[74] 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.
[75] 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.
[76] 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.
[77] Heikki Mannila and Kari-Jouko Räihä. 1994. Algorithms for inferring functional dependencies from relations. *Data & Knowledge Engineering* 12, 1 (1994), 83–99.
[78] 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.
[79] 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.
[80] Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2021. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943* (2021).
[81] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911* (2022).
[82] 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.
[83] 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.
[84] 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.
[85] 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.
[86] 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.
[87] 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.
[88] 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.
[89] Clement Pit-Claudel, Zelda Mariet, Rachael Harding, and Sam Madden. 2016. Outlier detection in heterogeneous datasets using automatic tuple expansion. (2016).
[90] Abdulhakim Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern functional dependencies for data cleaning. *PVLDB* 13, 5 (2020), 684–697.
[91] 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).
[92] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.
[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] Murray Shanahan. 2024. Talking about large language models. *Commun. ACM* 67, 2 (2024), 68–79.
[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] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[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] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223* (2023).

[123] 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*.

A MODEL ARCHITECTURE OVERVIEW

In Figure 8, we present the model architecture overview of the proposed GEIL in detail.

B PROMPTS

As the prompts in GEIL 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 2.3.

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 2.3..

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 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:

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

B.4 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 ˆ[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:

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

B.5 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 ˆ\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:

```
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.6 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 9/10/11.

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 GEIL 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 [70, 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 12 by using LLMs of different parameter sizes from 560M(Million) to 7B(Billion), including BLOOM-560M [101], ChatGLM-6B [32], Mistral-7B[58]. As the parameter size increases, there is a notable improvement in precision, recall, and F1 score of GEIL 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 GEIL 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 GEIL 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.

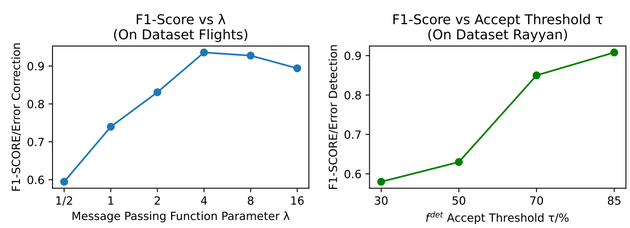


Figure 7: Performance evaluation with respect to the hyper-parameter λ on error correction for Flights**, and τ for f^{det} on error detection for** Rayyan

Hyper-parameter analysis. In Figure 7, 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

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 9: 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 10: 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 11: 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 12: Parameter Size Impact for $\mathcal{M}_G$

performance. However if τ is too high, no valid functions will be accepted within the given conversation turns budget n'. Thus we set $\tau = 85\%$ by default.

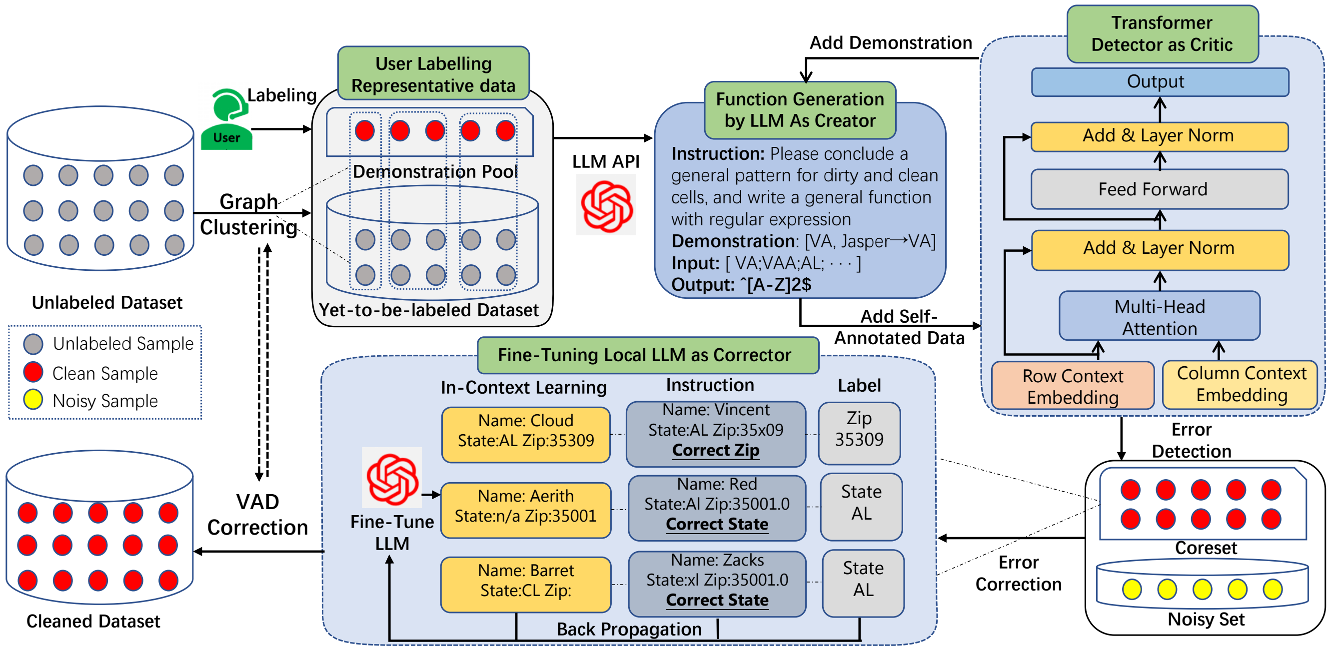


Figure 8: Overview of model architecture for GEIL