Automated Data Quality Validation in an End-to-End GNN Framework

Sijie Dong
Université Paris Cité
Paris, France
sijie.dong@etu.u-paris.fr

Soror Sahri
Université Paris Cité
Paris, France
soror.sahri@parisdescartes.fr

Themis Palpanas
Université Paris Cité
Paris, France
themis@mi.parisdescartes.fr

Qitong Wang
Harvard University
Boston, USA
qitong@seas.harvard.edu

ABSTRACT

Ensuring data quality is crucial in modern data ecosystems, especially for training or testing datasets in machine learning. Existing validation approaches rely on computing data quality metrics and/or using expert-defined constraints. Although there are automated constraint generation methods, they are often incomplete and may be too strict or too soft, causing false positives or missed errors, thus requiring expert adjustment. These methods may also fail to detect subtle data inconsistencies hidden by complex interdependencies within the data. In this paper, we propose DQuag, an end-to-end data quality validation and repair framework based on an improved Graph Neural Network (GNN) and multi-task learning. The proposed method incorporates a dual-decoder design: one for data quality validation and the other for data repair. Our approach captures complex feature relationships within tabular datasets using a multi-layer GNN architecture to automatically detect explicit and hidden data errors. Unlike previous methods, our model does not require manual input for constraint generation and learns the underlying feature dependencies, enabling it to identify complex hidden errors that traditional systems often miss. Moreover, it can recommend repair values, improving overall data quality. Experimental results validate the effectiveness of our approach in identifying and resolving data quality issues.

1 INTRODUCTION

In the era of artificial intelligence and large models, ensuring data quality is crucial. High-quality data is essential for reliable decision-making, particularly in machine learning, where data quality directly impacts model performance. This paper addresses data quality validation, focusing on innovative approaches that overcome the limitations of existing methods.

Background: Quality validation is important to ensure that data meets specific requirements. Its main steps start with data profiling [5], generating metadata to describe quality like missing values and data dependencies [25, 27]. This metadata forms the basis for assessing data quality against established metrics. However, this often relies on expert-defined constraints, which are resource-intensive and time-consuming to develop, maintain, and adjust.

	Name	Age	City	Country	Occupation	Income Type	Error Type
1	John	110	New York	USA	Truck Driver	Working	Anomalies
2	Jane	40	Sydney	Australia	Softwera Engneer	Working	Typo
3	Alice	35	Toronto	USA	Marketing Specialist	Working	Conflict
4	Bob	30	London	UK	Chef	Pensioner	Conflict

Figure 1: Examples of Data Errors in tabular data: anomalies, typos, and conflicts between attributes.

Automated constraint generation [9, 35] attempts to learn from dataset characteristics, but these constraints are often incomplete or either too strict or too soft. Strict constraints can lead to false positives, such as flagging acceptable minor data entry variations as errors. While soft constraints may miss critical issues, such as overlooking small discrepancies in numerical data. They need expert intervention for fine-tuning, which limits practical effectiveness.

Furthermore, these automated constraint generation methods often fail to detect hidden relationships and dependencies within the data, which we refer to as hidden errors. Indeed, data quality involves not only individual data points but also the complex interrelationships among them.

Motivation: To motivate our work, consider a dirty tabular dataset with four data quality errors that require verification. Figure 1 shows common data errors: numeric anomalies, typos, and attribute mismatches. Tuple 1 includes a numeric anomalies value in the Age field, and tuple 2 has a typo in the Occupation field. Existing data quality validation systems, such as Amazon Deequ [35] and Google TFDV [9], can automatically generate constraints (e.g., the *age* should be less than 100) to detect errors in tuples 1 and 2. However, these constraints are often difficult to set accurately, as valid cases may exist where age exceeds 100. In tuple 1, being employed as a truck driver at that age is highly unlikely and should be flagged as an error. Traditional systems often set such constraints either too strictly or too leniently, resulting in false positives or missed errors. Tuple 3 contains a simple dependency issue, such as a city listed as *Toronto* while the country is listed as the *USA*. This type of error can often be addressed through expert-defined constraints. However, tuple 4 involves deeper dependencies, which is an illogical combination of age and income, referred to as hidden errors. Although data profiling [5, 25, 27] methods can discover such dependencies, existing automated data quality validation approaches only rely

on expert-based strategies to identify such dependencies. As a result, they frequently overlook errors like those in tuple 4.

Our goal is to detect challenging errors, such as hidden errors (e.g., tuple 4), without relying on expert input, while also detecting ordinary errors like missing data and typos. Additionally, we provide suggestions for repairing the identified errors.

Challenges: Addressing the limitations of existing methods involves several challenges. First, obtaining expert input for defining constraints is costly and often impractical, making an automated approach essential. Second, existing methods struggle to detect subtle issues embedded within complex feature relationships, which we refer to as hidden data quality errors. Third, the approach must be capable of identifying which specific features of a sample are contributing to the data quality issue, ensuring a precise assessment of problematic areas. Lastly, the model should provide suggestions to repair problematic features, while maintaining consistency with the overall clean data distribution.

Our Solution: To address these challenges, we propose DQuag, a multi-task learning framework, which integrates a GNN encoder [23] for feature embedding and dual decoders for data quality validation and repair. This combination leverages graph-based learning and multi-task optimization to detect and repair data quality issues.

The method includes the following key steps in the training phase: First, we construct a feature graph from clean tabular data to capture relationships between features. This graph representation enables the model to leverage relational dependencies inherent in the data. Next, a GNN encoder generates feature embeddings that encapsulate these relationships. The generated feature embeddings are then passed to two separate decoders. The Data Quality Validation Decoder reconstructs the original data from the feature embeddings, aiming to maximize the reconstruction accuracy for clean data and identify errors based on reconstruction loss. The Data Repair Decoder generates corrected values for problematic features. These decoders are optimized with distinct loss functions.

During the verification phase, we apply the trained model to unseen datasets and use reconstruction errors to measure data quality, with large errors indicating potential data quality issues. The repair decoder then suggests corrections for these features. This approach detects both explicit and underlying data quality problems, reducing the need for expert-defined constraints and manual intervention.

Contributions: This paper makes three key contributions. First, we introduce a novel advanced data quality validation and repair framework based on graph neural networks and multi-task learning, eliminating the need for expert-defined constraints. Second, we develop a novel dual-decoder architecture to address two critical tasks: data quality validation and data repair. The Data Quality Validation Decoder is designed to highlight potential quality issues, while the Data Repair Decoder provides correction suggestions. Finally, we validate our approach through comprehensive and extensive experiments, thoroughly demonstrating its effectiveness and adaptability across various scenarios.

2 RELATED WORK

Research in data quality has traditionally emphasized the calculation of data quality dimensions and statistics to assess data quality. Fundamental components of this field include data profiling, which identifies issues like missing values and syntax violations [10, 18, 30], and data quality assessment, which measures attributes such as accuracy, completeness, and consistency [38, 44] to ensure datasets meet standards applicable to various domains [19, 20, 47].

Several studies have investigated data quality within ML pipelines, examining its influence on ML performance [7], methods for high-quality data preparation [15, 17], ensuring data completeness [21, 36], detecting data drift [12, 41, 45], and validating data quality [9, 14, 35, 39]. Frameworks like Dagger [33] and Mlinspect [16] incorporate advanced debugging and interactive query mechanisms to streamline systematic data management. Furthermore, tools such as HoloClean [32] and ActiveClean [22] extend these efforts to data cleaning and repair, leveraging probabilistic models and active learning for iterative dataset refinement.

Existing quality validation systems typically use statistical methods or statistics to train machine learning models for detecting data quality problems. These methods often rely on predefined rules and constraints that require expert adjustment [14, 39], e.g., automated data quality verification systems, such as Deequ [35] and TFDV [9], which generate constraints automatically. However, these constraints are not always accurate and often need expert fine-tuning to effectively detect hidden data relationships and dependencies [34]. Other systems like ADQV [31] use k-nearest neighbors to evaluate data quality by computing limited metrics, but fail to detect hidden errors and cannot pinpoint the incorrect samples.

Our approach uses a multi-task learning framework that combines GNN encoder [23] and dural decoders. While GNNs and other machine learning models are used for data imputation [8, 40, 42], anomaly detection [13], and fraud detection [24, 29], their use in an integrated manner for direct data quality validation remains underexplored. Our work applies GNN to automatically validate data quality, focusing on both explicit data errors and complex interdependencies without relying on expert-defined rules and manual adjustments and optimizes for both detection and repair tasks, setting our approach apart from conventional data management techniques that typically rely on isolated, task-specific methods.

3 OUR APPROACH: DQUAG

In this section, we present DQuaG (Data Quality Graph), a novel approach for data quality validation. Figure 2 illustrates the framework of our approach, which includes two main phases: model training on a clean dataset and data quality validation and repair for new data. In Phase 1, we train a model using a clean dataset to learn the normal patterns and relationships between features. In Phase 2, we use the trained model to assess the quality of new data and provide repair suggestions for any detected errors.

3.1 Phase 1: Training GNN on Clean Data

We assume the availability of a high-quality, clean dataset, $\mathcal{D}_{\text{clean}}$, that has undergone rigorous quality control and is free from errors. This dataset serves as the foundation for training our model. For feature encoding and normalization, categorical features are converted to numerical form using label encoding, where the encoder is fitted on both clean data and any possible future data to ensure consistency. For numerical features, we apply min-max normalization to scale values to the range [0, 1], which helps improve training stability and ensures that all features are on a comparable scale.

3.1.1 *Feature Graph Construction*. We use ChatGPT-4 [28] to automate the feature graph construction. Given a clean dataset,
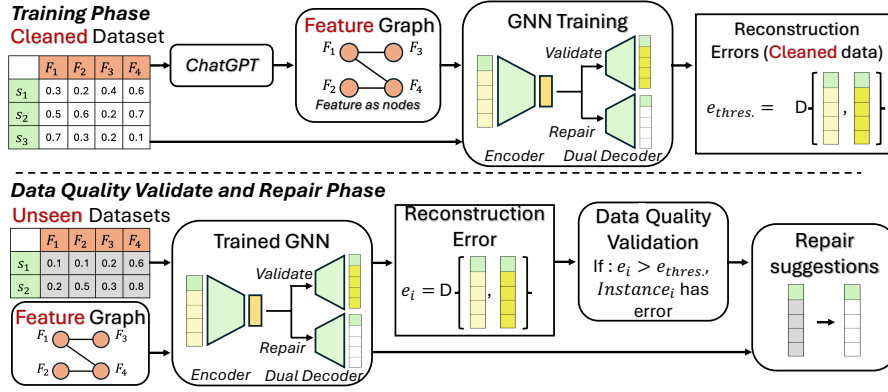
Figure 2: Data Quality Validation Framework Using GNN. Top: Training on clean data. Bottom: Validating unseen data by reconstruction error comparison.

we extract the feature names (F) and their descriptions (D) from the data source. We then randomly sample 100 data points from the dataset, denoted as (S). These feature names, descriptions, and sample points are provided to ChatGPT-4 in a structured format to infer potential relationships between features. The output from ChatGPT-4 is a JSON file capturing feature relationships, which we denote as Feature_Relationships = $\{(f_i, f_j) \mid f_i, f_j \in F\}$, indicating that there is a relationship between features f_i and f_j.

Prompt for Feature Relationship Inference

Given the following information, please infer the relationships between features. Provide your output in JSON format, capturing the type of relationships.
Feature Names: List of feature names (F)
Feature Descriptions: List of descriptions (D) for each feature
Sample Data Points: 100 data samples (S) from the dataset
Output: Please return a JSON object in the format:

```
{"relationships": [{"feature1", "feature2"},
                   {"feature3", "feature4"}, ...]}
```

Using these relationships, we construct the knowledge-based feature graph $G = (V, E)$, where V represents features and E represents edges indicating relationships between features.

3.1.2 ***GNN Model Architecture***. Our model architecture combines the strengths of different graph neural network variants to effectively capture complex feature relationships. The architecture consists of three main components: an improved GNN encoder that fuses Graph Attention Network (GAT) [43] layers and Graph Isomorphism Network (GIN) [46] layers, and two specialized decoders for quality validation and repair suggestion generation.

GNN Encoder (GAT + GIN). Our encoder consists of four layers: alternating *Graph Attention Network (GAT)* and *Graph Isomorphism Network (GIN)* layers, in the order of GAT-GIN-GAT-GIN. This design is inspired by recent findings in the field of graph representation learning that demonstrate how combining different types of graph layers can yield improved performance in feature extraction and relational representation tasks [48]. Our experimental results demonstrate the advantages of this structure.

The GAT layers compute attention weights between connected features, enabling the model to adaptively assign importance to significant relationships in the data. This allows the model to focus on critical connections and ignore irrelevant information,

which enhances its ability to learn meaningful feature representations. Our approach uses GAT layers, which automatically learn edge weights through attention mechanisms during training. This eliminates the need to manually assign weights in the initial feature graph.

The GIN layers aggregate feature information from neighboring nodes to capture structural information more effectively. By using GIN, the encoder gains a strong ability to represent the underlying structure of the data, preserving key relationships crucial for data quality validation and repair tasks.

This alternating GAT and GIN structure enhances the model's ability to both prioritize important features and learn intricate structural relationships, thereby making it more effective at representing complex feature dependencies in the data. Specifically, the GNN encoder processes the feature graph $G = (V, E)$ along with the input data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the number of nodes (features) and d is the dimensionality of each feature vector. The output from the GNN encoder is a feature embedding matrix $\mathbf{Z} \in \mathbb{R}^{n \times h}$, where h represents the size of the learned feature embeddings.

Dual Decoder Structure. Our model employs two separate decoders to address the tasks of Data Quality Validation and Repair Suggestion, enabling focused optimization for each objective.

Data Quality Validation Decoder is responsible for reconstructing the original feature space from the learned embeddings, denoted as $\mathbf{Z}$. The primary objective of this decoder is to learn the correct patterns from clean data and reconstruct the features in a way that captures the underlying structure of the dataset. This allows us to identify abnormalities by measuring reconstruction errors. We have designed a unique loss function that ensures the model focuses on learning accurate representations of clean data while effectively distinguishing abnormal samples.

For normal data samples, the decoder should ideally have a low reconstruction error, as the learned embeddings should effectively capture the true relationships between the features, resulting in an accurate reconstruction. For abnormal samples, the reconstruction error will be higher, indicating that these samples do not conform to the learned patterns from the clean data. The reconstruction loss is defined as $L_{\text{validation}} = \frac{1}{N} \sum_{i=1}^{N} w_i \left\| \mathbf{X}_i - \hat{\mathbf{X}}_i \right\|_2^2$, where $\mathbf{X}_i$ represents the original input features, and $\hat{\mathbf{X}}_i$ is the reconstructed feature vector for the i-th sample. The weights w_i are assigned to each sample based on its reconstruction error.

We assign larger weights to normal data samples (w_i s higher for samples with smaller reconstruction errors), giving them a greater influence in minimizing their reconstruction loss. This encourages the model to accurately reconstruct the normal data and effectively learn the correct data distribution. For samples with potential quality issues, the weights w_i are reduced, meaning that their influence on the overall loss is diminished. This allows the model to focus on minimizing the reconstruction loss for normal data while maintaining high reconstruction errors for problematic samples during the backpropagation. By using this weighting mechanism, we ensure that the validation decoder can distinguish between normal data and data with potential issues based on reconstruction errors.

Data Repair Decoder, on the other hand, takes the same learned embeddings $\mathbf{Z}$ as input, but its goal is different: it aims to suggest repaired values for features identified as erroneous. Unlike the Data Quality Validation Decoder, which reconstructs data to highlight discrepancies, the Data Repair Decoder attempts to produce an output that aligns with the clean, underlying data distribution, effectively suggesting corrections for the detected errors. The objective of this decoder is defined through the following loss function: $L_{\text{repair}} = \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{X}_i - \tilde{\mathbf{X}}_i \right\|_2^2$. Here, $\tilde{\mathbf{X}}_i$ represents the feature values repaired by the decoder, while $\mathbf{X}_i$ stands for the corresponding clean feature values from the input dataset. Since the input is already clean, $\mathbf{X}_i$ can directly serve as the target for the repair task.

The combination of these two decoders is essential for effectively handling data quality issues. The overall loss function is a weighted sum of the validation and repair losses: $L_{\text{total}} = \alpha L_{\text{validation}} + \beta L_{\text{repair}}$, where α and β are hyperparameters used to balance the contributions of reconstruction and repair, both of which are set to 1 in our experiments.

The two decoders serve different purposes: the *Data Quality Validation Decoder* is optimized to detect data issues by maximizing reconstruction errors for problematic instances, while the *Data Repair Decoder* aims to provide realistic corrections for identified issues. By separating these tasks, the model avoids conflicting optimization goals, ensuring it is both effective at identifying problems and providing reliable repairs.

Multi-Task Learning Framework. The encoder is shared between the quality validation and repair tasks, while each task-specific decoder learns independently. This multi-task framework enables the model to exploit shared information between these tasks, allowing the model to learn a unified representation beneficial for both.

3.1.3 **Training Process**. We train the model on the clean dataset using an optimizer Adam to minimize L_{total}.

3.1.4 **Collecting the statistics of reconstruction errors.** During training, we record the reconstruction error for each instance. The reconstruction error is essentially the loss for each instance. Let e_i denote the reconstruction error for instance i, and let $\mathcal{E}$ be the set of all reconstruction errors from the clean dataset. Given that even cleaned datasets may contain undetected errors, we do not set the maximum reconstruction error as the threshold for identifying problematic instances. Instead, we set the threshold at the 95th percentile of $\mathcal{E}$, denoted as $e_{threshold}$.

Instances in the next phase with reconstruction errors above $e_{threshold}$ are flagged as potentially problematic.

3.2 Phase 2: Data Quality Validate and Repair

3.2.1 **Data Quality Validation Process**. In this phase, we validate the quality of incoming data by comparing it to the patterns learned during model training.

The new unseen data is preprocessed in the same manner as the clean dataset to ensure consistency in feature encoding, normalization, and feature graph construction. These new unseen datasets must keep the same schema as the original clean dataset. **Detecting Data Quality Issues by Reconstruction Errors**. After preprocessing, the model uses the validation decoder to reconstruct the features of the new data. For each data instance, the reconstruction error e_i is calculated. We then obtain a list of reconstruction errors, denoted as $\mathcal{E}_{\text{new}}$. Next, we compare each reconstruction error in $\mathcal{E}_{\text{new}}$ with the threshold $e_{threshold}$ from the clean dataset. We calculate the proportion of instances in the new dataset with reconstruction errors exceeding $e_{threshold}$, denoted as R_{error}. Since the threshold was set at the 95th percentile for the clean dataset, we expect around 5% of clean data instances to exceed this value.

To account for data variability, if R_{error} exceeds $5\% \times n$, we classify the new dataset as problematic. This means if more than $5n\%$ of instances in the new dataset have errors greater than $e_{threshold}$, we will report the dataset has data quality issues. The parameter n can be adjusted based on observed reconstruction errors after deployment. In our experiments, we set $n = 1.2$, which exhibited good performance. Finally, we report the indices of all instances in the new dataset with reconstruction errors above $e_{threshold}$, clearly identifying problematic samples. **Detecting Feature Errors**. Each instance's reconstruction error e is a list corresponding to each feature's loss. To identify specific problematic features, we detect outliers with significantly higher reconstruction errors. For an instance $\mathbf{x}_i$, let $\mathbf{e}_i = [e_{i1}, e_{i2}, \ldots, e_{in}]$ be the reconstruction errors for the n features. We calculate the mean μ_i and standard deviation σ_i of the errors. Features with errors greater than $\mu_i + 5\sigma_i$ are flagged as problematic.

3.2.2 **Repair Suggestion Generation**. In this phase, we provide repair suggestions for detected errors to improve the quality of the data for downstream use. The repair decoder is used to generate a repaired feature vector, which includes suggested repaired values for all features. In the previous step, we flagged which specific instances and features were problematic. Then we selectively apply modifications only to the flagged problematic features. For categorical features, the repair decoder predicts the most likely corrected category, while for numerical features, it predicts a value that aligns with the learned data distribution.

4 EXPERIMENTAL EVALUATION

4.1 Experimental Setup

The experiments were conducted using Python 3.11 and PyTorch 1.12.1. All computations were performed on an NVIDIA A100 GPU. The source code and data have been made available[1].

4.1.1 *Datasets.* We evaluate the robustness and generality of our approach using datasets with varied error types and data structures, following methodologies outlined in prior research [31]. *Datasets with ground-truth errors.* **Airbnb Data[1]** which contains information about Airbnb listings in New York City, including attributes such as price, location, and property type; and **Chicago Divvy Bicycle Sharing Data[2]** includes trip data

---

[1]Source code and data: https://github.com/SiSijie/DQuaG

from the Divvy bike-sharing program in Chicago, with details like the trip duration, start and end locations, and bike ID. **Google Play Store Apps Data[11]** includes app data, capturing ratings, downloads, categories, and trends in app performance. For these datasets, we have uncleaned versions with real-world errors. We involved data-cleaning techniques to create cleaned version datasets, such as removing duplicates, handling missing data, and filtering out illogical records.

Datasets without ground-truth errors. **New York Taxi Trip Data[4]** which comprises taxi trip records in New York City, detailing pickup and dropoff locations, fares, and trip durations; **Hotel Booking Data[6]** contains booking information for a city hotel and a resort hotel; and **Credit Card Data[3]** includes information on credit card applications. For these datasets, we directly utilized clean versions of data from reliable sources [3, 4, 6]. These datasets were publicly released, carefully collected, and cleaned before use. From these clean versions, we generated four types of data errors to create the corresponding dirty version datasets.

Clean datasets are not entirely error-free. Clean data is defined as having higher quality than dirty datasets and meeting the user's standards.

4.1.2 Synthetic Errors in Datasets without ground-truth errors. We simulate three ordinary errors and propose two potential hidden errors to evaluate the effectiveness of our approach:

Ordinary Errors: We introduce three types of errors affecting 20% of values in three selected attributes: missing Values arise from empty cells due to collection or integration errors; numeric anomalies occur when sensor malfunctions or scaling issues result in out-of-range values; and String typos are caused by spelling errors simulated by randomly replacing letters with neighboring keys on a "qwerty" keyboard.

Hidden Errors: **Logical and Temporal Conflicts between Attributes**, which occur when related attributes contain values that are either conflicting or illogical when considered together, or when time-related data does not follow chronological logic. In the Credit Card dataset, we set two hidden conflicts. First, a conflict arises when *DAYS_EMPLOYED* exceeds *DAYS_BIRTH*, implying employment before birth. Second, a conflict involves *AMT_INCOME_TOTAL*, *NAME_EDUCATION_TYPE*, and *OCCUPATION_TYPE*, producing improbable combinations (e.g., high education and advanced occupation but extremely low income) that simple domain rules rarely cover. In the Hotel Booking dataset, a hidden error was generated for bookings labeled *customer_type* as 'Group' with zero *adults* and more than zero *babies*, conflicting with logic.

4.1.3 Baselines. We selected four SOTA baselines, each widely recognized for their effectiveness in detecting and validating data quality issues. **Deequ [35]:** A tool by Amazon for scalable data quality validation, using constraints and metrics. *Deequ auto* generates constraints automatically, while *Deequ expert* incorporates expert-tuned adjustments for higher precision. **TFDV [9]:** TensorFlow Data Validation for scalable ML pipeline validation, automatically detecting anomalies and schema violations. *TFDV auto* uses auto-generated constraints, while *TFDV expert* supports expert fine-tuning. **ADQV [31]:** A tool leveraging adaptive learning to dynamically adjust validation criteria, excelling in evolving datasets. **Gate [37]:** A machine learning-based method for automated detection and correction of data quality issues, offering an adaptive alternative to rule-based systems.

Similar to previous work [31], we manually performed the fine-tuning work required by experts for Deequ and TFDV.

Table 1: Accuracy and recall across different methods and two datasets with synthetic data errors (Ordinary errors: N = Numeric Anomalies, S = String Typos, M = Missing Values; Hidden Errors: Conflicts = Logical and Temporal Conflicts between Attributes). Note: * Indicates average value.

Dataset	Error Types	Methods	Acc.	Recall
Hotel Booking	N, S, M	Deequ auto	0.530*	1
	N, S, M	Deequ expert	1	1
	N, S, M	TFDV auto, expert	1	1
	N, S, M	ADQV	0.963*	1
	N	Gate	0.500	0
	S, M	Gate	0.980*	0.960*
	N, S, M	**DQuaG**	1	1
	Conflicts	Deequ expert	0.500	0
	Conflicts	TFDV expert	0.500	0
	Conflicts	ADQV	0.970	1
	Conflicts	Gate	0.820	0.640
	Conflicts	**DQuaG**	**1**	**1**
Credit Card	N, S, M	Deequ auto	0.550*	1
	N, S, M	Deequ expert	0.970*	1
	S, M	TFDV auto	1	1
	N	TFDV auto	0.500	0
	N, S, M	TFDV expert	1	1
	N, S, M	ADQV	0.960*	1
	N, S, M	Gate	0.510*	1
	N, S, M	**DQuaG**	1	1
	Conflicts-1	Deequ expert	0.500	0
	Conflicts-1	TFDV expert	0.500	0
	Conflicts-1	ADQV	0.500	1
	Conflicts-1	Gate	0.510	1
	Conflicts-1	**DQuaG**	**1**	**1**
	Conflicts-2	Deequ expert	0.500	0
	Conflicts-2	TFDV expert	0.500	0
	Conflicts-2	ADQV	0.960	1
	Conflicts-2	Gate	0.560	1
	Conflicts-2	**DQuaG**	**1**	**1**

## 4.2 Accuracy of Synthetic Error Detection

We used a clean dataset, randomly sampling 10% to generate 50 batches of clean data, and did the same with a dirty dataset to generate 50 batches of dirty data. We then used these 100 batches to test our method and baselines.

According to Table 1, our method performs well in detecting both ordinary and hidden errors, achieving accuracy and recall of 1. For ordinary errors, Deequ-auto and TFDV-auto methods are inaccurate due to overly strict or soft constraints but perform well after expert tuning. However, they cannot detect hidden errors. ADQV can automatically detect ordinary errors but fails to identify hidden errors. For hidden errors in the Hotel Booking dataset, ADQV shows an accuracy of 0.97 but actually flags change in numeric feature distribution, missing the real issues. For the Gate, the results are also unstable. The constraints it sets are too strict and cannot distinguish between dirty and clean datasets. It also cannot distinguish well between hidden errors. The experimental results highlight our method's robustness, particularly in identifying complex data interdependencies.

4.3 Accuracy of Real-World Error Detection

We used the same method to generate 100 test batches from Airbnb and Bicycle datasets as in Section 4.2. In Figure 3, our approach showed excellent performance in detecting problematic data in both real-world datasets, achieving an accuracy of 1. ADQV and Gate performed poorly, flagging all batches due to overly strict error detection. Deequ auto and TFDV auto performed poorly on the Airbnb dataset but well on the bicycle dataset. Deequ expert and TFDV expert performed well on both datasets but required manual tuning of constraints. Therefore,
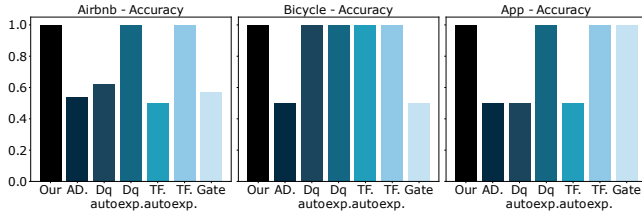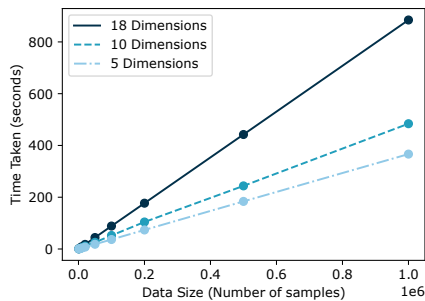
Figure 3: Accuracy across different methods and two datasets with real-world data errors. (All methods have Recall=1)

Table 2: Difference (%) in flagged errors for clean vs. dirty data. (Higher is better.)

Dataset	Graph2Vec	GCN	GCN+GAT	GCN+GIN	GAT+GIN
Airbnb	2.72	1.83	2.60	**4.55**	**4.17**
Bicycle	21.49	11.06	12.36	17.51	**21.72**

Figure 4: Scalability Analysis: data quality validation time of our approach, when varying the data dimensionality and data size, on the New York Taxi dataset.

our method effectively detects real-world errors without the need for manual intervention.

4.4 Comparison with Encoder Architectures

To validate the effectiveness of the GAT+GIN architecture, we tested five encoder architectures: *Graph2Vec* [26], *GCN, GCN+GAT, GCN+GIN*, and *GAT+GIN* on Airbnb and Bicycle datasets. Table 2 shows that *GAT+GIN* achieves the highest difference in flagged errors, reflecting its stronger ability to distinguish clean from dirty data. We attribute this to GAT's attention mechanism (focusing on important neighbors) and GIN's injective aggregation (capturing nuanced relationships). For hyperparameters, all models used four layers, a hidden dimension of 64, a learning rate of 0.01, and a batch size of 128.

4.5 Scalability Analysis

Figure 4 illustrates the scalability analysis of our data quality validation method on the New York Taxi dataset, showing the validation time across different data dimensions (5, 10, and 18) and varying data sizes (up to 1 million samples). The results demonstrate that as the dataset size and the number of dimensions increases, our method's computation time increases linearly rather than exponentially, which is logical. When processing datasets with millions of data points, our method only takes ten minutes. These findings indicate that our method is scalable.

To further assess scalability and robustness, we conducted an experiment using different sample sizes (10 to 1000). The accuracy results are summarized in Table 3. The sample size represents the number of new data instances used for quality validation.

Table 3: Summary of Overall Accuracy for Different Sample Sizes across 3 Datasets.

Sample Size	10	20	50	100	500	1000
Airbnb (Accuracy %)	85.0	93.0	99.0	99.0	**100.0**	**100.0**
Bicycle (Accuracy %)	86.0	92.0	89.0	97.0	**100.0**	**100.0**
NY Taxi (Accuracy %)	83.0	89.0	98.0	97.0	**100.0**	**100.0**

The experimental results show that as the sample size increases, the model's accuracy also improves. When the sample size exceeds 500, the model achieves 100% accuracy across both datasets, indicating stable performance at larger scales. These findings highlight the effectiveness of our method for large sample sizes but also suggest limitations in scenarios with smaller data availability.

4.6 Data Repair Evaluation

To assess the effectiveness of our data repair process, we conducted experiments on the Airbnb and Bicycle datasets. According to our data quality validation results of the Airbnb dataset, The error rate of the original dirty dataset was 10.52%. After applying repair suggestions generated by our repair decoder, the error rate was reduced to 4.97%, closely matching the error rate of 4.95% observed in the clean dataset. For the results of the Bicycle dataset, the repair decoder reduced the error rate of the dirty data from 21.11% to 2.75%. Importantly, the repaired dataset was classified as clean under our quality validation framework.

5 CONCLUSIONS AND FUTURE WORK

We proposed DQuaG, a novel multi-task learning framework that combines a GNN encoder with dual decoders for data quality validation and repair. Our approach effectively detects and repairs data quality issues without expert-defined constraints or manual interventions. By utilizing a dual-decoder structure, our model independently optimizes data quality validation and repair tasks, with a specially designed validation decoder loss that enhances the detection of erroneous data. We evaluated our approach using real-world datasets containing inherent data quality issues, where it performed well. However, in extreme scenarios—such as datasets with only a few thousand samples where only a single erroneous instance exists—our method may lack sensitivity. In such cases, traditional statistical methods often prove to be more accurate, as they can better identify isolated anomalies within smaller datasets. We aim to further refine our approach, improving its robustness in extreme conditions. Additionally, we plan to extend our approach to encompass post-validation tasks, such as data cleaning and data selection. To improve the interpretability of our models, we will focus on optimizing the GNN frameworks.

REFERENCES

[1] [n.d.]. Airbnb Data. https://www.kaggle.com/datasets/sandeepmajumdar/airbnbnyccleaned.

[2] [n.d.]. Chicago Divvy Bicycle Sharing Data. https://www.kaggle.com/datasets/yingwurenjian/chicago-divvy-bicycle-sharing-data/data.

[3] [n.d.]. Credit Card Data. https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction?select=application_record.csv.

[4] [n.d.]. New York Taxi Trip Data. https://data.cityofnewyork.us/Transportation/2015-Yellow-Taxi-Trip-Data/2yzn-sicd/about_data.

[5] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2017. Data profiling: A tutorial. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1747–1751.

[6] Nuno Antonio, Ana de Almeida, and Luis Nunes. 2019. Hotel booking demand datasets. *Data in brief* 22 (2019), 41–49.

[7] Lukas Budach, Moritz Feuerpfeil, Nina Ihde, Andrea Nathansen, Nele Sina Noack, Hendrik Patzlaff, Hazar Harmouch, and Felix Naumann. 2022. The Effects of Data Quality on Machine Learning Performance.

[8] Riccardo Cappuzzo, Saravanan Thirumuruganathan, and Paolo Papotti. 2024. Relational Data Imputation with Graph Neural Networks. In *EDBT/ICDT 2024, 27th International Conference on Extending Database Technology*.

[9] Emily Caveness, Paul Suganthan GC, Zhuo Peng, Neoklis Polyzotis, Sudip Roy, and Martin Zinkevich. 2020. Tensorflow data validation: Data analysis and validation in continuous ml pipelines. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2793–2796.

[10] Fei Chiang and Renée J Miller. 2008. Discovering data quality rules. *Proceedings of the VLDB Endowment* 1, 1 (2008), 1166–1177.

[11] Yusuf Delikkaya. 2023. Google Play Store Apps Dataset. https://www.kaggle.com/datasets/yusufdelikkaya/google-play-store-apps-dataset?resource=download

[12] Sijie Dong, Qitong Wang, Soror Sahri, Themis Palpanas, and Divesh Srivastava. 2024. Efficiently Mitigating the Impact of Data Drift on Machine Learning Pipelines. *Proceedings of the VLDB Endowment* 17, 11 (2024), 3072–3081.

[13] Xusheng Du, Jiong Yu, Zheng Chu, Lina Jin, and Jiaying Chen. 2022. Graph autoencoder-based unsupervised outlier detection. *Information Sciences* 608 (2022), 532–550.

[14] Hadi Fadlallah, Rima Kilany, Houssein Dhayne, Rami el Haddad, Rafiqul Haque, Yehia Taher, and Ali Jaber. 2023. Context-aware big data quality assessment: a scoping review. *ACM Journal of Data and Information Quality* (2023).

[15] Daniele Foroni, Matteo Lissandrini, and Yannis Velegrakis. 2021. Estimating the extent of the effects of Data Quality through Observations. *2021 IEEE 37th International Conference on Data Engineering (ICDE)* (2021), 1913–1918.

[16] Stefan Grafberger, Shubha Guha, Julia Stoyanovich, and Sebastian Schelter. 2021. Mlinspect: A data distribution debugger for machine learning pipelines. In *Proceedings of the 2021 International Conference on Management of Data*. 2736–2739.

[17] Nitin Gupta, Hima Patel, Shazia Afzal, Naveen Panwar, Ruhi Sharma Mittal, Shanmukha Guttula, Abhinav Jain, Lokesh Nagalapatti, Sameep Mehta, Sandeep Hans, et al. 2021. Data Quality Toolkit: Automatic assessment of data quality and remediation for machine learning datasets. *arXiv preprint arXiv:2108.05935* (2021).

[18] Ihab F Ilyas and Xu Chu. 2015. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases* 5, 4 (2015), 281–393.

[19] Vimukthi Jayawardene, Shazia Sadiq, and Marta Indulska. 2013. The curse of dimensionality in data quality. (2013).

[20] Vimuthki Jayawardene, Shazia Sadiq, and Marta Indulska. 2015. An analysis of data quality dimensions. (2015).

[21] Bojan Karlaš, Peng Li, Renzhi Wu, Nezihe Merve Gürel, Xu Chu, Wentao Wu, and Ce Zhang. 2020. Nearest neighbor classifiers over incomplete information: From certain answers to certain predictions. *arXiv preprint arXiv:2005.05117* (2020).

[22] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *PVLDB* 9, 12 (2016), 948–959.

[23] Cheng-Te Li, Yu-Che Tsai, Chih-Yao Chen, and Jay Chiehen Liao. 2024. Graph Neural Networks for Tabular Data Learning: A Survey with Taxonomy and Directions. arXiv:cs.LG/2401.02143

[24] Yang Liu, Xiang Ao, Zidi Qin, Jianfeng Chi, Jinghua Feng, Hao Yang, and Qing He. 2021. Pick and choose: a GNN-based imbalanced learning approach for fraud detection. In *Proceedings of the web conference 2021*. 3168–3177.

[25] Dror E Maydan, John L Hennessy, and Monica S Lam. 1991. Efficient and exact data dependence analysis. In *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*. 1–14.

[26] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. 2017. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005* (2017).

[27] Felix Naumann. 2014. Data profiling revisited. *ACM SIGMOD Record* 42, 4 (2014), 40–49.

[28] OpenAI. 2024. GPT-4: OpenAI's most advanced system. https://openai.com/index/gpt-4/.

[29] Susie Xi Rao, Shuai Zhang, Zhichao Han, Zitao Zhang, Wei Min, Zhiyao Chen, Yinan Shan, Yang Zhao, and Ce Zhang. 2022. xFraud: explainable fraud transaction detection. *VLDB2022* (2022).

[30] Simon Razniewski and Werner Nutt. 2011. Completeness of queries over incomplete databases. *Proceedings of the VLDB Endowment* 4, 11 (2011), 749–760.

[31] Sergey Redyuk, Zoi Kaoudi, Volker Markl, and Sebastian Schelter. 2021. Automating Data Quality Validation for Dynamic Data Ingestion.. In *EDBT*. 61–72.

[32] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820* (2017).

[33] El Kindi Rezig, Lei Cao, Giovanni Simonini, Maxime Schoemans, Samuel Madden, Nan Tang, Mourad Ouzzani, and Michael Stonebraker. 2020. Dagger: A data (not code) debugger. In *CIDR 2020, 10th Conference on Innovative Data Systems Research, Amsterdam, The Netherlands, January 12-15, 2020, Online Proceedings*.

[34] Sebastian Schelter, Stefan Grafberger, Philipp Schmidt, Tammo Rukat, Mario Kiessling, Andrey Taptunov, Felix Biessmann, and Dustin Lange. 2019. Differential data quality verification on partitioned data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1940–1945.

[35] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. 2018. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1781–1794.

[36] Sebastian Schelter, Tammo Rukat, and Felix Bießmann. 2020. Learning to validate the predictions of black box classifiers on unseen data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1289–1299.

[37] Shreya Shankar, Labib Fawaz, Karl Gyllstrom, and Aditya Parameswaran. 2023. Automatic and Precise Data Validation for Machine Learning. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 2198–2207.

[38] Fatimah Sidi, Payam Hassany Shariat Panahy, Lilly Suriani Affendey, Marzanah A Jabar, Hamidah Ibrahim, and Aida Mustapha. 2012. Data quality: A survey of data quality dimensions. In *2012 International Conference on Information Retrieval & Knowledge Management*. IEEE, 300–304.

[39] Phanwadee Sinthong, Dhaval Patel, Nianjun Zhou, Shrey Shrivastava, Arun Iyengar, and Anuradha Bhamidipaty. 2021. DQDF: data-quality-aware dataframes. *Proceedings of the VLDB Endowment* 15, 4 (2021), 949–957.

[40] Indro Spinelli, Simone Scardapane, and Aurelio Uncini. 2020. Missing data imputation with adversarially-trained graph convolutional networks. *Neural Networks* 129 (2020), 249–260.

[41] Ashraf Tahmasbi, Ellango Jothimurugesan, Srikanta Tirthapura, and Phillip B Gibbons. 2020. DriftSurf: A Risk-competitive Learning Algorithm under Concept Drift. *arXiv preprint arXiv:2003.06508* (2020).

[42] Lev Telyatnikov and Simone Scardapane. 2023. EGG-GAE: scalable graph neural networks for tabular data imputation. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2661–2676.

[43] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[44] Richard Y. Wang and Diane M. Strong. 1996. Beyond Accuracy: What Data Quality Means to Data Consumers. *J. Manag. Inf. Syst.* 12, 4 (1996), 5–33.

[45] G. Widmer and M. Kubat. 1996. Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning* 23, 1 (1996), 69–101.

[46] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[47] Naiem K. Yeganeh, Shazia Sadiq, and Mohamed A. Sharaf. 2014. A framework for data quality aware query systems. *Information Systems* 46 (2014), 24–44. https://doi.org/10.1016/j.is.2014.05.005

[48] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 793–803.