UniClean: A Scalable Data Cleaning Solution for Mixed Errors based on Unified Cleaners and Optimized Cleaning Workflow

Xiaoou Ding
Harbin Institute of Technology
dingxiaoou@hit.edu.cn

Zekai Qian
Harbin Institute of Technology
qzk010728@gmail.com

Hongzhi Wang*
Harbin Institute of Technology
wangzh@hit.edu.cn

Siying Chen
Harbin Institute of Technology
23S003001@stu.hit.edu.cn

Yafeng Tang
Harbin Institute of Technology
tangyf@stu.hit.edu.cn

Hongbin Su
Harbin Institute of Technology
24S103394@stu.hit.edu.cn

Huan Hu
Huawei Cloud Computing
Technologies Co., Ltd., China
huhuan18@huawei.com

Chen Wang
Tsinghua University
wang_chen@tsinghua.edu.cn

ABSTRACT

Data cleaning is an essential technique to enhance data quality. Despite the proposal of various algorithms with different cleaning strategies, current automated cleaning technologies still fall short of practical requirements when dealing with large-scale data containing mixed errors. This paper presents **UniClean** to efficiently solve the mixed error cleaning problem with three key technical contributions. (1) A unified construction and extension method for cleaners, enabling cleaning methods to easily utilize various cleaners to perform cleaning tasks. (2) Three optimization strategies to achieve efficiency-oriented cleaning preparation. (3) A cleaning algorithm based on an optimized cleaning process to effectively clean mixed errors. **UniClean** achieves a time complexity of $O(|D_{\text{error}}|^4 \cdot |Op| + |D| \cdot |D_{\text{error}}|)$, significantly enhancing scalability. Experiments on public and large-scale enterprise datasets demonstrate that **UniClean** achieves over 40% improvement across five metrics, compared to five state-of-the-art cleaning methods, and delivers more than 30% gains in *F1* and *REDR* on complex datasets, while completing the cleaning process within hours even for millions of records.

PVLDB Reference Format:
Xiaoou Ding, Zekai Qian, Hongzhi Wang, Siying Chen, Yafeng Tang, Hongbin Su, Huan Hu, and Chen Wang. UniClean: A Scalable Data Cleaning Solution for Mixed Errors based on Unified Cleaners and Optimized Cleaning Workflow. PVLDB, 18(11): 4117 - 4130, 2025.
doi:10.14778/3749646.3749681

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/qzkinhit/UniClean-bench-Result.

1 INTRODUCTION

In recent years, data science has driven profound transformations in scientific exploration and societal progress, fostering a growing recognition that high-quality datasets are fundamental to the effectiveness of data analysis and applications [1, 9, 37, 45]. High-quality data ensures the accuracy and reliability of information, serving as an indispensable prerequisite for trustworthy decision-making[24]. However, the rapid accumulation of data have given rise to frequent and increasingly complex data quality issues, which not only elevate the human labor costs involved in data preprocessing but also undermine the precision of data analysis outcomes [23, 30, 49, 51].

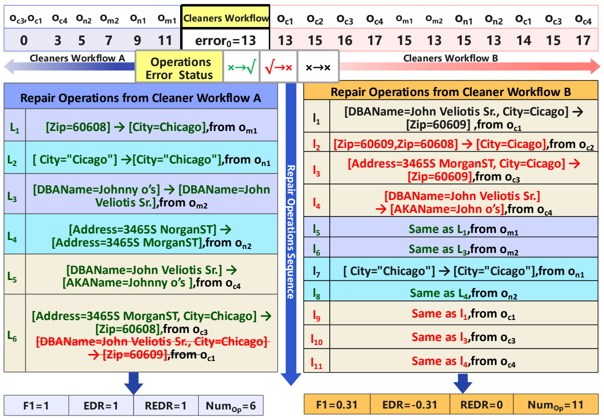
Data cleaning is crucial for addressing data quality issues, involving detecting and repairing errors [6, 18, 19]. Various algorithms have been proposed, rule-based [7, 11, 25, 35, 48], data distribution-based [20, 28, 41, 53], knowledge bases [10, 27, 40], and human-in-the-loop (including crowdsourcing and LLM) [52, 54] approaches. Researchers are exploring the integration of *multiple cleaning signals* [47] for better solutions, including user-defined rules, expert knowledge, and dataset characteristics, to devise superior cleaning solutions. However, current techniques still fall short in practical applications, especially with multiple error types (e.g., missing values, outliers, rule violations). While error detection achieves high accuracy and recall (exceeding 0.9 and 0.8) according to recent experiments [2, 32, 42, 43, 54], automated repair remains unsatisfactory and unstable, demanding improvements in existing technologies:

(1) Multiple cleaning methods (refer to *cleaners* in Definition 3.1) are essential for handling *heterogeneous error* types in dynamic and large-scale datasets. However, existing techniques typically target a single error type per scenario [38], and lack unified mechanisms for configuring and coordinating multiple cleaners. As a result, users often face repeated configuration and manual intervention, which limits cleaning efficiency and scalability. **(2) Downstream users** expect both improved *data quality* and *interpretability* in data cleaning, requiring transparent explanations of error repair mechanisms and reuse of proven historical strategies. However, existing frameworks prioritize cell-level fixes while neglecting global cleaner orchestration and workflow optimization, limiting adaptability and interpretability in dynamic data scenarios. Thus, ensuring a reasonable repair operation sequence and coordinating multiple cleaning methods are vital for effective cleaning on mixed errors. We illustrate this issue with Example 1.1.

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749681
*Corresponding author.

Table 1: Multiple errors in a restaurant dataset.

	DBAName	AKAName	City	Zip	Address
t_1:	John Veliotis Sr.	Johnny o's	Cicago	–	3465S MorganST
t_2:	John Veliotis Sr.	John o's	Chicago	60608	3465S NorganST
t_3:	Johnny o's	Johnny o's	Cicago	60608	3465S MorganST
t_4:	Johnny o's	Johnny o's	Chicago	60609	3465S MorganST
t_5:	John Veliotis Sr.	John o's	Cicago	60609	3465S MorganST
t_6:	El Cafetal Del Tio Corp	MI CAFETRAL	Cicago	60608	3465S NorganST



o_{c3},o_{c1} o_{c4} o_{n2} o_{m2} o_{n1} o_{m1}
0 3 5 7 9 11
Cleaners Workflow
error$_0$=13
o_{c1} o_{c2} o_{c3} o_{c4} o_{m1} o_{m2} o_{n1} o_{n2} o_{c1} o_{c3} o_{c4}
13 15 16 17 15 13 15 13 14 15 17
Cleaners Workflow A
Operations Error Status
×→√
√→×
×→×
Cleaners Workflow B
Repair Operations from Cleaner Workflow A
L_1 [Zip=60608] → [City=Chicago],from o_{m1}
L_2 [City="Cicago"] →[City="Chicago"],from o_{n1}
L_3 [DBAName=Johnny o's] → [DBAName=John Veliotis Sr.],from o_{m2}
L_4 [Address=3465S NorganST] → [Address=3465S MorganST],from o_{n2}
L_5 [DBAName=John Veliotis Sr.] → [AKAName=Johnny o's],from o_{c4}
L_6 [Address=3465S MorganST, City=Chicago] → [Zip=60608],from o_{c3}
~~[DBAName=John Veliotis Sr., City=Chicago] → [Zip=60609],from o_{c1}~~
Repair Operations Sequence
Repair Operations from Cleaner Workflow B
l_1 [DBAName=John Veliotis Sr., City=Cicago] → [Zip=60609] ,from o_{c1}
l_2 [Zip=60609,Zip=60608] → [City=Cicago],from o_{c2}
l_3 [Address=3465S MorganST, City=Cicago] → [Zip=60609],from o_{c3}
l_4 [DBAName=John Veliotis Sr.] → [AKAName=John o's],from o_{c4}
l_5 Same as L_1,from o_{m1}
l_6 Same as L_3,from o_{m2}
l_7 [City="Chicago"] → [City="Cicago"],from o_{n1}
l_8 Same as L_4,from o_{n2}
l_9 Same as l_1,from o_{c1}
l_{10} Same as l_3,from o_{c3}
l_{11} Same as l_4,from o_{c4}
F1=1 EDR=1 REDR=1 Num$_{Op}$=6
F1=0.31 EDR=-0.31 REDR=0 Num$_{Op}$=11

(a)Cleaner workflow with UniClean (b)Cleaner Workflow with greedy approach

Figure 1: Comparison of cleaning effectiveness

Example 1.1. *There are various data quality issues in Table 1, including incorrect values (t_4*`[DBAName]`*), missing values (t_1*`[Zip]`*), rule violations (t_4*`[City]`* and t_3*`[AKAName]`*), and typos (t_4*`[Address]`*). These mixed errors are addressed using three types of cleaners, illustrated in Figure 1: ① **Rule-based cleaners:** Cleaner o_{c1} identifies a violation between t_1 and t_5 and fills the missing* `Zip` *in t_1 with "60609", ② **Distribution-based cleaners:** Cleaner o_{n1} marks t_2 and t_4 as outliers for the* `City`*, replacing them with "Cicago" (a typo error). ③ **External knowledge-enhanced cleaners:** With external dictionaries or annotations, cleaner o_{m2} corrects inconsistent values like "Johnny o's" in t_3 and t_4, replacing them with "John Veliotis Sr".*

However, we find that globally, relying on a single cleaner (e.g., o_{c1} or o_{n1}) proves insufficient for mixed-error cleaning due to limited coverage (low recall*) and the risk of introducing new errors* (low accuracy*). For instance, the rule-based cleaner o_{c1} may detect rule-based inconsistencies (e.g., zip-code mismatches between t_1 and t_5) but fails to repair spelling mistakes or outlier distributions. Conversely, the distribution-based cleaner o_{n1} can address abnormal numeric values but is unable to handle rule conflicts or missing references.*

To handle mixed-error types while minimizing error propagation, we compare a greedy approach (iteratively applying cleaners based on detected errors) with an *optimal cleaner execution sequencing method* for cleaning (i.e., **UniClean** proposed in this paper in Fig. 1). The greedy method's local repair of `AKAName` (l_4) inadvertently introduces new `DBAName` inconsistencies, necessitating a suboptimal follow-up fix (l_{11}). In contrast, **UniClean** first resolves `DBAName` via o_{m2}, then applies o_{c4} to `AKAName`, reducing total operations and propagated errors. By prioritizing critical attributes (e.g., `City`) early, **UniClean** enables subsequent cleaners (e.g., o_{c3}) to generate higher-quality repairs without interference from errors.

Furthermore, when different cleaners propose conflicting repairs for the same attribute, choosing the repair with lower cost and higher global benefit is essential. In Figure 1(b), both o_{c3} and o_{c1} can repair `Zip`, but o_{c1} requires modifying more cells, resulting in a higher repair cost. **UniClean** selects o_{c3}'s repair, thus reducing downstream inconsistencies and overall repair effort. In contrast, the greedy method's local fix (e.g., l_2) satisfies o_{c2} but raises new errors for o_{c1}, o_{c3}, and o_{n1}, demonstrating that global sequencing and strategic selection are crucial for handling mixed-error data.

Challenges. It is evident that orchestrating effective repair operation sequence for multi-error data cleaning is crucial for maximizing cleaner efficacy [21]. However, achieving this is not straightforward. Technical challenges include: **(1) High complexity of repair operations sequence combinations**. Mixed-error datasets demand collaborative cleaners, leading to a combinatorial explosion in repair planning. Each error instance (cell) tie to a candidate repair operation, but global cleaning quality depends heavily on the execution order of these rules due to interdependencies (e.g., Example 1.1). This necessitates exploring $O(|D_{\text{error}}|!)$ repair operations sequences over $|D_{\text{error}}|$ errors. Each step may invoke one of $|Op|$ candidate cleaners, typically with $O(|D|^2)$ time complexity [43], leading to an overall cost of $O(|D_{\text{error}}|! \cdot |Op| \cdot |D|^2)$. The problem thus constitutes a repair scheduling challenge rather than a direct error-type-to-cleaner mapping. **(2) High maintenance costs for diverse cleaners**. Despite advances in cleaner automation, many still require manual parameter tuning, particularly for multi-error scenarios. Integrating new cleaners demands labor-intensive framework adjustments, complicating system maintenance. This complexity hinders the global evaluation of cleaning effectiveness and the development of optimal repair strategies. **(3) High adaptability costs in dynamic big data**. Rapid data changes make it difficult to reuse historical cell-level repair strategies due to shifting error locations. This often requires re-executing the cleaning framework, increasing costs.

Motivated by this, we propose the **UniClean** solution to address the key issue in this paper: how to automatically generate the optimal cleaning operations in a mixed-error data environment at a low time cost. Our **contributions** are as follows.

- **Unified construction and extensibility of cleaners**. We design **UniClean** to unify a comprehensive set of row-based, column-based, and knowledge-enhanced cleaners (currently 73 in our library, with over 8K lines of codes), which accommodating external cleaners without altering the overall framework structure. We theoretically prove its *completeness* (**CCC Theorem 4.1**), ensuring that all potential cleaning scenarios are covered.
- **Efficiency-oriented cleaning preparation.** We develop three efficiency-optimized strategies for large-scale mixed-error cleaning: (1) *Dependency-based cleaner classification* with topological ordering to minimize repair conflicts; (2) a *quality-centric coreset extraction* method to reduce computational load via error-distribution sampling; (3) *data partitioning* to separate repair operations into independent blocks according to operation scopes. The strategies jointly reduce the overall complexity from $O\big((|D_{\text{error}}|!) \cdot |Op| \cdot |D|^2\big)$ to $O\big(|D_{\text{error}}|^4 \cdot |Op| + |D| \cdot |D_{\text{error}}|\big)$, while maintaining cleaning effectiveness, guaranteed by the *Independence and Consistency* properties (**ICC Theorem 4.2**).
- **Iterative optimized cleaning workflow.** We propose a multi-granularity cleaning workflow mechanism (ICW Algorithm 4) that combines *provenance analysis* and *dynamic weight updating*. At a fine-grained level, ICW incrementally selects optimal record-level

repair by optimizing a global objective function Q derived from cleaners. At a coarse-grained level, it builds a reusable workflow for low-overhead migration to multi-version or dynamic data scenarios. **CQD Theorem 4.3** confirms that ICW progressively improves data quality under iterative updates.

- We evaluate **UniClean** on seven real-world datasets using two novel metrics: Record Error Reduction Rate (*REDR*), a record-level metric designed to reflect the overall usability of data records after cleaning, and Cleaning Time cost per 100 Records (*CTR*). Compared to five SOTA methods, **UniClean** achieves: >40% improvement in cleaning effectiveness across most cases, <10% performance degradation on large-scale vs. smaller native data, and >30% gains in *F1*/*REDR* on complex datasets while halving processing time for 10K+ records. Results validate its superior *effectiveness* and *scalability* for mixed-error data cleaning.

Organization. Section 2 presents related work. Section 3 outlines the studied cleaning problem. Section 4 introduces the steps of **UniClean** and provides theoretical analysis. Section 5 reports the experimental results, and Section 6 draws the conclusion.

2 RELATED WORK

To achieve intelligent data cleaning, various methods have been proposed, which can be categorized into three types. ❶ **Rule-based Cleaners** rely on logical or semantic constraints between attributes to detect and repair data errors. Common attribute constraints include *denial constraints (DC)* [29] and *functional dependencies (FD)* [3, 5], as well as their relaxed forms [14, 15]. Frameworks such as **Holistic** [8], **BigDansing** [35], and **Horizon** [48] improve cleaning efficiency and accuracy using techniques like conflict hypergraphs, rule engines, and functional dependency graphs. While effective for large-scale data cleaning, these methods are limited by the completeness and accuracy of the rules, especially in complex or ill-defined scenarios [43]. ❷ **Distribution-based Cleaners** detect and repair errors by analyzing the contextual distribution or statistical characteristics of rows. They can rely on explicit statistical rules [26] or machine learning models [22, 39]. **Baran** [41] optimizes cleaning using contextual information. Machine learning-based methods, like **Scared** [53], handle complex distribution anomalies but often require high model complexity and labeled data [38]. ❸ **External Knowledge-enhanced Cleaners** enhance traditional rule-based or distribution-based cleaners by incorporating external knowledge, such as ground-truth labels, domain dictionaries, or semantic embeddings (e.g., Wikipedia or expert annotations). These cleaners improve detection precision by enabling additional constraint discovery and prune low-quality repair candidates using label- or distribution-informed priors. For instance, **CoCo** [27] extends rule-based cleaning by supporting interactive acquisition of consistency constraints [10], while **Baran** [41] augments distribution-based methods with annotated labels to enhance repair accuracy. Despite their effectiveness, such approaches often rely on substantial human supervision, increasing development and integration overhead.

Advancements also focuses on integrating multiple cleaning strategies to enhance repair effectiveness. Broadly, they can be categorized as: ❹ **Fixed-cleaner Integration Cleaners** automate repair by leveraging predefined cleaner sets. For example, **HoloClean** [47] employs probabilistic graphical models to integrate repairs from outlier detection, constraint violation resolution, and missing value imputation. While effective in static, well-defined scenarios, these methods lack adaptability and flexibility for complex data challenges [10]. ❺ **User-logic Extension Cleaners** provide flexibility by supporting custom repair logic. **NADEEF** [10] combines multi-signal inputs with user-defined rules, enabling cleaning workflows for diverse tasks. However, such methods heavily rely on manual effort, reducing automation and increasing implementation complexity [47].

We survey several advanced cleaning systems in Table 3 in Section 5. Despite advancements, existing methods still lack automated, flexible framework for robust mixed-error handling with minimal preconfiguration. Our proposed **UniClean** addresses this gap by offering a scalable solution for data cleaning.

3 OVERVIEW OF UNICLEAN

3.1 Preliminaries

$Attr(\mathcal{R}) = (A_1, ..., A_m)$ denotes the set of attributes in a database relation $\mathcal{R}$. D is an instance of $\mathcal{R}$ containing n tuples, each in the domain $Dom(A_1)\times\cdots\times Dom(A_m)$. $Dom(A)$ represents the domain of attribute A. A cell $t[A]$ denotes the data value of tuple t on A.

Definition 3.1. *A **cleaner** is an instance of a data cleaning method. Formally, a cleaner o defined on* $\mathcal{R}$ *is a mapping* $[\texttt{Detect}, f] : [A_s] \rightarrow [A_T]$*, where* $A_s = (A_1, \ldots, A_{m'})$ *is a set of source attributes, and* A_T *is a single target attribute. The **error repair** logic* f*:* $Dom(A_1) \times \cdots \times Dom(A_{m'}) \rightarrow Dom(A_T)$ *determines the corrected value in the single target cell* $t.A_T$ *based on the specific values of the source cells* $t.A_s$*. The **error detector*** $\texttt{Detect} : Dom(A_1) \times \cdots \times Dom(A_{m'}) \times Dom(A_T) \rightarrow [0, 1]$ *evaluates at the cell level the error status of each target cell* $t.A_T$ *during the mapping process.*

`Detect` in our framework returns a probability of error *for each target cell*, and each error repair logic f derived from the source cells $t.A_s$ overwrites exactly one target cell $t.A_T$. Figure 3(a) illustrates a series of data cleaners. Cleaner $o_{m1} : f_{\text{Zip}} = \texttt{Ext.Zip} \rightarrow \texttt{City} = \texttt{Ext.City}$. Here, $A_s = \{\texttt{Zip}\}$ and $A_T = \texttt{City}$. The detector `Detect` evaluates at the cell level whether the cell $t.\texttt{City}$ with $\texttt{Zip} = 60608$ has the correct value *Chicago*.

In complex data cleaning workflows, multiple errors across attributes often require collaboration among cleaners. As shown in Figure 3(a), cleaners o_{c1}, o_{m1}, and o_{n1} all target the `City` for correction. However, o_{m1} and o_{n1} have conflicting repair operations, leading to redundancy and interference. Moreover, o_{c1} introduces a misspelled *Cicago*, which disrupts the outlier detection logic of o_{n1}. The root causes of these issues are: (*i*) o_{m1} and o_{n1} both modify the target $A_T = \texttt{City}$ but use different repair logic f and detection logic `Detect`, and (*ii*) o_{c1}'s target $A_T = \texttt{City}$ is the source attribute $A_s = \texttt{City}$ for o_{n1}. Even for cleaners targeting entire tuples, their detectors operate at the cell level, assessing each tuple t separately for each source-target attribute pair $(t.A_s, t.A_T)$, with actual repairs conducted at the cell level on the target cell $t.A_T$.

When multiple cleaners modify overlapping attributes, grouping *mutually dependent* cleaners into a *cleaning block* is essential. This block-based organization (*i*) *reduces redundant conflict resolution*, (*ii*) *enables efficient search for an execution order*, and (*iii*) *provides a natural unit for cost-aware optimization.* We therefore first introduce *cleaner association rules* to systematically build such blocks before presenting the overall processing flow.

Definition 3.2. *A* ***cleaner association rule*** *r defines the association conditions between two data cleaners. For a set of cleaners $Op = \{o_1, \ldots, o_n\}$, where $o_i = A_{S_i} \rightarrow A_{T_i}$, there exist i and j ($i \neq j$) such that: (i) If $A_{T_i} \in A_{S_j}$, then there is an association rule $r : o_i \rightarrow o_j$. (ii) If $A_{T_i} = A_{T_j}$, there is a mutual rule $r : o_i \rightarrow o_j$ and $o_j \rightarrow o_i$. That is, if A_{T_i} of o_i is in the source attribute set A_{S_j} of o_j, the result of o_i affects the execution of o_j. Similarly, if two cleaners share the same target attribute, they mutually influence each other.*

Based on these mappings, cleaners can be classified and ordered to achieve an optimal cleaning process with minimal interference.

Definition 3.3. *A* ***cleaner workflow*** *CL is an ordered set of cleaning blocks, each representing a sequence of cleaners, denoted as $CL = (C_1, \ldots, C_m)$, where each C_i is an ordered set representing an independent* ***cleaning block****. Specifically, (i) within each block $C_i = (o_{i1}, o_{i2}, \ldots, o_{in})$, for all $o_{ip}, o_{iq} \in C_i$ ($p \neq q$), there exists mutual dependency $o_{ip} \rightarrow o_{iq}$ and $o_{iq} \rightarrow o_{ip}$, and ($ii$) between different blocks C_j and C_i ($j < i$), for all $o_{jr} \in C_j$ and $o_{ip}, o_{iq} \in C_i$, there does not exist a dependency $o_{ip} \rightarrow o_{jr}$ or $o_{iq} \rightarrow o_{jr}$.*

The execution order within the same cleaning block C_i follows the cleaner sequence. Cleaning blocks $C_1, \ldots, C_m$ are executed in topological order derived from their cleaning dependencies.

Definition 3.4. *In a flow CL, each cleaning block C_i produces a set of data repair operations $L_i = \{l_1, \ldots, l_q\}$. A* ***data repair operations*** *$l \in L_i$ is defined as $\exists o_i \in C_k$, $o_i = f : A_{S_i} \rightarrow A_{T_i}$, such that $l : A_{S_i}, v_{A_{S_i}} \rightarrow A_{T_i}, v_{A_{T_i}}$, where A_{S_i} is a set of source attributes. $v_{A_{S_i}}$ is a set of values in $Dom(A_{S_i})$ that specifies the tuples to be repaired.*

Example 3.1. *As shown in Figure 1, the clean workflow $CL = \{o_{m1}, o_{n1}, o_{m2}, o_{n2}, o_{c4}, o_{c3}, o_{c1}\}$ includes a repair block $C = \{o_{c3}, o_{c1}\}$ consisting of two cleaners. Both o_{c3} and o_{c1} target* `Zip` *for repairing, but their source attributes and repair logic differ. Additionally, the target* `City` *of o_{c3} serves as a source attribute for o_{c1}, making the execution order critical to repair effectiveness. The repair operations generated by this block, $L = \{l_1, l_2\}$, are as follows:*

① *Operation l_1:* `Address` = *3465S MorganST,* `City` = *Chicago* → `Zip` = 60608, *c*3.
② *Operation l_2:* `DBAName` = *John Veliotis Sr.,* `City` = *Chicago* → `Zip` = 60609, *c*1.

Conflict exists between l_1 and l_2 as generated by o_{c3} and o_{c1}. Evaluation reveals that applying l_1 maximizes the block's overall satisfaction even though it repairs fewer cells. Consequently, the repair block is updated to $L = \{l_1\}$ and $C = \{o_{c3}\}$, generating an optimal cleaning workflow that balances fine- and coarse-grained operations.

3.2 Problem statement

Given a library of cleaners Op and a data instance D under schema $\mathcal{R}$, **the mixed error cleaning problem** is to determine a cleaning process flow CL and the corresponding set of *cell-level* repair operations, such that the cleaned dataset $D' = L_1 \cdot \ldots \cdot L_m(D)$ achieves optimal data quality. Both detection and repair are performed explicitly at the cell level: even when cleaners aggregate evidence across entire tuples (such as row distribution-based cleaners), their resulting repair operations still individually target single cells $t.A_T$.

In mixed-error cleaning tasks, we aim to globally optimize the cleaning process to leverage each cleaner's maximum potential. At each repair step, repair operations are carefully selected within cleaning blocks based on cell-level detection scores, ensuring the greatest cumulative improvement in data quality. Additionally, the cleaning cost is evaluated by comparing the original D and the repaired D', selecting the repair configuration that optimally balances quality improvement and minimal data modification. Formally, we define the global cleaning effectiveness objective Q as follows:

$$CL, Operations = \arg \min_{(CL, Operations)} [Q(D, CL, Operations)] \quad (1)$$

where

$$Q = \sum_{\substack{o \in C_i \in CL \\ L_i \in Operations}} w_o \cdot \sum_{t \in L_i(D)} \texttt{Detect}_o(t) + cost(D, L_i(D)). \quad (2)$$

Here, $L_i(D)$ denotes the dataset cleaned by block C_i, w_o is the confidence weight of cleaner o in block C_i. The cleaning cost function $cost(D, L_i(D))$ represents the modification cost (e.g., the number of records or fields modified). This helps minimize large-scale data changes and reduces the intrusiveness of the cleaning process.

The cleaning capability of a block C_i is measured by the weighted results of its detectors. Specifically, for each cleaner o with [`Detect`, f], $\texttt{Detect}_o(t)$ evaluates the probability of an error in t of the cleaned dataset $L_i(D)$. Therefore, $\sum_{t \in L_i(D)} \texttt{Detect}_o(t)$ measures the error level in $L_i(D)$. The lower the error level, the higher the cleaning quality and capability of C_i. Our optimization goal is to adjust the contribution weights $W = \{w_o \mid o \in \bigcup_{C_i \in CL} C_i\}$ to generate the optimal cleaner order CL and the corresponding set of data repair *Operations*, such that $Q(D, CL, Operations)$ is minimized.

UniClean optimizes the cleaning process flow and repair operations through standardized interfaces, dependency analysis, and cost-aware strategies. Section 4 presents its detailed steps.

4 THE PIPELINE OF UNICLEAN

We first outline the key steps of **UniClean**, and then provide detailed descriptions of each step in Sections 4.1-4.3. As shown in Figure 2, **UniClean** integrates three core components:

In **cleaner construction** phase, we implement standardized interfaces in **UniClean** to support the specification and collaborative use of different cleaners. Cleaners are classified into *Column rule-based* and *Row distribution-based cleaners*, that can be optionally extended with external knowledge. In addition, *Cleaning cost function* is introduced to quantify the cost of cleaning operations.

In **cleaning preparation** phase, we partition cleaners into independent *cleaning blocks CL* and extract *core sample sets S_i* for each block, which are further divided into sub-blocks $Block_i = \{B_1, \ldots, B_m\}$ (Algorithms 1, 2, 3). These steps ensures that the cleaning objective function Q on the core set is equivalent to that on the full dataset, as guaranteed by Theorem 4.2. The theoretical support narrows the search space for repair operations, reduces computational complexity, and ensures both the consistency and efficiency of the cleaning process.

In **data cleaning** phase, we propose a global optimization framework to dynamically generate and apply repair operations *Operation* = $\{L_1, \ldots, L_n\}$ across the cleaning workflow CL, guided by the objective function Q (Algorithm 4). Cleaner weights $W_i = \{w_o \mid o \in \bigcup_{C_i \in CL} C_i\}$ are iteratively updated through repair tracing to optimize execution order and adapt strategies to varying data distributions. We guarantee global error reduction and cleaning consistency, as formalized in Theorem 4.3.

Upon completion, **UniClean** outputs the **macro cleaning workflow** CL, **cleaner weights** W_i, **fine-grained repair operation sequences** L_i, and the **final cleaned dataset** D'.

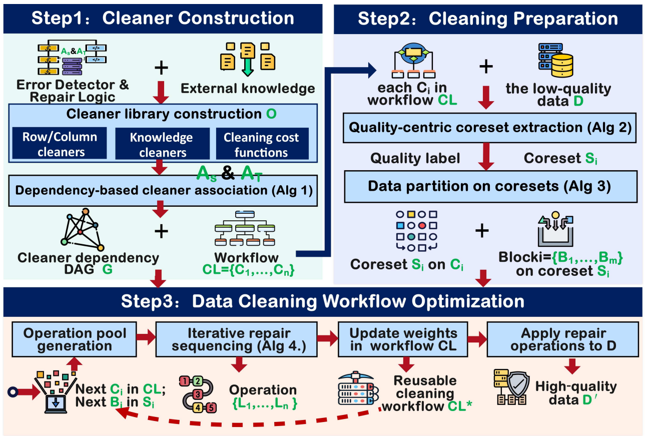

Step1：Cleaner Construction
A_s&A_T
Error Detector & Repair Logic
+
External knowledge
Cleaner library construction O
Row/Column cleaners
Knowledge cleaners
Cleaning cost functions
A_s & A_T
Dependency-based cleaner association (Alg 1)
Cleaner dependency DAG G
+
Workflow CL={C1,...,Cn}
Step2：Cleaning Preparation
each Ci in workflow CL
+
the low-quality data D
Quality-centric coreset extraction (Alg 2)
Quality label
Coreset Si
Data partition on coresets (Alg 3)
Coreset Si on Ci
+
Blocki={B1,...,Bm} on coreset Si
Step3：Data Cleaning Workflow Optimization
Operation pool generation
Iterative repair sequencing (Alg 4.)
Update weights in workflow CL
Apply repair operations to D
Next Ci in CL; Next Bi in Si
1 2 3 4 5
Operation {L1,...,Ln }
Reusable cleaning workflow CL*
High-quality data D′

Figure 2: The overview of UniClean.

4.1 Unified construction of cleaners

4.1.1 Cleaner classification and construction. To address diverse errors, we observe that existing cleaners exhibit heterogeneous interfaces and procedures. Thus, as the initial step in **UniClean**, we establish standardized cleaner interfaces to unify error-handling capabilities. We categorize the cleaners in Definition 3.1 into two categories, along with the cleaning cost function.

Definition 4.1. ***Column rule-based cleaners*** *($o_c : [A_s] \rightarrow [A_T]$) capture column dependencies such as CFDs, or DCs, where A_s functionally determines A_T. For a tuple $t \in D$, $\Pr(t.A_T \mid t.A_s)$ is the correctness probability, and the detector is* $\texttt{Detect}(t.A_T) = 1 - \Pr(t.A_T \mid t.A_s)$. ***Row distribution-based cleaners*** *($o_r : [A_s] \rightarrow [A_T]$) enforce that the values of (A_s, A_T) across tuples follow an expected joint distribution constructed from prior knowledge. Such prior information may include external data rules (e.g., historical business knowledge) or internal statistical preferences (e.g., frequency distributions and co-occurrence patterns mined by ML models [13, 34, 41, 47]). The detector is* $\texttt{Detect}(t.A_T) = 1 - \Pr(t.A_s, t.A_T)$.

Cleaning cost function. UniClean *also supports cost-sensitive cleaning. Formally, for each $t \in D$, the repair cost is defined as $Cost(t) = cost(t.A_T, t.A'_T)$, where $cost(a, b)$ quantifies the modification cost between the original and repaired values.*

The complexity of both cleaner types depends on the underlying detection logic: it is typically $O(1)$ for simple rules (e.g., regex or type checks), and commonly $O(|D|)$ for most statistical or learning-based detectors [43]. The cost function supports $O(1)$ evaluation and is customizable. Users may incorporate cleaner execution complexity into $cost(a, b)$ via adjustable penalty weights, allowing trade-offs between cleaning effectiveness and computational cost.

We emphasize that **UniClean** supports optional integration of external knowledge (e.g., data constraints, dictionaries, custom repair operations) to enhance cleaning capabilities. This extensibility enables adaptation to diverse data scenarios, leveraging external information to refine error detection and repair processes.

Example 4.1. *Figure 3 shows that **UniClean** incorporates three representative cleaners to address data quality issues: (1)* ***Column rule-based cleaner*** *o_{c1} detects errors using strict attribute relationships, $\mathbf{A}_s$ = (*$\texttt{City}, \texttt{DBAName}$*) and $\mathbf{A}_T$ = Zip. The detection function is* $\texttt{Detect}_{o_{c1}}(t) = 1 - \Pr(t.\texttt{Zip} \mid t.\texttt{City}, t.\texttt{DBAName})$. *For t.*$\texttt{City}$ = *Chicago and t.*$\texttt{DBAName}$ = *Hilton Garden Inn, D contains t_8.*$\texttt{Zip}$ = 60608 *and $t_7, t_{10}, t_{11}, t_{12}$.*$\texttt{Zip}$ = 60611, *giving* $\texttt{Detect}_{o_{c1}}(t_8) = 1 - \frac{1}{5} = \frac{4}{5}$, *and* $\texttt{Detect}_{o_{c1}}(t_7) = \texttt{Detect}_{o_{c1}}(t_9) = \frac{1}{5}$. *(2)* ***Row distribution-based cleaner*** *o_{n2} ensures consistency by evaluating the distribution of $\mathbf{A}_s$ =* $\texttt{Address}$, *with* $\texttt{Detect}_{o_{n2}}(t) = 1 - \Pr(t.\texttt{Address})$. *Records t_2, t_6 in Table 1 form one cluster, and t_1, t_3, t_4, t_5 form another, with* $\Pr(t_2.\texttt{Address}) = \Pr(t_6.\texttt{Address}) = \frac{2}{6}$, *yielding* $\texttt{Detect}_{o_{n2}}(t_2) = \texttt{Detect}_{o_{n2}}(t_6) = 1 - \frac{2}{6} = \frac{2}{3}$. *(3)* ***Knowledge-enhanced cleaners*** *(o_{m1} :* $\texttt{Zip} \longrightarrow \texttt{City}$, *$o_{m2}$:* $\texttt{DBAName} \longrightarrow \texttt{DBAName}$*). o_{m1} integrates column rules with external dictionaries to validate $\mathbf{A}_T$ =* $\texttt{DBAName}$, *while o_{m2} combines row distributions with external knowledge to repair $\mathbf{A}_s$ =* $\texttt{Address}$ *by aligning with patterns from* $\texttt{Ext\_DBAName}$. *Each cleaner in **UniClean** operates independently, iterating until all tuples satisfy* $\texttt{Detect}_o(t) = 0$, *ensuring thorough cleaning.*

4.1.2 Completeness of Cleaner Construction. The proposed cleaner construction approach unifies the use of row and column correlations, offering a structured methodology for data cleaning. This approach integrates advanced techniques such as denial constraint repair [29] and stream data cleaning under speed constraints [12, 50]. Next, we prove the completeness of our approach, ensuring effective and accurate cleaning through row- and column-related cleaners.

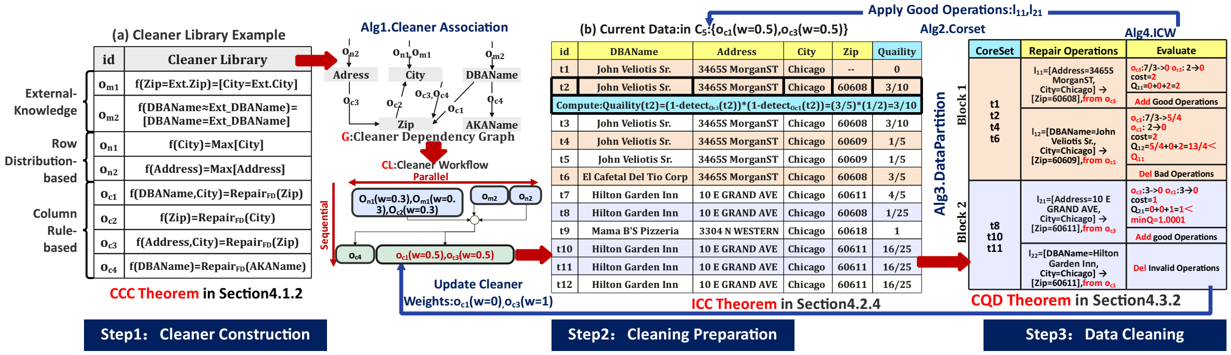
Theorem 4.1 (Completeness of Cleaner Construction (CCC)). *If $A_T \in Attr(\mathcal{R})$ has neither row dependency (detectable by row distribution cleaners) nor column dependency (detectable by column rule cleaners), then cleaning A_T is equivalent to random replacement in a probabilistic sense, assuming a maximum-entropy (i.e., uniform) distribution when no external or prior knowledge is available.*

Proof. Assume cleaning the target attribute $A_T \in Attr(\mathcal{R})$ under the known correct values of the source attribute set A_S. (*i*) For any tuple $t \in D$, if A_T has *no column dependency* with the source attributes $\{A_{S1}, \ldots, A_{Sk}\}$, as modeled by column rule-based cleaners, the probability that $t.A_T$ is correct is: $\Pr(A_T = t.A_T \mid A_{S1} = t.A_{S1}, A_{S2} = t.A_{S2}, \ldots, A_{Sk} = t.A_{Sk}) = \Pr(A_T = t.A_T)$. (*ii*) If A_T exhibits *no row dependency* with the source attributes and no external or prior distributional information is available, we invoke the maximum-entropy principle, which posits that in the absence of any known constraints or domain knowledge, the most unbiased probabilistic model is the one that maximizes entropy [33], i.e., the uniform distribution: $\Pr(A_T = t.A_T) = \frac{1}{|Dom(A_T)|}$.

Combining the above, when there is *neither row dependency nor column dependency*, and given the absence of external knowledge or detectable internal correlations, the probability that $t.A_T$ is correct defaults to the maximum-entropy assumption: $\Pr(A_T = t.A_T \mid A_{S1} = t.A_{S1}, A_{S2} = t.A_{S2}, \ldots, A_{Sk} = t.A_{Sk}) = \frac{1}{|Dom(A_T)|}$. Thus, under these assumptions, the success probability of correctly repairing $t.A_T$ is equivalent in expectation to uniformly choosing a value at random from the domain $Dom(A_T)$. □

This equivalence, however, is strictly in the probabilistic sense under the maximum-entropy assumption rather than implying that A_T is truly uniform in real-world data. In practice, if any skewed or structured distribution exists, it can be exploited by row distribution cleaners. thus, maximum entropy is only assumed after exhaustively failing to detect any informative dependency, either internal or external. Therefore, without dependencies or additional external knowledge, no deterministic cleaning strategy can consistently outperform random guessing in terms of repair accuracy.

In summary, Theorem 4.1 establishes the boundary of cleaner construction: If a target attribute A_T has repairable errors (i.e.,


(a) Cleaner Library Example
External-Knowledge
Row Distribution-based
Column Rule-based
CCC Theorem in Section4.1.2
Step1: Cleaner Construction
Alg1.Cleaner Association
o_{n2}
o_{n1},o_{m1}
o_{m2}
Adress
City
DBAName
o_{c3}
o_{c2}
o_{c3},o_{c4}
o_{c1}
o_{c4}
Zip
AKAName
G:Cleaner Dependency Graph
CL:Cleaner Workflow
Parallel
Sequential
O_{n1}(w=0.3),O_{m1}(w=0.3),O_{c2}(w=0.3)
o_{m2}
o_{n2}
o_{c4}
o_{c1}(w=0.5),o_{c3}(w=0.5)
Update Cleaner Weights:o_{c1}(w=0),o_{c3}(w=1)
(b) Current Data:in C_5:{o_{c1}(w=0.5),o_{c3}(w=0.5)}
Apply Good Operations:l_{11},l_{21}
Alg2.Corset
Alg3.DataPartition
Alg4.ICW
ICC Theorem in Section4.2.4
Step2: Cleaning Preparation
CQD Theorem in Section4.3.2
Step3: Data Cleaning

id	Cleaner Library
o_{m1}	f(Zip=Ext.Zip)=[City=Ext.City]
o_{m2}	f(DBAName≈Ext_DBAName)=[DBAName=Ext_DBAName]
o_{n1}	f(City)=Max[City]
o_{n2}	f(Address)=Max[Address]
o_{c1}	f(DBAName,City)=Repair_{FD}(Zip)
o_{c2}	f(Zip)=Repair_{FD}(City)
o_{c3}	f(Address,City)=Repair_{FD}(Zip)
o_{c4}	f(DBAName)=Repair_{FD}(AKAName)

id	DBAName	Address	City	Zip	Quaility
t1	John Veliotis Sr.	3465S MorganST	Chicago	--	0
t2	John Veliotis Sr.	3465S MorganST	Chicago	60608	3/10
Compute:Quaility(t2)=(1-detect_{Oc3}(t2))*(1-detect_{Oc1}(t2))=(3/5)*(1/2)=3/10					
t3	John Veliotis Sr.	3465S MorganST	Chicago	60608	3/10
t4	John Veliotis Sr.	3465S MorganST	Chicago	60609	1/5
t5	John Veliotis Sr.	3465S MorganST	Chicago	60609	1/5
t6	El Cafetal Del Tio Corp	3465S MorganST	Chicago	60608	3/5
t7	Hilton Garden Inn	10 E GRAND AVE	Chicago	60611	4/5
t8	Hilton Garden Inn	10 E GRAND AVE	Chicago	60608	1/25
t9	Mama B'S Pizzeria	3304 N WESTERN	Chicago	60618	1
t10	Hilton Garden Inn	10 E GRAND AVE	Chicago	60611	16/25
t11	Hilton Garden Inn	10 E GRAND AVE	Chicago	60611	16/25
t12	Hilton Garden Inn	10 E GRAND AVE	Chicago	60611	16/25

	CoreSet	Repair Operations	Evaluate
Block 1	t1 t2 t4 t6	l_{11}=[Address=3465S MorganST, City=Chicago] → [Zip=60608],from o_{c3}	o_{c3}:7/3->0 o_{c1}: 2→0 cost=2 Q_{11}=0+0+2=2
			Add Good Operations
		l_{12}=[DBAName=John Veliotis Sr., City=Chicago] → [Zip=60609],from o_{c1}	o_{c3}:7/3->5/4 o_{c1}: 2→0 cost=2 Q_{12}=5/4+0+2=13/4< Q_{11}
			Del Bad Operations

Block 2	t8
	t10
	t11
		l_{21}=[Address=10 E GRAND AVE, City=Chicago] → [Zip=60611],from o_{c3}	o_{c3}:3->0 o_{c1}:3→0
			cost=1
			Q_{21}=0+0+1=1<
			minQ=1.0001

			Add good Operations
		l_{22}=[DBAName=Hilton Garden Inn, City=Chicago] → [Zip=60611],from o_{c1}	Del Invalid Operations

Figure 3: The demonstration of the cleaning process implemented by UniClean.

repair success probability exceeds random guessing), it must exhibit structural patterns, i.e., patterns that our proposed rule-based cleaners or distribution-based cleaners can exploit.

4.2 Cleaning preparation

To address mixed-error cleaning complexity, we optimize operational execution after unified cleaner construction. Direct application of multiple cleaners to large datasets is impractical due to excessive repair operations. We tackle this via two strategies: (*i*) *Cleaners Pruning*: minimize repair operations per block by limiting the number of cleaners used, and (*ii*) *Data Scaling*: reduce per-block processing volume to constrain candidate operations. This dual optimization ensures cleaning efficiency while maintaining effectiveness and minimizing downstream selection complexity.

4.2.1 Data blocking based on cleaner association analysis. We propose a multi-cleaner association method to handle dependencies between cleaners. This approach groups cleaners sharing related attributes into blocks and isolates unrelated ones, thereby reducing candidate repair operations per block. Starting with the cleaner library Op, we identify the source attribute set A_S and target repair attribute A_T for each cleaner. Based on dependency conditions (Definition 3.2), we build a cleaner dependency network (attribute as vertices and cleaners as edges). We apply directed graph techniques (e.g., vertex merging, cycle detection) to resolve dependencies and generate a directed acyclic graph (DAG), and then derive an initial cleaning workflow CL and core set S. Algorithm 1 outlines this process, including the following steps:

① **Dependency graph construction.** After initialing, for each cleaner in Op, it identifies the source A_S and target A_T and adds the edge (A_S, A_T) to G. This step costs $O(|Op| \cdot |Attr|)$ in times.

② **Cycle detection and merging.** Cycles in the dependency graph are detected and resolved by merging cycle nodes into a single node, eliminating cyclic dependencies to ensure a DAG. The time complexity of cycle detection and merging is $O(|V| + |E|)$, where $|V| \leq |Attr|$ represents the number of nodes, and $|E| \leq |Op|$ represents the number of dependencies. In the worst case, merging cycles across all nodes costs $O(|Attr| \cdot (|Attr| + |Op|))$ in time.

③ **Topological sorting and block partitioning.** The algorithm performs a topological sort on the DAG to generate the execution order of cleaning blocks, ensuring that each cleaner is processed only after its dependencies have been resolved. Cleaners within the same level can be executed in parallel, while blocks in different levels are processed sequentially. The time complexity of topological sorting is $O(|V| + |E|)$, i.e., $O(|Attr| + |Op|)$.

Time Complexity. From the above, in the worst case, the overall time complexity of Algorithm 1 is $O(|Attr|^2 + |Attr| \cdot |Op|)$.

Algorithm 1: CleanerAssociation

```
Input: Cleaner Set $Op$, Attribute Set $Attr$
Output: Cleaner Association DAG: $G$, Initial Cleaner Workflow: $CL$
Edges ⟵ $\emptyset$, $G$ ⟵ InitializeGraph($Attr$)
foreach $op \in O$ do
    src ⟵ $op.As$, tgt ⟵ $op.At$, Edges.add(src, tgt)
$G$.AddEdges(Edges) // Construct dependency graph
while exists cycles in $G$ do
    CycleNodes ⟵ DetectCycles($G$) // Identify cycles
    MergedNode ⟵ MergeNodes(CycleNodes)
    $G$.Update(MergedNode)
$CL$⟵TopologicalSort($G$) // Get initial cleaner workflow
return $G, CL$
```

Example 4.2. *As illustrated in Figure 3, for the data in Table 1, we construct graph G with attributes like* City, Zip, *and* Address *as vertices, and cleaners like o_{c1} connecting* (City, DBAName) ⟶ Zip. *Cyclic dependencies in G are detected and resolved to produce a DAG, ensuring the feasibility of topological sorting. Then, cleaners are grouped into blocks CL based on dependency levels in the DAG. For example, o_{c1} and other cleaners in C_1 are executed in parallel, followed by cleaners in C_2. By iteratively analyzing dependencies and scheduling execution blocks,* ***UniClean*** *ensures efficient and dependency-respecting data cleaning.*

4.2.2 Core set extraction based on quality evaluation. We next extract a subset of the large dataset to determine the best cleaning strategy. Unlike conventional ML tasks that seek clean subsets, our method retains dirty data to ensure comprehensive error coverage while including clean data to guide accurate repair operation generation. To this end, we propose a quality evaluation-based method for core set extraction, implemented via Spark-based sampling. As outlined in Algorithm 2, we extract dirty data blocks and selects clean data based on error label distribution. The process involves:

① **Data quality evaluation.** For each cleaning block $C_i \in CL$, it evaluates data quality on the full D. Using the detection functions $\texttt{Detect}_o$ of the cleaners in C_i, the quality score of each tuple t is approximated as the product of the quality scores from all cleaners in C_i: $Quality(t) = \prod_{o \in C_i} (1 - \texttt{Detect}_o(t))$. This score represents the probability that a is correct under C_i. An error threshold is estimated as: $\text{error}_\theta = \frac{1}{|D|} \sum_{t \in D} (1 - Quality(t))$. It represents tuples that are below the average quality threshold and have a higher error probability. These tuples can aid in pruning repair operations. This step incurs a time cost of $O(|Op| \cdot |D|^2)$.

② **Core set sampling.** Based on the quality scores, the algorithm extracts a core set while preserving the quality distribution of the original dataset. Records with $Quality(t) = 1$ are removed, as they are considered absolutely correct and irrelevant for cleaning. Subsequently, records with $Quality(t) < \text{error}_\theta$ are selected, representing erroneous data that require immediate inclusion in the core set. For these selected tuples, related tuples are identified based on the attribute set $\mathbf{A_T}$ of the cleaning block C_i, grouping all tuples sharing the same values in $\mathbf{A_T}$. Within each group, tuples are sorted by $Quality(t)$ in descending order, and sampling is performed to retain a proportional representation of the quality distribution.

This approach ensures the core set retains both representative correct and erroneous records, maintaining the quality distribution of the original dataset while significantly reducing its size.

Time Complexity. For a cleaning workflow CL containing $n \leq |Op|$ cleaning blocks, Algorithm 2's complexity on D is $O(|Op| \cdot |D|^2)$. Ideally, the core set includes all dirty data along with corresponding clean data for assistance, reducing its size to $O(|D_{\text{error}}|)$, where $|D_{\text{error}}|$ is the number of erroneous records.

Algorithm 2: Quality-based CoreSet

```
Input: Data $D$, Cleaning Block $C_i = \{o_{ij}\}$
Output: Core Set $S_i$, Quality Scores $Quality_i$, Threshold $\text{error}_\theta$
Initialize $S_i \leftarrow \emptyset$, $Quality_i \leftarrow \emptyset$, $\text{error}_\theta \leftarrow 0$
// Step 1: Quality analysis
foreach $t \in D$ do
    $Quality_i(t) \leftarrow \prod_{o \in C_i} (1 - \text{Detect}_o(t))$
$\text{error}_\theta[i] \leftarrow \frac{1}{|D|} \sum_{t \in D} (1 - Quality_i(t))$
// Step 2: Core set selection
$D_{\text{error}} \leftarrow \{t \in D \mid Quality_i(t) < \text{error}_\theta[i]\}$
$S_i \leftarrow S_i \cup D_{\text{error}}$, $D' \leftarrow D \setminus D_{\text{error}}$
Group $D'$ by attributes $C_i.\mathbf{A_T}$
foreach group $G$ in $D'$ do
    Sort $G$ by $Quality_i(t)$ in descending order
    $S_i \leftarrow S_i \cup \text{Sample}(G, \text{ProportionPreserve}(Quality_i))$
        // Sample from each group
return $S_i$, $Quality_i$, $\text{error}_\theta[i]$
```

Example 4.3. *Figure 3 shows that when processing the cleaning block $C_i = \{o_{c1}(w = 0.5), o_{c3}(w = 0.5)\}$, core set extraction proceeds as follows: (1) calculate detection values $\text{Detect}_{o_{c1}}$ and $\text{Detect}_{o_{c3}}$ for all tuples, and derive quality scores $Quality(t)$. (2) Classify tuples as clean ($Quality(t) = 1$), erroneous ($Quality(t) < \frac{1}{5}$), or intermediate. (3) Remove clean tuples, fully sample erroneous tuples (e.g., t_1, t_8), and proportionally sample intermediate tuples based on related group quality distribution. The resulting core set is balanced, retaining high-quality and erroneous data for effective repair evaluation while preserving the original dataset's quality distribution.*

4.2.3 *Data partitioning based on cleaner classification.*

After classifying cleaners by attribute associations, conflicts or independence may arise among fine-grained cleaners. For example, interdependent cleaners o_{c_3} and o_{c_1} may generate overlapping repair operations (Figure 1), requiring selection and ordering. Non-conflicting operations can be applied directly. To simplify cleaning search, we propose a data blocking method based on cleaner classification. We group tuples likely to generate conflicting operations into blocks, ensuring cleaners between blocks do not conflict. This reduces the time for searching and evaluating repair operations. The process is detailed in Algorithm 3, consisting of two iterative phases.

①**Preliminary partitioning.** For each cleaner $o_{ij} \in C_i$, the core set S_i is aggregated based on the cleaner's source attributes A_{sj}. Tuples with identical values in A_{sj} are assigned to the same preliminary block. The result is an initial block list $Block_i^{\text{init}}$, where each block groups records that are likely to share the same repair path. At this stage, blocks ensure attribute-level consistency but do not yet account for potential inter-cleaner interactions.

② **Iterative block expansion.** To enhance the repair effect for low-quality tuples (with $Quality(t) < \text{error}_\theta$), blocks are iteratively expanded. For each low-quality t, additional cleaners $o_{ik} \in C_i$ ($j \neq k$) are applied based on their source attributes A_{sk}. Related tuples sharing the same A_{sk} values are identified and merged into the block containing t. This iterative expansion continues until all low-quality tuples are adequately covered, ensuring blocks capture cross-cleaner interactions while preserving high-quality tuples.

Specifically, these data blocks exhibit the following properties.

Proposition 4.1. *The final* stable *block list $Block_i$ exhibits the following properties: ① Each low-quality record t belongs to a single block and does not cross blocks: $\forall t \in B_i \in Block, j \neq i, t \notin B_j$. ② Repair operations generated by cleaners $o_{ij} \in C_i$ for each block B_i are independent and non-conflicting. ③ The core set S_i is completely partitioned by Algorithm 3, i.e., $S_i = \bigcup_{B \in Block_i} B$.*

Algorithm 3: DataPartition

```
Input: Core Set $S_i$, Cleaning Block $C_i = \{o_{ij}\}$, Threshold $\text{error}_\theta$
Output: Stable Block List $Block_i = \{B_1, \ldots, B_m\}$
Initialize $Block_i^{\text{init}} \leftarrow \emptyset$
foreach $o_{ij} \in C_i$ do
    foreach $t \in S_i$ do
        if $t.A_{sj}$ is unique then
            Assign $t$ to a preliminary block $B$ based on $A_{sj}$
$Block_i^{\text{init}} \leftarrow$ Preliminary blocks
Iterative Block Expansion:
foreach Record $t \in S_i$ where $Quality(t) < \text{error}_\theta$ do
    foreach $o_{ik} \in C_i \mid j \neq k$ do
        Identify related records $b \in S_i$ where $b.A_{sk} = t.A_{sk}$
        Merge $b$ into the block containing $t$
Finalize Blocks:
foreach Block $B \in Block_i^{\text{init}}$ do
    Ensure $S_i \leftarrow \bigcup_{B \in Block_i} B$
    Remove duplicates and conflicts between blocks
return $Block_i$
```

Time Complexity. Algorithm 3 requires performing one partitioning operation on each block C_i in the cleaning process $CL = \{C_1, ...C_n\}$ with a time complexity of $O(|C_i| \cdot |D|)$. Therefore, the time complexity of applying Algorithm 3 to all cleaning blocks in CL on D is $O(|Op| \cdot |D|)$.

Example 4.4. *As illustrated in Figure 3, consider the sampled data $\{t_1, t_2, t_3, t_4, t_6, t_8, t_{10}, t_{11}\}$ for data partitioning. The cleaning* *block C_i includes two cleaners, o_{c1} and o_{c2}, with their respective target attributes $\mathbf{A}_\mathrm{T}$. Initially, the data is partitioned based on $o_{c1}.\mathbf{A}_\mathrm{T}$. Records are grouped by identical values in $o_{c1}.\mathbf{A}_\mathrm{T}$, resulting in the following blocks: $Block_i^{\mathrm{init}} = \{\{t_1, t_2, t_3, t_4\}, \{t_6\}, \{t_8, t_{10}, t_{11}\}\}$. Next, the algorithm examines $o_{c2}.\mathbf{A}_\mathrm{T}$ to identify relationships between blocks. Record t_6 shares attributes in $o_{c2}.\mathbf{A}_\mathrm{T}$ with records in the block $\{t_1, t_2, t_3, t_4\}$. These blocks are merged, resulting in $Block_i = \{\{t_1, t_2, t_3, t_4, t_6\}, \{t_8, t_{10}, t_{11}\}\}$. For tuples with $Quality(t) < \mathrm{error}_\theta = \frac{1}{5}$ (e.g., t_1 and t_8), the algorithm ensures no further blocks need to be merged, as no other tuples are associated with these low-quality tuples. Thus, the partitioning process terminates with the final block list: $Block_i = \{\{t_1, t_2, t_3, t_4, t_6\}, \{t_8, t_{10}, t_{11}\}\}$.*

Considering the operations conducted during data preprocessing, We report the theoretical guarantees of the algorithms employed in Theorem 4.2 for data cleaning preparation phase.

Theorem 4.2 (Independence and consistency of cleaning workflow (ICC)). *For independent cleaning blocks within one core set, the cleaning objective function is equivalent to the global cleaning objective function Q on the entire D. This is guaranteed through three key properties: independence of cleaning blocks, consistency of quality distribution in sampled data, and independence of partitioned data.*

Proof. (*i*) The **independence of cleaning blocks** ensures that repair operations in different blocks do not interfere with each other, and each block's cleaning results are independent of others. By performing a topological sort on the dependency graph (as described in Section 4.2.1), the execution order of cleaners adheres to the dependency relationships defined in Definition 3.2. For cleaning blocks $C_{i_k} \in \mathrm{level}_i$ and $C_{j_m} \in \mathrm{level}_j$ $(i < j)$, repair operations L_{i_k} and L_{j_m} satisfy $L_{i_k} \cap L_{j_m} = \emptyset, \forall C_{i_k} \in \mathrm{level}_i, C_{j_m} \in \mathrm{level}_j$. This ensures that repair operations across levels are non-conflicting.

(*ii*) **Consistency of quality distribution in sampled data.** The consistency of sampled data ensures that the cleaning objective on sampled data aligns with that on the full dataset. As defined in Section 3.2, the cleaning objective function Q minimizes: $\sum_{C_i \in CL} \sum_{o \in C_i} \sum_{t \in l_i(D)} w_o \cdot \mathtt{Detect}_o(t)$. $\mathtt{Detect}_o(t)$ denotes the error probability of t in cleaner o. Algorithm 2 ensures the quality distribution of sampled data maintains the same relative order as D, enabling repair operations generated on the core set S to match those on D: $L_S(t) = L_D(t), \forall t \in S \subseteq D$.

(*iii*) **Independence of partitioned data.** The partitioning in Algorithm 3 within a cleaning block C_i ensures that each low-quality tuple belongs to only one data block. For any $t \in B_i$ and $b \notin B_i$: $\forall o_{ij} \in C_i$, if $b.A_{sij} = t.A_{sij}$, then $b \in B_i$. This guarantees that repair operations within each block are independent and non-conflicting, avoiding interference between blocks. □

Accordingly, our algorithms for data cleaning and cleaner preparation reduce the search space and computational complexity, while preserving cleaning consistency with the full dataset.

4.3 Data cleaning

To address complex data errors where repair operations impact subsequent detection efficacy, we propose a globally optimized cleaning orchestration strategy. This method selects repair operations to maximize the collective performance of all detectors within cleaning blocks, thereby fully exploiting cleaner capabilities. For a given workflow CL and its corresponding core set S, we determine the optimal repair operation set *Operation* that maximizes data quality while satisfying the objective function Q. Our cleaning process, outlined in Algorithm 4, comprises the following steps:

Step 1: Layered processing of cleaning blocks. Cleaning blocks $C_i \in CL$ are processed in a topological order of layers level_i. For each block, we extract the core set S_i with Algorithm 3, generate a candidate repair operation parameter pool and arrange the sequences of repair operations within the pool (see Sect. 4.3.1-4.3.2).

Step 2: Dynamic adjustment cleaning strategy. After executing each cleaning block C_i, we apply the generated set of repair operations to D. We are then able to trace the cleaners based on the effectiveness of the operations and dynamically adjust the cleaner weights (see Sect. 4.3.3). Subsequently, we re-execute Step 1, optimizing the cleaning effect of Step 2 based on the current block's cleaner weights and reducing unnecessary computations.

The cleaning process terminates when no executable repair operations can be identified for any cleaning block.

4.3.1 Candidate Repair Parameter Pool Generation (`GenerateFixes`*).* The generation of candidate repair operations is based on the process outlined in Definition 3.3, with additional pruning using external knowledge bases to remove unnecessary parameters. Repairs are generated by iterating over partitioned tuples $t \in B_i$ in each block, guided by quality scores and the error threshold. The steps are as follows: ① **Generate predicate conditions.** For such tuples that $Quality(t) < \mathrm{error}_\theta$, collect values of source attributes $o.A_s$ to form predicates, defining the range of records needing repair. ② **Generate repair values.** For tuples that $Quality(b) > \mathrm{error}_\theta$, filter repair values using external knowledge bases, excluding invalid entries. Valid repair values are retained as candidates, enhancing accuracy and coverage. ③ **Generate repair operations.** Combine predicates and repair values for each cleaner's target attribute to define repair operations. Conflicting or redundant repairs are resolved using additional knowledge-base-defined rules.

Time Complexity. Based on Theorem 4.2, the repair operations within each block are independent. Let the size of the dataset after sampling and blocking within C_i be $|B_i|$ with an average error rate of $rate_i$, the number of erroneous tuples is $|B_i| \cdot rate_i$, and each tuple generates $O(1)$ predicates and repair values. Thus, for a block C_i, it total costs $O(|C_i| \cdot |B_i| \cdot rate_i)$.

4.3.2 Operations Sequence Orchestration (`FixSearch`*).* To address the proposed cleaning problem, **UniClean** is designed to prioritize applying repair operations that minimize the detector scores of all cleaners within the cleaning block. By iteratively analyzing the impact of each repair operation on data quality and cleaner performance, it dynamically adjusts the operation sequence to ensure that each repair step contributes optimally towards the global optimization goal. The core of function `FixSearch` function comprises the following steps: ① **Evaluate candidate repairs.** For each repair $l \in L_i$, assess its impact on the objective function Q, including the weighted sum of detector scores and the introduced repair cost. ② **Select and apply optimal repair.** Iteratively select the repair l^* that minimizes the overall error probability of all cleaners within the block. Apply l^*, update the dataset and its $Quality(t), \forall t \in B_i$, and remove l^* from the candidate pool. ③ **Dynamic repair operations sequence adjustment.** Continuously update the candidate pool by removing invalid or conflicting repairs and stopping when no repairs further reduce errors or all errors are resolved.

Time Complexity. Evaluating repairs in a block C_i requires computing new data quality scores and detection functions with complexity $O(|B_i|^2 \cdot C_i)$. Iterating through $O(|B_i| \cdot rate_i)$ repair operations leads to total complexity $O(|B_i|^4 \cdot rate_i^2 \cdot |C_i|)$.

4.3.3 *Traceability of Repair Operations and Iteration (*`UpdateWeights`*).*

UniClean continuously optimizes the cleaning workflow by dynamically adjusting cleaner weights through repair traceability. Specifically, each repair operation is traced back to its originating cleaner and quantitatively evaluated based on its impact on the global Q. An operation is considered effective if its application leads to a significant decrease in Q; otherwise, it is treated as redundant or even harmful. This enables **UniClean** to (1) degrade the weight of cleaners that consistently yield low-impact or ineffective repairs and exclude them from subsequent fix generation (`GenerateFixes`), and (2) prioritize high-weight cleaners when searching for high-quality repairs (`FixSearch`). As a result, the cleaner library is progressively refined by pruning underperforming cleaners and preserving only impactful operations. This iterative mechanism ensures that the cleaning workflow remains adaptive for the current error distribution.

Time Complexity of Algorithm 4. Applying repairs to D requires $O(|D|^2 \cdot \text{rate}_i)$ per repair operation. Over k iterations, we have $\sum_{i=1}^{k}(\text{rate}_i \cdot |B_i|) \approx |D_{\text{error}}|$, yielding total complexity $O\big(|B|^2 \cdot |D_{\text{error}}|^2 \cdot |Op| + |D| \cdot |D_{\text{error}}|\big)$. As $|D_{\text{error}}|$ diminishes with iterations, the ideal core set $|B| = \sum |B_i|$ should minimally cover all dirty data. Under this optimality condition, complexity reduces to $O(|D_{\text{error}}|^4 \cdot |Op| + |D| \cdot |D_{\text{error}}|)$. In practical execution, due to optimized sampling block sizes, the introduction of additional knowledge bases, and a gradual decrease in error rate across rounds, the complexity tends towards the optimal case.

Recall the time complexity of $O(|D_{\text{error}}|! \cdot |Op| \cdot |D|^2)$ for the cleaning problem mentioned in Section 1, our proposed **UniClean** significantly reduces the computational cost of data cleaning. This demonstrates its advantage for large-scale data cleaning tasks.

Example 4.5. *As shown in Figure 3, after the partitioning, candidate repairs are generated per block. In Block 1, l_1 is selected over l_2 based on lower Q values. Similarly, in Block 2, repair l_3 is selected as it yields the best improvement in Q among available candidates. Once selection is completed, suboptimal repairs (l_2) and outdated or conflicting repairs (l_4) are discarded. Finally, the selected repairs are analyzed to trace their origins. All chosen repairs l_1 and l_3 are found to originate from the cleaner o_{c3}. As a result, the weight of the final cleaning block in CL is updated to prioritize o_{c3} in future iterations.*

4.3.4 *Validity of the Cleaning Objective Function.*

We discuss the validity of the cleaning objective proposed in this paper, namely the effectiveness of the cleaning operations executed with respect to the objective function Q, in Theorem 4.3.

Theorem 4.3 (Consistency of Q with Data Quality Improvement (CQD)). *The cleaning workflow CL generated by* ***UniClean****, when minimizing Q, ensures a* monotonic reduction in the expected number of data errors across workflow levels.

Algorithm 4: Iterative Cleaning Workflow (ICW)

```
Input: Input: Dataset $D$, Cleaner Set $Op$
Output: Cleaned Data $D'$, Cleaner Workflow $CL'$
$(G, CL) \leftarrow$ CleanerAssociation($D, Op$)// Algorithm 1
$W \leftarrow$ InitializeWeights($CL$), $Operations \leftarrow \emptyset$
foreach Cleaning Block $C_i \in CL$ do
    $S_i$, Quality$_i$, error$_t[i] \leftarrow$ CoreSet($D, C_i$)// Algorithm 2
    $B_i \leftarrow$ DataPartition($S_i, C_i$)// Algorithm 3
    CandFixes $\leftarrow$ GenerateFixes($B_i, C_i, Quality_i$, error$_\theta[i]$)
    OptimalFixes $\leftarrow$ FixSearch(CandFixes, $W[i], C_i, B_i$)
    if OptimalFixes $\neq \emptyset$ then
        $D \leftarrow$ ApplyFixes($D$, OptimalFixes)
        $Operations \leftarrow Operations \cup$ OptimalFixes
    else
        break;
    $W \leftarrow$ UpdateWeights($W[i]$, OptimalFixes)
if NoFurtherFixes($CL, D$) then
    break;
foreach Cleaning Block $C_i \in CL$ do
    Sort the cleaners in $C_i$ by $W[i]$ in descending order
return $D, CL$
```

Proof. Assume the following for each cleaner $o = [\texttt{Detect}, f] \in Op$: (A1) **Probabilistic coverage.** Every erroneous cell is detectable by at least one cleaner $o \in Op$ once its source attributes A_S are correct, and $\mathbb{E}[\texttt{Detect}_o(t) \mid A_S \text{ correct}] > \delta$, for some $\delta > 0$. (A2) **Reliable repair in expectation.** If A_S is correct, the expected repair output $f(A_S)$ is close to the true value A_T^*, i.e., $\mathbb{E}[\texttt{Loss}(f(A_S), A_T^*)] \leq \epsilon$, for a small $\epsilon > 0$. (A3) **Topological execution.** Cleaners are applied level by level according to the attribute dependency order. We assume that all first-layer attributes that do not depend on other attributes (i.e., primary key attributes in D) are initially correct.

We then analyze how the expected number of remaining errors evolves across workflow levels. At level i, we express the total expected errors as:

$$\text{Num}_{\text{error}}^{(i)} = \sum_{o \in Op} \sum_{t \in D} \texttt{Detect}_o(t \mid A_S \text{ is correct at level } i).$$

This term represents the cumulative detection scores for tuples whose source attributes A_S have been correctly repaired by level i. Under assumptions (A1)–(A3), each repair at level i strictly reduces the number of detectable errors in subsequent levels. In particular, for any cleaner $o_j \in \text{level}_i$ ($i \geq 2$), its dependent attributes A_{S_j} have already been repaired by cleaners in level_{i-1} or earlier. Thus, we have

$$\mathbb{E}[\text{Num}_{\text{error}}^{(i)}] \leq \mathbb{E}[\text{Num}_{\text{error}}^{(i-1)}],$$

with the inequality being strict unless no fixable errors remain at that stage. This ensures the global error trend decreases as the cleaning workflow progresses.

Consider the objective function Q in Section 3.2, where $w_o > 0$ is the cleaner-specific weight. Since $\texttt{Detect}_o(t)$ approximates the probability that t is erroneous under cleaner o, and Q aggregates these scores across CL, we have $\mathbb{E}[Q] \propto \mathbb{E}[\text{Num}_{\text{error}}^{(k)}]$, implying that minimizing $Q(CL)$ is equivalent to minimizing the expected number of remaining data errors.

Therefore, minimizing Q over CL guarantees a *monotonic improvement in expected data quality across levels.* Note that this monotonicity holds in expectation. Individual tuple errors may fluctuate, but the global error trend decreases consistently. □

5 EXPERIMENTAL EVALUATION

5.1 Experimental settings

Datasets. We evaluated our approach on six standard data cleaning benchmark datasets and one large enterprise dataset (Table 2). These datasets exhibit real-world error patterns including missing values (MV) [46], typo outliers (T) [31, 44], attribute dependency violations (VAD) [36], and format inconsistencies (FI) [6]. **C-Rate** and **R-Rate** denote cell-level and record-level error rates respectively.

Table 2: Summary of datasets.

Datasets	#row	#col	C-Rate,%	R-Rate, %	Error Types
Hospital	1,000	19	2.67	40.70	T, VAD
Flights	2,376	7	34.51	80.13	MV, FI, VAD
Beers	2,410	11	13.93	100.00	MV, FI, VAD
Rayyan	1,000	11	9.03	78.40	MV, T, FI, VAD
Tax	200,000	15	0.10	1.46	T, FI, VAD
Soccer	200,000	10	1.56	15.64	T, VAD
Commercial	40,000,000	107	6.76	84.43	MV, T, VAD

Comparison Methods. Besides our **UniClean**, we evaluate five mainstream systems: **Holistic**, **BigDansing**, **Horizon**, **Baran**, and **HoloClean**. These cover the five cleaning strategies in Section 2 and are summarised in Table 3. All baselines are executed with their *official open-source* implementations and *default* hyper-parameters. We prepare each system with the necessary pre-configurations (e.g., various rules and ground-truth labels GT) as required.

Metrics. Five kinds of metrics are used to evaluate the cleaning performance: (1) $P = \frac{\#\text{correctRepCells}}{\#\text{RepCells}}$ represents the ratio of correctly modified errors to the total number of repairs, $R = \frac{\#\text{correctRepCells}}{\#\text{ErrCells}}$ denotes the ratio of correctly modified errors to the total number of errors, $F1 = \frac{2PR}{P+R}$. (2) *Hybrid Distance* (*HD*)[43] quantifies the discrepancy between repaired and ground-truth values by integrating structured and unstructured attribute differences: $HD = w_1 \cdot \text{MSE} + w_2 \cdot \text{Jac\_dis}$, where MSE measures numerical deviation and Jac_dis captures categorical/textual divergence. Unless otherwise specified, we set $w_1 = w_2 = 0.5$ to ensure balanced treatment across attribute types. (3) *Error-Reduction Rate* (*EDR*) [43] measures the reduction of errors *from a cell perspective.* A cell is counted as *correct* only when its post-clean value exactly matches the ground truth; partial fixes are still regarded as errors. In addition, we propose two novel metrics specifically for evaluating mixed-error cleaning problems: (4) *Record Error Reduction Rate* (*REDR* = $\frac{\text{dis}_{d2c} - \text{dis}_{r2c}}{\text{dis}_{d2c}}$), where dis_{d2c} is the distance between dirty data and clean data, and dis_{r2c} is the distance between repaired and clean data. Different from the cell-level *EDR*, where each cell is independently evaluated, *REDR* assesses consistency at the *record* level: *any* remaining error in a single cell makes the entire record unusable. *REDR* highlights improvements in the number of completely error-free records, which is crucial for downstream tasks (e.g., ML model training) that require fully consistent inputs. (5) *Cleaning Time per 100 Records (s)* $\left(CTR = \frac{\text{Time (s)}}{\#\text{Records}}\right)$ evaluates cleaning efficiency, expressed as the processing time per hundred records, enabling an intuitive comparison across data scales.

Experimental reproducibility. UniClean is executed with *three* explicit inputs: (1) Dirty data D, (2) *Optional* rule files, and (3) *Cleaners.* Our public *Cleaner Library* contains a total of 73 unified cleaners, composed as follows:

$$\texttt{\#Cleaners} = \underbrace{(8+6)}_{\text{Base Cleaners}} \times \underbrace{5}_{\text{Knowledge}} + \underbrace{3}_{\text{Cost Function}} = \boxed{\text{73 types}} \quad (3)$$

Table 3: Summary of comparison data cleaning methods.

No.	Methods	Strategies	Target Error Types	Preconfig
1	**Holistic** [8]	❶	VAD	DC
2	**BigDansing** [35]	❶	VAD	DC
3	**Horizon** [48]	❶	VAD	FD
4	**Baran** [41]	❷❸	T, FI	GT
5	**HoloClean** [47]	❶❷❹	MV, FI, VAD	DC
6	**UniClean** (Ours)	❶❷❸❺	MV, T, FI, VAD	FD,RE

The base cleaners include 8 row-distribution-based cleaners (e.g., `Date`, `Float`, `Pattern`) and 6 column-rule-based cleaners (e.g., `CFD`, `DC`). These are optionally enhanced with 5 knowledge modules such as `DictValue` and `EditRule`, and evaluated using 3 cost models including `Jaccard` and `EditDistance`. All datasets, cleaners, rules, injection scripts, and execution logs are available at our GitHub.

Implementation. We run experiments under macOS (M2, 16GB RAM), Linux Ubuntu 18.04.6 LTS (Intel Xeon Silver 4210R CPU, 40 cores, 500GB RAM). Note that we also implement a visualization interface for **UniClean**. We note that **UniClean**'s demo version is recently accepted in [16], and the video is available at [17].

5.2 Performance evaluation on native errors

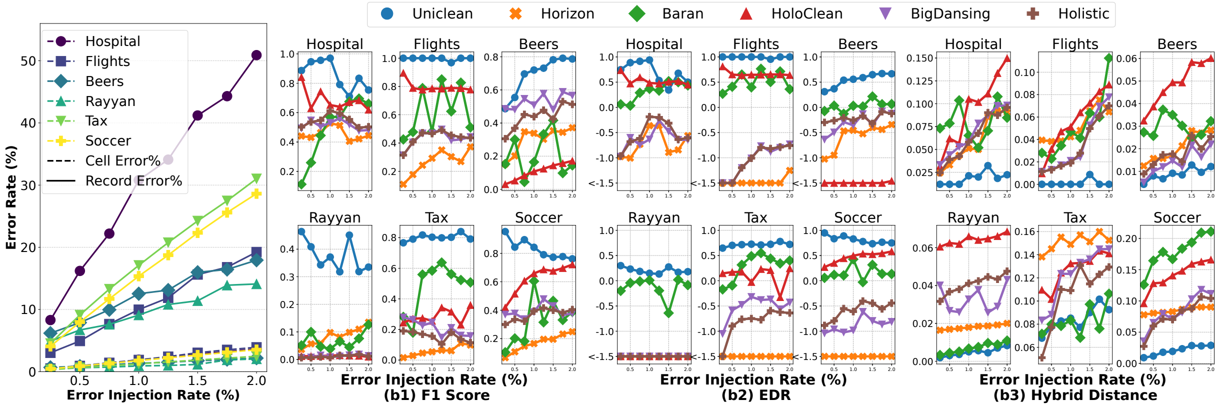
We evaluate the adaptive cleaning capabilities on datasets with native errors, as reported in Table 4. Due to the inability of many baselines to complete within 24 hours on the large-scale **Tax** and **Soccer** datasets, we report the results for each algorithm on a relatively extreme data volume. The ranking of each method on each metric is indicated by circles following the specific scores.

Table 4: Overall performance comparison on datasets.

Datasets	Algorithms	F1 Score ↑	EDR ↑	HD ↓	REDR ↑	CTR(s) ↓
Hospital	UniClean	**0.8847①**	**0.7839①**	**0.0521①**	**0.7543①**	10.8115②
	Horizon	0.5841⑤	0.0570④	0.0974②	0.0270⑤	**0.3245①**
	Baran	0.5753⑥	0.4165③	0.1304⑤	0.3612③	43.1077⑤
	HoloClean	0.6262②	0.4558②	0.1523⑥	0.4324②	15.0942③
	BigDansing	0.6050④	-0.0766⑥	0.1239④	0.0221⑥	23.2969④
	Holistic	0.6080③	-0.0236⑤	0.1157③	0.0442④	105.3257⑥
Flights	UniClean	**0.6537①**	**0.5175①**	**0.0953①**	**0.1129①**	3.5603③
	Horizon	0.4049⑤	0.1148④	0.1782③	-0.1534⑥	**1.4109①**
	Baran	0.6278②	0.4478②	0.1231②	0.0326③	19.3508④
	HoloClean	0.4763③	0.3508③	0.2018⑥	0.0604②	1.9327②
	BigDansing	0.3870⑥	-0.1382⑥	0.1999⑤	-0.0693⑤	2694.6635⑥
	Holistic	0.4067④	-0.1191⑤	0.1944④	-0.0636④	231.8536⑤
Beers	UniClean	**0.8373①**	**0.8329①**	**0.0306①**	**0.7730①**	1.2966②
	Horizon	0.1051③	0.0027③	0.0794③	0.0000③	1.4270③
	Baran	0.7976②	0.7868②	0.0538②	0.7224②	24.7956⑤
	HoloClean	0.0535⑥	-0.1704⑥	0.0942⑥	0.0000③	9.6334④
	BigDansing	0.0940④	-0.0104④	0.0822④	0.0000③	**1.2669①**
	Holistic	0.0939⑤	-0.0113⑤	0.0823⑤	0.0000③	65.4283⑥
Rayyan	UniClean	**0.9213①**	**0.9005①**	**0.0078①**	**0.8827①**	5.2378②
	Horizon	0.0091③	-0.5281③	0.0589③	-0.1918③	**2.3133①**
	Baran	0.2983②	0.1757②	0.0458②	0.1686②	27.3773④
	HoloClean	0.0088④	-1.9204⑤	0.1070⑥	-0.2258⑥	10.8541③
	BigDansing	0.0000⑥	-1.3667④	0.0688④	-0.1699⑤	83.3452⑤
	Holistic	0.0006⑤	-2.0654⑥	0.0807⑤	-0.1956④	2017.2680⑥
Tax	UniClean(**200k**)	**0.5011①**	**0.1005①**	0.0018②	**0.4250①**	25.1793③
	Horizon(**10k**)	0.0073⑤	-87.5904⑥	0.0943⑥	-57.1333⑥	**11.1435①**
	Baran(**10k**)	0.0288④	0.0181②	**0.0012①**	0.0182②	38.0591④
	HoloClean(**10k**)	0.0000⑥	-0.6687③	0.0057③	-0.6545⑤	12.4377②
	BigDansing(**10k**)	0.0855③	-1.2640⑤	0.0144④	-0.0271④	103.8026⑤
	Holistic(**10k**)	0.0876②	-1.2427④	0.0241⑤	-0.0353③	574.0921⑥
Soccer	UniClean(**200k**)	**0.5341①**	**0.3301①**	0.0354②	**0.3442①**	**0.0343①**
	Horizon(**200k**)	0.3338③	-2.9908⑥	0.0443④	-2.9908⑥	0.1834②
	Baran(**10k**)	0.3276②	0.2338②	**0.0071①**	0.2338②	23.7848③
	HoloClean(**10k**)	N/A	0.0000③	0.0135③	0.0000③	2.8439②
	BigDansing(**6k**)	0.4121④	-0.0858④	0.0509⑤	-0.0133④	973.3974④
	Holistic(**6k**)	0.4121⑤	-0.0858⑤	0.0504⑥	-0.0133⑤	230.0340⑤

Summary. Table 4 shows that **UniClean** achieves the best cleaning effectiveness in most cases and demonstrates superior efficiency.

- **Comprehensive data quality improvement. UniClean** effectively addresses mixed errors in real-world data, consistently outperforming state-of-the-art baselines across both *F1* and *REDR.* In particular, *REDR* reflects a more realistic evaluation of record-level usability, which better aligns with the needs of data management and downstream analysis tasks than traditional cell-level metrics.



(a) Record vs. Cell Error **(b) Performance comparison of methods on three datasets.**

Figure 4: Experiments under 8 different error injection ratios with mixed error types.

UniClean achieves over 40% *REDR* improvement compared to the best-performing baselines on all datasets except **Beers**, indicating its superior capability in enhancing holistic record correctness.

- **Large-scale data processing capability. UniClean** demonstrates strong scalability by efficiently cleaning large datasets such as **Tax** and **Soccer**, processing hundreds of thousands of records within hours without truncation. It achieves *REDR* and *F1* scores exceeding 0.3 and 0.5, respectively, substantially outperforming most baselines, which are typically limited to thousands of records and prone to higher error rates.

We note that our reproduced baseline results align with a recent study [43]. Scores in original basline papers often rely on per-dataset hyperparameter tuning, and this limits comparability under a unified, fully reproducible evaluation protocol.

5.3 Experiments for mixed error injection

To assess the robustness of cleaning methods under diverse error patterns, we inject multiple error types into real-world datasets using BART [4]. Following an attribute-level independent injection strategy, each non-key attribute receives each error type with a fixed probability $r_e \in [0.25\%, 2\%]$ per cell. The expected number of erroneous cells is $m \cdot n \cdot k \cdot r_e$, where m is the number of attributes, n the number of records, and k the number of error types. As n increases, errors tend to occur on distinct tuples, yielding an approximate upper bound for the record-level error rate: $RecordErrorRate \approx m \cdot k \cdot r_e$. As illustrated in Figure 4a, the resulting record-level error rates range from 8% to 30% across datasets, exceeding the native error levels and reflecting realistic yet challenging dirty data scenarios.

Summary. Datasets with more attributes and constraints are more susceptible to quality degradation under error injection, due to increased opportunities for error propagation and rule violations. Record-level error rates escalate more rapidly and reach higher levels than cell-level rates, as a single erroneous cell can compromise an entire record. This disparity underscores the compounding effect of inter-attribute dependencies, where errors in one field can cascade through related attributes, amplifying their impact on data integrity and usability.

5.4 Evaluation under Varying Parameters

5.4.1 Performance with varying error rates. Figure 4b compares all methods across 8 error injection rates, and Table 5 provides detailed results for our proposed *REDR* and *CTR* metrics.

Summary. **UniClean** demonstrates stable and effective performance across diverse error modes, consistently outperforming baselines whose cleaning quality degrades significantly as error rates increase. Its dependency-graph topological scheduling and blockwise repair guarantee a *monotone* global-error reduction (Theorem 4.3), and the learned workflows can be directly reused across different error distributions without retraining, achieving strong transferability. In scalability, **UniClean** efficiently processes 10^6–10^7 tuples within hours, whereas baseline systems often fail beyond 10K–20K records. Although **UniClean** may not always achieve the fastest *CTR* on small datasets, faster baselines typically suffer from much higher residual errors, as reflected by their significantly worse *EDR* and *REDR* scores. In most cases, **UniClean** surpasses the best baseline by over 20% in overall cleaning quality, and consistently achieves the best *REDR* across all datasets and error rates, demonstrating superior stability and robustness.

5.4.2 Ablation study on cleaning preparation strategy. To assess cleaning preparation impacts in **UniClean**, two experiments are conducted: (*i*) **Small-scale data**. Four datasets (**Hospital**, **Flights**, **Beers**, **Rayyan**) are tested without preparation strategies (Section 4.2), using baseline attribute partitioning and direct repairs (Fig. 5a). (*ii*) **Large-Scale data**. Public datasets are scaled to 1M records, with additional tests on a **40M**-record commercial dataset (Table 6).

Summary. Figure 5a shows that, without preparation strategies, the repair operation generation and selection process substantially increase the search space, leading to redundant operations, especially in complex datasets like **Rayyan**. **UniClean** achieves hour-level cleaning speeds on expanded datasets (millions of records), with under 10% performance deviation compared to small datasets. On commercial data with tens of millions of records, **UniClean** leverages multiple cleaners to complete the cleaning tasks within hours, fulfilling large-scale enterprise processing demands.

5.4.3 Ablation study on cleaning workflow optimization. The cleaner execution order is generated based on the original errors by **UniClean** and applied to all datasets with varying error ratios. As a comparison, cleaners were executed independently in a greedy order on the same dataset. The results are shown in Figure 5b.

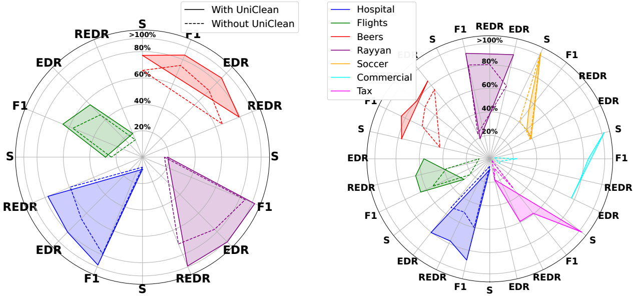
Summary. **UniClean**'s optimized cleaning order effectively coordinates dependencies among cleaners, significantly reducing redundant repairs and conflicting operations. Results align with the

Table 5: Performance comparison across our proposed metrics under 8 error injection ratios on real-world datasets.

Datasets	Methods	REDR ↑								CTR(s/100 Records) ↓							
		0.25%	0.5%	0.75%	1%	1.25%	1.5%	1.75%	2%	0.25%	0.5%	0.75%	1%	1.25%	1.5%	1.75%	2%
Hospital	UniClean	0.75	0.88	0.91	0.93	0.79	0.77	0.84	0.81	14.98	15.25	13.24	13.80	14.00	13.41	16.13	15.91
	Horizon	-0.01	-0.35	-0.03	0.40	0.42	-0.14	0.09	0.15	0.25	0.25	0.25	0.25	0.25	0.26	0.26	0.26
	Baran	0.06	0.00	0.28	0.33	0.26	0.47	0.44	0.34	33.56	41.65	40.40	40.26	41.65	40.85	40.32	41.20
	HoloClean	0.73	0.46	0.58	0.48	0.44	0.50	0.48	0.39	15.69	14.79	15.06	14.58	13.99	14.29	13.86	15.98
	BigDansing	-0.18	-0.08	-0.07	-0.01	-0.02	-0.01	-0.02	-0.00	22.10	32.45	24.63	23.13	34.72	29.10	32.67	29.70
	Holistic	-0.19	-0.14	-0.07	-0.04	-0.02	-0.01	-0.01	0.00	147.43	197.16	187.46	187.42	190.15	191.45	221.49	214.45
Flights	UniClean	1.00	1.00	1.00	1.00	1.00	0.95	1.00	1.00	2.51	2.60	2.83	2.87	3.11	2.48	2.54	2.50
	Horizon	-9.33	-5.73	-3.46	-2.45	-2.08	-1.60	-1.73	-1.16	0.13	0.14	0.14	0.14	0.14	0.14	0.14	0.15
	Baran	0.26	0.36	0.63	0.36	0.72	0.46	0.70	0.36	11.01	12.19	12.48	12.19	12.17	12.34	12.28	12.80
	HoloClean	0.82	0.71	0.69	0.71	0.76	0.73	0.76	0.74	3.51	3.47	3.41	3.50	3.17	3.29	3.04	3.00
	BigDansing	-1.60	-0.97	-0.60	-0.45	-0.36	-0.29	-0.27	-0.21	5.47	5.46	5.52	5.74	5.78	5.72	5.90	6.02
	Holistic	-1.60	-0.97	-0.60	-0.45	-0.38	-0.29	-0.27	-0.21	150.62	148.09	147.34	148.69	147.84	149.48	150.75	150.80
Beers	UniClean	0.35	0.39	0.55	0.57	0.58	0.66	0.65	0.64	2.32	2.32	2.19	3.08	2.66	2.57	2.63	2.72
	Horizon	-0.49	-0.43	-0.10	-0.08	-0.14	-0.01	-0.03	-0.03	1.35	1.39	1.37	1.39	1.40	1.41	1.48	1.44
	Baran	-0.07	-0.03	-0.10	0.04	-0.03	0.20	0.07	0.04	23.47	23.53	23.82	24.62	24.43	23.38	23.91	24.08
	HoloClean	-5.48	-4.18	-3.13	-2.35	-2.21	-1.71	-1.64	-1.45	9.16	8.92	8.85	8.70	8.69	8.86	8.57	8.55
	BigDansing	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.53	1.54	1.58	1.40	1.58	1.40	1.47	1.71
	Holistic	-0.14	-0.14	-0.11	-0.11	-0.08	-0.10	-0.09	-0.09	122.52	120.34	122.23	121.86	121.94	120.41	122.15	121.42
Rayyan	UniClean	0.29	0.25	0.13	0.14	0.10	0.22	0.13	0.10	2.27	2.34	3.18	4.83	8.20	9.43	4.47	4.60
	Horizon	-10.53	-7.54	-6.58	-5.29	-4.62	-4.18	-3.41	-3.16	0.35	0.36	0.36	0.36	0.30	0.37	0.33	0.36
	Baran	-0.18	-0.04	-0.01	0.01	-0.20	-0.70	0.02	-0.08	39.85	30.16	32.65	33.59	33.72	26.19	22.02	28.99
	HoloClean	-19.02	-13.63	-11.92	-9.77	-8.33	-7.64	-6.32	-5.92	9.81	9.98	9.43	11.29	9.71	9.95	11.42	12.15
	BigDansing	-47.27	-19.37	-14.67	-9.94	-9.09	-6.96	-5.58	-5.37	255.70	275.95	300.32	275.21	261.89	263.52	265.38	245.68
	Holistic	-46.67	-19.80	-14.78	-10.92	-8.92	-7.73	-6.13	-5.45	3062.12	2091.99	2778.68	2693.08	1892.64	1963.02	2430.18	2679.38
Tax	UniClean(200k)	0.66	0.69	0.70	0.71	0.69	0.70	0.76	0.72	1.43	1.98	0.72	0.68	0.72	0.78	1.77	0.97
	Horizon(10k)	-19.97	-10.11	-4.81	-4.67	-3.04	-3.28	-1.90	-2.25	12.48	13.13	12.42	13.12	12.89	12.51	13.14	12.43
	Baran(10k)	-0.16	-0.09	0.29	0.46	0.52	0.41	0.31	0.35	39.49	37.72	37.74	39.52	39.85	38.42	41.76	42.79
	HoloClean(5k)	0.14	0.17	0.16	-0.01	0.20	0.19	-0.36	0.20	22.19	20.37	21.62	21.64	20.46	20.68	21.10	20.91
	BigDansing(2k)	-0.48	-0.25	-0.26	-0.15	-0.22	-0.15	-0.27	-0.17	5.35	5.37	54.29	193.82	267.31	188.78	65.16	167.46
	Holistic(2k)	-0.81	-0.33	-0.37	-0.31	-0.32	-0.22	-0.26	-0.26	25.53	25.32	26.26	25.64	25.44	25.73	25.37	25.55
Soccer	UniClean(200k)	0.94	0.90	0.92	0.90	0.84	0.81	0.79	0.77	0.08	0.09	0.09	0.10	0.10	0.11	0.11	0.12
	Horizon(200k)	-18.15	-8.80	-5.79	-4.23	-3.26	-2.64	-2.20	-1.84	0.75	0.80	0.77	0.78	0.82	0.93	0.92	1.00
	Baran(10k)	0.06	0.11	0.11	0.39	-0.09	0.30	0.13	0.11	24.63	24.77	24.31	25.04	25.08	25.09	25.10	25.69
	HoloClean(5k)	0.26	0.34	0.46	0.49	0.56	0.51	0.55	0.59	5.76	5.36	4.82	4.82	3.95	4.47	4.37	4.60
	BigDansing(2k)	-0.11	-0.10	-0.03	-0.04	-0.02	-0.02	-0.05	-0.05	34.55	36.58	91.53	32.53	34.74	40.11	66.53	83.74
	Holistic(2k)	-0.09	-0.11	-0.04	-0.08	-0.09	-0.03	-0.08	-0.05	36.10	36.09	35.50	36.45	37.45	38.61	37.31	37.92

Table 6: Performance on scaled datasets with cleaning preparation strategies in the proposed UniClean.

Datasets	F1 ↑	EDR ↑	REDR ↑	Time (hours)
Hospital (1M)	0.87	0.77	0.74	0.40
Flights (1M)	0.64	0.51	0.10	0.25
Beers (1M)	0.83	0.82	0.76	0.21
Rayyan (1M)	0.91	0.89	0.87	0.23
Tax (1M)	0.49	-0.04	0.41	3.50
Soccer (1M)	0.52	0.32	0.33	0.79
Commercial (40M)	0.84	0.70	0.77	3.10



(a) Evaluation of cleaning preparation on small datasets

(b) Evaluation of cleaner execution workflows on all datasets

Figure 5: Performance comparison before and after cleaning strategy ablation.

simulation in Figure 1, i.e., compared to greedy strategies, **UniClean** generates more accurate repair operations (faster) while avoiding error propagation (better performance). Importantly, the workflow optimised on the *original error distribution* retains over 80% of its original *F1*, *EDR*, and *REDR* across datasets with various increased error rates, whereas baseline systems require retraining or full re-execution of the cleaning process to maintain reasonable performance. This confirms the transferability and robustness of **UniClean**'s cleaning workflow compared to unordered multi-cleaner strategies. In particular, on datasets with complex dependency structures, **UniClean** improves *F1*, *EDR*, and *REDR* by over 30%, while reducing the overall cleaning time for large-scale datasets by more than half. By generating the optimal execution order based on error distributions, **UniClean** ensures low-cost migration to datasets with varying error versions.

6 CONCLUSION

This paper introduces **UniClean**, a system for large-scale mixed-error cleaning through three integrated stages: unified cleaner construction, efficiency-optimized preparation, and workflow execution. It incorporates core-set extraction and quality-driven optimizations to reduce computational costs while preserving cleaning efficacy. Theoretical guarantees are provided for cleaner completeness, preparation independence, and objective function effectiveness. Experiments on real-world datasets show **UniClean** outperforms state-of-the-art methods across metrics, achieving >30% *F1*/*EDR*/*REDR* improvement on complex data with 1M+ tuples, while completing cleaning within hours.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China (2023YFB3308003), the National Natural Science Foundation of China (NSFC) (62202126, 62232005, 92267203), and the Natural Science Foundation Project of Heilongjiang Province of China (YQ2024F005).

REFERENCES

[1] Mohamed Abdelaal, Christian Hammacher, and Harald Schöning. 2023. REIN: A Comprehensive Benchmark Framework for Data Cleaning Methods in ML Pipelines. In *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*, Julia Stoyanovich, Jens Teubner, Nikos Mamoulis, Evaggelia Pitoura, Jan Mühlig, Katja Hose, Sourav S. Bhowmick, and Matteo Lissandrini (Eds.). OpenProceedings.org, 499–511. https://doi.org/10.48786/EDBT.2023.43

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

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases.* Addison-Wesley.

[4] 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. *Proc. VLDB Endow.* 9, 2 (2015), 36–47. https://doi.org/10.14778/2850578.2850579

[5] George Beskales, Ihab F. Ilyas, Lukasz Golab, and Artur Galiullin. 2013. On the relative trust between inconsistent data and inaccurate constraints. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou (Eds.). IEEE Computer Society, 541–552. https://doi.org/10.1109/ICDE.2013.6544854

[6] Xu Chu and Ihab F. Ilyas. 2016. Qualitative Data Cleaning. *Proc. VLDB Endow.* 9, 13 (2016), 1605–1608. https://doi.org/10.14778/3007263.3007320

[7] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Discovering Denial Constraints. *Proc. VLDB Endow.* 6, 13 (2013), 1498–1509. https://doi.org/10.14778/2536258.2536262

[8] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou (Eds.). IEEE Computer Society, 458–469. https://doi.org/10.1109/ICDE.2013.6544847

[9] Carolina Cortes, Camila Sanz, Lorena Etcheverry, and Adriana Marotta. 2024. Data Quality Management for Responsible AI in Data Lakes. In *Proceedings of Workshops at the 50th International Conference on Very Large Data Bases, VLDB 2024, Guangzhou, China, August 26-30, 2024*. VLDB.org. https://vldb.org/workshops/2024/proceedings/TaDA/TaDA.13.pdf

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

[11] Xiaoou Ding, Genglong Li, Hongzhi Wang, Chen Wang, and Yichen Song. 2024. Time Series Data Cleaning Under Expressive Constraints on Both Rows and Columns. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*. IEEE, 3682–3695. https://doi.org/10.1109/ICDE60146.2024.00283

[12] Xiaoou Ding, Genglong Li, Hongzhi Wang, Chen Wang, and Yichen Song. 2024. Time Series Data Cleaning under Expressive Constraints on Both Rows and Columns. In *40th IEEE International Conference on Data Engineering, ICDE*. IEEE.

[13] Xiaoou Ding, Yingze Li, Hongzhi Wang, Chen Wang, Yida Liu, and Jianmin Wang. 2024. TSDDISCOVER: Discovering Data Dependency for Time Series Data. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*. IEEE, 3668–3681. https://doi.org/10.1109/ICDE60146.2024.00282

[14] Xiaoou Ding, Yida Liu, Hongzhi Wang, Chen Wang, Yichen Song, Donghua Yang, and Jianmin Wang. 2024. Efficient Relaxed Functional Dependency Discovery with Minimal Set Cover. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*. IEEE, 3519–3531. https://doi.org/10.1109/ICDE60146.2024.00271

[15] Xiaoou Ding, Yixing Lu, Hongzhi Wang, Chen Wang, Yida Liu, and Jianmin Wang. 2024. DAFDiscover: Robust Mining Algorithm for Dynamic Approximate Functional Dependencies on Dirty Data. *Proc. VLDB Endow.* 17, 11 (2024), 3484–3496. https://doi.org/10.14778/3681954.3682015

[16] Xiaoou Ding, Zekai Qian, Hongzhi Wang, Zhe Sun, Siying Chen, Hongbin Su, and Huan Hu. 2025. UniClean: A Multi-Signal Fusion Pipeline for Optimizing Data Cleaning Workflow. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 4600–4603.

[17] Xiaoou Ding, Zekai Qian, Hongzhi Wang, Zhe Sun, Siying Chen, Hongbin Su, and Huan Hu. 2025. *UniClean Demo (ICDE Demo 2025).* https://youtu.be/BGYHj0gMN2g Video demonstration.

[18] Xiaoou Ding, Yichen Song, Hongzhi Wang, Chen Wang, and Donghua Yang. 2024. MTSClean: Efficient Constraint-based Cleaning for Multi-Dimensional Time Series Data. *Proc. VLDB Endow.* 17, 13 (2024), 4840–4852. https://www.vldb.org/pvldb/vol17/p4840-wang.pdf

[19] Xiaoou Ding, Yichen Song, Hongzhi Wang, Donghua Yang, Chen Wang, and Jianmin Wang. 2024. Clean4TSDB: A Data Cleaning Tool for Time Series Databases. *Proc. VLDB Endow.* 17, 12 (2024), 4377–4380. https://doi.org/10.14778/3685800.3685879

[20] Xiaoou Ding, Hongzhi Wang, Jiaxuan Su, Zijue Li, Jianzhong Li, and Hong Gao. 2019. Cleanits: A Data Cleaning System for Industrial Time Series. *Proc. VLDB Endow.* 12, 12 (2019), 1786–1789. https://doi.org/10.14778/3352063.3352066

[21] Xiaoou Ding, Hongzhi Wang, Jiaxuan Su, Muxian Wang, Jianzhong Li, and Hong Gao. 2022. Leveraging Currency for Repairing Inconsistent and Incomplete Data. *IEEE Trans. Knowl. Data Eng.* 34, 3 (2022), 1288–1302. https://doi.org/10.1109/TKDE.2020.2992456

[22] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *Proc. VLDB Endow.* 11, 11 (2018), 1454–1467. https://doi.org/10.14778/3236187.3236198

[23] Equifax. 2022. *Equifax Statement on Recent Coding Issue.* https://www.equifax.com/newsroom/all-news/-/story/equifax-statement-on-recent-coding-issue/ Accessed: 2025-01-01.

[24] Wenfei Fan and Floris Geerts. 2012. *Foundations of Data Quality Management.* Morgan & Claypool Publishers. https://doi.org/10.2200/S00439ED1V01Y201207DTM030

[25] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33, 2 (2008), 6:1–6:48. https://doi.org/10.1145/1366102.1366103

[26] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. 2009. Reasoning about Record Matching Rules. *Proc. VLDB Endow.* 2, 1 (2009), 407–418. https://doi.org/10.14778/1687627.1687674

[27] Anna Fariha, Ashish Tiwari, Alexandra Meliou, Arjun Radhakrishna, and Sumit Gulwani. 2021. CoCo: Interactive Exploration of Conformance Constraints for Data Understanding and Data Cleaning. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2706–2710. https://doi.org/10.1145/3448016.3452750

[28] Congcong Ge, Yunjun Gao, Xiaoye Miao, Bin Yao, and Haobo Wang. 2021. A Hybrid Data Cleaning Framework Using Markov Logic Networks (Extended Abstract). In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 2344–2345. https://doi.org/10.1109/ICDE51399.2021.00258

[29] Stella Giannakopoulou, Manos Karpathiotakis, and Anastasia Ailamaki. 2020. Cleaning Denial Constraint Violations through Relaxation. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 805–815. https://doi.org/10.1145/3318464.3389775

[30] Shubha Guha, Falaah Arif Khan, Julia Stoyanovich, and Sebastian Schelter. 2023. Automated Data Cleaning Can Hurt Fairness in Machine Learning-based Decision Making. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 3747–3754. https://doi.org/10.1109/ICDE55515.2023.00303

[31] D. M. Hawkins. 1980. *Identification of Outliers.* Springer. https://doi.org/10.1007/978-94-015-3994-4

[32] 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, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 829–846. https://doi.org/10.1145/3299869.3319888

[33] Edwin T Jaynes. 1982. On the rationale of maximum-entropy methods. *Proc. IEEE* 70, 9 (1982), 939–952.

[34] Rui Kang, Shaoxu Song, and Chaokun Wang. 2022. Conditional Regression Rules. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2481–2493. https://doi.org/10.1109/ICDE53745.2022.00231

[35] Zuhair Khayyat, Ihab F. Ilyas, Alekh Jindal, Samuel Madden, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2015. BigDansing: A System for Big Data Cleansing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). ACM, 1215–1230. https://doi.org/10.1145/2723372.2747646

[36] Solmaz Kolahi and Laks V. S. Lakshmanan. 2009. On approximating optimum repairs for functional dependency violations. In *Database Theory - ICDT 2009, 12th International Conference, St. Petersburg, Russia, March 23-25, 2009, Proceedings (ACM International Conference Proceeding Series, Vol. 361)*, Ronald Fagin (Ed.). ACM, 53–62. https://doi.org/10.1145/1514894.1514901

[37] Dimitrios Koutsoukos, Renato Marroquín, Ingo Müller, and Ana Klimovic. 2025. Adaptive Data Transformations for QaaS. In *Conference on Innovative Data Systems Research (CIDR)*. https://vldb.org/cidrdb/papers/2025/p20-koutsoukos.pdf

[38] 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 *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021.* IEEE, 13–24. https://doi.org/10.1109/ICDE51399.2021.00009

[39] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (2020), 50–60. https://doi.org/10.14778/3421424.3421431

[40] Yuyu Luo, Chengliang Chai, Xuedi Qin, Nan Tang, and Guoliang Li. 2020. VisClean: Interactive Cleaning for Progressive Visualization. *Proc. VLDB Endow.* 13, 12 (2020), 2821–2824. https://doi.org/10.14778/3415478.3415484

[41] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *Proc. VLDB Endow.* 13, 11 (2020), 1948–1961. http://www.vldb.org/pvldb/vol13/p1948-mahdavi.pdf

[42] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 865–882. https://doi.org/10.1145/3299869.3324956

[43] Wei Ni, Xiaoye Miao, Xiangyu Zhao, Yangyang Wu, Shuwei Liang, and Jianwei Yin. 2024. Automatic Data Repair: Are We Ready to Deploy? *Proc. VLDB Endow.* 17, 10 (2024), 2617–2630. https://www.vldb.org/pvldb/vol17/p2617-miao.pdf

[44] Keith Ord. 1996. Outliers in statistical data. *International Journal of Forecasting* 12, 1 (1996), 175–176.

[45] Conor Power, Hiren Patel, Alekh Jindal, Jyoti Leeka, Bob Jenkins, Michael Rys, Ed Triou, Dexin Zhu, Lucky Katahanas, Chakrapani Bhat Talapady, Josh Rowe, Fan Zhang, Rich Draves, Ivan Santa, and Amrish Kumar. 2021. The Cosmos Big Data Platform at Microsoft: Over a Decade of Progress and a Decade to Look Forward. *Proc. VLDB Endow.* 14, 12 (2021), 3148–3161. https://doi.org/10.14778/3476311.3476390

[46] Zhixin Qi, Hongzhi Wang, Fanshan Meng, Jianzhong Li, and Hong Gao. 2017. Capture Missing Values with Inference on Knowledge Base. In *Database Systems for Advanced Applications - DASFAA 2017 International Workshops: BDMS, BDQM, SeCoP, and DMMOOC, Suzhou, China, March 27-30, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10179)*, Zhifeng Bao, Goce Trajcevski, Lijun Chang, and Wen Hua (Eds.). Springer, 185–194. https://doi.org/10.1007/978-3-319-55705-2_14

[47] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proc. VLDB Endow.* 10, 11 (2017), 1190–1201. https://doi.org/10.14778/3137628.3137631

[48] El Kindi Rezig, Mourad Ouzzani, Walid G. Aref, Ahmed K. Elmagarmid, Ahmed R. Mahmood, and Michael Stonebraker. 2021. Horizon: Scalable Dependency-driven Data Cleaning. *Proc. VLDB Endow.* 14, 11 (2021), 2546–2554. https://doi.org/10.14778/3476249.3476301

[49] Manasi S. 2021. *How to Improve Your Data Quality.* Gartner Research. https://www.gartner.com/smarterwithgartner/how-to-improve-your-data-quality [Accessed: 2025-01-01].

[50] Shaoxu Song, Aoqian Zhang, Jianmin Wang, and Philip S. Yu. [n. d.]. SCREEN: Stream Data Cleaning under Speed Constraints. In *SIGMOD*. 827–841.

[51] Motley Fool Transcribing. 2022. *Unity Software Inc. (U) Q1 2022 Earnings Call Transcript.* https://www.fool.com/earnings/call-transcripts/2022/05/11/unity-software-inc-u-q1-2022-earnings-call-transcr Accessed: 2025-01-01.

[52] Yihai Xi, Ning Wang, Xinyu Chen, Yiyi Zhang, Zilong Wang, Zhihong Xu, and Yue Wang. 2022. EasyDR: A Human-in-the-loop Error Detection&Repair Platform for Holistic Table Cleaning. *Proc. VLDB Endow.* 15, 12 (2022), 3578–3581. https://doi.org/10.14778/3554821.3554848

[53] Mohamed Yakout, Laure Berti-Équille, and Ahmed K. Elmagarmid. 2013. Don't be SCAREd: use SCalable Automatic REpairing with maximal likelihood and bounded changes. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias (Eds.). ACM, 553–564. https://doi.org/10.1145/2463676.2463706

[54] Haojun Zhang, Chengliang Chai, AnHai Doan, Paris Koutris, and Esteban Arcaute. 2020. Manually Detecting Errors for Data Cleaning Using Adaptive Crowdsourcing Strategies. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, Angela Bonifati, Yongluan Zhou, Marcos Antonio Vaz Salles, Alexander Böhm, Dan Olteanu, George H. L. Fletcher, Arijit Khan, and Bin Yang (Eds.). OpenProceedings.org, 311–322. https://doi.org/10.5441/002/EDBT.2020.28