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


Mengyi Yan[1] Yaoshu Wang*[2] Yue Wang[2] Xiaoye Miao[3] Jianxin Li[1]

[1]Beihang University [2]Shenzhen Institute of Computing Sciences [3]Zhejiang University


## A. Introduction

- Data cleaning is a critical but labor-intensive task.
- Challenges of Rule-Based Methods:
  - Require **significant domain knowledge** to define rules/constraints.
  - High barrier to entry for non-expert users.
- Challenges of configuration free/ML methods:
  - Lack of **interpretability**.
  - Require extensive labeled data for training.
- Challenges of directly applying LLM for data cleaning:
  - **Understanding Dependencies:** Token limitations prevent LLMs from grasping the full context of relational tables.
  - **Hallucination**: LLMs may generate plausible but incorrect data repairs when context is irrelevant.
  - **Efficiency**: The size and complexity of LLMs make sequential processing of all tuples impractical.

## B. Problem Definition

- Given a dirty relational table $T$ and a limited labeling budget $\theta$, where users are only able to label at most $\theta$ tuples, our objective is to cleanse the table $T$ aiming to identify and rectify all errors, among with interpretable rule set $\mathcal{F}$ and dependency FDs.
- **Input**: A dirty relational table $T$, a limited labelling budget $\theta$: user need to annotate at most $\theta$ tuples.
- **Output**: A cleaned relational table $T_{clean}$, and interpretable patterns include:
  (1) Error Detection Pattern $\mathcal{F}^{det}$, generated by LLM.
  (2) Error Correction Function $\mathcal{F}^{corr}$, generated by LLM.
  (3) Functional Dependencies, retrieved with graph structural learning.

*Sorry for the misleading result. Based on the given examples, the pattern of clean cells seems to contain abbreviations of month and 2 digits. Therefore, a simple regular expression to check if a given cell is dirty or not can be:*

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

    # Dirty pattern
    dirty_pattern1 = re.compile(r'^\d{2}-\d$')

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

Fig.1 LLM-Generated Error Detection pattern $\mathcal{F}^{det}$. This case comes from benchmark dataset **Rayyan** column **article-pagination**.
- Error: 4-digits pages + 1/2-digits month(e.g. 1972-4)
- Correct: 3-letter month + 2-digits page (e.g. Apr-72)

```
def Rayyan_Correct_pagination(cell):
    clean_pattern1 = re.compile(r'^(Jan|Feb|Mar|Apr|May|Jun|
        Jul|Aug|Sep|Oct|Nov|Dec)-\d{2}$')
    clean_pattern2 = re.compile(r'^(\d{2})-(Jan|Feb|Mar|Apr|
        May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)$')

    # Dirty patterns
    dirty_pattern1 = re.compile(r'^(\d{2})-(\d{1,2})$')
    dirty_pattern2 = re.compile(r'^(\d{4})-(\d{1,2})$') #
        XXXX-MM format

    # Check clean patterns first
    if clean_pattern1.match(cell) or clean_pattern2.match(
        cell):
        return cell

    # Handle XXXX-MM format
    match = dirty_pattern2.match(cell)
    if match:
        yy = match.group(1)[-2:] # Always take last 2 digits
        month_num = match.group(2)
        month_map = {
            '1': 'Jan', '2': 'Feb', '3': 'Mar', '4': 'Apr',
            '5': 'May', '6': 'Jun', '7': 'Jul', '8': 'Aug',
            '9': 'Sep', '10': 'Oct', '11': 'Nov', '12': 'Dec'
        }
        if month_num in month_map:
            return f"{month_map[month_num]}-{yy}"

    match = dirty_pattern1.match(cell)
    if match:
        return f"{match.group(2)}-{match.group(1)}"

    return cell
```

Fig.2 LLM-Generated Error Correction pattern $\mathcal{F}^{corr}$.

## C: Contributions

### 1. An End-to-End LLM-based Data Cleaning Framework:

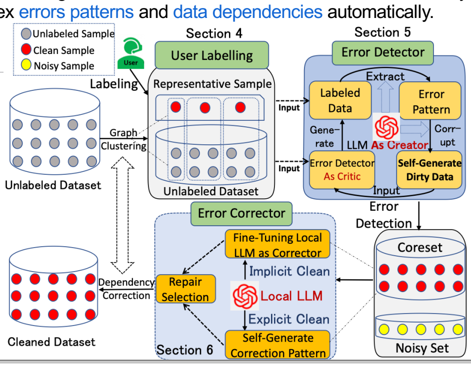
- We introduce GIDCL, a systematic framework that integrates LLMs for a complete data cleaning workflow, from user labeling to error detection and correction, fully utilize the LLM's capability of in-context learning, code generation and generative cleaning ability with high precision.

### 2. An Iterative workflow via knowledge distillation:

- We design an innovative creator-critic workflow where an LLM (creator) generates interpretable detection rules, and distills a transformer-based error detection model, achieving high accuracy with only a few labeled samples.

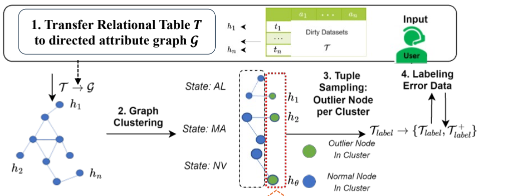
### 3. Graph-Enhanced LLM-based Correction:

- We propose a graph-enhanced, retrieval-augmented method for fine-tuning local LLMs to generate reliable and efficient corrections, effectively handling complex errors patterns and data dependencies automatically.




## D. Graph Structural Learning

- **Process**: Converts the input table $T$ into a directed attribute graph $\mathcal{G}$.
- **Representation Learning**: Uses GNN model to learn tuple embeddings, capturing structural similarities.
- **Tuple Selection**: Employs k-means clustering and an outlier selection strategy to identify a small set of representative and ambiguous tuples for user labeling, maximizing information gain.

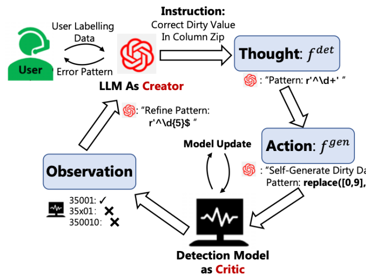



Objective Function:

$$\underbrace{\max_{S\subseteq T, |S|=\theta_{\text{label}}} \Big(\sum_{i=1}^{k}\sum_{t\in S\cap C_i}\sum_{p\in C_i}\text{dist}(t,p)\Big)}_{\text{Maximize outlier sample distance with intra-cluster members}} \text{ s.t. } \underbrace{\min_{C}\Big(\sum_{l=1}^{k}\sum_{t_i,t_j\in C_l}\text{dist}(t_i,t_j)\Big)}_{\text{Minimization average distance within clusters}}, \underbrace{|S\cap C_i| \le \lceil \theta_{\text{label}}/k \rceil}_{\text{Sample } \lceil \frac{\theta}{k} \rceil \text{ outlier Per cluster}}$$

## E. Creator-Critic Workflow for Error Detection

- **Creator (LLM)**: Prompted with few-shot examples, the LLM generates interpretable error detection functions $\mathcal{F}^{det}$ and data corruption functions $\mathcal{F}^{gen}$ for data augmentation. (As Fig.1)
- **Critic (PLM)**: A smaller, fine-tuned PLM $\mathcal{M}_{det}$ acts as a fast and efficient classifier to identify erroneous cells, feeding predictions back to the creator to refine the rules and divide coreset of clean data.




## F. Graph-Enhanced Error Correction

- **Implicit Correction**: A local LLM $\mathcal{M}_{corr}$ is fine-tuned, providing with graph clustering-based RAG to generate high-quality corrections.
- **Explicit Correction**: The LLM also generates interpretable correction functions $\mathcal{F}^{corr}$ for simpler, pattern-based errors. (As Fig.2)
- **Repair Selection**: The critic $\mathcal{M}_{det}$ is used as a ranker to select the best repair from the implicit and explicit methods, avoiding hallucination.
- **Dependency Correction**: The framework re-learns the graph structure on the cleaned data to discover FDs to resolve remained inconsistencies.

## G. Experiments

- Real-life Datasets. Hospital, Flights, Beers, Rayyan, Tax, IMDB.
- Baselines. Raha/Baran/Garf /HoloClean/Rotom/JellyFish(LLM-Based method).
- Measurements. F1-Score on end-to-end data cleaning (Including error detection and correction.)

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

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

- Effectiveness:
  - Rule Generation: Offline 7B-LLM can generate >80% detection rules, and >70% correction rules automatically, on average of 9.7 queries per attribute.
  - Correction Generation: Graph clustering-based RAG can constraint LLM from generating hallucinations, even the labeled tuples are as few as 20.
- Efficiency: By leveraging function-based cleaning, GIDCL's runtime does not increase linearly with dataset size, with high label efficiency.
- Robustness: GIDCL demonstrates strong robustness, maintaining a high F1-score even when the data error rate is increased to 50%.