

LLM-based Data Operators for Data Processing

Deepak Raj Iti¹, Vandana Jada², Praveen Kandukuri³, Divyanjali Peraka⁴

¹Student, Dept. of Electronics Engineering, Indian Institute of Technology (ISM), Dhanbad

²Student, Dept. of Data Science, Texas A&M University, College Station

³Student, Dept. of Metallurgical Engineering, National Institute of Technology, Tiruchirapalli

⁴Operations Research, UC Berkeley, Berkeley

Abstract - Data processing is essential in machine learning pipelines to ensure data quality. Many applications use user-defined functions (UDFs) for this, offering flexibility and scalability. However, the rising demands on these pipelines present three challenges: low-code limitations, dependency issues, and a lack of knowledge awareness. To tackle these, we propose a new design pattern where large language models (LLMs) serve as a generic data operator (LLM-GDO) for effective data cleansing, transformation, and modeling. In this pattern, user-defined prompts (UDPs) replace specific programming language implementations, allowing LLMs to be easily managed without runtime dependency concerns. Fine-tuning LLMs with domain-specific data enhances their effectiveness, making data processing more knowledge-aware. We provide examples to illustrate these benefits and discuss the challenges and opportunities LLMs bring to this design pattern.

Key Words: Large Language Models, Data Modeling, Data Cleansing, Data Transformations, Design Pattern

1. INTRODUCTION

Machine learning (ML) drives a variety of data-driven applications across different use cases. A typical machine learning pipeline comprises several steps: data processing, feature engineering, model selection, model training, hyperparameter tuning, evaluation, testing, and deployment [1]. Many of these steps require high-quality data to ensure that machine learning applications perform as expected, and they often benefit from large volumes of data to support effective training[2].

However, most reliable datasets are generated through human annotations, a process that is both expensive and time-consuming, making it difficult to scale. As machine learning models become more complex and involve increasing numbers of parameters, the demand for vast amounts of high-quality data for training continues to grow. Data processing tasks must also adapt to this increasing need for effective data cleansing, transformation, and modeling. Therefore, this work focuses primarily on the transformation process as defined in a typical ETL (Extract, Transform, Load) framework in data warehousing.

To support the growing demand for data transformation, user-defined functions (UDFs) are commonly utilized. These functions clean, transform, and model data within a data warehouse or data lake[3]. A typical UDF template is written in a Pythonic style, allowing users to import runtime dependencies, implement processing logic, and handle input data. When a UDF is applied to a database, following a narrow transformation model in Spark [4], it processes each row of data individually, storing the transformed rows.

The UDF design pattern offers three significant advantages in large-scale data processing:

1. Flexibility: Users can implement their own data processing logic, even if it isn't supported by built-in functions.
2. Modularity: UDFs provide abstraction, facilitating better understanding, debugging, and reusability of code.
3. Scalability: UDFs can easily scale using big data processing engines like Spark.

Despite these advantages, UDFs also face several challenges:

1. Not low-code or zero-code: Users must possess substantial programming skills and experience to create UDFs.
2. Not dependency-free: UDFs can require complex runtime environments, making dependency management difficult during development and deployment. If UDFs have completely non-overlapping runtime dependencies, separate pipelines are needed for each.
3. Not knowledge-aware: It is challenging to incorporate prior knowledge into UDFs for data processing tasks. This knowledge is often task-specific; for instance, classifying e-commerce item categories requires extensive domain expertise to identify item attributes. Integrating such knowledge deterministically into UDFs is complicated due to the vast combinations of item attributes.

Recently, the development of artificial intelligence (AI) has made significant progress with the emergence of Large Language Models (LLMs). Models like Llama2 and GPT-4 have demonstrated their effectiveness in addressing a wide range of downstream tasks, such as question answering and multi-step reasoning, thanks to their emergent abilities. This

progress has narrowed the gap between natural language processing and programming.

With well-crafted prompts, users who lack extensive experience in data processing can effectively use an LLM to extract product features—tasks that typically require expertise from e-commerce professionals [6]. This capability, which relies solely on natural language instructions and input-output examples without the need to optimize any parameters, is known as in-context learning. This in-context learning allows LLMs to understand few-shot and even zero-shot learning tasks, such as tabular data classification [7] and anomaly detection in system logs [8].

Like other pre-trained models, the performance of LLMs can be further enhanced through fine-tuning techniques. Approaches like LoRA and QLora optimize the rank decomposition matrices of the dense layers within a neural network, leading to LLMs that can achieve human-comparable results in many tasks [9]. This advancement reduces the human effort required for labeling and annotation.

In our paper, we address the limitations of current user-defined function (UDF)-based data processing practices and summarize a new design pattern that incorporates LLMs and user-defined prompts (UDPs) to balance flexibility and human-level accuracy. We propose that LLMs with UDPs can serve as Generic Data Operators (LLM-GDOs) for data cleansing, transformation, and modeling. An example of the LLM-GDO design pattern is illustrated in Figure 1-(b).

In the LLM-GDO design, we simplify the UDF concept with two main changes. First, instead of defining a programming-language-based UDF, users can create prompts or prompt templates to describe data processing logic, resulting in low-code and zero-code solutions. When the data distribution changes, users can easily update the processing logic by modifying the instructions and examples in the prompts, rather than altering the processing code. Unlike a traditional UDF, which requires runtime dependencies for execution, an LLM (whether pre-trained or fine-tuned) can act as a compiler for the prompt and execute requests independently, without dependencies.

By maintaining the same LLM under appropriate version control, we can align offline development with online serving. In data processing, the database communicates with the remote LLM resource through LLM gateways (such as APIs or agents), where the LLMs are managed to handle requests. We refer to this function as an "LLM call," with the implementation details handled by platforms, effectively abstracting complexity from users. By fine-tuning LLMs, we can seamlessly integrate domain-specific knowledge into them using a small dataset, thereby enhancing their performance on specific tasks.

Although LLMs are versatile tools, they do have limitations that we must continue to address. To provide a comprehensive overview of this design pattern, we also explore the challenges it presents.

Our contributions are summarized as follows:

- We introduce the LLM-GDO design pattern for machine learning (ML) pipelines in big data contexts.
- We summarize the potential applications of LLM-GDOs.
- We discuss the challenges and opportunities associated with LLM-GDO.

2. PRELIMINARIES

1. Narrow Transformations and Wide Transformations:

Data transformations refer to the instructions used to modify the rows of a database. In Spark, data transformations can be categorized into narrow transformations (Figure 2-(a)) and wide transformations (Figure 2-(b)), depending on the dependencies between data points. Typically, the output of a narrow transformation depends on only one input dataset, meaning there is no data shuffling involved. In contrast, the output of a wide transformation relies on multiple input rows and involves data shuffling. In our paper, we focus on narrow transformations, as they are predominantly used in early-stage data processing steps to enhance data quality.

2. LLMs: Function Calling and Fine-Tuning:

There are various methods to access large language models (LLMs), with the OpenAI API being one popular option. In our paper, we assume that the LLM gateways are mainly maintained by the platforms, whether they are remote or local. In Figure 1-(b), we represent LLM function calling through the 'llm call' function. This function takes a formatted user-defined prompt (UDP) and returns the processed output. LLMs can be fine-tuned using a small sample of high-quality data. As new data enters the database, the platform can seamlessly fine-tune the LLMs by extracting a small sample of high-quality data.

```

(a) an UDF template
# define an UDF
def user_define_function(inputs):
    # Import the dependencies
    # implement the logic based on the use cases
    # process the inputs
    # return the processed data
    return processed_inputs

# SQL query
SELECT user_define_function(inputs)

(b) an UDF template with LLMs and UDP (LLM-GDO)
# define a user-defined prompt template
prompt_template = "
<define the system prompt parameter>
<define the prompt incorporating with the (inputs) and the
output.>"

def user_defined_function(inputs):
    # an LLM_call function communicates with LLM engine
    # and returns the processed result
    processed_inputs = llm_call(prompt_template.format(inputs))
    return processed_inputs

# SQL query,
SELECT user_define_function(inputs)

```

Fig -1: (a) UDF design pattern and (b) LLM-GDO design pattern.

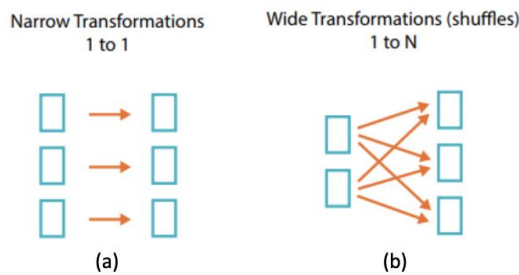


Fig -2: (a) Narrow Transformations and (b) Wide Transformations

3. METHODOLOGY

In this section, we present a list of tasks where LLM-GDOs (Large Language Model-Generated Data Operations) could enhance User Defined Functions (UDFs). It is important to note that LLMs are still evolving, so we primarily focus on design patterns in this section to illustrate the connection between UDFs and LLM-GDOs. A comprehensive discussion about the challenges of current LLM-GDO design will be provided in Section IV. For brevity, we will reuse the definition of "user defined function" from Figure 1-(b) in the following case studies. All example LLM outputs in this paper were generated using ChatGPT 3.5.

A. Data Cleansing and Transformation

Data cleansing and transformation are crucial steps to ensure the performance of a machine learning pipeline. To compare UDFs and LLM-GDOs in data cleansing and transformation, we consider a sample table titled "item rating," as defined in Figure 3. The following three tasks illustrate the low-code feature and dependency-free attribute of LLM-GDOs:

item_id	user_id	user_rating	date
"101"	"201"	3	"20220305"
"102"	"201"	4	"2022/10/23"
"101"	"202"	5	"7th April 2021"
"101"	"203"	2	"Feb 03 2020"

Fig -3: A sample table 'item rating' with columns 'item id', 'user id', 'user rating' and 'date'.

1. Data Structural Consistency: Ensuring data structural consistency is usually one of the initial steps. It helps organize the data to improve downstream transformations. In the "item rating" table (Figure 3), the "date" column contains date strings in different formats. Figure 4 shows an example of how to standardize the date data. With LLM-GDOs in UDFs, we can specify the desired output format (YYYYMMDD) in the prompt and allow the LLMs to manage the data processing (see Figure 4-(b)). Traditional UDFs can complete this structuralization as well, but they typically require either the enumeration of date formats or the use of various packages.

2. Data Type Conversion: After achieving better structural consistency, we can convert data from one type to another. Figure 5 presents an example of converting date data into UNIX epoch time. Again, the LLM-GDO performs this data transformation based on the instructions provided in the prompt, while traditional UDF users must understand the definition of UNIX epoch time to implement it or know which packages to utilize.

3. Data Standardization: Another important task is normalizing numeric data. In many machine learning pipelines, the normalization of numeric values enhances the stability of model training. Figure 5 illustrates an example in which LLM-GDO is employed to normalize the "user rating" column by providing the appropriate rating range.

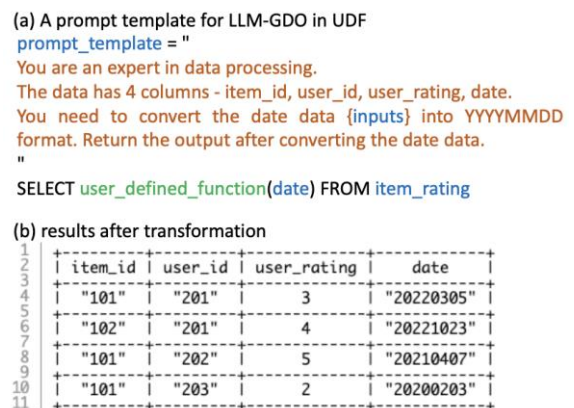


Fig -4: LLM-GDO-based data transformation for data structural consistency.

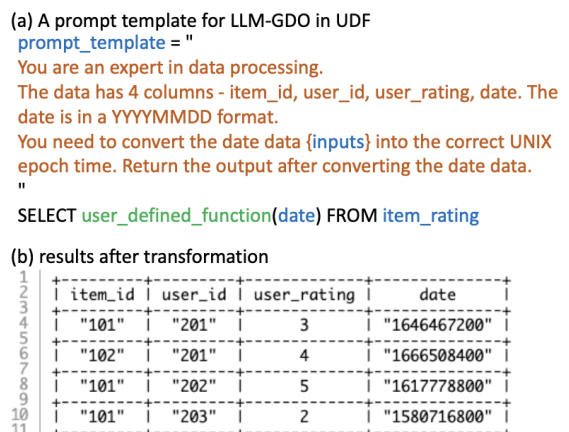


Fig -5: LLM-GDO-based data transformation for data type conversion.

B. Data Modeling

In addition to data cleansing and transformation, many feature engineering steps require machine learning models for basic reasoning and generating high-quality features. For

example, in many natural language processing (NLP) tasks, word annotations and tagging are critical features for modeling [10] [11]. These features often necessitate a dedicated machine learning pipeline to extract them, which can be inconvenient to maintain without proper support from machine learning operations. Users must thoroughly understand the machine learning pipeline to execute a user-defined function (UDF) that utilizes these complex models, managing dependencies and an additional layer of data processing.

However, LLM-GDOs (Large Language Model-based Generalized Data Operations) maintain the same convenience with a uniform approach. We consider a new sample table, 'item information,' as defined in Figure 6. We highlight this advantage through the following two tasks:

1. Reasoning: In this task, we need to parse each row of data and perform reasoning (e.g., classification) to generate output. A common use case is detecting anomalous values in the database. For instance, in many system log databases, system error messages are categorized by severity levels or types of events. Another example is categorizing items into product types in e-commerce for item display. While classification may yield invalid outcomes, identifying anomalies can be challenging due to the vast data volume. Anomaly detection could be achieved through separate machine learning pipelines, but integrating them into the database poses difficulties. First, these machine learning pipelines come with different dependencies and require domain knowledge. Second, as the underlying data changes, models may not be updated without a proper orchestration system for model retraining.

item_id	item_type	item_name
"101"	"game console"	"Xbox Series S - 1TB SSD All-Digital Gaming Console"
"102"	"game controller"	"Xbox Core Wireless Controller - Shock Blue"
"103"	"game controller"	"Controller Charger for Xbox Series"
"104"	"computer monitor"	"SAMSUNG 23.5" Curved Computer Monitor"

Fig -6: A sample table 'item information' with columns 'item id', 'item types' and 'item name'.

```
(a) LLM-GDO in UDF
prompt_template = "
You are the expert on product catalog classification for an e-commerce platform.
Product type is a group of products which fulfill a similar need for a market segment or market as a whole.
For example, the product type of the product 'SAMSUNG 65-Inch Class Crystal UHD 4K' is '4K Television'.
Check whether '{item_type}' is the correct product type for a product '{item_name}'.
If yes, stay with the given product type. If no, you need to come up the correct product type.
"

def user_defined_function(item_type, item_name):
    # an LLM_call function communicates with LLM engine
    # and returns the processed result
    processed_inputs = llm_call(prompt_template.format(item_type, item_name))
    return processed_inputs

SELECT user_defined_function(item_type, item_name) FROM item_information

(b) results after transformation
item_id | item_type | item_name
-----|-----|-----
"101" | "game console" | "Xbox Series S - 1TB SSD All-Digital Gaming Console"
"102" | "game controller" | "Xbox Core Wireless Controller - Shock Blue"
"103" | "wireless controller charger" | "Controller Charger for Xbox Series"
"104" | "computer monitor" | "SAMSUNG 23.5" Curved Computer Monitor"
```

Fig -7: An LLM-GDO application for data reasoning and anomaly detection on item type classification.

LLM-GDO can perform reasoning to detect anomalies in such cases. Figure 7 illustrates an example of LLM-GDO for anomaly detection in the 'item information' table. We can observe that LLM-GDO successfully identifies and corrects the incorrect item type for item '103' shown in Figure 6. In this prompt, we can leverage the in-context learning capabilities of LLMs by designing a suitable prompt. In contrast to traditional UDFs, where users must define complex heuristics or different versions of deep learning models to complete the task, LLM-GDO simplifies logic development and maintenance. LLMs on the back end can be managed centrally, allowing tasks to be powered without exposing the underlying details to users.

2. Embedding Database: Another significant use case for data processing is embedding generation. With LLM-GDO, we can produce high-quality embeddings based on the 'item name' column. Due to space constraints, we will omit the prompt examples and output embeddings here. However, this approach enables timely updates to the embeddings for downstream modeling.

4. DISCUSSION

Despite the remarkable human-compatible and programmable performance of large language models (LLMs) in data transformation, it is essential to highlight the opportunities and challenges inherent in the LLM-Generative Data Optimization (GDO) design pattern. This understanding can illuminate potential future research directions and enhance our comprehension of this evolving field.

1. LLM Inference and Scalability: While LLMs demonstrate greater scalability compared to traditional user-defined functions (UDFs), they require significantly more computational resources. This is due to the massive number of parameters in LLMs, which demand extensive computational resources for model inference. For instance, a GPT-3-sized LLM necessitates the use of eight 80G A100 GPUs for inference. Additionally, current commercial LLMs, such as GPT-3.5 and PaLM2, have limited response speeds. Nonetheless, research on LLM knowledge distillation [12] and compression [13] presents promising avenues for reducing LLM size and computational demands, ultimately making them more cost-effective, efficient, and scalable.

2. LLM Hallucination: Hallucination refers to instances where an LLM generates content that does not exist in the real world or does not meet the criteria specified in the prompt. While hallucination can be advantageous in creative use cases, it typically poses challenges in data processing [14]. For example, when the desired output format is tab-separated values (TSV), it is critical to avoid inconsistencies, such as using alternative separators, especially when integrating an LLM into a production data engineering pipeline. Various methods have been proposed to mitigate and quantify hallucinations in LLMs, including reducing the model's temperature, providing more context in the prompt,

employing a chain-of-thought approach, ensuring self-consistency, and specifying precise formatting requirements within the prompt. Despite these efforts, effectively detecting and preventing hallucinations in LLMs during data processing remains a significant challenge.

3. LLM Unit Testing and Evaluation: Traditional UDFs are typically deterministic, which facilitates the development of unit tests to assess their correctness. In contrast, LLMs use a generative approach to produce results based on probability [15], making universal testing more challenging. Recently, some teams have leveraged larger LLMs to generate unit test cases (e.g., expected outputs) for prospective LLMs. We believe that LLM unit testing will garner more attention in LLM development and application.

4. LLM Privacy: Fine-tuning an industry-level LLM for data processing may require data from various departments, which involves data transition and sharing. For instance, when processing data from a social application, customer profiles and preferences are highly sensitive. Such data sharing increases the likelihood of data leaks and may violate data privacy policies [16]. Recent studies in federated learning address privacy concerns related to LLMs. We anticipate growing research interest in LLM privacy issues.

5. CONCLUSIONS

In this paper, we introduce a novel design pattern called LLM-GDO, which aims to enhance the efficiency and reliability of data transformation. LLM-GDO leverages low-code and dependency-free implementations for knowledge-aware data processing. However, it also faces challenges associated with large language models (LLMs). We explore these challenges and opportunities, providing a comprehensive perspective on the LLM-GDO design pattern.

REFERENCES

- [1] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwin-ski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe et al., "Accelerating the machine learning lifecycle with mlflow." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.
- [2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [3] A. Nambiar and D. Mundra, "An overview of data warehouse and data lake in modern enterprise data management," *Big Data and Cognitive Computing*, vol. 6, no. 4, p. 132, 2022.
- [4] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin et al., "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.
- [5] <https://www.anyscale.com/blog/fine-tuning-llama-2-a-comprehensive-case-study-for-tailoring-models-to-unique-applications>
- [6] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale et al., "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.
- [7] OpenAI, "Gpt-4 technical report," 2023.
- [8] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM computing surveys (CSUR)*, vol. 54, no. 2, pp. 1–38, 2021.
- [9] C.-Y. Hsieh, C.-L. Li, C.-K. Yeh, H. Nakhost, Y. Fujii, A. Ratner, R. Krishna, C.-Y. Lee, and T. Pfister, "Distilling step-by-step! Outperforming larger language models with less training data and smaller model sizes," *arXiv preprint arXiv:2305.02301*, 2023.
- [10] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang, "A survey on model compression for large language models," *arXiv preprint arXiv:2308.07633*, 2023.
- [11] 2023.
- [12] M. Zhang, O. Press, W. Merrill, A. Liu, and N. A. Smith, "How language model hallucinations can snowball," *arXiv preprint arXiv:2305.13534*, 2023.
- [13] Z. Zhang, A. Zhang, M. Li, and A. Smola, "Automatic chain of thought prompting in large language models," *arXiv preprint arXiv:2210.03493*, 2022.
- [14] [23] J. Huang, S. S. Gu, L. Hou, Y. Wu, X. Wang, H. Yu, and J. Han, "Large language models can self-improve," *arXiv preprint arXiv:2210.11610*, 2022.
- [15] Y. Chang, X. Wang, J. Wang, Y. Wu, K. Zhu, H. Chen, L. Yang, X. Yi, C. Wang, Y. Wang et al., "A survey on evaluation of large language models," *arXiv preprint arXiv:2307.03109*, 2023.
- [16] S. Ghayyur, J. Averitt, E. Lin, E. Wallace, A. Deshpande, and H. Luthi, "Panel: Privacy challenges and opportunities in LLM-Based chatbot applications." Santa Clara, CA: USENIX Association, Sep. 2023.