The Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI-25)

Data Wrangling Task Automation Using Code-Generating Language Models

Ashlesha Akella, Krishnasuri Narayanam

IBM Research India
ashlesha.akella@ibm.com, knaraya3@in.ibm.com

Abstract

Ensuring data quality in large tabular datasets is a critical challenge, typically addressed through data wrangling tasks. Traditional statistical methods, though efficient, cannot often understand the semantic context and deep learning approaches are resource-intensive, requiring task and dataset-specific training. To overcome these shortcomings, we present an automated system that utilizes large language models to generate executable code for tasks like missing value imputation, error detection, and error correction. Our system aims to identify inherent patterns in the data while leveraging external knowledge, effectively addressing both memory-dependent and memory-independent tasks.

Introduction

Tabular datasets in industrial use cases represent structured data with numerous rows and columns. Since data is pivotal in business decision-making, preserving data quality has become paramount. Data wrangling tasks like missing value imputation, error identification, and error correction are crucial to enhancing the overall data quality of tabular datasets by resolving data inconsistencies and errors.

State-of-the-art data quality enrichment approaches found in the literature face distinct challenges. Traditional statistical methods (Van Buuren 2018; Gong et al. 2021; Thomas and Rajabi 2021) applied to large tabular datasets efficiently leverage statistical properties of structured data with minimal time and computational requirements, however, they usually fail to account for the semantic context of the data (e.g., impute the column *state* given *city*). While deep learning approaches (Lin, Tsai, and Zhong 2022; Samad, Abrar, and Diawara 2022; Huang et al. 2024) show promise, they require large training datasets and models tailored to specific tasks and datasets, making them time-consuming and resource-intensive to develop and deploy.

Large Language Models (LLMs) offer a novel approach (Iida et al. 2021; Narayan et al. 2022; Huh, Shin, and Choi 2023; Jaimovitch-López et al. 2023; Liu et al. 2023, 2024; Ashlesha et al. 2024) to address these challenges in data quality enrichment due to their extensive knowledge gained from training on massive datasets. However, data wrangling with LLMs applied on row-level tabular data can become computationally expensive for large datasets, making it difficult to balance accuracy and efficiency.

Memory-Dependent Task

```
def task(input_dict, reference_table):
  try:
    input_value = input_dict.get('Airport Country Code')
    value = reference_table.loc[reference_table['Country code'] ==
input_value, ['Country']]
    return str(value.values[0][0])
  except:
    return "Unknown"
```

Memory-Independent Task

```
def task(input_dict):
  sub_category = input_dict['sub_category']
  if re.match(r'Biscuits', sub_category):
    return 'Snacks & Branded Foods'
  elif re.match(r'Organic Fruits', sub_category):
    return 'Fruits & Vegetables'
  else:
    return 'Unknown'
```

Figure 1: Auto generated code by our system for Missing Value Imputation task. Impute 'Category' in *BigBasket* dataset (above) and 'Country Name' in *Airline* dataset.

To address this challenge, code-generating LLMs can be leveraged to automatically translate data patterns into concise executable code, enabling efficient tasks like imputation, error detection, and correction. While (Li and Döhmen 2024) work has proposed code generation workflows, they often face limitations that require external knowledge or iterative refinement, highlighting the need for more adaptable systems that incorporate both inherent data patterns and external information.

Our system uses LLMs to automatically generate code for data-wrangling tasks, incorporating relevant external knowledge as needed. As accommodating numerous columns in a dataset is resource-intensive, the system selects only the columns semantically relevant to the task. This targeted approach enables the LLM to efficiently generate code, allowing the system to execute data-wrangling tasks without additional LLM calls per row, thereby enhancing scalability for large datasets. Moreover, our system employs an iterative refinement technique to optimize the generated code snippets.

Leveraging Code-Generating LLMs

Tabular datasets often contain inherent patterns with dependencies between specific columns. E.g., the '24-Hour Ser-

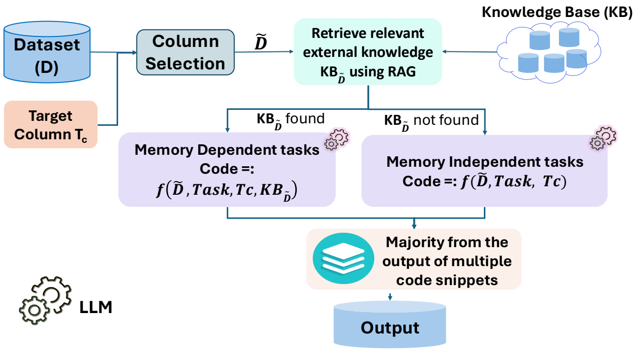

Figure 2: Workflow of data wrangling task automation through code generation

vice' column in the 'Starbucks' dataset (Alice 2017) can be set to 'True' if both 'Opening Time' and 'Closing Time' are midnight. Such patterns can be represented as concisely formulated executable code. Our system automates code generation for data-wrangling tasks leveraging LLMs trained on extensive code datasets (Mishra et al. 2024). It reduces manual effort and minimizes repeated LLM calls, enhancing computational efficiency. Figure 1 shows an example code generated by our system using an iterative prompting approach. It even allows the LLM to refine its output based on previous responses.

Methodology

Our system classifies tasks into two primary categories based on the necessary external knowledge:

1. **Memory-Dependent tasks**: They require the incorporation of external or organizational data, such as established mappings (e.g., cities to states) or evolving business mappings (e.g., imputing job role based on job title)
2. **Memory-Independent tasks**: They solely depend on the input table and can be further categorized into two groups: (i) *Row-alone Tasks*: Data present in the current row is sufficient to accomplish these tasks without any contextual knowledge (e.g., imputing the '24-Hour Service' column using 'Opening Time' and 'Closing Time' columns). (ii) *Few-shot Tasks*: Such tasks can be inferred using contextual knowledge learnt from a small number of examples provided as part of the prompt.

Figure 2 depicts the overall architecture of our system. Given a task dataset D and a target column, our system employs a Histogram-based Gradient Boosting Classification Tree (Guryanov 2019) method to identify columns relevant to the target column, resulting in the filtered dataset $\widetilde{D}$. Inspired by the RAG approach (Lewis et al. 2020), our system finds if there is any data in the external Knowledge Base (KB) that is semantically similar to $\widetilde{D}$. If there exists any such data ($KB_{\tilde{D}}$) exceeding a preset similarity threshold, the *Memory-Dependent* workflow is executed using the selected $KB_{\tilde{D}}$; else, the *Memory-Independent* workflow proceeds. Both workflows use iterative prompting for code generation.

Code Generation

For data imputation and error correction, the system uses non-null rows in the target column as ground truth G. For error detection, users provide a small annotated dataset with 'Yes'/'No' labels as ground truth. Our system employs k-fold cross-validation on G, generating code with k-1 folds and validating with the k^{th} fold. It generates multiple code snippets through k-fold cross-validation, each capturing different patterns. Data-wrangling task is executed with each code snippet, and final output is determined through a majority consensus among the outputs of all code snippets.

Iterative Prompt Generation: The prompts for the system are constructed through an iterative process (Wang, Deng, and Sun 2022). The initial prompt contains instructions and a small set of randomly selected rows from the ground truth data G. In subsequent iterations, the prompt is enhanced by incorporating the most effective code generated in previous iterations.

In each iteration, for *Memory-Dependent* tasks, code is generated using external data $KB_{\tilde{D}}$ and validated on ground truth G. For *Memory-Independent* tasks, system tries the *Few-shot* method only after exhausting *Row-alone* method. In *Row-alone* method, a few diverse samples are selected from G using K-means clustering algorithm (with samples representing centroids of the clusters) to generate a code snippet, which is applied on G. If accuracy of this operation is ≥ 0.9, then the same code is applied on D. In *Few-shot* method, one randomly selected sample r from G is provided along with few-shot examples from G that are semantically similar to r to generate a code snippet. We used the LLM model *granite-34b-code-instruct-8k* (Mishra et al. 2024).

The prompt is structured with instructions: *Task Description*: Directing the LLM to "Write a Python code..". *Function Behavior*: Instructing to detect patterns, and return "Unknown" when no pattern is found. *Example Data*: A small randomly selected sample of n rows from the input table, with each row separated by newlines and column values by semicolons. *Example Code*: The most effective code is selected from the previous iteration of k-fold cross-validation. *Reference Table*: For Memory-Dependent tasks, a sample from the reference data $KB_{\tilde{D}}$ is added to the prompt.

Results

The results (using Airline (Banerjee 2023) and BigBasket (SJ 2022) datasets) show that our system achieves comparable performance to a system that calls the LLM for each row, while significantly reducing the number of LLM calls.

Task	Dataset	row-wise	code-gen
Imputation	Airline	0.97 (*#1376*)	0.99 (*#20*)
Error Detection	Airline	0.98 (*#1376*)	0.98 (*#15*)
Error Correction	Airline	0.98 (*#1376*)	0.98 (*#15*)
Imputation	BigBasket	0.94 (*#1512*)	0.94 (*#30*)
Error Detection	BigBasket	0.94 (*#1512*)	0.98 (*#32*)
Error Correction	BigBasket	0.92 (*#1512*)	0.94 (*#34*)

Table 1: Accuracy of row-wise LLM invocation vs. code generation approach (with the number of LLM calls #)

References

Alice. 2017. Startbucks Dataset. https://data.world/alice-c/starbucks/workspace/file?filename=Starbucks+in+California.csv. Accessed on 3-Oct-2024.

Ashlesha, A.; Manatkar, A.; Chavda, B.; and Patel, H. 2024. An Automatic Prompt Generation System for Tabular Data Tasks. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL): Human Language Technologies (Industry Track)*, 191–200.

Banerjee, S. 2023. Airline Dataset (Navigating the Skies: Exploring Insights from Synthetic Airline Data). https://www.kaggle.com/datasets/iamsouravbanerjee/airline-dataset. Accessed on 3-Oct-2024.

Gong, Y.; Li, Z.; Zhang, J.; Liu, W.; Yin, Y.; and Zheng, Y. 2021. Missing value imputation for multi-view urban statistical data via spatial correlation learning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 35(1): 686–698.

Guryanov, A. 2019. Histogram-based algorithm for building gradient boosting ensembles of piecewise linear decision trees. In *Proceedings of the 8th International Conference on Analysis of Images, Social Networks and Texts (AIST)*, 39–50.

Huang, B.; Zhu, Y.; Usman, M.; and Chen, H. 2024. Semi-supervised learning with missing values imputation. *Knowledge-Based Systems (KBS)*, 284: 111171.

Huh, J. S.; Shin, C.; and Choi, E. 2023. Pool-Search-Demonstrate: Improving Data-wrangling LLMs via better in-context examples. In *NeurIPS 2023 Second Table Representation Learning Workshop*.

Iida, H.; Thai, D.; Manjunatha, V.; and Iyyer, M. 2021. TABBIE: Pretrained Representations of Tabular Data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL): Human Language Technologies*, 3446–3456.

Jaimovitch-López, G.; Ferri, C.; Hernández-Orallo, J.; Martínez-Plumed, F.; and Ramírez-Quintana, M. J. 2023. Can language models automate data wrangling? *Machine Learning (ML)*, 112(6): 2053–2082.

Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.-t.; Rocktäschel, T.; Riedel, S.; and Kiela, D. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 9459–9474.

Li, X.; and Döhmen, T. 2024. Towards Efficient Data Wrangling with LLMs using Code Generation. In *Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning (DEEM)*, 62–66.

Lin, W.-C.; Tsai, C.-F.; and Zhong, J. R. 2022. Deep learning for missing value imputation of continuous data and the effect of data discretization. *Knowledge-Based Systems (KBS)*, 239: 108079.

Liu, S.-C.; Wang, S.; Chang, T.; Lin, W.; Hsiung, C.-W.; Hsieh, Y.-C.; Cheng, Y.-P.; Luo, S.-H.; and Zhang, J. 2023. JarviX: A LLM no code platform for tabular data analysis and optimization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP): Industry Track*, 622–630.

Liu, Y.; Tao, S.; Zhao, X.; Zhu, M.; Ma, W.; Zhu, J.; Su, C.; Hou, Y.; Zhang, M.; Zhang, M.; et al. 2024. CoachLM: Automatic Instruction Revisions Improve the Data Quality in LLM Instruction Tuning. In *Proceedings of the IEEE 40th International Conference on Data Engineering (ICDE)*, 5184–5197.

Mishra, M.; Stallone, M.; Zhang, G.; Shen, Y.; Prasad, A.; Soria, A. M.; Merler, M.; Selvam, P.; Surendran, S.; Singh, S.; et al. 2024. Granite code models: A family of open foundation models for code intelligence. *arXiv preprint arXiv:2405.04324*.

Narayan, A.; Chami, I.; Orr, L.; Arora, S.; and Ré, C. 2022. Can Foundation Models Wrangle Your Data? *Proceedings of the VLDB Endowment (PVLDB)*, 16(4): 738–746.

Samad, M. D.; Abrar, S.; and Diawara, N. 2022. Missing value estimation using clustering and deep learning within multiple imputation framework. *Knowledge-Based Systems (KBS)*, 249: 108968.

SJ. 2022. BigBasket Entire Product List (Analyzing BB Products and their performance across). https://www.kaggle.com/datasets/surajjha101/bigbasket-entire-product-list-28k-datapoints. Accessed on 3-Oct-2024.

Thomas, T.; and Rajabi, E. 2021. A systematic review of machine learning-based missing value imputation techniques. *Data Technologies and Applications (DTA)*, 55(4): 558–585.

Van Buuren, S. 2018. *Flexible imputation of missing data*. CRC press.

Wang, B.; Deng, X.; and Sun, H. 2022. Iteratively Prompt Pre-trained Language Models for Chain of Thought. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2714–2730.