Data Cleaning Approaches in Software Engineering Tasks with LLM Integrations

Jay Casey
Department of Software Engineering (Data Science)
Rochester Institute of Technology
Rochester, New York
jwc7817@rit.edu

Emily Nover
Department of Software Engineering (Data Science)
Rochester Institute of Technology
Rochester, New York
en7367@rit.edu

***Index Terms*—meaning-typed programming, byLLM, data cleaning, dirty datasets, prompt engineering, regex-based scripts**

I. Introduction

Software engineers now can reap benefits from the speed and productivity improvements of working with Large Language Models (LLMs). Programs recently gained the ability to directly implement artificial intelligence (AI) solutions into their code without needing to write thousands of lines to achieve the same result.

Prompt engineering is the current standard of this integration, and it involves connecting to a LLM (via APIs, websites, apps) and writing specific instructions in the code to explain exactly what should be done. This is a lengthy process and leaves much room for error when it comes to clarity within the codebase. As prompt engineering is considered as a newer topic, many engineers are still illiterate when it comes to speaking the LLM's language. ByLLM is a solution to this problem. By using Meaning-Typed Programming (MTP), byLLM completely removes the need to write these lengthy prompts. MTP uses minimal written code in order to interpret what functionality the user wants the LLM to contribute to the program [1].

However, this shift of responsibility from explicit prompt creation from the user to MTP also introduces another risk, misinterpretation. Since the user does not directly inform the LLM of the desired action, there might be preventable errors where the model's output does not match the intended action. This project investigates the accuracy, reliability, and maintenance overhead of using byLLM's MTP-driven approach for automated data cleaning compared to traditional prompt-based and regex-based approaches.

II. Problem Statement

ByLLM's MTP-driven structure simplifies the process of utilizing AI-assisted integrations in programming tasks down to simply using the by operator with a model of the user's choice. Overhead within tasks is reduced with byLLM by delegating a variety of tasks to an user-specified LLM provider such as Anthropic, Google, OpenAI, or a local model. The complexity of using AI integration in programming tasks is disguised, as code is generated based on the user's intent without the excessive prompting that comes with prompt-based programming.

There is a promising nature of MTP and byLLM, but an omission lies in the research on the capability of these tools to use the `by` operator to detect and repair dirty data automatically compared to traditional data cleaning methods such as regex-based scripting [1]. Traditional approaches in data quality tasks can be prone to error due to the naturally restrained nature of data cleaning methods to specific domains. Consideration is not given to the ability to adapt across domains, datasets, or prevent the influence of real-world noise in datasets. This is a case where the data cleaning pipeline does not produce output that matches the intended result (*e.g.,* incorrect values or inconsistent formats), and we define that as misinterpretation in this project.

MTP and byLLM appear to be potential solutions to tackling problems in data quality tasks that occur in traditional approaches, but there are challenges. MTP is found to be sensitive to poor coding practices [1], struggling with cryptic and bad namings. Many, not all, but many datasets are considered to be dirty due to cryptic and bad naming conventions, along with missing or corrupted data. These issues can appear across datasets in various domains (*e.g.,* government data, sports data, business intelligence data). ByLLM also relies mainly on application programming interfaces (APIs) and/or local-only models to access AI. As this project is not a funded project, API access (and by extension, the number of API calls) is limited to the free tiers of providers. Having limits on API access and calls will narrow down the amount of datasets usable for this project, reducing the full scope of the analysis.

Regardless, in this project, we will implement a data cleaning pipeline in three ways: *i)* byLLM with the `by` operator, *ii)* prompt-based scripts that use the same underlying LLM directly, and *iii)* traditional regex-based scripts. We will use subsets of real-world generic dirty datasets that have common data quality problems such as inconsistent date formats, missing values, and noisy categorical fields. Subsetting the datasets allows us to stay within the API limits while allowing for other people to reproduce this experiment to some semblance but with more computing power or a better API budget. Our goal is not to analyze the domain itself, but to evaluate how

byLLM, prompt-based LLM scripts, and regex-based scripts perform as engineering solutions for cleaning such datasets in terms of misinterpretation rate (*e.g.,* schema violations or incorrect repairs), development effort, and robustness across domains.

III. Related Work

(*PHASE 3*) PLACEHOLDER FOR PHASE3 CONTENT

IV. Study Design

We will use Google Gemini via byLLM, direct API calls, and the webpage or app for Gemini. The rationale behind this decision lies in the fact that Rochester Institute of Technology (RIT) gives its student access to GenAI Tools [2] and Gemini is one of the two LLM on that list. Microsoft 365 Copilot is the second option on that list, but ease of access is an important aspect of this project, and Google Gemini becomes the winner due to students not having straightforward access to the Microsoft 365 suite compared to Google Gemini. Data cleaning using manual prompting will utilize the Gemini 3 model, as the current Gemini web/chat interface only allows the Gemini 3 model in Fast (free), Thinking (free), or Pro (paid) options. All byLLM-based cleaning will be used in the Gemini 2.5 family, which offers sufficient free-tier limits (see Section IX for details) compared to the Gemini 3 lineup for our sampled datasets. We estimate that we will make 250-500 [3] requests or less per day during the study and experiment phases of this project.

Approach Overview:

- Create subsets of dirty datasets to stay below the API limits.
- Data loading and basic preprocessing.
- Basic regex cleaning (date normalization, etc).
- Prompt-based LLM cleaning (hand-typed prompts, measure prompts, LOC, etc).
- byLLM cleaning (MTP types and `by` operator, measure LOC, misinterpretation, etc).
- Evaluation (compare outputs to a manually cleaned subset of datasets).

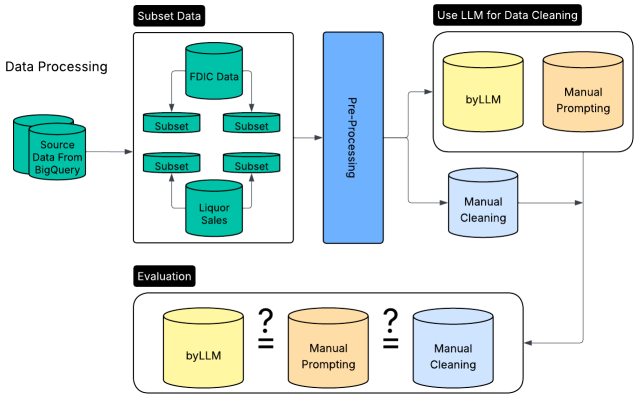

Fig. 1. Approach Overview Diagram.

As mentioned earlier in Section II, we define misinterpretation in this project as any instance where byLLM produces an output that violates the intended result (*e.g.,* wrong type, missing fields, or incorrect data) given a clearly defined prompt or specification. We will measure misinterpretation as the proportion of records with violations or incorrect repairs based on a ground-truth cleaned dataset.

As this project utilizes public LLM methods by the chat/web interface and the API access, we chose to provide the LLMs with subset data from the datasets. This is to stimulate mitigating information leakage in real-world workflow scenarios. The data subsets provided to the LLMs were only 250 rows, and the LLMs were primarily used to generate the cleaning logic, which was then checked locally with the full dataset. This prevents the entire "dirty" dataset from being used with the model's training or memory.

We will use a sample from two "dirty" datasets:

- *Dataset 1*: FDIC-Insured Banks and Branches [4]
- *Dataset 2*: Iowa Liquor Retail Sales [5]

With these two datasets, we will follow the same steps and compare them to the "ground-truth" dataset. With the manual prompting (using Google Gemini's web/chat interface) and the byLLM, we will use prompts with different levels of specificity. With the web/chat interface, this prompt was used following the suggested guidelines (links can be found in the `manual_prompting.ipynb` file):

> *You are an expert software engineer and data scientist. Your task is to help me build a robust data cleaning and processing pipeline for a dataset that may contain inconsistencies, missing values, or formatting errors. I will provide a small sample of the dataset (250 rows) for demonstration purposes. Your pipeline should generalize to larger datasets without exposing the full dataset.*
> *Requirements:*
> *1. Data Ingestion*
> - *Suggest and write code to load a dataset (.csv, .json, or similar).*
> - *Handle potential read errors gracefully.*
>
> *2. Data Cleaning*
> - *Detect and repair dirty records, including but not limited to:*
>   - *Inconsistent or invalid date formats*
>   - *Missing or null values in any column*
>   - *Misformatted numeric fields (extra characters, wrong decimal separators)*
>   - *Misformatted categorical fields (typos, inconsistent capitalization)*
>   - *Duplicate rows*
>   - *Trailing or leading whitespace*
>   - *Unexpected or invalid values (e.g., negative ages, impossible categories)*
>   - *Outliers that are likely data entry errors*
>   - *Any other records that might be considered dirty*

- *Prioritize high precision (avoid incorrect corrections) and high recall (detect most errors).*
- *Provide clear, step-by-step logic and code for each type of cleaning.*

3. Pipeline Skeleton

- *Structure the pipeline with modular functions (e.g., load_data(), clean_data(), analyze_data()).*
- *Include docstrings and comments for maintainability.*
- *Include a short demonstration of running the pipeline on the sample dataset.*

4. Validation & Testing

- *Suggest methods to check that cleaning worked correctly (e.g., counts of missing values, sample checks, consistency tests).*
- *Optionally, provide small example inputs and expected outputs.*

5. Maintainability Notes

- *Explain why this approach is easy to maintain and extend.*
- *Highlight potential bottlenecks or limitations.*

Output:

- *Python code implementing the pipeline, modular and well-documented.*
- *Brief explanation of each module and function.*
- *Demonstration of the pipeline applied to the sample dataset.*

Due to byLLM being built on MTP, a more simple, straightforward prompt was used:

Clean the data presented.

To ensure that the experiment stays within free API tiers of Google Gemini, we will limit LLM-based cleaning to subset samples of each dataset if they exceed 250 rows and cache all responses [**?**] to avoid repeated calls. We will primarily use the Gemini 2.5 model, which falls in the free tier usage for APIs, for any byLLM-based cleaning, the Gemini 3 model for the web/chat interface for any manual prompting cleaning, and run manually-typed/regex scripts on the full datasets for ground-truth and baseline comparisions. This approach design allows us to compare the three ways while staying within rate limits and cost barriers.

V. Research Question

The research question for this project is primarily focused on the data quality abilities of MTP using byLLM.

RQ1 (Data Quality): For dirty datasets, can the `by` operator (via byLLM) detect and repair dirty records (*e.g.*, inconsistent date formats, missing labels) with lower misinterpretation rate and comparable or better precision/recall than regex-based scripts and prompt-based LLM calls?

VI. Experiments and Preliminary Results

A. Precision, Recall, F1 Score

The averages for the columns of the manually prompted datasets, using the manually typed dataset as a ground-truth, the scores are as follows:

- Average Precision: 0.9693
- Average Recall: 0.9671
- Average F1 Score: 0.9671

These results are very promising. The average precision being 0.9693 means there is close to a perfect match against our ground truth dataset for the rows that persisted. In addition, the average recall is 0.9671, demonstrating that a large amount of the data that is expected is present. Unsurprisingly, the average F1 score is also very high at 0.9671. This shows that the manual prompting method succeeds at delivering high quality data cleaning without losing or hallucinating many records. Based on theses scores we can almost trust implicitly what the manual prompting produces.

B. Confusion Matrix

There are over 100 cols between both datasets, so exact confusion matrix analysis will happen in Phase 3 as we determine what cols are the best, or worst performing.

C. Schema Integrity

Did (if so, how many times) or did not the LLM "invent" data with the datasets?

Did the LLM change the data types or drop columns it wasn't supposed to?

Schema diffs between manual type/manual prompts:

- **Dropped Columns:**
  - `change_code_2`
  - `change_code_3`
  - `change_code_4`
  - `end_effective_date`
- **Invented Columns:** Null
- **Row Count Difference:** -1

The LLM demonstrated high precision, but lacked in recall at times. The resulting dataset dropped four columns. Three of which being `change_code_n` columns, which are used to signify a structural event relating to an institution. These were likely dropped if the sample contained all null values for these columns, as many records are empty for these fields. The last dropped column, `end_effective_date` can pose more of an issue when conducting time sensitive analysis. This column shows the date that ends the structural event related to these change codes. This may be an issue, as this could signify a date a branch closed operation.

Fortunately, the LLM did not hallucinate any columns, and there was only one less row in the result vs. the ground-truth dataset. The context matters in many datasets, creating a need for a subject matter expert to review the LLM's response to ensure crucial data is not being handled incorrectly.

D. Processing Time Across Different Subset Sizes

Will discuss the time differences of writing/running the manual typed, manual prompting, and the byLLM executions here once both datasets are added and all types of approaches are run. So far, it seems like there is a negligible difference between subset sizes for the manual cleaning and manual prompting approaches with the FDIC dataset. We shall see with the byLLM approach!

TABLE I
MANUAL TYPING PROCESSING TIMES (SECONDS) FOR THE FDIC DATASET AND THE IOWA LIQUOR SALES DATASET

	Subset Size (# of Rows)			
Dataset	250 rows	500 rows	1,000 rows	Full Dataset
FDIC	0.49s	0.50s	0.48s	0.77s
Iowa Liquor Sales	0.00s	0.00s	0.00s	0.00s

TABLE II
MANUAL PROMPTING PROCESSING TIMES (SECONDS) FOR THE FDIC DATASET AND THE IOWA LIQUOR SALES DATASET

	Subset Size (# of Rows)			
Dataset	250 rows	500 rows	1,000 rows	Full Dataset
FDIC	0.42s	0.51s	0.46s	0.76s
Iowa Liquor Sales	0.00s	0.00s	0.00s	0.00s

TABLE III
BYLLM PROCESSING TIMES (SECONDS) FOR THE FDIC DATASET AND THE IOWA LIQUOR SALES DATASET

	Subset Size (# of Rows)			
Dataset	250 rows	500 rows	1,000 rows	Full Dataset
FDIC	0.00s	0.00s	0.00s	0.00s
Iowa Liquor Sales	0.00s	0.00s	0.00s	0.00s

E. Technical Challenges

We encountered some challenges in building this study. The biggest challenge was the lack of uniformity between model versions. The byLLM API integration only supported Gemini 2.5, while the web/chat interface used for manual prompting was limited only to the Gemini 3 model. This is a bit of an issue since each model version will have differences in architecture and capabilities, meaning fully comparable outputs across approaches might not happen. This introduces a potential bias in evaluating performance and misinterpretation rates, since the differences between model versions may be at fault rather than the experimental approach. Since this challenge could not be avoided due to budget limitations, we tried to standardize the prompts as closely as possible between the two experiments that utilize LLMs, documented everything (from prompts to outputs), and made sure we considered model mismatches in our interpretation of results.

Building the byLLM pipeline proved to be a difficult task. The library is optimized for primitive data types or objects, both of which are not well optimized for this level of data manipulation within python. Another hurdle came in the form of output limits. As the process involved reading the entire data set, altering and returning all the data, the LLM reached a barrier with the amount it could generate.

With that in mind, we also had limitations with API access and rates. We are using only the free tier of Google Gemini, and it restricts the number of requests, which in turn limits our dataset size and results. We try to mitigate this by using batching and caching [**?**] as needed, and subsetting our full datasets to stay within the quota limits while also maximizing our ability to clean data and test the pipeline.

On the LLM side, we had to deal with potential model misinterpretation/semantic ambiguity, reproducibility between domains and datasets, and the dependence on a certain model type. Most of the current LLMs are black boxes with changing architectures and versions, which might cause different results after updates or small changes. This impacts reproducibility over time and changes the cleaning behaviors of the models used. To reduce this, we made sure to stay with specific API versions, logging data, and repetitions of prompts/outputs. One of the biggest issues with MTP and byLLM is their inability to infer on crytpic or bad naming conventions, and MTP's reduction of explicit prompting might cause the model to misinterpret the intent of the prompt and data, causing incorrect or incomplete data cleaning. We mitigated that by making sure to choose dirty datasets with clear naming conventions or descriptive variable names, and validation approaches to double-check outputs.

Another model misinterpretation issue is making sure misinterpretations are measured objectively without accidentally bringing in bias or skewing the output results. What is considered to be "wrong", or "dirty"? We made sure to use our manually typed and cleaned datasets as ground truth and to set up strict rules for what was a "violation" or a incorrect repair.

Lastly, despite the LLM tendency to vary in their outputs based on how the prompt or data is structured, we made sure to select datasets that were as generic as possible to ensure that we could create clear evaluation metrics. We also documented everything, and created clear organizational structure within the repository and the files so any changes can be documented and made across all files.

VII. THREATS TO VALIDITY

(*PHASE 3*) PLACEHOLDER FOR PHASE3 CONTENT

VIII. CONCLUSION

(*PHASE 3*) PLACEHOLDER FOR PHASE3 CONTENT

REFERENCES

[1] Dantanarayana, Jayanaka L. and Kang, Yiping and Sivasothynathan, Kugesan and Clarke, Christopher and Li, Baichuan and Kashmira, Savini and Flautner, Krisztian and Tang, Lingjia and Mars, Jason. 2025. MTP: A Meaning-Typed Language Abstraction for AI-Integrated

Programming. Proc. ACM Program. Lang. 9, OOPSLA2, Article 314 (October 2025), 29 pages. https://doi.org/10.1145/3763092
[2] GenAI Tools Available at RIT. https://www.rit.edu/its/generative-ai
[3] Gemini Rate Limits. https://ai.google.dev/gemini-api/docs/rate-limits#gsc.tab=0&gsc.q=free%20rate%20limits&gsc.sort=
[4] FDIC-Insured Banks and Branches. https://console.cloud.google.com/marketplace/product/fdic/insured-institutions?authuser=1&project=infra-core-488315-g1
[5] Iowa Liquor Retail Sales. https://console.cloud.google.com/marketplace/product/iowa-department-of-commerce/iowa-liquor-sales?authuser=1&project=infra-core-488315-g1

IX. Appendix

A. Free Tier Rate Limits (Per Project)

- Gemini 2.5 Pro: 2–5 requests per minute (RPM) and 50–100 requests per day (RPD).
- Gemini 2.5 Flash: 10–15 RPM and 250–1,500 RPD.
- Gemini 2.5 Flash-Lite: 15 RPM and up to 1,000–1,500 RPD.
- Token Limit: All models share a limit of 250,000 tokens per minute (TPM).
- Context Window: The 1 million token context window is available on the free tier for compatible models.
- *(THE ABOVE NEEDS A LINK FOR CITATION OR REFERENCE REASONS)*