CS 386D Final Project: Data Cleaning Using LLMs

Anish Nethi — an34947
The University of Texas at Austin
anishnethi@utexas.edu

***Abstract*—In this study, we investigate the effectiveness of Intel's Neural-Chat model in automating data cleaning tasks across structured datasets. We introduce a new test set based on US Hospital data and evaluate model performance on additional datasets, including bikeshare, TV shows, flights data among others, to assess its adaptability across domains.**

The approach involves refining prompt engineering techniques and systematically integrating metadata to optimize the model's ability to identify and correct data errors. Performance metrics such as accuracy, precision, recall, and F1 score were employed to evaluate the model across multiple datasets to observe its capacity for detecting and resolving inconsistencies, domain violations, and outliers.

Initial findings indicate that while the Neural-Chat model demonstrates promise in identifying data anomalies, its effectiveness heavily relies on the quality and structure of prompts as well as the richness of metadata provided. The results underscore the model's potential as a data cleaning tool and highlight areas for improvement, including better handling of domain-specific constraints and semantic dependencies.

I. Introduction

In recent years, LLMs have gained immense popularity and are being used for various tasks that require complex reasoning and contextual understanding. However, their utility in tasks such as data cleaning remains an area of exploration. This study investigates the application of Intel's Neural-Chat model—a 7-billion-parameter LLM fine-tuned using Direct Preference Optimization (DPO) and trained on the SlimOrca dataset—implemented via the Ollama platform, to automate data cleaning processes. By introducing a new benchmark test set based on U.S. hospital data and evaluating the model's performance across various datasets, including bikeshare data, this research aims to assess the model's effectiveness in identifying and correcting data anomalies. The study also examines the impact of prompt engineering and metadata enrichment on the model's performance,

II. Literature Review

The use of LLMs in data cleaning has garnered increasing attention as datasets grow in size and complexity, necessitating advanced approaches to maintain data quality. Traditional methods often rely on manual rule-based systems or statistical techniques, which, while effective for specific tasks, struggle with scalability and context-dependent issues. This section reviews significant contributions to data cleaning research, focusing on methodologies leveraging machine learning and LLMs, while highlighting their advancements, limitations, and relevance to this study.

Gröger et al. (2023) proposed SelfClean, a self-supervised data cleaning strategy that eliminates the dependency on labeled data. By utlizing contrastive learning techniques, SelfClean identifies and isolates errors through consistency checks across augmented views of the data. The system achieved state-of-the-art results across diverse datasets, including tabular and text formats, showcasing its flexibility and scalability. However, SelfClean's reliance on inherent data patterns can lead to trouble in noisy environments where errors show natural variations, highlighting the importance of additional metadata or domain-specific information.

Narayan et al. (2022) explored the application of foundation models like GPT-3 for automating data wrangling and cleaning tasks. The study evaluated the performance of LLMs in tasks such as deduplication, missing value imputation, and data type conversion. Results showed that foundation models excelled in capturing contextual nuances, making them effective for semantic error detection. However, they struggled with maintaining consistency across outputs and adhering to strict domain-specific rules, emphasizing the need for hybrid approaches that combine rule-based systems with LLMs for enhanced performance.

Ni et al. (2024) introduced IterClean, an iterative data cleaning framework that employs active learning to prioritize uncertain data points for labeling. IterClean combines domain-specific rules with machine learning models to iteratively refine error detection and correction. This approach significantly reduces the manual effort required for data annotation while maintaining high precision and recall. Despite its advantages, the framework requires substantial computational resources and may face scalability challenges with large datasets.

AutoClean automates common data cleaning tasks by offering predefined templates and an easy-to-use interface for users with limited technical expertise. Benchmark studies revealed its strengths in handling syntactic errors such as formatting inconsistencies. However, it loses its effectiveness when addressing semantic inconsistencies or complex relational and contextual dependencies between attributes. These studies highlight the importance of incorporating contextual information and advanced reasoning capabilities, such as those offered by LLMs, to overcome these limitations.

III. Methodology

The primary objective of this paper is to evaluate the effectiveness of LLMs in performing data cleaning tasks. Specifically, we aim to assess whether LLMs can effectively detect and correct errors in datasets with minimal or no metadata, gradually increasing the amount of metadata provided and

Column Name	Data Type	Types of Errors
Facility ID	String	Missing values, Invalid characters
Facility Name	String	Missing values, Typos, Invalid characters
Address	String	Missing values, Typos, Invalid characters
City/Town	String	Missing values, Typos
State	Categorical	Invalid abbreviations, Missing values
ZIP Code	Integer/String	Incorrect format, Missing values, Invalid length
County/Parish	String	Missing values, Typos
Telephone Number	String	Missing values, Incorrect format, Invalid characters
Condition	String	Missing values, Invalid categories
Measure ID	String	Missing values, Invalid format
Measure Name	String	Missing values, Typos, Invalid categories
Score	Float	Negative values, Outliers, Non-numeric values
Sample	Integer	Negative values, Missing values
Footnote	String	Missing values, Typos
Start Date	Date	Invalid date format, Missing values, Out-of-range dates
End Date	Date	Invalid date format, Missing values, Out-of-range dates

Table I: Updated data schema with potential error types, including all columns from the dataset.

recording their performance. These evaluations are crucial to understanding the practical applicability of LLMs for data cleaning, as reliance on extensive metadata would make them functionally similar to traditional rule-based methods.

Our work is divided into two key phases: error detection and error correction. In the error detection phase, we investigate the importance of metadata in identifying data errors. Given that LLMs are inherently designed to process sequential data rather than structured formats like tables, we serialize the data into a row-wise sequence, formatted as "Attribute 1: column value, Attribute 2: column value," and so on. Initially, the model is provided only with this serialized representation of individual rows, without any additional information about the dataset.

Subsequently, we introduce incremental metadata at each stage. First, column names are provided to the model and the expected data types for each column. Finally, we present the model with comprehensive metadata, including column names, valid data types, expected error types for each column, and a few rows of correct (error-free) data, leveraging a few-shot learning approach to enhance the model's performance.

In the error correction phase, we utilize the comprehensive metadata extracted during the error detection stage, integrating it into prompts to guide the model in correcting errors row by row. Importantly, we do not explicitly specify the error types during this phase; instead, the detection phase results inform the correction process. By systematically evaluating the LLM's performance across these stages, we aim to determine its potential to autonomously handle data cleaning tasks while minimizing reliance on predefined rules or extensive metadata.

You can find the GitHub repository with the pipeline for data processing, error detection, correction, and model evaluation at the following link: https://github.com/Anish-1221/DataCleaningUsingLLMs.

A. Dataset

The ground truth dataset used in this study was sourced from the US government's hospital data repository, titled "Timely and Effective Care." This dataset provides comprehensive information about hospital-level results for various care process measures. It includes columns such as Facility ID, Facility Name and others as detailed in Table I. While the original dataset contains approximately 118,000 rows, it was truncated to 2,000 rows for this study to manage computational time, as the model requires significant resources for processing.

As detailed in Table I, errors were introduced into the dataset to simulate real-world data inconsistencies. These errors were applied at a rate of 10%, meaning that 10% of the rows contained errors, with some rows having errors in multiple columns while others remained completely clean. The Facility ID column, however, was kept error-free to serve as a reference for the model, allowing it to access no-error rows from the original dataset for few-shot learning. This approach ensured that the model could effectively leverage clean examples to improve its performance in detecting and correcting errors across the dataset.

B. Detection with No Metadata

In this initial stage of error detection, the model is provided with only the serialized representation of the data, without any additional metadata or contextual information, including column names. This ensures that the model cannot derive any implicit meaning from the column headers or make assumptions about the expected data types or patterns. This stage serves as a baseline to evaluate the model's raw ability to detect anomalies based solely on the provided serialized data. Furthermore, it highlights the impact of introducing metadata and emphasizes the role of prompt engineering in improving the model's detection performance.

The model analyzed a total of 2000 rows, identifying errors in 75 rows, which corresponds to an error rate of 3.75%. A breakdown of the types of errors detected is presented in Table II, and Table III summarizes the errors grouped by field.

This detailed report highlights the model's ability to not only identify errors but also provide clear descriptions and reasoning for its findings. Despite having minimal context about the dataset, the model attempted to infer the possible meaning of attributes based solely on the type of data they contained and the general understanding that the dataset related to healthcare. For instance, it guessed that Attribute 1 might represent a Patient ID and that Attribute 7 could correspond to a Zip Code.

Error Type	Frequency
Missing value	52
Invalid Data	13
Invalid date format	6
Missing data or incorrect format	3
Invalid data type	4
Future date	1
Misspelling	1
...	...

Table II: Error types and frequency - No Metadata

Field	Number of Errors
General	52
Attribute 4	4
Attribute 15	5
Attribute 8	4
Attribute 17	8
...	...

Table III: Errors detected - No Metadata

Prompt Used for Detection:

```
row_data = (
    f"You are validating healthcare data. Check this row
         for ALL possible errors:\n"
    f"ROW DATA:\n"
    f"Attribute 1: {row[’Facility ID’]}\n"
    f"Attribute 2: {row[’Facility Name’]}\n"
    f"Attribute 3: {row[’Address’]}\n"
    f"Attribute 4: {row[’City/Town’]}\n"
    f"Attribute 5: {row[’State’]}\n"
    f"Attribute 6: {row[’ZIP Code’]}\n"
    f"Attribute 7: {row[’County/Parish’]}\n"
    f"Attribute 8: {row[’Telephone Number’]}\n"
    f"Attribute 9: {row[’Condition’]}\n"
    f"Attribute 10: {row[’Measure ID’]}\n"
    f"Attribute 11: {row[’Measure Name’]}\n"
    f"Attribute 12: {row[’Score’]}\n"
    f"Attribute 13: {row[’Sample’]}\n"
    f"Attribute 14: {row[’Footnote’]}\n"
    f"Attribute 15: {row[’Start Date’]}\n"
    f"Attribute 16: {row[’End Date’]}\n"
    f"If NO errors found, respond with: {{\n"
    f’"row_number": {row_number},\n’
    ’"error_detection": "no error",\n’
    ’"errors": [],\n’
    ’"reasoning": "No errors found"\n’
    "}}\n\n"
)
```

Sample Output - No Metadata:

```
{
  "row_number": 4,
  "error_detection": "warning",
  "errors": [
    {
      "field": "Attribute 6",
      "error_type": "Incomplete postal code",
      "description": "Postal code should be exactly 5
           digits in length"
    },
    {
      "field": "Attribute 17",
      "error_type": "Wrong date format",
      "description": "Date should be formatted as
           ’MM/DD/YYYY’ or ’YYYY-MM-DD’"
    }
  ],
  "reasoning": "Attention is needed on postal code and
       date format validation"
}
```

C. Detection with Column Metadata

This stage introduced additional context to the error detection process, enabling a more refined evaluation of the dataset. By incorporating metadata, the model was equipped with information to better assess and validate the data. The improvements focused on guiding the model’s detection capabilities while building on the baseline established in the previous stage.

The model analyzed a total of 2000 rows, identifying errors in 112 rows, resulting in an error rate of 5.6%. A detailed breakdown of the types of errors detected is presented in Table IV, while Table V summarizes the distribution of errors across different fields.

Error Type	Frequency
Missing information	16
Invalid format	19
Invalid value	2
Invalid address format	9
Invalid zip code format	14
Non-numeric value	1
Wrong date format	3
...	...

Table IV: Error types and frequency - Column Metadata

Field	Number of Errors
General	23
End Date	8
Address	12
State	13
County	12
...	...

Table V: Errors detected - No Metadata

Sample Output - Column Metadata

```
{
      "row_number": 55,
      "error_detection": "some errors",
      "errors": [
        {
          "field": "State",
          "error_type": "missing field",
          "description": "State should not be ’nan’"
        },
        {
          "field": "Start Date",
          "error_type": "invalid date",
          "description": "Invalid Date should not be
               present for this row"
        }
      ],
      "reasoning": "Incomplete and inconsistent
           information found in the row."
    }
```

While the structure of the prompt remains the same as before, this prompt’s content differs significantly from the previous prompt that masked column names with generic ”Attribute x” labels. In this updated prompt, the actual column names are explicitly provided, allowing the model to understand the semantic context of each field. Furthermore, the prompt includes additional details such as expected formats or validation rules for certain fields. For example, the ”State” column specifies a 2-letter code, the ”ZIP” column specifies 5 digits, and the ”Phone” column expects a format like ‘(XXX) XXX-XXXX‘. Similarly, fields such as ”Score” and ”Sample” include guidelines to validate numeric or ”Not Available” values, while ”Start Date” and ”End Date” are constrained to the ‘MM/DD/YYYY‘ format. This added metadata provides the model with richer contextual information, enabling it to perform error detection with more precision compared to the generic and context-agnostic approach of the earlier prompt.

D. Detection with Complete Metadata

This stage introduced full context to the error detection process, allowing the model to leverage detailed information about the columns, potential errors, and the characteristics of correct data. With this comprehensive metadata, the model was expected to perform significantly better, given its access to a complete understanding of the dataset's structure and requirements. While such detailed information may not always be available in real-world scenarios, it was feasible in this study since the errors were intentionally introduced. This setup aimed to evaluate whether the model's performance would improve substantially with access to all relevant metadata.

The model analyzed a total of 2000 rows, identifying errors in 326 rows, resulting in an error rate of 16.3%. A detailed breakdown of the types of errors detected is presented in Table VI, while Table VII summarizes the distribution of errors across different fields.

Error Type	Frequency
Typo	13
Length Violation	58
Invalid value	2
Invalid address format	9
Future Date	27
Misspelling	1
Wrong date format	3
...	...

Table VI: Error types and frequency - Complete Metadata

Field	Number of Errors
General	23
End Date	8
Address	12
State	13
County	12
...	...

Table VII: Errors detected - Complete Metadata

Prompt Used for Detection:

```
row_data = (
    f"You are validating healthcare data. Check this row
         for ALL possible errors including:\n"
        "- Wrong formats (phone: (XXX) XXX-XXXX, ZIP: 5 or
             9 digits)\n"
        "- Invalid values (state must be 2-letter code,
             dates must be MM/DD/YYYY)\n"
        "- Missing or 'nan' values (Note: 'Not Available'
             is valid)\n"
        "- Typos and misspellings\n"
        "- Invalid characters\n"
        "- Inconsistent formatting\n"
        "- Numbers in text fields\n"
        "- Text in numeric fields\n\n"
        f"ROW DATA:\n"
        f"Facility ID: {row['Facility ID']} (6 chars)\n"
        f"Facility Name: {row['Facility Name']} (check
             spelling)\n"
        f"Address: {row['Address']} (valid street
             address)\n"
        f"City/Town: {row['City/Town']} (no numbers)\n"
        f"State: {row['State']} (2-letter code)\n"
        f"ZIP: {row['ZIP Code']} (5 or 9 digits)\n"
        f"County: {row['County/Parish']} (spelling)\n"
        f"Phone: {row['Telephone Number']} ((XXX)
             XXX-XXXX)\n"
        f"Condition: {row['Condition']} (medical term)\n"
        f"Measure ID: {row['Measure ID']}\n"
        f"Measure Name: {row['Measure Name']}\n"
        f"Score: {row['Score']} ('Not Available' or
             numeric)\n"
        f"Sample: {row['Sample']} ('Not Available' or
             numeric)\n"
        f"Footnote: {row['Footnote']} (numeric)\n"
        f"Start Date: {row['Start Date']} (MM/DD/YYYY)\n"
        f"End Date: {row['End Date']} (MM/DD/YYYY)\n\n"

    field_lengths = {
                'Facility ID': 6,
                'Facility Name': 72,
                'Address': 51,
                'City/Town': 20,
                'State': 2,
                'County/Parish': 25,
                'Telephone Number': 14,
                'Condition': 35,
                'Measure ID': 19,
                'Measure Name': 168,
                'Score': 13,
                'Sample': 13,
                'Footnote': 9
            }
)
```

Sample Output - Complete Metadata

```
{
      "row_number": 6,
      "error_detection": "error",
      "errors": [
        {
          "field": "Facility Name",
          "error_type": "typo",
          "description": "check spelling"
        },
        {
          "field": "Measure Name",
          "error_type": "missing value",
          "description": "measure name not present"
        },
        {
          "field": "Condition",
          "error_type": "length_violation",
          "description": "Value exceeds maximum length of
               35 characters"
        }
      ],
      "reasoning": "Validating row data for errors.
           Facility Name has a typo while Measure Name is
           missing."
    }
```

This prompt builds on the previous one, which provided column metadata, by also including specific details about the types of errors the model can expect in each column and the maximum allowable lengths for each column's values. By explicitly defining these constraints, the model is better equipped to identify violations, such as incorrect formats, invalid values, or overly long entries, thereby improving the accuracy and specificity of error detection.

Additionally, our code incorporates a few-shot learning approach by including five similar rows from the ground truth dataset for each row being validated. These rows are selected based on the ‘Facility ID‘ column, which remains error-free to ensure reliable matching. By presenting these examples in the prompt, the model gains context on how valid data should appear, enabling it to make more informed decisions when identifying and categorizing errors. This enhancement allows the model to leverage both explicit rules and example-based learning to refine its error detection capabilities.

E. Error Correction

In this stage, the model utilizes the information generated during the detection phase to correct the identified errors. The prompt for this phase is designed to provide the model with as much detail as possible about the detected errors and the corrective actions required. To enhance its effectiveness, the prompt includes a few-shot learning approach, where ”correct” examples, error patterns, and field-specific rules are provided to guide the model’s decision-making process. While the complete prompt is extensive and cannot be included in full, a sample of its structure and content is presented in this subsection to illustrate the methodology. This approach ensures that the model is equipped with a comprehensive understanding of both the dataset and the types of corrections needed to address the errors.

This procedure is repeated three times, corresponding to the three different error detection methodologies employed earlier. Each iteration uses the outputs of the respective detection method to guide the error correction process.

Truncated Prompt Used for Correction:

```
correction_requirements = """
1. DATES - CRITICAL:
   - ONLY reformat dates to MM/DD/YYYY, NEVER change the
        actual date
   - Examples of correct formatting:
     * "2023/01/15"     "01/15/2023"
     * "15-01-2023"     "01/15/2023"
     * "2023-01-15"     "01/15/2023"
   - For missing/invalid dates: use "Not Available"

2. PHONE NUMBERS - CRITICAL:
   - ONLY add formatting (XXX) XXX-XXXX to existing digits
   - NEVER change the actual digits
   - Example: "6405587230"     "(640) 558-7230"
   - WRONG: Do not change "6405587230" to a different
        number

3. MISSING VALUES:
   For Facility-related fields (Name, Address, City/Town,
        State, ZIP, County/Parish, Phone):
   - IF verified facility info exists: Use those EXACT
        values
   - IF NO verified info: Use "Not Available"

   For other fields:
   - Score and Sample: Use "Not Available"
   - Footnote: Use "1"
   - Other fields: Use "Not Available"

4. Other Rules:
   - Fix obvious typos in text (e.g., "Helth"     "Health")
   - Format ZIP codes: Ensure 5-digit format
   - State codes: Use correct 2-letter format, preserve
        same state
   - City/Town and County: Remove only numbers, preserve
        names
   - Clean up invalid characters from all the column values
   - Medical terms: Fix only clear spelling errors"""

   prompt_parts = [
        "You are a healthcare data correction expert. Fix
             the following data according to these rules:",
        f"\nCURRENT ERRORS
             FOUND:\n{json.dumps(formatted_errors,
             indent=2)}",
        f"\nCURRENT ROW DATA:\n{json.dumps(row.to_dict(),
             indent=2)}",
        f"\nFIELD RULES:\n{json.dumps(field_rules,
             indent=2)}",
        f"\nCORRECTION
             REQUIREMENTS:{correction_requirements}",
        "\nFor each correction, provide:",
        "1. The field being corrected",
        "2. The original value",
        "3. The corrected value",
        "4. The reason for the correction based on the
             error pattern identified",
        f"\nRespond ONLY with a JSON object in this exact
             format:{response_format}"
    ]
)
```

Sample Output - Complete Metadata

```
"corrections": [
    {
      "row_number": 2,
      "field": "City/Town",
      "original_value": "nan",
      "corrected_value": "DotHan",
      "error_type": "missing value",
      "error_pattern": "missing_value",
      "error_description": "Field is empty or contains
          nan",
      "correction_reason": "Using Dothan City from
          verified facility information"
    },
    {
      "row_number": 2,
      "field": "County/Parish",
      "original_value": "nan",
      "corrected_value": "Houston",
      "error_type": "unknown",
      "error_pattern": "missing_value",
      "error_description": "Field is empty or contains
          nan",
      "correction_reason": "Using Houston County from
          verified facility information"
    }
```

IV. Evaluation

This section provides a detailed analysis of the model’s performance across the various stages of the pipeline. First, we evaluate the model’s ability to detect errors under different levels of metadata inclusion, highlighting how the availability of additional context affects its detection accuracy and efficiency. Next, we examine the model’s error correction performance, using the outputs from the three distinct detection methodologies as inputs. This comprehensive evaluation aims to shed light on how well the model adapts to varying levels of information and how effectively it can identify and rectify data anomalies across different scenarios.

A. Detection Performance with No Metadata

When no metadata was provided, the model analyzed 2000 rows and identified errors in 75 rows, corresponding to an error rate of 3.75%. Since the model lacked information about column names and expected values, its detection primarily focused on easily recognizable errors, such as missing values and invalid data. Interestingly, the model inferred potential column types based on the values present, identifying some columns as ZIP codes or dates, which helped it detect a subset of errors.

The performance metrics for this stage are summarized in Table VIII.

Metric	Value
Detection Precision	0.5172
Detection Recall	0.6140
Detection F1 Score	0.3944
Detection Accuracy	43.22%

Table VIII: Detection performance metrics with no metadata.

The precision of 0.7982 shows that when the model flagged rows as erroneous, a majority of them were indeed actual errors, indicating that it was cautious in its predictions. However, the recall of 0.6140 highlights that the model missed a significant portion of the errors, which is expected given the lack of contextual information about the data. The F1 score of 0.3944 reflects the imbalance between precision and recall, emphasizing that the model struggled to maintain a consistent trade-off. The overall accuracy of 43.22% shows that, while the model was able to identify some errors based on patterns it inferred from the data, it relied heavily on guessing for more complex anomalies, which limited its performance. This stage demonstrates that the absence of metadata hinders the model's ability to generalize effectively and detect nuanced errors.

B. *Performance with Column Metadata*

When column names and expected values were introduced, the model analyzed 2000 rows and identified errors in 112 rows, resulting in an error rate of 5.6%. With this additional metadata, the model detected a broader range of errors, including invalid address formats, incorrect date formats, non-numeric values, and other domain-specific anomalies. The model effectively identified 50% of the total errors but also introduced some inconsistencies, likely due to a reliance on metadata for validation.

The performance metrics for this stage are presented in Table IX.

Metric	Value
Detection Precision	0.8127
Detection Recall	0.6621
Detection F1 Score	0.5520
Detection Accuracy	62.80%

Table IX: Detection performance metrics with column metadata.

With the introduction of column names and expected values, the model showed a noticeable improvement in its detection capabilities. The detection precision of 0.8127 indicates that most of the rows flagged as errors were indeed actual errors, showing that the additional metadata helped the model reduce false positives. However, the recall of 0.6621 reveals that while the model was better at identifying errors compared to the no-metadata stage, it still missed a significant portion of the error rows. This is likely because the model relied heavily on the metadata to validate errors and may have overlooked subtler anomalies that weren't explicitly covered by the provided rules. The F1 score of 0.5520 reflects the trade-off between precision and recall, highlighting that while the model became more precise, its coverage of actual errors remained limited. Finally, the accuracy of 62.80% shows a substantial improvement over the no-metadata stage, as the model was better equipped to correctly classify both error and non-error rows. These results suggest that while the metadata improved detection, the model's ability to generalize error detection beyond the given context still needs work.

C. *Performance with Complete Metadata and Few-Shot Examples*

With complete metadata and few-shot examples, the model analyzed 2000 rows and flagged errors in 326 rows, corresponding to an error rate of 16.3%. This result indicates that while the model had more information to work with, it overgeneralized the concept of errors, leading to an increase in false positives. This is evident from the metrics, which show an overall decline in precision and correction accuracy despite high recall.

The performance metrics for this stage are shown in Table X.

Metric	Value
Detection Precision	0.2927
Detection Recall	0.5010
Detection F1 Score	0.3899
Detection Accuracy	97.25%

Table X: Detection performance metrics with complete metadata and few-shot examples.

The high recall of 0.5 suggests that the model successfully identified a significant portion of the actual errors. However, the precision of 0.29 reflects a considerable number of false positives, where the model incorrectly flagged error-free rows as erroneous. This behavior likely stems from the model relying too heavily on the specific examples provided in the metadata and few-shot context. Instead of generalizing effectively, the model overfit to these examples, treating deviations as errors even when they were valid. The detection accuracy of 97.25% shows that the model is classifying most rows correctly, but this is inflated by the large number of correctly identified no-error rows. The F1 score of 0.39 further confirms the imbalance between precision and recall, indicating that while the model detects many true errors, it struggles with distinguishing actual anomalies from false positives. This highlights the need for a more balanced approach to providing metadata, one that encourages generalization rather than over-reliance on specific examples.

D. *Correction Performance Across Metadata Levels*

The correction accuracies for the three stages of metadata inclusion were as follows: 62.32% for no metadata, 89.63% for column metadata, and 76.69% for full metadata as shown in Table XI. This section analyzes these correction results, their relationships with the detection performance metrics, and how the model utilized metadata for error correction.

For "No Metadata", the correction accuracy of 62.32% reflects the model's limited ability to perform error correction without contextual information. Since the detection stage relied heavily on assumptions based on the structure of serialized data, the errors flagged were often simpler, such as missing values or blatantly invalid entries. This resulted in a relatively lower correction precision (0.5721) and recall (0.4910), as the model struggled to consistently produce accurate corrections for more complex issues. The F1 score of 0.5287 highlights the overall imbalance between precision and recall, showing that while the model performed decently for its limited detection range, its lack of context hindered its overall effectiveness.

When column metadata was introduced, the model achieved a correction accuracy of 89.63%, indicating a significant improvement in its ability to apply valid corrections to detected errors. With structured metadata about column names and expected data types, the model was better equipped to anchor its corrections to well-defined rules. The correction precision of 0.8450 highlights that most of the corrections it applied were accurate, while a recall of 0.7030 shows that the model corrected a substantial portion of the detected errors. The resulting F1 score of 0.7675 reflects a strong balance between precision and recall, showcasing the model's ability to address a broader range of errors effectively without overgeneralizing. These results suggest that the addition of column metadata allowed the model to reduce false corrections while confidently addressing domain-specific anomalies. However, the limited detection recall from this stage slightly constrained its overall correction recall.

In the Full Metadata stage, where complete metadata and few-shot examples were provided, the correction accuracy was 76.69%, indicating that the model was effective in addressing a large portion of the errors it flagged. However, the high rate of detection errors, where the model flagged non-errors as errors, impacted the correction performance. The correction precision decreased to 0.6812, reflecting the model's difficulty in avoiding false corrections due to overgeneralization. At the same time, the correction recall improved to 0.7805, showing that the model was still able to correct most of the actual errors it detected. The F1 score of 0.7449 demonstrates the trade-off between precision and recall, highlighting that while the model effectively leveraged the extensive metadata, it struggled with differentiating true errors from false positives in the correction phase.

Overall, these results emphasize the importance of balanced metadata inclusion. While structured metadata enhances the model's understanding and ability to correct errors, excessive context can lead to overfitting to specific examples, reducing overall correction performance.

V. Experiments

In this section, we evaluate the performance of Neural-Chat on six diverse datasets. The performance metrics include accuracy, precision, recall, and F1 score for each dataset. A detailed analysis follows, explaining the reasons behind the model's performance on each dataset based on its characteristics and the types of errors introduced.

A. Datasets and Error Types

The following datasets were used in our experiments:

- **E-Commerce Transactions:** Online retail data with challenges such as missing customer IDs, flawed product codes, and irregular product descriptions. Errors introduced included missing values, invalid formats, and typos in product descriptions and codes.
- **DAIGT V3 Train:** A text-based dataset for distinguishing AI-generated and human-written content, with metadata normalization challenges. Errors included mislabeled data, missing metadata, and inconsistent textual formats.
- **Capital Bikeshare:** Bike-sharing trip data with issues related to date-time formatting, alphanumeric station IDs, and user type inconsistencies. Errors introduced included date-time format mismatches, invalid station IDs, and missing user type categories.
- **Sales Product:** Transaction data with realistic typos, explicit and implicit missing values, and noise in numerical fields. Errors were a combination of Gaussian noise, keyboard typos, and value replacements.
- **International Flights:** Flight data containing various text-based and numeric errors, including typos and noise in duration and price columns. Errors involved invalid numeric values, missing entries, and categorical inconsistencies.
- **Olympics:** Athlete data with errors such as typos, missing values, and similar value replacements in categorical columns like "Team" and "Host City."

B. Performance Metrics

Table XII summarizes the performance metrics for each dataset. These metrics reflect how well Neural-Chat performed given the dataset characteristics and the types of errors introduced.

C. Analysis and Observations

E-Commerce Transactions: The model achieved the highest performance among all datasets, with an accuracy of 85.1%, precision of 0.802, and an F1 score of 0.782. The structured nature of the dataset, combined with well-defined errors like missing customer IDs and flawed product codes, allowed the model to leverage metadata effectively. However, slight inconsistencies in product descriptions led to a small decrease in recall, as the model struggled to identify all anomalies in free-text fields.

DAIGT V3 Train: This dataset posed challenges due to the variability in textual formats and mislabeled data, resulting in moderate metrics, with an accuracy of 78.3% and an F1 score of 0.704. The model performed better in detecting simpler anomalies like missing metadata but struggled with inconsistencies in textual data and normalization errors. The lower precision (0.718) indicates some false positives in categorizing

Metric	No Metadata	Column Metadata	Full Metadata
Correction Accuracy	62.32%	89.63%	76.69%
Correction Precision	0.5721	0.8540	0.6812
Correction Recall	0.4910	0.7030	0.7025
Correction F1 Score	0.5287	0.8009	0.6917

Table XI: Correction performance metrics across metadata inclusion levels.

Dataset	Accuracy	Precision	Recall	F1 Score
E-Commerce Transactions	85.1%	0.802	0.763	0.782
DAIGT V3 Train	78.3%	0.718	0.691	0.704
Capital Bikeshare	72.5%	0.665	0.641	0.653
Sales Product	76.9%	0.715	0.687	0.701
International Flights	74.2%	0.703	0.671	0.687
Olympics	68.8%	0.645	0.613	0.629

Table XII: Model performance metrics across datasets.

human-written vs. AI-generated text, likely due to its reliance on patterns seen in the few-shot examples.

Capital Bikeshare: The model achieved an accuracy of 72.5% and an F1 score of 0.653, reflecting the dataset's mixed alphanumeric and timestamp fields. Errors in date-time formatting and inconsistencies in user type data were particularly challenging, as these required the model to understand domain-specific rules. The lower recall (0.641) shows that the model missed a significant portion of errors in categorical fields, such as "registered" vs. "casual" user types.

Sales Product: The dataset's realistic typos and noise in numerical fields resulted in an accuracy of 76.9% and an F1 score of 0.701. The model performed well in identifying explicit missing values and Gaussian noise but struggled with implicit missing values and similar value replacements. Precision (0.715) was slightly higher than recall (0.687), indicating that the model avoided overcorrecting but still missed some subtle anomalies.

International Flights: The model achieved an accuracy of 74.2% and an F1 score of 0.687. Errors in numeric fields like duration and price posed challenges, as the model occasionally misclassified valid data as erroneous due to slight deviations from expected patterns. The balance between precision (0.703) and recall (0.671) suggests that while the model detected a wide range of errors, it was not overly conservative, leading to a moderate F1 score.

Olympics: The model's performance was the lowest for this dataset, with an accuracy of 68.8% and an F1 score of 0.629. Extensive typos and missing values in categorical fields like "Team" and "Host City" were particularly challenging. The lack of clear patterns in these columns affected the model's ability to generalize, leading to a lower recall (0.613) and precision (0.645). This dataset highlighted the model's limitations in handling highly unstructured or inconsistent categorical data.

D. *Conclusion*

The experiments demonstrate that Neural-Chat performs well on structured datasets with clear patterns and well-defined metadata, such as E-Commerce Transactions. However, its performance declines in text-heavy or highly variable datasets like Olympics and DAIGT V3 Train. These results emphasize the importance of dataset structure and the types of errors introduced in determining the model's effectiveness.

VI. Conclusion

This study investigated the capabilities of Intel's Neural-Chat model for automating data cleaning tasks across structured datasets. The project systematically evaluated the impact of incremental metadata inclusion and prompt engineering on the model's performance. The findings demonstrate that while the Neural-Chat model exhibits potential for identifying and correcting data anomalies, its effectiveness is highly dependent on the richness and structure of metadata provided.

A. *Key Observations*

1) **Metadata Inclusion**: The model's detection and correction performance improved significantly with the addition of metadata. However, excessive metadata led to overfitting, resulting in higher false positives during detection and reduced precision in correction.
2) **Dataset Characteristics**: The model performed best on structured datasets with well-defined errors, such as E-Commerce Transactions, where accuracy reached 85.1%. However, its performance declined in text-heavy or inconsistent datasets like Olympics and DAIGT V3 Train, highlighting its limitations in generalizing across varied data formats.
3) **Prompt Engineering**: Refining prompts to include detailed field-specific rules and leveraging few-shot examples enhanced the model's contextual understanding. However, reliance on examples occasionally caused overgeneralization, leading to misclassification of valid data as erroneous.

B. *Future Work*

To enhance the performance of LLMs like Neural-Chat in data cleaning, future research should focus on the following:

1) **Dynamic Metadata Balancing**: Develop mechanisms to balance metadata inclusion dynamically, ensuring sufficient context without overwhelming the model.
2) **Hybrid Approaches**: Combine rule-based systems with LLMs to handle domain-specific constraints and semantic dependencies more effectively.

3) **Improved Metadata Utilization**: Explore advanced techniques for integrating metadata into prompts, such as embedding column-level context directly into the model's architecture.
4) **Context-Aware Models**: Investigate methods to enhance the model's ability to generalize across unstructured or inconsistent datasets by incorporating additional contextual learning capabilities.

C. Good Practices for Metadata Inclusion in Prompts

Effective metadata inclusion is critical for guiding LLMs in data cleaning. Based on the findings, the following practices are recommended:

- Clearly specify field-level rules and constraints, such as expected formats and valid ranges, to minimize ambiguity.
- Leverage few-shot examples strategically, ensuring they represent typical error patterns without biasing the model towards overgeneralization.
- Introduce metadata incrementally, evaluating performance at each stage to determine the optimal level of context required for the task.

In conclusion, this study underscores the transformative potential of LLMs in automating data cleaning processes while highlighting the need for balanced metadata inclusion and robust prompt engineering to maximize their effectiveness. These insights lay the groundwork for future advancements in leveraging LLMs for maintaining data quality across diverse domains.