UNIVERSITEIT VAN AMSTERDAM

VU VRIJE UNIVERSITEIT AMSTERDAM

MSc Computer Science

Big Data Engineering

Master Thesis

Efficient Table Error Detection with LLMs

Clustering-Based Representative Labeling without Manual Annotations

---

by

Yifeng Zhao

2807122

Supervisor:	Dr. Hazar Harmouch
Assessor:	Dr. Victoria Degeler
Date:	August 15th, 2025
Credits & Period:	120 ECTS, September 2023 - August 2025

Abstract

Error detection (ED) in tabular data is essential for ensuring data quality in modern data-driven applications. Traditional ED methods often rely on manual rules, statistical thresholds, or labeled data, making them resource-intensive and less adaptable to diverse datasets. While recent approaches with large language models (LLMs) are promising, they still require some supervision and are time-consuming to scale. In this work, we propose an efficient, minimally-supervised framework for tabular ED using open-source LLMs. Our method combines clustering-based representative sampling with zero-shot and batch prompting to identify data errors without manual annotations. Evaluation on real-world datasets shows that the method delivers competitive accuracy while significantly reducing reliance on human labeling.

Contents

1

Introduction

Data plays a fundamental role in modern data-driven applications, and ensuring its quality is critically important. Erroneous data not only hampers machine learning effectiveness [1], but also undermines a business's situational awareness, leading to poor decisions and missed opportunities [2], [3]. As a result, ED is recognized as a vital component of effective data preprocessing and wrangling workflows [4].

Structured tabular data, prevalent in fields such as finance, manufacturing and healthcare, exhibits a variety of error types, including pattern violations, typographical errors, rule breaches, and inconsistent units. Traditional ED approaches typically rely on hand-crafted rules, statistical outlier detection, or supervised learning. However, the demand on extensive domain knowledge and manual annotation makes them difficult to generalize across different datasets or schemas. To reduce labeling costs, some work explore semi-supervised and weak supervision methods, yet these approaches still rely on labeled seed data or domain heuristics.

More recently, LLMs have shown remarkable performance in various natural language processing (NLP), data analysis, code generation [5], question answering [6] and mathematic reasoning [7] tasks. Beyond these domains, LLMs have also been explored for data-centric tasks, such as schema matching, entity resolution, and text-based anomaly detection. Their capacity to integrate world knowledge and perform reasoning motivates growing interest in applying them to ED in tabular data.

In this work, we propose a novel ED framework designed to minimize human supervision by utilizing the generalization and inference capabilities of LLMs. Specifically, we explore whether small-scaled open-source LLMs can identify errors in tabular data through unsupervised or weakly supervised methods without the need to labeled training data. Our main contributions are listed below:

- A clustering-based representative sampling technique that selects diverse candidate data instances for LLM inspection based on extracted character-level, token-level and statistical features;
- A zero-shot prompting strategy tailored for ED, using task-specific instructions, rule-based guidance, and batch prompting to enhance LLM inference efficiency;
- A comprehensive empirical evaluation on real-world datasets, demonstrating the

effectiveness of our approach against existing baselines, and providing in-depth analysis of its limitations, challenges, and potential avenues for future improvement.

The thesis is structured as follows:

- Chapter 2 reviews relevant background and related work on tabular data ED, tokenization of tables, and the LLMs.
- Chapter 3 presents our proposed methodology, including feature extraction, clustering-based sampling, and LLM-based error labeling.
- Chapter 4 describes the experimental setup, and reports results from multiple benchmark datasets.
- Chapter 5 discusses limitations, potential improvements, and future directions.
- Finally, Chapter 6 concludes the thesis.

2

Background and Related Work

This chapter provides a brief introduction of key topics relevant to our work, including error detection, tokenization of tabular data, and the large language model.

2.1. Error Detection in Tabular Data

ED in structured data aims to identify values that are erroneous, inconsistent, or implausible with respect to the expected semantics, schema, or domain constraints. Such errors may arise from various sources, such as human input mistakes, system malfunctions, type conversion issues, or outdated information [8]. Common types of error include typographical errors, formatting issues, value outliers, and semantic inconsistencies.

2.1.1. Error Types

Data errors can be broadly classified into syntactic and semantic categories [9]. Syntactic errors violate data type or format constraints and are often easily detectable with rule-based methods. In contrast, semantic errors involve contextually incorrect but well-formed values, making them more challenging to identify due to requirement of a deeper understanding of data semantics or domain knowledge.

To illustrate the difference more concretely, we list some common examples of each error type as follow:

- **Syntactic Errors**. Errors related to the format or structure of the data.
    - **Typos / Misspellings**: Incorrectly spelled words or character transpositions (e.g., "New Yorrk" instead of "New York").
    - **Format / Pattern Violations**: Data not conforming to a specified pattern (e.g., an email address without an "@" symbol or a date in an unexpected format).
    - **Domain Constraint Violations**: Values falling outside an allowed range or set for the given attribute (e.g., age = -5).
- **Semantic Errors**. Errors related to the meaning or consistency of the data in relation to real-world values or expected data distributions.

- **Value Outliers**: A value outlier is a data point that significantly different from other observations in a dataset.
- **Rule violation**: Rule violation or inconsistencies refer to contradictory or conflicting information within a dataset.
- **Missing Values**: Absence of expected data, which can sometimes be considered an error if the value is mandatory.
- **Duplicates**: Multiple entries representing the same entity, leading to redundancy and potential inconsistencies.

Figure 2.1 illustrates the aforementioned error types using a sample table of board games. To be specific, the players field in the first row contains "2~5" instead of the correct format "2–5", representing a pattern violation. For the game *Chess*, the player count is listed as "222", which is implausibly high and constitutes a value outlier. Listing *Catan* as from "Italy" instead of its actual origin, "Germany", is an example of a semantic rule violation. The year for the game *Mahjong* is "3025", far outside the typical range of publication years, indicating a domain constraint violations. Furthermore, the game *Azul* lacks a specified designer, the game *Splendor* contains a typographical error ("Splendorr"). Lastly, *Ticket to Ride* and *Zug um Zug* are duplicated entries with identical entries across all fields.

Figure 2.1: Examples of different error types

These error types are closely tied to core dimensions of data quality commonly discussed in the literature [10]–[12]. Syntax errors like typos, pattern violations, and domain constraint violations primarily affect validity and accuracy by failing to conform to expected formats or ranges. Semantic errors, including value outliers, rule violations, missing values, and duplicates, span a wider set of dimensions. Although valid in format, value outliers are often inaccurate signals due to significant inconsistencies with the expected distribution. Rule violations introduce logical contradictions within the dataset, breaking internal consistency

and suggesting at least one value is incorrect. Missing values reflect incomplete information and directly reduce completeness. Moreover, duplicates violate uniqueness and may compromise consistency and accuracy by introducing conflicting information about the same entity. Table 2.1 summarizes how each type of error maps to specific data quality dimensions, highlighting their impact on overall data integrity.

Table 2.1: Mapping of common error types to data quality dimensions

Error Type	Impacted Data Quality Dimension(s)
Typos / Misspellings	Accuracy, Validity
Format / Pattern Violations	Validity
Domain Constraint Violations	Accuracy, Validity
Value Outliers	Accuracy, Consistency
Rule Violations	Accuracy, Consistency
Missing Values	Completeness
Duplicates	Accuracy, Consistency, Uniqueness

2.1.2. Traditional Approaches

Traditional approaches to ED in structured data primarily rely on expert knowledge, data constraints, metadata or statistic methods. These methods are interpretable and effective in domains where rules and data semantics are well-understood, though they often struggle with scalability and generalization.

Rule and Constraint
Rule-based and constraint-based approaches rely on predefined logic and dependencies that must hold true within the dataset. A widely used form is functional dependencies (FDs) [13], which assert that the value of one attribute uniquely determines the value of another. For example, a valid zip code value often dictates the corresponding city. These FDs can be further extended to conditional functional dependencies (CFDs) [14] that apply only under specific conditions, and matching dependencies (MDs) [15], [16] that capture similarity-based relationships for duplicate detection. Another important class is denial constraints (DCs) [17]–[20], which prohibit certain combinations of values across attributes or tuples.

These constraints are typically enforced through violation detection, where the system scans the dataset for records that do not comply with the specified rules. While these methods are effective at capturing logical inconsistencies and certain types of errors, their practical utility is limited by the need for comprehensive and accurate domain knowledge to manually define the constraints. To mitigate this, several systems [21]–[23] provide flexible framework that support both extensible rule definitions and automatic violation detection.

Pattern and Knowledge Base
Pattern-based methods assume that format consistency implies correctness. They often model each column using regex-style patterns. For example, the system might learn that phone numbers follow a "(XXX) XXX-XXXX" pattern, or postal codes follow 5-digit formats.

Violations of such learned or predefined patterns are flagged as potential errors [24], [25]. Knowledge-based approaches such as KATARA [26] use external knowledge graphs or linked data (e.g., DBpedia, Freebase) to verify factual correctness and enrich the detection process. These systems can detect semantic errors by checking if a value conflicts with known facts.

Statistical and Outlier Detection
Statistical methods treat error detection as a problem of distributional anomaly detection. The core assumption is that most correct data points follow a consistent distribution, while erroneous ones deviate significantly. For numerical data, techniques such as z-score, IQR-based thresholds, or density-based outlier detection (e.g., LOF, DBSCAN) are commonly used to identify outliers [27].

For categorical data, some approaches use frequency analysis or entropy-based metrics to identify low-frequency or unexpected values [19], [28], [29]. Others build probabilistic models or Bayesian networks over relational data to detect violations in co-occurrence likelihoods [30].

Limitations
While effective in well-understood domains, these approaches suffer from several limitations. Rule-based methods require significant human effort to encode rules, which may not generalize across datasets or domains. Outlier detection techniques are typically limited to numerical anomalies and perform poorly when semantic errors occur, such as mismatches between related fields. Pattern-based and constraint-based approaches assume that formatting regularity correlates with correctness, which may not hold in noisy or inconsistent datasets. As noted by [31], traditional methods are often only effective for a narrow subset of error types and may fail to capture nuanced or context-sensitive inconsistencies.

2.1.3. Model-based Methods

To address the scalability and generalization limitations of traditional approaches, model-based approaches have attracted a lot of attention in recent years. These methods often treat ED as a prediction task that a model must judge whether a cell or record is dirty or not.

Machine Learning
A typical class of methods leverage supervised learning based on feature engineering, where features capturing abnormal data patterns are constructed by rules, constraints, or statistical properties. , and serve to numerically represent potential errors for model training. These features numerically encode potential errors to train classifiers capable of distinguishing erroneous from normal values [32]–[34]. However, model performance is sensitive to the amount of labeled data available. To address this, data augmentation techniques have been employed to expand training sets [35], [36].

Besides, unsupervised methods also show effectiveness in scenarios with limited labeled data. Some approaches [9], [37] use clustering algorithms to group similar data instances, then assign labels to a small subset of samples and propagate these labels within each cluster to identify potential errors.

Sudowoodo [38] adopts a contrastive self-supervised learning framework, enabling deep neural networks to learn robust data representations for multi-task error detection and

integration without human supervision. RLclean [39], on the other hand, formulates ED and correction as a sequential decision-making process and applies deep reinforcement learning to iteratively identify and clean noisy sensor data.

Representation Learning
Autoencoders are neural networks designed to learn efficient data representations by encoding input data into a compressed latent space and then reconstructing the original input from this representation [40]. The key principle is that, when trained on predominantly clean data, the model learns to accurately reconstruct typical patterns. This property allows autoencoders to be applied for ED, as dirty or erroneous data can be treated as noise that deviates from the learned clean data distribution. Some studies [41], [42] have explored their potential for identifying anomalies and correcting errors by leveraging reconstruction loss as an anomaly score.

Unlike autoencoders, transformer architectures [43] leverage self-attention mechanisms to effectively model intricate dependencies and contextual relationships within sequential data. Therefore, transformer-based approaches [8], [44], [45] can automatically learn intricate row-column dependencies without relying on handcrafted features or labeled data, demonstrating potential for generalizable error identification and data clean tasks.

Pre-trained Language Model
Recent studies have explored various ways to leverage pre-trained LLMs for data ED. These methods can be broadly categorized into four groups based on their use of LLMs: few-shot learning, fine-tuning, code generation, and other hybrid or novel approaches.

Few-shot prompting allows LLMs to learn from a limited set of labeled examples by performing implicit comparisons, thereby facilitating the understanding and identification of data errors. This approach leverages the pretrained knowledge and strong contextual understanding capabilities of LLMs without requiring any task-specific fine-tuning. Recent research [46], [47] demonstrates that pre-trained LLMs can effectively perform core data cleaning tasks without fine-tuning. Building on these foundational insights, later studies [48]–[50] have further explored the semantic capabilities of LLMs in context-aware data preprocessing, such as structural understanding of tables, type inference, and data transformation.

Fine-tuning pre-trained LLMs allows the models to adapt to specific domains or tasks by incorporating supervision signals during training. In the context of ED, these signals may include direct supervision using labeled error instances [51], [52], instruction tuning to improve adherence to task-specific prompts [53], or learning structured dependencies among clean attribute values [54]. Such strategies have enabled even relatively simple LLM architectures to achieve strong ED performance after fine-tuning.

LLMs are capable of generating task-specific code based on natural language instructions and input data. By providing contextual information and data specifications, LLMs can produce executable logic for data processing tasks. Leveraging this capability, recent work [55]–[57] has explored using LLMs for ED via code generation. Similar to how human programmers operate, these approaches often adopt an iterative interaction framework, where the model can revise and refine the generated code based on execution results or feedback.

What's more, hybrid approaches leverage LLMs in combination with complementary techniques to enhance tabular data cleaning and ED. LLMClean [58] generates context-aware ontological functional dependencies using LLMs to capture semantic relationships within datasets, enabling effective data cleaning without manual context modeling. RetClean [59] integrates retrieval-augmented generation, combining LLMs with external data lakes to support error correction across both public and private datasets in an end-to-end manner. ZeroED [60] presents a hybrid zero-shot framework that fuses LLM reasoning with clustering-based sampling and label propagation, producing high-quality training data and achieving superior detection performance with reduced token costs.

Building on these ideas, our method takes inspiration from ZeroED and Raha by adopting clustering-based sampling and label propagation, but differs in two key aspects. First, we adopt a simplified yet effective feature extraction scheme with subword-level tokenization. Second, we design lightweight prompt strategies tailored for small LLMs, striking a balance between detection performance and efficiency. These design choices not only distinguish our approach from prior hybrid methods, but also motivate the next section.

2.2. Tokenization of Structured Data

2.2.1. Existing Tokenization Techniques

Tokenization [61] is a preprocessing step in NLP and LLMs, where raw text is segmented into smaller, discrete units known as tokens. These tokens can represent characters, subwords, or full words, depending on the tokenization strategy.

Broadly, three primary types of tokenizers are widely used:

- **Character/word-level tokenizers**: These traditional tokenizers split input text based on whitespace and punctuation marks. While straightforward and intuitive, they tend to produce large vocabularies and often lack the ability to generalize well to rare or compound tokens, resulting in sparse representations.
- **Subword tokenizers**: Predominantly used by modern LLMs, subword tokenization methods segment text into frequently occurring subword units, balancing vocabulary size and model generalization. This approach efficiently handles rare or out-of-vocabulary words by breaking them into known subunits. Examples include Byte Pair Encoding (BPE) [62] and WordPiece tokenization [63].
- **Byte-level tokenizers:** By operating directly on the raw byte representation of text rather than characters, byte-level tokenizers (e.g., Byte-level BPE, or BBPE) [64] improve robustness to noisy inputs, multilingual text, and unusual character sequences. This approach reduces vocabulary size while preserving expressiveness, particularly in scenarios with rich character diversity.

Overall, each tokenization strategy presents unique trade-offs in vocabulary compactness, representational granularity, and robustness to linguistic variation. The choice of tokenizer significantly influences how well downstream models, particularly LLMs, can interpret and reason over structured or semi-structured data inputs.

2.2.2. Challenges in Tokenizing Tables

Despite advances in tokenization techniques for textual data, structured tables present unique challenges due to their heterogeneous, multi-dimensional nature. Unlike plain text, tabular data is inherently structured along rows and columns, with each cell potentially containing values of vastly different types (e.g., strings, numerics, timestamps, categorical codes, or even missing values). This structural and semantic diversity complicates tokenization and subsequent processing by language models.

One key challenge lies in preserving the table's two-dimensional layout and relational semantics. Naively flattening a table into a row-wise or column-wise sequence may result in the loss of inter-cell dependencies or functional constraints (e.g., FDs, DCs). Moreover, simple tokenization may fail to meaningfully segment domain-specific terms, abbreviations, or compound values commonly found in tabular data. For instance, ED2 quantifies the frequency of n-grams within a cell using TF-IDF scores, effectively creating a bag-of-words (BoW) representation [65]. Similarly, Raha employs a bag-of-characters (BoC) representation by extracting character-level features to perform clustering-based detection [9]. While both methods employ character- or word-level tokenization and capture coarse-grained textual patterns, they struggle to represent fine-grained subword structures. As we discuss in Chapter 3, our approach resolve this issue by extracting features using a subword-level tokenizer combined with token occurrence frequency statistics.

2.3. Large Language Models

LLMs are deep neural networks trained on massive corpora of text data to learn statistical patterns of language. Leveraging architectures such as the Transformer [43], LLMs are capable of understanding and generating human-like text by predicting the next token in a sequence. Due to their scale, often containing billions of parameters, these models exhibit strong abilities in various natural language processing tasks, including text generation, comprehension, translation, and reasoning, often without task-specific training. Their capacity to generalize across diverse domains has made them a powerful tool in fields ranging from machine translation to data cleaning and error detection.

2.3.1. Closed-Source and Open-Sourced LLMs

LLMs have rapidly evolved, with several notable architectures dominating research and industry. Among the most influential closed-source models are OpenAI's GPT series [66]–[69], which pioneered large-scale autoregressive language modeling and have been widely adopted for a range of natural language understanding and generation tasks. Many SOTA closed-source models, such as Claude, Gemini [70], Grok, and PaLM [71], similarly demonstrated strong capabilities across multiple domains. However, their limited access to model architectures, training data, and fine-tuning methodologies constrains transparency, reproducibility, and community-driven research progress.

On the open-source front, Meta's LLaMA series [72] has made large-scale models more accessible to the research community, balancing performance with smaller model sizes for easier deployment. Other open-source models like BERT [73], DeepSeek [74], [75], Mistral [76],

and Qwen [77] also provide alternatives for developers seeking customizable LLMs without restrictive licensing. Considering the high cost and computational demands of closed-source models, we adopt open-source LLMs of medium and small sizes to execute inference tasks, as detailed in Chapter 4.

2.3.2. LLMs for Table Understanding

Tables represent a kind of structured data composed of rows, columns, and cells, whereas LLMs are inherently designed to process sequential textual inputs. To enable LLMs to understand tabular data, a serialization process is typically employed, converting the table into a linear textual format. This can be achieved through natural language or structured text representations such as Markdown tables, CSV, or TSV formats, which preserve the inherent row-column relationships in a textual form. Recent research [78]–[80] demonstrates that simple text serialization strategies, where each feature–value pair is rendered in a descriptive natural-language sentence, enable LLMs to perform surprisingly well on classification tasks when fine-tuned or prompted in a few-shot setting.

Beyond serialization, specialized training frameworks [8], [81], [82] facilitate deeper semantic understanding of tabular inputs and improve performance on semantic-parsing tasks. These models extend the Transformer architecture by incorporating row and column embeddings alongside textual inputs. Even though such specialized models can capture both the structural and semantic relationships inherent in tabular data, they typically require extensive pretraining on large-scale table-specific datasets, and often lack the flexibility and general-purpose reasoning capabilities exhibited by recent instruction-following LLMs.

In contrast, as detailed in Chapter 3, we design a lightweight serialization strategy that enriches inputs with row- and column-level context, while leveraging JSON formatting to enforce structured input–output control.

3

Methodology

In this chapter, we present the overall methodology of our LLM-based ED framework. Our approach combines traditional feature engineering with LLMs to identify erroneous values in structured tabular data without relying on extensive manual labeling. We first introduce the main components of the pipeline in Section 3.1, followed by detailed descriptions of the feature extraction process, which incorporates character-level, token-level, and inter-column structural signals (Section 3.2). These features are then used for clustering-based sampling (Section 3.3), which selects representative instances for efficient LLM annotation. We further elaborate on our LLM-based labeling strategy, including prompt design, zero-shot and few-shot prompting, and table representation techniques (Section 3.4). Finally, we describe how we propagate the obtained labels to the entire dataset in Section 3.5.

3.1. Overview

We propose a comprehensive framework for efficient ED in tabular data that leverages both structural representations and LLMs. As shown in Figure 3.1, the framework operates in three major stages: column feature construction, representative instance selection via clustering, and prompt-based error annotation and propagation.

For each attribute in the table, we construct a rich column feature representation that captures its underlying pattern, format, and relational dependencies. This representation integrates character-level, token-level, and statistical features to capture the format, semantics, and structural properties of each column. Together, these features provide a comprehensive understanding of column content and inter-column relationships.

To minimize the manual labeling burden, we introduce a clustering-based sampling strategy. First, data instances are clustered based on their column-level representations. Then, we identify the centroid of each cluster and select it as the representative instance. This method ensures both diversity and representativeness in the sampled data, allowing the LLM to observe a wide range of potential error patterns with minimal input. The number of clusters is a tunable hyperparameter that enables a trade-off between annotation cost and detection performance.

The selected representative instances are transformed into natural language inputs for the

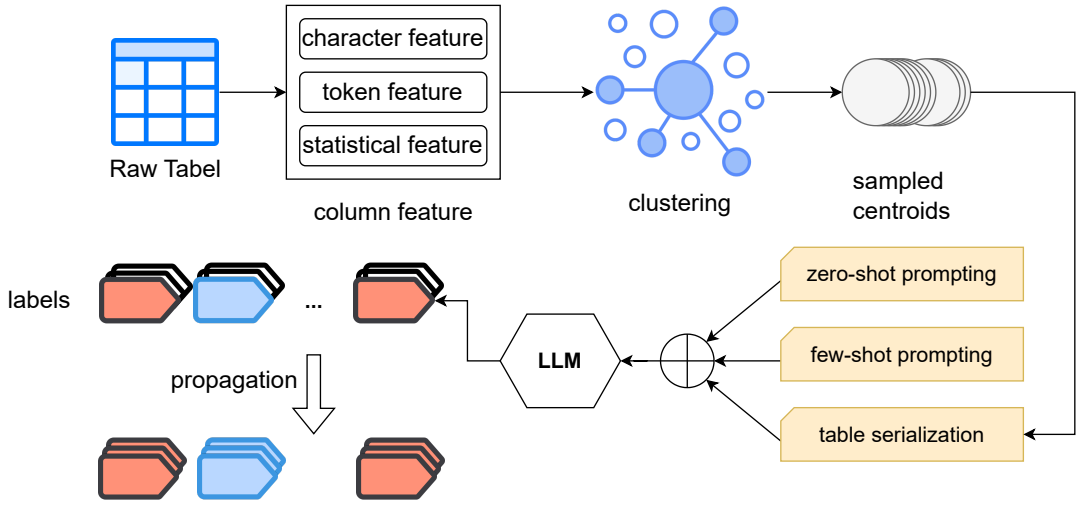


Figure 3.1: Framework of ED with LLMs

LLM through a carefully designed prompting strategy. Each prompt is composed of three key components: serialized tabular input, instructions and rule-based prompts. Our prompting strategy enables batch annotation of multiple data instances and achieves efficient execution on smaller LLMs through rule-guided design. Using the LLM's contextual reasoning capabilities, each selected instance is labeled with error types or correctness flags. Finally, based on the assumption of intra-cluster label similarity [83], the predicted labels are propagated to all instances within the same cluster, thus extending the annotations to the entire dataset with minimal supervision.

3.2. Feature Extraction

To effectively detect anomalies in table cells, we extract a diverse set of features at multiple levels, capturing both local textual properties and global structural relationships. At the character level, inspired by prior clustering-based error detection methods [9], we adopt features such as character length and character composition, which are effective for identifying irregular distribution and non-standard symbols in table cells (Section 3.2.1). At the token level, we propose frequency-based features derived from subword tokenization, enabling the detection of more complex and subtle pattern violations as well as typographical errors that may not be captured by coarse character-level statistics (Section 3.2.2). Beyond individual cells, at the structural level, we incorporate inter-column statistical features inspired by association rule mining, which capture dependencies and correlations between different attributes in the table (Section 3.2.3). This hierarchical combination of character-, token-, and structure-level features ensures a comprehensive representation of the data, facilitating more accurate anomaly detection.

3.2.1. Character-Level Features

Character-level representations can learn deviations in spelling, style, and grammar by modeling variations in character combinations [9]. By counting the frequency of all characters within each cell value, it is possible to distinguish entries that contain uncommon characters in the same attribute, without requiring predefined rules or data patterns. Specifically, for the j-th attribute in the dataset D, we define the character set C_j and count the total number of occurrences of each character $c_k \in C_j$ in all cells of the attribute, denoted as $f_j(c_k)$. After computing the character frequency for each cell, we apply TF-IDF (Term Frequency-Inverse Document Frequency) normalization to obtain a weighted character-level representation. TF-IDF emphasizes characters that are frequent in a particular cell but rare across the column, helping highlight subtle irregularities such as inconsistent punctuation or non-standard symbols. Finally, the character-level feature extraction function of the cell value $D_{i,j}$ is formally defined as:

$$\phi_{ch}(D_{i,j}) = \left[\text{tfidf}_{i,j}(c_1), \text{tfidf}_{i,j}(c_2), \ldots, \text{tfidf}_{i,j}(c_k)\right]$$

3.2.2. Token-Level Features

At the token level, we apply a subword tokenizer to each cell value to decompose the text into a sequence of meaningful subword units. As a widely used subword segmentation method, BPE [62] tokenizer can effectively balance vocabulary size and coverage, helping capture partial words and common substrings, making it suitable for handling typographical errors. Compared to character-level features, token-level representations preserve the subword order and capture the co-occurrence patterns of common tokens within one attribute. To mitigate the over-fragmentation caused by directly applying token-level representations, we simplify token categories by binning them according to their occurrence frequency within the attribute. In particular, for each tokenized value in an attribute, we compute the total frequency of each token across the attribute. Tokens that are purely numeric are decomposed into individual characters to avoid sparsity. For example, the token "2025" would be split into the characters "2", "0", "2", and "5" rather than treated as a single token. Then, each token is assigned to a frequency bin based on predefined thresholds (e.g., $\leq 1, \leq 2, \leq 5, \ldots, > 500$). For a given cell, we count how many of its tokens fall into each frequency bin, resulting in a fixed-length feature vector. Let $f_j(t)$ denotes the frequency of token t across all cells in attribute j, and $B = \{b_1, b_2, \ldots, b_n\}$ denotes a set of frequency bins, where each b_k is a count interval like $(p, q], p \leq q$. Then, the binned token-frequency feature vector for each cell will be:

$$\phi_{\text{bin}}(D_{i,j}) = \left[\sum_{t \in T_{i,j}} \mathbf{1}_{f_j(t) \in b_1}, \ldots, \sum_{t \in T_{i,j}} \mathbf{1}_{f_j(t) \in b_n}\right].$$

To reduce the impact of cell length, we normalize the feature vector as:

$$\hat{\phi}_{\text{bin}}(D_{i,j}) = \frac{\phi_{\text{bin}}(D_{i,j})}{\sum_{k=1}^{n} \phi_{\text{bin}}^{k}(D_{i,j}) + \varepsilon},$$

where $\varepsilon = 1 \times 10^{-10}$ is a small constant to prevent division by zero.

3.2.3. Structural and Inter-Column Features

In addition to character and token-level statistics, we also extract inter-column statistical features inspired by association rule mining. Rather than exhaustively enumerating all possible value combinations between attributes, functional dependencies can be approximated and efficiently represented using support and confidence measures. For example, suppose the j-th attribute as the right-hand side (RHS) attribute and an arbitrary left-hand side (LHS) attribute attribute l in a dataset D. The support measures the relative frequency of the co-occurrence of a specific LHS value x and RHS value y across the entire dataset, while the confidence measures the conditional probability of observing the RHS value y given the LHS value x:

$$\text{support}_l(x, y) = s_l = \frac{|\{i : D_{i,l} = x \wedge D_{i,j} = y\}|}{|D|}$$
$$\text{confidence}_l(y \mid x) = c_l = \frac{|\{i : D_{i,l} = x \wedge D_{i,j} = y\}|}{|\{i : D_{i,l} = x\}|}$$

Then, we construct a structural feature vector based on the support and confidence values between the target attribute j and all other m attributes:

$$\phi_{\text{fd}}(D_{i,j}) = [s_1, c_1, s_2, c_2, \ldots, s_m, c_m] = [s_k, c_k]_{k=1}^{m}.$$

To reduce the influence of the number of attributes on the magnitude of the structural feature vector, we normalize the feature by dividing it with $\sqrt{m}$: $\hat{\phi}_{\text{fd}}(D_{i,j}) = \frac{1}{\sqrt{m}} \cdot \phi_{\text{fd}}(D_{i,j})$. This ensures the vector remains scale-consistent across datasets with varying schema sizes.

3.2.4. Feature Aggregation

Finally, the complete feature vector for each cell $D_{i,j}$ is constructed by concatenating the character-level, token-level, and structural (FD-based) features:

$$\phi(D_{i,j}) = \left[\phi_{\text{ch}}(D_{i,j}) \,\|\, \hat{\phi}_{\text{bin}}(D_{i,j}) \,\|\, \hat{\phi}_{\text{fd}}(D_{i,j})\right]$$

Here, $\|$ denotes vector concatenation.

3.3. Clustering-based Sampling

To ensure diversity and representativeness in sampled data, we adopt a clustering-based sampling strategy. By partitioning the feature space into clusters and selecting representative samples from each cluster, this approach preserves the overall distribution of the data while mitigating the randomness inherent in purely random sampling. Such a strategy is particularly useful in tabular ED, where rare or subtle error patterns may be underrepresented in a naive random subset. Clustering-based selection ensures that each distinct region of the feature space contributes at least one exemplar, enhancing the quality and coverage of subsequent LLM labeling.

The mini-batch k-means clustering algorithm [84] is applied to the feature vectors of each attribute, as it efficiently extracts representative samples based on distance while maintaining computational scalability. Compared to the standard k-means algorithm, mini-batch k-means

updates cluster centroids using small, randomly selected subsets (mini-batches) of the data at each iteration. This makes it suitable for large-scale datasets, especially when clustering needs to be performed independently on multiple high-dimensional feature matrices. Formally, let $\mathbf{X}^{(j)} \in \mathbb{R}^{n \times d}$ denote the feature matrix corresponding to the j-th attribute of the dataset D, where n is the number of rows, and d is the dimensionality of the extracted feature vector $\phi(D_{i,j})$. We constrain the number of clusters for each column's feature matrix using a predefined parameter k: $\{C_1^{(j)}, C_2^{(j)}, \ldots, C_k^{(j)}\} = \text{MiniBatchKMeans}(\mathbf{X}^{(j)}, k)$. For each cluster $C_r^{(j)}$, calculate its centroid:

$$\boldsymbol{\mu}_r^{(j)} = \frac{1}{|C_r^{(j)}|} \cdot \sum_{D_{i,j} \in C_r^{(j)}} \phi(D_{i,j}),$$

and select the sample closest to the centroid as the representative data instance:

$$\mathbf{x}_r^{(j)} = \arg \min_{\mathbf{x}_i^{(j)} \in C_r^{(j)}} \|\mathbf{x}_i^{(j)} - \boldsymbol{\mu}_r^{(j)}\|_2.$$

Since the centroid of a cluster minimizes the average squared distance to all other points within that cluster, the point nearest to the centroid is naturally situated in the densest region of the cluster. This makes it a statistically central and prototypical member of the group.

3.4. LLM-based Error Labeling

Humans can accurately identify erroneous values in tabular data by leveraging real-world knowledge and observation on data feature. However, obtaining supervision data through this manual approach is undeniably expensive and time-consuming. To reduce reliance on such manually labeled data, LLMs can be employed to automatically annotate error labels for tabular data, replacing costly human supervision. Previous studies [47], [48], [50], [60] has demonstrated that state-of-the-art LLMs can effectively perform ED tasks by utilizing the vast corpora and knowledge acquired during their training process under specific conditions.

To assess the effectiveness of LLMs on ED tasks, we apply them to a subset of representative samples obtained via clustering and investigate multiple prompting strategies and improvement techniques.

```
1 You are an error detection assistant. Respond ONLY in valid JSON.
2 [Zero-shot prompt]
3 [Few-shot prompt]
4 [Table representation]
5 [Output template]
```

Listing 3.1: Prompt overview

3.4.1. Prompt Design

We propose a modular and extensible prompting framework, as illustrated in listing 3.1. The prompt begins by explicitly defining the model's role and strictly enforcing a valid JSON output format. This ensures that the model remains focused and produces structured responses. The core of the prompt consists of three key components: zero-shot instructions,

```
Instructions:
- You are given a list of data rows extracted from a dataset, where the target
    column named ‘county‘. Label each value from the target column as either an
    error or not an error. An error could be a typo or a formatting issue.
- Only evaluate the ‘county‘ value.
- Only label values you are confident about. If you are not sure, do not label
    it as an error.
```

Listing 3.2: Zero-shot instruction prompt example

few-shot demonstrations, and table representations. They can be flexibly combined to support different prompting strategies. Finally, the prompt concludes with an output template derived from the input data schema, which reinforces the expected response format and improves consistency during inference.

3.4.2. Zero-shot Prompting

LLMs can generalize to unseen tasks via zero-shot prompting, where a model follows instructions without seeing any examples. In particular, models trained via instruction tuning [85] or reinforcement learning from human feedback [86] demonstrated improved zero-shot capabilities. In the zero-shot setting, we design a general instruction prompt that describes the ED task without providing explicit examples. Listing 3.2 shows an example, where the prompt clearly states the task, the target column to be evaluated, and the decision rules.

While general instruction prompts are effective in capturing common error types such as typos or formatting inconsistencies, they may fail to handle dataset-specific requirements. In practical scenarios, the definition of an error can vary across datasets. For instance, certain placeholder values like "N/A" or "empty" might be acceptable in one column but considered invalid in others. Similarly, expected output formats can differ significantly depending on the data context. To address these challenges, we further augment the instruction prompt with human-crafted rules that specify column-level semantics, value constraints, or formatting expectations. By manually inspecting each column's data distribution, we derive lightweight and declarative rules that can be injected directly into the prompt in natural language. For a detailed list of such rules, see Appendix A.

3.4.3. Few-shot In-Context Learning

To further enhance model's performance, we also apply few-shot prompting [68] by including a small number of annotated examples within the prompt. These examples illustrate both correct and incorrect values, allowing the LLM to generalize its labeling decision through robust pattern recognition. Each example presents a single data instance, corresponding to a raw table row converted into a textual format (detailed in section 3.4.4), and specifies the original and corrected values of the target column, as shown in listing 3.3.

Compared to zero-shot prompting, few-shot prompting allows the model to observe concrete input-output patterns, which is especially beneficial for columns with dataset-specific error types or subtle anomalies that may not be fully captured by general instructions. By

```
Here are some examples:

Example 1:
Row: [Data Instance 1]
Column: county
Original value: dx kalb
Corrected value: de kalb

Example 2:
Row: [Data Instance 2]
Column: county
Original value: tallaxega
Corrected value: talladega

...
```

Listing 3.3: Few-shot prompt example

```
{
  "row_id": <row index>,
  "attr_1": <value_1>,
  "attr_2": <value_2>,
  ...
  "attr_n": <value_n>
}
```

Listing 3.4: Example of a serialized table row in JSON format

providing a small, representative set of examples, the LLM can infer implicit rules and correction strategies without requiring full fine-tuning. This approach also reduces the risk of under-labeling or mislabeling common patterns that were not explicitly described in the instruction.

3.4.4. Table Representation

Since LLMs are fundamentally sequence-to-sequence models, they cannot directly process two-dimensional tabular data [87]. Therefore, a key challenge in table representation lies in converting the 2-dimensional structure into a linearized or serialized one-dimensional string that LLMs can understand. Common and effective serialization methods for tabular understanding tasks include using natural language descriptions or structured formats such as CSV, JSON, XML, Markdown, HTML and etc. [88]. Given its clear key-value structure and natural alignment with table semantics, we serialize each table row into a JSON-style structure in the following format: In this format, each key corresponds to a column name, and each value to the respective cell content. The special key "row_id" denotes the index of the row in the original table and is used to identify each data instance. To optimize how LLMs process this serialized data, we consider two distinct dimensions for input construction: column-wise selection and row-wise grouping.

Column-Wise Input Selection
To investigate the influence of contextual column information on the model's ED ability, we explore two input selection strategies for each serialized row:

- **Target-Only**: This minimalist strategy includes only the value of the target column, served as a baseline to assess the model's performance when no additional contextual information is available.
- **Top-N Informative Columns**: This strategy augments the input with values from informative columns in the same row, selected based on their normalized mutual information (NMI) scores with respect to the target column. Specifically, for each column we compute pairwise NMI scores, sort the related columns, and then select between one and five with the highest scores (subject to a minimum of one). If no column exceeds the given NMI score threshold, the top column is still selected to ensure coverage.

Row-Wise Grouping
Beyond intra-row information, inter-row relationships within the same target column often provide valuable signals for ED. To exploit this, we employ a batch prompting strategy [89], concatenating multiple serialized rows into a single prompt. This allows the model to compare entries across rows and better identify inconsistencies or anomalies that might not stand out when processing rows independently. For example, consider a `birthday` column: if most rows contain dates in the format `MM-DD-YYYY` but one row includes only the year, grouping rows allows the model to recognize this irregularity more reliably by contrasting it with neighboring entries. Moreover, batching improves inference efficiency by reducing token consumption and sharing prompt overhead across instances.

3.4.5. Structured Output

To ensure consistency and facilitate automatic parsing of model predictions, we enforce a structured output format. Specifically, given serialized table rows as input, the LLM is instructed to generate predictions following the JSON template in Listing 3.5. The template explicitly defines a list of row-level labels, where each entry contains the row index and a Boolean field `is_error` indicating whether the table cell is detected as erroneous.

Such an explicit output specification serves two purposes. First, it constrains the model to return results in a machine-readable format, which simplifies subsequent evaluation and integration into the ED pipeline. Second, for small LLMs in particular, clearly defined output templates reduce the probability of missing or inconsistent labels and help the model better align with the expected task semantics. Furthermore, for models that support constrained decoding or JSON schema validation, we specify the corresponding schema in runtime parameters, which stabilizes the generation process and ensures the LLM reliably produces structured outputs.

3.5. Label Propagation

Once the representative samples has been labeled by the LLM, we propagate these labels to the remaining unlabeled samples within the same feature space to enable large-scale annotation

```
Please fill in the "is_error" field (either ‘true‘ or ‘false‘) in the following
    JSON template:
{
  "labels": [
    {
      "row_id": <row index>,
      "is_error": null
    },
    ...
  ]
}
```

Listing 3.5: Example of output template

without incurring additional labeling cost. According to the cluster assumption [9], [83], instances within the same cluster are likely to share the same class label. Therefore, by clustering the data cells in a column using their feature vectors, we can group similar cells together and distinguish between clean and dirty samples. Specifically, the model's responses are first parsed according to a predefined JSON schema to extract each labeled instance, including its row index and the predicted label. These row indices are then mapped to their corresponding clusters, and the labels of the representative samples are propagated to all other members of each cluster.

For robustness, our system tolerates imperfect LLM outputs. If the model's responses deviate from the expected JSON format, omit instances, or contain ambiguities, the affected samples retain their original (clean) labels. Given that the proportion of erroneous entries in the dataset is typically low, this conservative strategy minimizes the risk of introducing incorrect labels during propagation while maintaining high overall annotation reliability.

4

Experiments

In this chapter, we conduct a comprehensive evaluation of our proposed method on multiple real-world and synthetic datasets. The goal of our experiments is to assess the effectiveness, robustness, and generalizability of our approach in comparison to state-of-the-art baselines, as well as to understand the contribution of individual components through ablation analysis.

4.1. Experimental Setup

To ensure reproducibility and to mitigate the influence of randomness in clustering-based methods, we fix the random seed for all clustering algorithms. This guarantees that different models are provided with consistent prompts derived from identical cluster assignments in our experiments. The source code for our experiments is publicly available on GitHub.[1]

4.1.1. Datasets

We first evaluate on three widely used open-source benchmarks from prior work [9], [19], [60], including *Flights*, *Hospital*, and *Beers*. These datasets are commonly adopted in the error detection literature and span a diverse range of domains. *Hospital* contains structured data related to healthcare providers. Errors are synthetically injected by randomly replacing characters in certain fields with the character 'x', leading to typos and rule violations, such as incorrect ZIP codes or provider IDs. *Flights* includes airline flight information and is primarily corrupted by violating FDs. This is done by modifying or deleting departure and arrival times to introduce inconsistencies, as well as injecting formatting issues. *Beers* [90] is a semi-curated dataset collected from the web, containing a wide range of error types, including typos, missing values, schema inconsistencies, and semantic violations.

In addition, we construct a new synthetic dirty dataset *Spotify* [91] by injecting realistic errors, aiming to better reflect the complexity of data quality issues encountered in real-world scenarios. We noticed that only the Hospital dataset contains typos among the three benchmark datasets. These typos are artificially generated by replacing a single character with an "x", which doesn't reflect the diverse nature of real-world typographical errors. To address this, our synthetic dataset includes a richer variety of error types, primarily focusing

[1] `https://github.com/montpelllier/EffiED-LLM`

on realistic typos and rule violations. The simulated typos encompass several common operations:

1. **Character replacement**, where characters are randomly replaced with adjacent keys on a standard English keyboard layout (QWERTY) or with visually or semantically similar characters. For example, letter 'q' might be replaced with its neighbors 'a', 'w' or 's'. And the lowercase letter 'o', the digit '0', and the uppercase letter 'O', may be substituted for one another;
2. **Character deletion**, where a randomly selected character is removed from a string;
3. **Character duplication**, where a randomly selected character is duplicated and inserted immediately after itself.

For simulating rule violations, we focus on attributes involved in known FDs. Specifically, we randomly select a set of cells and replace their values with values from other tuples in the same column, thereby violating the underlying FDs. Additionally, we introduce missing values by randomly replacing a subset of cells with nulls to simulate incompleteness, which is another common form of data corruption in real-world datasets.

Table 4.1: Summary of benchmark datasets used for evaluation. Error types consist of typo (T), rule violation (RV), missing value (MV), pattern violations (PV) and outlier (O). Error rate is rounded to two decimal places.

Dataset	Err. Rate	Tuples	Attributes	Err. Types
Hospital	2.96%	1,000	18	T, RV
Flights	29.58%	2,376	7	RV, MV, PV
Beers	16.71%	2,410	11	RV, MV, PV
Spotify	10.00%	10,000	11	T, RV, MV, O

Table 4.1 provides details on the number of records, attributes, error rates and existing error types for each dataset.

4.1.2. Evaluation Metrics

As evaluation metrics, we report precision, recall, and the F_1 score, which collectively measure the accuracy and completeness of ED performance. Precision reflects how many predicted errors are correct, while recall measures how many real errors are detected. F_1 score serves as a harmonic mean, offering a balanced evaluation of both. This suite of metrics is preferred over simple accuracy, particularly in low-error-rate datasets, where accuracy can be misleading due to class imbalance.

4.1.3. Baseline

We use Raha [9] and ZeroED [60] as baseline methods for evaluating our approach. Raha integrates a variety of data cleaning tools to generate diverse error detection strategies. It then clusters these strategies and propagates a small number of human labels to the remaining data. This enables Raha to achieve strong ED performance with minimal supervision, making it highly effective in scenarios where labeled data is limited. ZeroED, in contrast, explores LLMs for zero-shot ED tasks. It leverages LLMs' abilities in code generation, natural language

annotation, and reasoning to detect potential errors without requiring labeled data. Those methods provide strong low-supervision and zero-supervision baselines for evaluating the effectiveness of our proposed method.

4.2. Results

4.2.1. Comparison Study

We conducted a comprehensive comparison of different ED approaches across multiple datasets, as presented in Table 4.2. To emphasize the applicability of our approach to small and medium-sized LLMs, we selected models from the LLaMA 3.1 [92], Qwen3 [93], and Gemma 3 (instruction-tuned version) [94] families. Specifically, LLaMA 3.1 and Qwen3 models were deployed using the Ollama framework, while the Gemma 3 series models were accessed via the Google AI Studio APIs. This setup ensures a representative evaluation across different model families and deployment platforms. For the clustering parameters, we fix the number of clusters k for each column to 20. The `Ground Truth` represents the performance when labeling representative samples for each cluster with correct labels, serving as an upper-bound reference for comparison.

Among all evaluated models, `Gemma-3B-27B-IT` consistently achieved the best or near-best performance across datasets. This aligns with the scaling law hypothesis [95], indicating that larger model sizes generally lead to improved capabilities in error detection tasks. Although `LLaMA-3.1-8B` exhibits the weakest overall performance among the evaluated models, its low precision and relatively high recall suggest that it is overly sensitive to potential errors, frequently misclassifying correct values as erroneous. Many of these false positives could be avoided through better adherence to instructions or rule-based constraints, which are correctly handled by other models in the benchmark. This is because `LLaMA-3.1-8B` has not been instruction-tuned, making it less capable of accurately interpreting the prompts and task-specific instructions in the input. Compared to the upper-bound performance (ground-truth labels), our method shows a slight performance gap on the Hospital and Flights datasets, but nearly reaches the theoretical optimum on *Beers* and *Spotify*. This demonstrates the effectiveness of our prompt design tailored for small and mid-sized language models in guiding accurate ED. To better understand the limitations of LLMs in this task, we analyzed the erroneous predictions across datasets. We observed several bottlenecks that persist even in the strongest models:

1. **Abstract or complex values**: LLMs struggle with uncommon abbreviations, internal codes, or concatenated forms. For example, in the Hospital dataset, the value "scix-inf-2" (a typo for "scip-inf-2") corresponds to the measure name “surgery patients who were given the right kind of antibiotic to help prevent infection.” Such typos make it difficult for the model to distinguish between domain-specific abbreviations and actual errors.
2. **Numeric data**: Due to the tokenization mechanism in Transformer-based models, numeric values are spitted and embedded without inherent quantitative understanding. This leads to difficulties in detecting numeric outliers or violations of numerical rules, as observed in multiple datasets.
3. **Statistical or distributional errors**: In datasets like Flights, many errors involve small

deviations in time values that violate latent functional or statistical dependencies. Such errors are challenging for LLMs to detect solely based on local context or co-occurrence patterns, as they require fine-grained statistical reasoning.

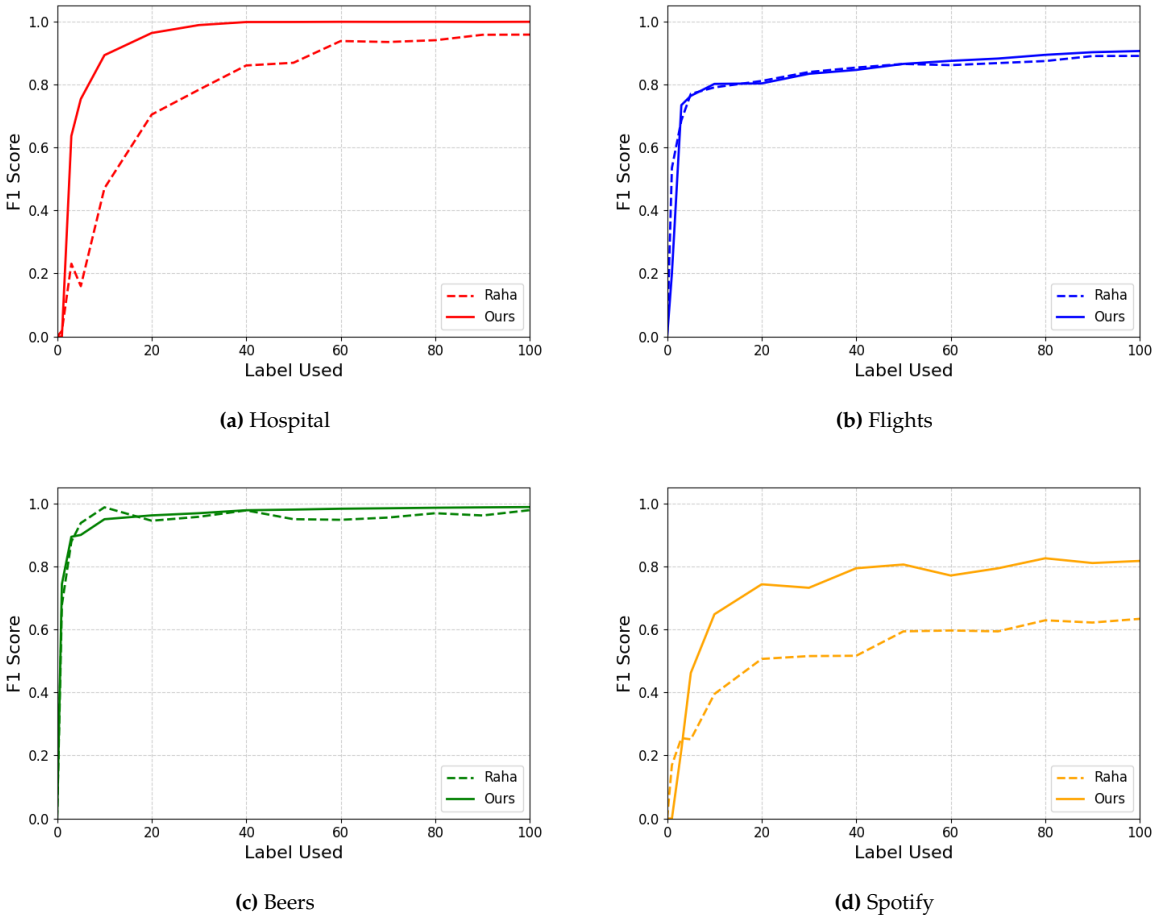
Compared to the baseline methods, our approach consistently outperforms both traditional and LLM-based methods on datasets where typos constitute the majority of errors. It is notable that ZeroED is also an LLM-based method that does not rely on human-labeled data, whereas Raha is a traditional weakly-supervised approach that requires a small amount of manual labeling. All three methods leverage clustering algorithms to minimize the annotation effort. Interestingly, on the *Flights* dataset, our method and ZeroED perform similarly, while both fall behind Raha. This is accompanied by relatively high precision but low recall, indicating that the model fails to identify a large number of actual errors, particularly when the dataset contains a substantial number of FD violations that are syntactically correct but contextually inconsistent. Such behavior indicates the difficulty LLMs face in identifying statistical or distributional errors without real labels or strong statistical priors. This observation further supports our earlier analysis regarding the limitations of LLMs in detecting subtle data quality issues, particularly those involving numerical deviations or latent functional dependencies.

Table 4.2: Comparative evaluation of different error detection approaches, measured in percentages (%). "–" indicates data not reported or applicable in prior works for this dataset.

Approach	**Hospital**			**Flights**			**Beers**			**Spotify**		
	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
Raha	81.9	54.3	64.1	84.6	**81.2**	**82.8**	96.2	**96.0**	96.0	58.8	43.6	49.4
Zero-ED	93.6	71.5	81.1	93.5	58.6	72.2	88.8	68.9	77.4	-	-	-
Llama-3.1-8B	23.8	**93.1**	37.9	56.9	57.9	57.4	51.8	74.5	61.1	41.9	**70.4**	52.5
Qwen3-8B	**99.8**	85.7	**92.2**	95.2	57.9	72.0	96.7	91.2	93.9	50.4	67.0	57.5
Qwen3-4B	82.1	85.0	83.5	91.6	57.2	70.4	96.0	81.0	87.9	53.3	64.5	58.3
Gemma-3-27B-IT	**99.8**	85.7	**92.2**	**100**	57.7	73.1	90.2	93.0	91.6	**79.5**	67.4	**73.0**
Gemma-3-12B-IT	86.8	87.4	87.1	96.6	57.6	72.2	**96.8**	95.7	**96.3**	70.9	62.2	66.3
Gemma-3-4B-IT	60.5	79.9	68.9	88.9	45.9	60.5	91.1	89.8	90.5	54.3	65.1	59.2
Ground Truth	99.8	93.1	96.3	77.6	89.9	79.2	96.8	95.7	96.3	75.4	70.9	73.0

To further demonstrate the effectiveness of our feature engineering, we conducted experiments by varying the number of clusters and comparing the results of our approach and Raha, both utilizing the true labels for evaluation. Figure 4.1 illustrates this comparison, highlighting the consistent performance advantages of our method across different cluster settings. Notably, our approach outperforms Raha on the *hospital* and *spotify* dataset, exhibiting both faster convergence and higher final performance. As the number of labeled samples increases, Raha shows significant improvement on the Hospital dataset, eventually approaching the performance of our method. However, it consistently lags behind on the *spotify* dataset. This discrepancy is primarily due to Raha's reliance on the BoC strategy, which struggles to distinguish between typos caused by character repetition (e.g., "aaaddress") and those caused by character omission (e.g., "adress"). In contrast, our method leverages the insight that typos typically disrupt the original token sequence structure, allowing for more comprehensive and fine-grained detection of such errors. Meanwhile, the performance of both is comparable on

the *beers* and *flights* datasets. This indicates that our support and confidence-based features can achieve performance on par with Raha's value-matching strategies across columns, particularly in the detection of FD violations. While Raha explicitly constructs rule violation detection strategies for each potential FD by exhaustively considering all attribute pairs, our method implicitly captures similar structural dependencies through statistical correlations, without requiring the enumeration of all rule templates. These results suggest that our incorporation of token-level features enables more effective detection of typographical errors, while the statistical features derived from support and confidence can achieve a level similar to that of Raha.



(a) Hospital

(b) Flights

(c) Beers

(d) Spotify

Figure 4.1: Error detection performance between Raha and our approach

4.2.2. Ablation Study

Table 4.3 presents an ablation study on different prompting strategies using `Qwen3-8B` model across different datasets. The results reveal several key insights. First, prompting with specific rules (**ZS-R**) consistently outperforms general instructions (**ZS-I**) across all datasets, which emphasize the importance of explicit task guidance in zero-shot settings. Additionally, incorporating values from related attributes (**A**) generally improves recall and F_1 scores, particularly for more structured datasets such as *Beers*, demonstrating the effectiveness of

contextual information from other columns. Among all combinations, the **ZS-R+A** strategy achieves the highest or near-highest performance across datasets, illustrating the robustness of combining precise rules with inter-column context. Interestingly, adding random few-shot examples (**FS**) provides modest gains when used with general instructions, suggesting that concrete examples can help compensate for the lack of specificity in zero-shot settings. However, when combined with rule-based prompts, few-shot examples sometimes lead to a performance drop. This suggests that the additional examples may introduce redundancy or conflict with the clear guidance provided by the rules, potentially confusing the model and reducing its overall effectiveness.

These observations underscore several practical considerations. First, careful prompt design, especially incorporating rule-based guidance, is critical for small to medium-sized LLMs. Second, context selection can substantially enhance detection of complex or subtle anomalies. Finally, random few-shot examples should be used judiciously: while they can help in general instruction settings, overloading prompts with examples in the presence of precise rules may be counterproductive.

Table 4.3: Ablation study of different prompting strategies across datasets using Qwen3-8B, measured in percentages (%). Batch size is 10.

Strategy	Hospital			Flights			Beers		
	P	R	F_1	P	R	F_1	P	R	F_1
ZS-I+V	62.45	83.30	71.38	**100**	56.67	72.26	88.65	17.01	28.54
ZS-R+V	95.41	85.74	90.32	**100**	57.66	**73.15**	96.05	75.72	84.68
ZS-I+FS+V	60.71	**93.06**	73.48	**100**	41.34	58.50	95.85	38.68	55.11
ZS-R+FS+V	84.07	90.06	88.34	**100**	57.66	**73.15**	95.59	67.52	79.14
ZS-I+A	68.62	83.30	75.25	96.78	47.62	63.83	85.26	11.14	19.71
ZS-R+A	**99.78**	85.74	**92.23**	95.19	**57.93**	72.02	96.70	**91.24**	**93.89**
ZS-I+FS+A	65.50	87.62	74.96	96.68	46.10	62.43	94.48	28.66	43.98
ZS-R+FS+A	96.01	85.74	90.59	97.32	57.66	72.42	**98.86**	65.43	78.74

ZS-I: zero-shot prompting with general instructions; **ZS-R**: zero-shot prompting with specific rules; **FS**: few-shot prompting; **V**: use value from the target attribute only; **A**: use values from related attributes.

As shown in Table 4.4, we also analyzed the impact of different batch sizes on model performance and computational efficiency. Both token count and total processing time decrease with larger batch sizes, indicating more compact inputs with reduced redundancy. On the other hand, the token processing rate shows a decreasing trend, from 198.97 tokens per second at batch size 1 to 165.99 tokens per second at batch size 20. This implies that the model spend more time per prompt reasoning about the relationships between individual cell values when multiple rows are processed together, likely due to the increased complexity of cross-row comparisons.

In terms of performance, a small increase on batch sizes significantly improve the F_1 score by about 0.2, from batch sizes 1 to 4, reflecting the benefit of inter-row context in detecting anomalies. Beyond batch size 4, the F_1 score stabilizes at a consistently high level, indicating diminishing returns from further increasing batch size. These results highlight a trade-off between computational efficiency and model reasoning capacity: moderate batch sizes can

leverage inter-row relationships to improve detection accuracy while keeping processing time manageable. Practically, selecting a batch size in this intermediate range offers an effective balance, maximizing F_1 performance without incurring excessive per-prompt computation or memory overhead.

Table 4.4: Effect of different batch sizes on performance and efficiency. Measured on the *Hospital* datatset, using Gemma-3-27B-IT, zero-shot prompting with specific rules and values from related attributes.

Batch Size	F_1 (%)	Token Count	Time (s)	Rate (token/s)
1	74.19	104,874	527	198.97
2	79.71	66,474	354	187.81
4	94.99	47,247	263	179.63
10	92.23	35,691	211	169.15
20	94.79	32,047	193	165.99

5

Discussion

5.1. Limitations

While the proposed framework demonstrates strong performance in error detection for tabular data, several limitations remain that warrant further investigation and improvement.

5.1.1. LLM Hallucination and Misinterpretation

One major challenge associated with LLMs is hallucination [96], i.e., the generation of outputs that are not grounded in the input data. In the context of error detection, hallucination manifests as incorrect error labels—either by mistakenly identifying correct values as erroneous or by overlooking genuine data issues. This is particularly likely when the data contains ambiguities, domain-specific abbreviations, or inconsistent formatting, which LLMs may struggle to disambiguate without sufficient context.

Furthermore, misinterpretation may arise even when the data is well-structured. Since LLMs rely heavily on surface-level token patterns and pretraining priors, they may be misled by syntactic irregularities rather than semantically meaningful cues. Given that our label propagation relies on a small number of LLM-labeled examples, such hallucinated or misinterpreted labels can propagate through clusters and negatively affect overall performance. Improving prompt clarity and incorporating verification mechanisms could help mitigate this issue.

5.1.2. Risk of Data Leakage

Another concern is the risk of data leakage, stemming from the possibility that pre-trained LLMs may have encountered portions of the benchmark datasets during training. Many widely used datasets in the data cleaning and error detection literature, such as *Flights* or *Beers*, are publicly available and frequently used in tutorials, blogs, or documentation. Although we introduced a new dataset, *Spotify* (updated in May 2025), similar public dataset exist on the internet. As such, there is a non-negligible chance that large-scale web-crawled corpora used for LLM pretraining may contain these datasets or similar samples.

This potential overlap can lead to inflated evaluation results, particularly if the model exhibits

memorization rather than generalization. While exact memorization is difficult to confirm due to the scale of training data, it raises fairness and reproducibility concerns.

5.1.3. Challenges with Numerical and Temporal Data

Despite their language understanding capabilities, LLMs face significant limitations when dealing with numerical and time-series data. This is largely due to the sub-word/sub-byte tokenization and embedding mechanisms used in Transformer-based models, which do not natively support arithmetic reasoning or numerical comparison. As a result, LLMs are poorly suited for identifying statistical outliers, temporal inconsistencies, or range violations—common types of errors in structured data.

For example, detecting whether a departure time is later than the arrival time, or whether a numeric value falls outside a contextually reasonable range, often requires direct numerical comparison rather than pattern-based inference. In such scenarios, LLMs are prone to both false positives and false negatives. Addressing this limitation may require integrating LLMs with symbolic reasoning modules or hybrid systems that combine language understanding with domain-specific numerical logic.

5.1.4. Limited Generalization Across Tables

The current framework is designed to operate on individual tables in isolation, without considering relationships or dependencies between different tables. However, many real-world data cleaning tasks involve datasets that are logically connected via foreign keys, shared schema elements, or semantic correspondences. For example, consistency checks may span multiple relational tables or require enforcing referential integrity constraints.

LLMs in their current form lack built-in mechanisms for modeling such inter-table relationships. Extending our method to support cross-table reasoning would involve complex prompt engineering, context fusion strategies, or even graph-based representations to encode multi-table structures. While some early research has explored LLMs for relational understanding, scalable and robust cross-table error detection remains an open and promising research direction.

5.1.5. Limitations of Small-Scale LLMs

To reduce computational cost, our framework supports the use of smaller LLMs (e.g., 4B–30B parameters) for tasks like zero-shot labeling and rule-based reasoning. However, these models often are less effective compared to larger instruction-tuned counterparts. In our experiments, small LLMs showed higher false positive rates and lower precision, especially when interpreting complex or abstract prompts involving domain-specific terminology or multi-step logic.

One key reason is that smaller models are less capable of following nuanced instructions or adapting to domain constraints without fine-tuning. In some cases, they may ignore rule conditions entirely or misinterpret the task objective. This limits the reliability and scalability of our method in low-resource or edge deployment scenarios. Enhancing the prompt design,

incorporating lightweight fine-tuning, or employing model distillation from larger LLMs are possible directions to mitigate these issues.

5.2. Future Work

While our current approach demonstrates promising results in LLM-based ED, several future directions can further enhance its robustness, adaptability, and practical relevance. Below, we outline potential areas of exploration.

5.2.1. Ensemble Approaches

A single error detection method often struggles to maintain consistent performance across diverse data types and error categories. To address this, integrating multiple complementary methods, such as rule-based systems, traditional statistical models, and machine learning classifiers, may yield a more balanced and robust detection framework. Ensemble strategies can leverage the strengths of each component and mitigate individual weaknesses, potentially improving both precision and recall.

5.2.2. Self-Improvement Capabilities

Small-scale LLMs are inherently limited in reasoning and generalization due to constrained model capacity. However, recent advances have shown that LLMs can improve their own outputs through self-improvement mechanisms with minimal human supervision, offering a promising direction for lightweight yet effective deployment.

Context Enrichment
One strategy is to enrich the input context to reduce hallucinations and improve output quality. This can include incorporating external knowledge [68], [97], historical interaction traces [98], or employing chain-of-thought prompting [99], [100] to encourage more coherent reasoning. By explicitly guiding the model's thought process, even small LLMs can demonstrate more robust behavior in complex tasks.

Feedback-Guided Refinement
Another direction involves iterative refinement based on feedback. This may include self-evaluation and self-correction mechanisms inspired by the LLM-as-a-Judge [101] paradigm, as well as learning from implicit signals such as uncertainty estimates [102]. In code generation, feedback from the execution environment [103], [104], such as error messages, output mismatches, or runtime exceptions captured by Python interpreters, can serve as valuable signals to guide iterative refinement as well. Feedback-driven methods allow the model to identify and correct its own errors progressively, imitating aspects of human self-reflection.

Collaborative Modeling
Leveraging multiple models can further enhance self-improvement. For instance, small LLMs can be supervised or guided by larger, more capable models [105]. Alternatively, ensembles of smaller models can collaborate [106], either through specialization (domain-specific expertise) or cross-validation to ensure output reliability [107].

Synthetic Data Augmentation
Self-generated synthetic data can serve as an additional resource for improving model performance. LLMs can be prompted to simulate plausible erroneous cases, which can then be used for fine-tuning or training error correction modules. This synthetic augmentation acts as a form of self-supervised learning, enhancing generalization without manual annotation.

5.2.3. Code Generation for Statistical Errors

While LLMs exhibit strong contextual understanding, they often struggle with numerical, logical, or statistical anomalies, which can be more effectively addressed through executable logic. Harnessing LLMs for code generation offers a promising solution: by translating natural language descriptions and data observations into executable code, models can perform programmatic data checks or transformations. Although small LLMs may underperform in complex code synthesis tasks, recent work [108] suggests that iterative code generation and multi-step reasoning can partially offset these limitations. Exploring such capabilities for lightweight models could bridge the gap between rule-based detection and LLM inference.

5.2.4. Extension to Data Wrangling

Our current focus is limited to single ED. However, in real-world applications, identifying errors is only the first step. Comprehensive data wrangling often requires cleaning, repairing, and transforming data. Future work should explore how LLMs can assist not only in detection but also in suggesting corrections, resolving inconsistencies, and enabling end-to-end data refinement workflows. By combining predictive modeling with external knowledge bases or interactive feedback, LLMs could be further developed into versatile agents for automated data wrangling tasks, thus enhancing their practical utility in data engineering pipelines.

6

Conclusion

This thesis presents a novel framework for efficient table ED using LLMs, with a particular focus on minimizing manual annotation efforts. By introducing a clustering-based representative labeling method, we demonstrate how representative samples from each cluster can be effectively annotated using prompting techniques, while the remaining samples are labeled via label propagation. Moreover, feature extraction based on tokenization plays a critical role in capturing both lexical and structural patterns from tabular data. This approach significantly reduces reliance on expensive labeled data and enables scalable error detection across diverse tabular domains.

The proposed method leverages zero-shot and batch prompting, balancing labeling efficiency with model accuracy. Through comprehensive experiments on real-world datasets, we show that our method achieves competitive performance compared to traditional baselines, with improved adaptability and lower annotation costs. In addition to empirical results, we critically analyze the limitations of current LLM-based systems in ED, such as hallucinations, data leakage risks, difficulties with numerical or temporal reasoning, and challenges in generalization across tables. These insights motivate several future directions, including ensemble methods, self-improvement strategies, LLM-driven code generation for handling statistical inconsistencies, and extending the framework toward broader data wrangling tasks such as data cleaning and transformation.

Overall, this work highlights the promising role of small LLMs in understanding structured data and provides a preliminary step toward more efficient and scalable approaches for data quality management.

Acknowledgement

First and foremost, I would like to express my heartfelt gratitude to my supervisor, Dr. Hazar, for her continuous support and guidance throughout this journey. From the initial stage of topic selection to the weekly meetings that kept me on track, her patience, constructive feedback, and wealth of ideas have been invaluable. Her suggestions greatly enhanced the quality and depth of the literature study and this work.

I am also deeply thankful to my fellow students and friends for their companionship, encouragement, and understanding. Their help, whether academic or emotional, made this journey much more enjoyable and less stressful.

Special thanks to my parents and family for their unwavering support and care. Their belief in me has been a constant source of motivation. Without them, I could not have made it this far.

And last but not least, I would like to thank the cool summer in Netherlands, those unforgettable experiences, environment and resources provided by the university...

References

[1] S. Mohammed, L. Budach, M. Feuerpfeil, *et al.*, "The effects of data quality on machine learning performance on tabular data," *Information Systems*, vol. 132, p. 102 549, 2025.

[2] T. C. Redman, "The impact of poor data quality on the typical enterprise," *Commun. ACM*, vol. 41, no. 2, pp. 79–82, Feb. 1998.

[3] A. Haug, F. Zachariassen, and D. van Liempd, "The costs of poor data quality," *Journal of Industrial Engineering and Management (JIEM)*, vol. 4, no. 2, pp. 168–193, 2011.

[4] T. Furche, G. Gottlob, L. Libkin, G. Orsi, and N. Paton, "Data wrangling for big data: Challenges and opportunities," in *Advances in Database Technology — EDBT 2016*, Nov. 2016, pp. 473–478.

[5] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation," *Advances in Neural Information Processing Systems*, vol. 36, pp. 21 558–21 572, 2023.

[6] Y. Tan, D. Min, Y. Li, *et al.*, "Can chatgpt replace traditional kbqa models? an in-depth analysis of the question answering performance of the gpt llm family," in *International Semantic Web Conference*, Springer, 2023, pp. 348–367.

[7] K. M. Collins, A. Q. Jiang, S. Frieder, *et al.*, "Evaluating language models for mathematics through interactions," *Proceedings of the National Academy of Sciences*, vol. 121, no. 24, e2318124121, 2024.

[8] M. Nashaat, A. Ghosh, J. Miller, and S. Quader, "Tabreformer: Unsupervised representation learning for erroneous data detection," *ACM/IMS Transactions on Data Science*, vol. 2, no. 3, pp. 1–29, 2021.

[9] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, *et al.*, "Raha: A configuration-free error detection system," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 865–882.

[10] R. Y. Wang and D. M. Strong, "Beyond accuracy: What data quality means to data consumers," *Journal of management information systems*, vol. 12, no. 4, pp. 5–33, 1996.

[11] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino, "Methodologies for data quality assessment and improvement," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–52, 2009.

[12] S. Mohammed, L. Ehrlinger, H. Harmouch, F. Naumann, and D. Srivastava, "The five facets of data quality assessment," *ACM SIGMOD Record*, vol. 54, no. 2, pp. 18–27, 2025.

[13] E. K. Rezig, M. Ouzzani, W. G. Aref, A. K. Elmagarmid, A. R. Mahmood, and M. Stonebraker, "Horizon: Scalable dependency-driven data cleaning," *Proceedings of the VLDB Endowment*, vol. 14, no. 11, pp. 2546–2554, 2021.

[14] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis, "Conditional functional dependencies for capturing data inconsistencies," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 2, pp. 1–48, 2008.

[15] W. Fan, "Dependencies revisited for improving data quality," in *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2008, pp. 159–170.

[16] L. Koumarelas, T. Papenbrock, and F. Naumann, "Mdedup: Duplicate detection with matching dependencies," *Proceedings of the VLDB Endowment*, vol. 13, no. 5, pp. 712–725, 2020.

[17] X. Chu, I. F. Ilyas, and P. Papotti, "Discovering denial constraints," *Proceedings of the VLDB Endowment*, vol. 6, no. 13, pp. 1498–1509, 2013.

[18] T. Bleifuß, S. Kruse, and F. Naumann, "Efficient denial constraint discovery with hydra," *Proceedings of the VLDB Endowment*, vol. 11, no. 3, pp. 311–323, 2017.

[19] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré, "Holoclean: Holistic data repairs with probabilistic inference," *arXiv preprint arXiv:1702.00820*, 2017.

[20] E. H. Pena, E. C. De Almeida, and F. Naumann, "Discovery of approximate (and exact) denial constraints," *Proceedings of the VLDB Endowment*, vol. 13, no. 3, pp. 266–278, 2019.

[21] M. Dallachiesa, A. Ebaid, A. Eldawy, *et al.*, "Nadeef: A commodity data cleaning system," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 541–552.

[22] Z. Khayyat, I. F. Ilyas, A. Jindal, *et al.*, "Bigdansing: A system for big data cleansing," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, 2015, pp. 1215–1230.

[23] X. Chu, I. F. Ilyas, and P. Papotti, "Holistic data cleaning: Putting violations into context," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, IEEE, 2013, pp. 458–469.

[24] Z. Huang and Y. He, "Auto-detect: Data-driven error detection in tables," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1377–1392.

[25] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer, "Wrangler: Interactive visual specification of data transformation scripts," in *Proceedings of the sigchi conference on human factors in computing systems*, 2011, pp. 3363–3372.

[26] X. Chu, J. Morcos, I. F. Ilyas, *et al.*, "Katara: A data cleaning system powered by knowledge bases and crowdsourcing," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, 2015, pp. 1247–1261.

[27] M. X. Ma, H. Y. Ngan, and W. Liu, "Density-based outlier detection by local outlier factor on largescale traffic data," *Electronic Imaging*, vol. 28, pp. 1–4, 2016.

[28] A. Reddy, M. Ordway-West, M. Lee, *et al.*, "Using gaussian mixture models to detect outliers in seasonal univariate network traffic," in *2017 IEEE Security and Privacy Workshops (SPW)*, IEEE, 2017, pp. 229–234.

[29] C. Pit-Claudel, Z. Mariet, R. Harding, and S. Madden, "Outlier detection in heterogeneous datasets using automatic tuple expansion," Massachusetts Institute of Technology, Tech. Rep. MIT-CSAIL-TR-2016-002, 2016. [Online]. Available: `https://dspace.mit.edu/handle/1721.1/101150`.

[30] C. Mayfield, J. Neville, and S. Prabhakar, "Eracer: A database approach for statistical inference and data cleaning," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 75–86.

[31] Z. Abedjan, X. Chu, D. Deng, *et al.*, "Detecting data errors: Where are we and what needs to be done?" *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 993–1004, 2016.

[32] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, "Activeclean: Interactive data cleaning for statistical modeling," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 948–959, 2016.

[33] S. Krishnan, M. J. Franklin, K. Goldberg, and E. Wu, "Boostclean: Automated error detection and repair for machine learning," *arXiv preprint arXiv:1711.01299*, 2017.

[34] F. Neutatz, M. Mahdavi, and Z. Abedjan, "Ed2: A case for active learning in error detection," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 2249–2252.

[35] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas, "Holodetect: Few-shot learning for error detection," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 829–846.

[36] M. Pham, C. A. Knoblock, M. Chen, B. Vu, and J. Pujara, "Spade: A semi-supervised probabilistic approach for detecting errors in tables.," in *IJCAI*, 2021, pp. 3543–3551.

[37] L. Visengeriyeva and Z. Abedjan, "Metadata-driven error detection," in *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, 2018, pp. 1–12.

[38] R. Wang, Y. Li, and J. Wang, "Sudowoodo: Contrastive self-supervised learning for multi-purpose data integration and preparation," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, IEEE, 2023, pp. 1502–1515.

[39] J. Peng, D. Shen, T. Nie, and Y. Kou, "Rlclean: An unsupervised integrated data cleaning framework based on deep reinforcement learning," *Information Sciences*, vol. 682, p. 121 281, 2024.

[40] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pp. 353–374, 2023.

[41] S. Eduardo, A. Nazábal, C. K. Williams, and C. Sutton, "Robust variational autoencoders for outlier detection and repair of mixed-type data," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 4056–4066.

[42] R. Mauritz, F. Nijweide, J. Goseling, and M. van Keulen, "A probabilistic database approach to autoencoder-based data cleaning," *arXiv preprint arXiv:2106.09764*, 2021.

[43] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[44] N. Tang, J. Fan, F. Li, *et al.*, "Rpt: Relational pre-trained transformer is almost all you need towards democratizing data preparation," *arXiv preprint arXiv:2012.02469*, 2020.

[45] Z. Liu, Z. Zhou, and T. Rekatsinas, "Picket: Guarding against corrupted data in tabular data during learning and inference," *The VLDB Journal*, vol. 31, no. 5, pp. 927–955, 2022.

[46] A. Narayan, I. Chami, L. Orr, S. Arora, and C. Ré, "Can foundation models wrangle your data?" *arXiv preprint arXiv:2205.09911*, 2022.

[47] G. Jaimovitch-López, C. Ferri, J. Hernández-Orallo, F. Martínez-Plumed, and M. J. Ramírez-Quintana, "Can language models automate data wrangling?" *Machine Learning*, vol. 112, no. 6, pp. 2053–2082, 2023.

[48] H. Zhang, Y. Dong, C. Xiao, and M. Oyamada, "Large language models as data preprocessors," *arXiv preprint arXiv:2308.16361*, 2023.

[49] Z. Huang and E. Wu, "Cocoon: Semantic table profiling using large language models," in *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics*, 2024, pp. 1–7.
[50] S. Zhang, Z. Huang, and E. Wu, "Data cleaning using large language models," *arXiv preprint arXiv:2410.15547*, 2024.
[51] D. Vos, T. Döhmen, and S. Schelter, "Towards parameter-efficient automation of data wrangling tasks with prefix-tuning," in *NeurIPS 2022 First Table Representation Workshop*, 2022, pp. 1–9.
[52] Z. Zhang, P. Groth, I. Calixto, and S. Schelter, "Directions towards efficient and automated data wrangling with large language models," in *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*, IEEE, 2024, pp. 301–304.
[53] H. Zhang, Y. Dong, C. Xiao, and M. Oyamada, "Jellyfish: A large language model for data preprocessing," *arXiv preprint arXiv:2312.01678*, 2023.
[54] P. Mehra *et al.*, "Leveraging structured and unstructured data for tabular data cleaning," in *2024 IEEE International Conference on Big Data (BigData)*, IEEE, 2024, pp. 5765–5768.
[55] L. Liu, S. Hasegawa, S. K. Sampat, *et al.*, "Autodw: Automatic data wrangling leveraging large language models," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 2041–2052.
[56] X. Li and T. Döhmen, "Towards efficient data wrangling with llms using code generation," in *Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning*, 2024, pp. 62–66.
[57] T. Bendinelli, A. Dox, and C. Holz, "Exploring LLM agents for cleaning tabular machine learning datasets," in *ICLR 2025 Workshop on Foundation Models in the Wild*, 2025.
[58] F. Biester, M. Abdelaal, and D. Del Gaudio, "Llmclean: Context-aware tabular data cleaning via llm-generated ofds," in *European Conference on Advances in Databases and Information Systems*, Springer, 2024, pp. 68–78.
[59] Z. A. Naeem, M. S. Ahmad, M. Eltabakh, M. Ouzzani, and N. Tang, "Retclean: Retrieval-based data cleaning using llms and data lakes," *Proceedings of the VLDB Endowment*, vol. 17, no. 12, pp. 4421–4424, 2024.
[60] W. Ni, K. Zhang, X. Miao, *et al.*, "Zeroed: Hybrid zero-shot error detection through large language model reasoning," *arXiv preprint arXiv:2504.05345*, 2025.
[61] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 39.
[62] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, K. Erk and N. A. Smith, Eds., Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725.
[63] Y. Wu, M. Schuster, Z. Chen, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.
[64] C. Wang, K. Cho, and J. Gu, "Neural machine translation with byte-level subwords," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 9154–9160.
[65] F. Neutatz, M. Mahdavi, and Z. Abedjan, "Ed2: Two-stage active learning for error detection–technical report," *arXiv preprint arXiv:1908.06309*, 2019.
[66] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," OpenAI, Tech. Rep., 2018. [Online]. Available: `https://openai.com/index/language-unsupervised/`.

[67] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
[68] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
[69] J. Achiam, S. Adler, S. Agarwal, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.
[70] G. Team, R. Anil, S. Borgeaud, *et al.*, "Gemini: A family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.
[71] A. Chowdhery, S. Narang, J. Devlin, *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.
[72] H. Touvron, T. Lavril, G. Izacard, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.
[73] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.
[74] A. Liu, B. Feng, B. Xue, *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.
[75] D. Guo, D. Yang, H. Zhang, *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.
[76] A. Q. Jiang, A. Sablayrolles, A. Mensch, *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.
[77] J. Bai, S. Bai, Y. Chu, *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.
[78] H. Gong, Y. Sun, X. Feng, *et al.*, "Tablegpt: Few-shot table-to-text generation with table structure reconstruction and content matching," in *Proceedings of the 28th International Conference on Computational Linguistics*, 2020, pp. 1978–1988.
[79] X. Zhang, S. Luo, B. Zhang, *et al.*, "Tablellm: Enabling tabular data manipulation by llms in real office usage scenarios," *arXiv preprint arXiv:2403.19318*, 2024.
[80] P. Li, Y. He, D. Yashar, *et al.*, "Table-gpt: Table fine-tuned gpt for diverse table tasks," *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–28, 2024.
[81] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, "Tabert: Pretraining for joint understanding of textual and tabular data," *arXiv preprint arXiv:2005.08314*, 2020.
[82] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos, "Tapas: Weakly supervised table parsing via pre-training," *arXiv preprint arXiv:2004.02349*, 2020.
[83] O. Chapelle, J. Weston, and B. Schölkopf, "Cluster kernels for semi-supervised learning," *Advances in neural information processing systems*, vol. 15, 2002.
[84] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 1177–1178.
[85] J. Wei, M. Bosma, V. Y. Zhao, *et al.*, "Finetuned language models are zero-shot learners," *arXiv preprint arXiv:2109.01652*, 2021.
[86] L. Ouyang, J. Wu, X. Jiang, *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[87] S. Hegselmann, A. Buendia, H. Lang, M. Agrawal, X. Jiang, and D. Sontag, "Tabllm: Few-shot classification of tabular data with large language models," in *International conference on artificial intelligence and statistics*, PMLR, 2023, pp. 5549–5581.

[88] Y. Sui, M. Zhou, M. Zhou, S. Han, and D. Zhang, "Table meets llm: Can large language models understand structured table data? a benchmark and empirical study," in *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, 2024, pp. 645–654.

[89] Z. Cheng, J. Kasai, and T. Yu, "Batch prompting: Efficient inference with large language model apis," *arXiv preprint arXiv:2301.08721*, 2023.

[90] J.-N. Hould, *Craft beers dataset*, `https://www.kaggle.com/nickhould/craft-cans`, Version 1, 2017.

[91] A. Shaw, *Top spotify listening history songs in countries*, `https://www.kaggle.com/datasets/anandshaw2001/top-spotify-songs-in-countries/data`, accessed August 5th, 2025.

[92] A. Grattafiori, A. Dubey, A. Jauhri, *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[93] A. Yang, A. Li, B. Yang, *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[94] G. Team, A. Kamath, J. Ferret, *et al.*, "Gemma 3 technical report," *arXiv preprint arXiv:2503.19786*, 2025.

[95] J. Kaplan, S. McCandlish, T. Henighan, *et al.*, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[96] Z. Ji, N. Lee, R. Frieske, *et al.*, "Survey of hallucination in natural language generation," *ACM computing surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[97] P. Lewis, E. Perez, A. Piktus, *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[98] W. Zhong, L. Guo, Q. Gao, H. Ye, and Y. Wang, "Memorybank: Enhancing large language models with long-term memory," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 19 724–19 731.

[99] J. Wei, X. Wang, D. Schuurmans, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[100] X. Wang, J. Wei, D. Schuurmans, *et al.*, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[101] L. Zheng, W.-L. Chiang, Y. Sheng, *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *Advances in neural information processing systems*, vol. 36, pp. 46 595–46 623, 2023.

[102] M. Xiong, Z. Hu, X. Lu, *et al.*, "Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms," *arXiv preprint arXiv:2306.13063*, 2023.

[103] L. Zhong, Z. Wang, and J. Shang, "Debug like a human: A large language model debugger via verifying runtime execution step-by-step," *arXiv preprint arXiv:2402.16906*, 2024.

[104] Z. Gou, Z. Shao, Y. Gong, *et al.*, "Critic: Large language models can self-correct with tool-interactive critiquing," *arXiv preprint arXiv:2305.11738*, 2023.

[105] X. Yu, B. Peng, M. Galley, J. Gao, and Z. Yu, "Teaching language models to self-improve through interactive demonstrations," *arXiv preprint arXiv:2310.13522*, 2023.

[106] V. Liventsev, A. Grishina, A. Härmä, and L. Moonen, "Fully autonomous programming with large language models," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023, pp. 1146–1155.

[107] Y. Zhang, M. Khalifa, L. Logeswaran, *et al.*, "Small language models need strong verifiers to self-correct reasoning," *arXiv preprint arXiv:2404.17140*, 2024.

[108] M. Hassid, T. Remez, J. Gehring, R. Schwartz, and Y. Adi, "The larger the better? improved llm code-generation via budget reallocation," *arXiv preprint arXiv:2404.00725*, 2024.

A

Manually Defined Rule Prompts

A.1. Rule for Hospital Dataset

```
{
  "columns": [
    {
      "name": "index",
      "meaning": "Unique identifier for each row in the dataset.",
      "data_type": "integer",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "provider_number",
      "meaning": "Identifier for healthcare providers, possibly a unique code or
          number.",
      "data_type": "string",
      "format_rule": "5-digit number",
      "null_value_rule": "not allowed"
    },
    {
      "name": "name",
      "meaning": "Name of the healthcare provider.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "address_1",
      "meaning": "First part of the address for the healthcare provider.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "city",
      "meaning": "City where the healthcare provider is located.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
```

```
    },
    {
      "name": "state",
      "meaning": "State where the healthcare provider is located.",
      "data_type": "string",
      "format_rule": "2-letter state code",
      "null_value_rule": "not allowed"
    },
    {
      "name": "zip",
      "meaning": "Zip code for the healthcare provider's location.",
      "data_type": "string",
      "format_rule": "5-digit zip code",
      "null_value_rule": "not allowed"
    },
    {
      "name": "county",
      "meaning": "County where the healthcare provider is located.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "phone",
      "meaning": "Phone number for the healthcare provider.",
      "data_type": "string",
      "format_rule": "10-digit phone number",
      "null_value_rule": "not allowed"
    },
    {
      "name": "type",
      "meaning": "Type of healthcare provider.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "owner",
      "meaning": "Ownership type of the healthcare provider.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "emergency_service",
      "meaning": "Whether the healthcare provider offers emergency services.",
      "data_type": "boolean",
      "format_rule": "yes or no",
      "null_value_rule": "not allowed"
    },
    {
      "name": "condition",
      "meaning": "Condition being measured for each healthcare provider.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
```

```
    },
    {
      "name": "measure_code",
      "meaning": "Unique code for the measure being reported.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "measure_name",
      "meaning": "Name of the measure being reported, possibly derived from
          'condition' and 'measure_code'.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "score",
      "meaning": "Score or percentage for the measure being reported.",
      "data_type": "float",
      "format_rule": "percentage (e.g., '95%', '64%')",
      "null_value_rule": "allowed values include string 'empty' for missing
          numeric data"
    },
    {
      "name": "sample",
      "meaning": "Number of patients included in the measure being reported.",
      "data_type": "string",
      "format_rule": "integer + 'patients' or 'patient'",
      "null_value_rule": "allowed values include string 'empty' for missing
          numeric data"
    },
    {
      "name": "state_average",
      "meaning": "Average score for the measure across all providers in a state,
          possibly derived from 'measure_code' and 'state'.",
      "data_type": "string",
      "format_rule": "combination of measure_code and state (e.g.,
          'state_measure_code')",
      "null_value_rule": "not allowed"
    }
  ]
}
```

A.2. Rule for Flights Dataset

```
{
  "columns": [
    {
      "name": "index",
      "meaning": "Unique identifier for each row in the dataset.",
      "data_type": "integer",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
```

```
    {
      "name": "tuple_id",
      "meaning": "A unique identifier for each row in the dataset.",
      "data_type": "integer",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "src",
      "meaning": "The source or origin of the data entry, such as a website,
          organization, or tracking system.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "flight",
      "meaning": "The flight identifier, which includes the airline code, flight
          number, and origin-destination pair.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "sched_dep_time",
      "meaning": "The scheduled departure time of the flight, formatted as
          'HH:MM a.m./p.m.'.",
      "data_type": "string",
      "format_rule": "HH:MM a.m./p.m.",
      "null_value_rule": "not allowed"
    },
    {
      "name": "act_dep_time",
      "meaning": "The actual departure time of the flight, formatted as 'HH:MM
          a.m./p.m.'.",
      "data_type": "string",
      "format_rule": "HH:MM a.m./p.m.",
      "null_value_rule": "not allowed"
    },
    {
      "name": "sched_arr_time",
      "meaning": "The scheduled arrival time of the flight, formatted as 'HH:MM
          a.m./p.m.'.",
      "data_type": "string",
      "format_rule": "HH:MM a.m./p.m.",
      "null_value_rule": "not allowed"
    },
    {
      "name": "act_arr_time",
      "meaning": "The actual arrival time of the flight, formatted as 'HH:MM
          a.m./p.m.'.",
      "data_type": "string",
      "format_rule": "HH:MM a.m./p.m.",
      "null_value_rule": "not allowed"
    }
  ]
```

```
}
```

A.3. Rule for Beers Dataset

```
{
  "columns": [
    {
      "name": "index",
      "meaning": "Unique identifier for each row in the dataset.",
      "data_type": "integer",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "id",
      "meaning": "Unique identifier for each beer entry.",
      "data_type": "integer",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "beer_name",
      "meaning": "Name of the beer.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "style",
      "meaning": "Classification of the beer style.",
      "data_type": "category",
      "format_rule": "no specific format",
      "null_value_rule": "allowed empty string '' for missing data"
    },
    {
      "name": "ounces",
      "meaning": "Volume of the beer in ounces.",
      "data_type": "integer",
      "format_rule": "Must be a plain integer without any units or symbols
          (e.g., 10, 18).",
      "null_value_rule": "not allowed"
    },
    {
      "name": "abv",
      "meaning": "Alcohol by volume percentage.",
      "data_type": "float",
      "format_rule": "Must be a plain float with up to 3 decimal places, no %
          symbols (e.g., 0.021, 0.01).",
      "null_value_rule": "allowed empty string '' for missing data"
    },
    {
      "name": "ibu",
      "meaning": "International Bitterness Unit, possibly with a numeric value.",
      "data_type": "integer",
      "format_rule": "no specific format",
```

```
      "null_value_rule": "allowed empty string '' for missing data"
    },
    {
      "name": "brewery_id",
      "meaning": "Unique identifier for each brewery entry.",
      "data_type": "integer",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "brewery_name",
      "meaning": "Name of the brewery, possibly with abbreviations or codes
          derived from other columns (e.g., city).",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "city",
      "meaning": "City where the brewery is located.",
      "data_type": "category",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "state",
      "meaning": "State where the brewery is located.",
      "data_type": "category",
      "format_rule": "two-letter state abbreviation (e.g., OR)",
      "null_value_rule": "not allowed"
    }
  ]
}
```

A.4. Rule for Spotify Dataset

```
{
  "columns": [
    {
      "name": "spotify_track_uri",
      "meaning": "Spotify URI that uniquely identifies each track in the form of
          \"spotify:track:<base-62 string>\"",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "ts",
      "meaning": "Timestamp indicating when the track stopped playing in UTC
          (Coordinated Universal Time)",
      "data_type": "datetime",
      "format_rule": "YYYY-MM-DD HH:MM:SS",
      "null_value_rule": "not allowed"
    },
    {
```

```
      "name": "platform",
      "meaning": "Platform used when streaming the track.",
      "data_type": "category",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "ms_played",
      "meaning": "Number of milliseconds the stream was played.",
      "data_type": "integer",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "track_name",
      "meaning": "The name of the musical track being played.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "artist_name",
      "meaning": "The name of the artist who performed the track.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "album_name",
      "meaning": "The name of the album to which the track belongs.",
      "data_type": "string",
      "format_rule": "no specific format",
      "null_value_rule": "not allowed"
    },
    {
      "name": "reason_start",
      "meaning": "Why the track started. E.g., fwdbtn, packbtn, trackdone,
          clickrow, appload...",
      "data_type": "category",
      "format_rule": "no specific format",
      "null_value_rule": "allowed empty string '' for missing data"
    },
    {
      "name": "reason_end",
      "meaning": "Why the track ended. E.g., fwdbtn, packbtn, trackdone,
          clickrow, appload...",
      "data_type": "category",
      "format_rule": "no specific format",
      "null_value_rule": "allowed empty string '' for missing data"
    },
    {
      "name": "shuffle",
      "meaning": "A boolean flag indicating whether the track was played as part
          of a shuffled playlist.",
      "data_type": "boolean",
      "format_rule": "TRUE or FALSE",
```

```
      "null_value_rule": "not allowed"
    },
    {
      "name": "skipped",
      "meaning": "A boolean flag indicating whether the track was skipped by the
          user.",
      "data_type": "boolean",
      "format_rule": "TRUE or FALSE",
      "null_value_rule": "not allowed"
    }
  ]
}
```