DATALORE: can a large language model find all lost scrolls in a data repository?

Yuze Lou[1*], Chuan Lei[2], Xiao Qin[2], Zichen Wang[2], Christos Faloutsos[2], Rishita Anubhai[2], Huzefa Rangwala[2+]
[1]*University of Michigan,* [2]*Amazon Web Services*
yuzelou@umich.edu, {chuanlei|drxqin|zichewan|faloutso|ranubhai|rhuzefa}@amazon.com

***Abstract*—How can we effectively generate missing data transformations among tables in a data repository? Multiple versions of the same tables are generated from the iterative process when data scientists and machine learning engineers fine-tune their ML pipelines, making incremental improvements. This process often involves data transformation and augmentation that produces an augmented table based on its base version and related tables. However, data transformations are often not well-documented or completely missing, resulting in poor traceability, reproducibility and explainability of ML pipelines. In this paper, we propose DATALORE, a framework that explains data changes between an initial dataset and its augmented version to improves traceability. Given a base table, DATALORE first discovers its potentially related tables from the data repository using a variety of data discovery techniques. DATALORE then effectively leverages a large language model (LLM) to generate a variety of data transformations that lead to the augmented table. DATALORE validates these transformations and selects the minimum number of related tables to ensure traceability and reproducibility of the ML pipelines. A preliminary experiment shows that DATALORE is able to effectively recovery data transformations on two benchmark datasets.**

I. INTRODUCTION

In a machine learning (ML) task, data scientists and ML engineers often work as a team, iteratively fine-tune ML pipelines, make incremental improvements, and ensure the model's robustness and generalization capabilities. Such iterative process involves tasks like data cleaning, data augmentation, and feature engineering. Unlike code, data changes often lack sufficient information regarding the specific source data used to generate the released data and the transformations applied to each source, which gives rise to significant concerns with *data traceability* and *reproducibility*.

Data traceability is important for establishing a well-documented ML pipeline and ensures the data integrity for model training and supports compliance with regulations and best practices. Without proper documentation, it becomes challenging to track what the original data has been used, how it has been transformed, and to ensure compliance with licensing requirements. Data reproducibility is a foundational aspect of data-driven decision-making and ensures that the findings and insights derived from ML pipelines are trustworthy, valid, and reproducible by others. Even the data is deemed trustworthy, certain data transformations may still need to be removed for legal and licensing reasons. Although datasets are available on open data portals and sharing platforms like data.gov and Auctus[1], data transformations are generally not shared. This absence of information hinders the ability to reproduce results and undermines the trustworthiness of the data among its users.

Motivating example. Figure 1 presents a toy example with a data repository consisting of five tables. The augmented table T' was derived from the base table T by joining with the other three related tables *district*, *school_national_center* and *schools*. Knowing how these tables can be joined, as indicated by the relations, to generate T' is not sufficient to understand and more importantly to reproduce the columns *grad_lvl*, *district* and *budget* in T'. The following information is needed to explain the changes.

1) T'*.district* column contains the abbreviations of the values in *district.name*.
2) T'*.budget* column is the result of dividing the values of *budget* column by the ones of *#school* column from *school_national_center* table.
3) T'*.grade_lvl* column is generated by bucketing each value in *T.grade_lvl* column.

As illustrated in the above example, a variety of possible transformations lead to an exponential number of changes to a data repository. These changes often involve different data types, diverse functions as well as multiple columns/tables. Transformation discovery methods [1]–[4] are often used to explain changes among different table versions in a data repository. Typically, they follow programming-by-example (PBE) [5] paradigm to synthesize a program that transforms a given input to a given output. Some of the systems [1] model supported transformations in a search space and design efficient search algorithms to find appropriate ones. Nevertheless, they often suffer from limited flexibility to handle complex and diverse data types and transformations. And they do not adapt well to evolving data distributions or new domains. Other approaches [4] use deep reinforcement learning to synthesize a program. However, high quality labeled data is needed for training. This entails a significant amount of effort from subject matter experts (SMEs), which is very expensive with respect to both cost and manual labor.

The DATALORE solution. We propose DATALORE, a novel system that automatically generates data transformations

*Work conducted during an internship at Amazon.

+Huzefa Rangwala is on LOA as a Professor of Computer Science at George Mason University . This paper describes work performed at Amazon.

[1]https://auctus.vida-nyu.org/

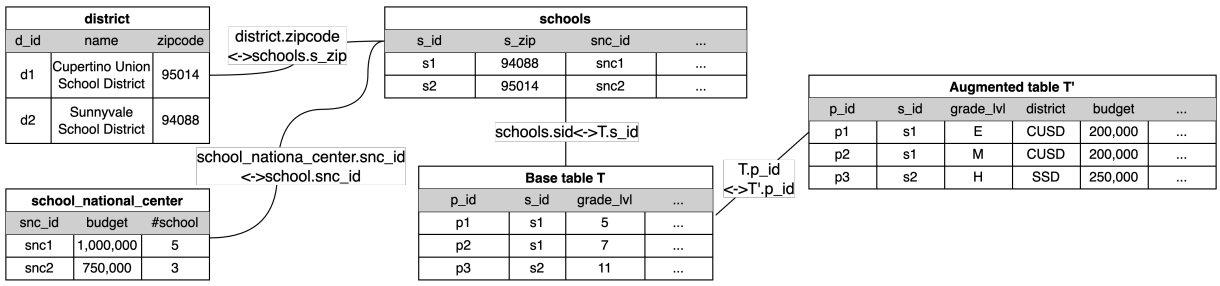


Fig. 1: A motivating example based on National Center for Education Statistics (NCES) data.

among tables in a shared data repository, which enables data scientists and ML engineers to collaboratively utilize the data repository for building ML pipelines. The key differences between DATALORE and the above discussed solutions are as follows. First, DATALORE takes a generative approach to tackle the missing data transformation problem. Namely, DATALORE leverages Large Language Models (LLMs), trained on billions of lines of code, as a synthesis tool for data transformation [6], reducing the semantic ambiguity and the need for manual work. Second, we also leverage data discovery methods to identify candidate tables that are potentially related to a given base table T. This not only makes our LLM-based system more effective, but also supports a sequence of data transformations. Third, DATALORE follows Minimum Description Length principle to minimize the number of related tables involved in deriving the augmented table T'. This avoids the expensive search space exploration, improving the efficiency of DATALORE.

DATALORE use cases. Cloud computing platforms, such as Amazon Web Services, Microsoft Azure, and Google Cloud, offer a broad portfolio for customers that can be benefited from DATALORE to unlock the data's potential, including data governance, data integration and machine learning services. Service users often spend a significant amount of time on discovering tables or datasets that are relevant to their search queries and then manually verifying the correctness and usefulness of the results. DATALORE improves the user experience in the following three aspects.

First, DATALORE's related table discovery can enhance search results by retrieving more relevant tables including semantic and transformation-based ones and categorize them into separate groups. DATALORE can be used to discover datasets that are derived from the ones that they already possess and such information will be indexed as part of a data catalog in an offline process. This essentially alleviates the limitation of statistical-based search methods by enriching the data catalog with additional information on related tables in a data repository. Second, if a user is interested in any related table, DATALORE's LLM-based data transformation generation can greatly improve the explainability of the return results by showing the possible transformation between two or more tables. The user can use the generated data transformation to bootstrap ETL pipelines, significant reducing the burden on the user to write their own code. Third, a user often needs to reproduce and validate every stage of a machine learning workflow to reduce the risk of errors. DATALORE's table selection refinement recovers data transformations among a minimal number of related tables to ensure the reproducibility of the user's dataset and to avoid inconsistencies in the ML workflow.

II. RELATED WORK

Data discovery systems [7]–[9] are often equipped with joinable [10], [11] and unionable [12], [13] table search capabilities. Recently semantic join and union discovery techniques are introduced to capture semantic relationships between tables and columns. However, they are still not sufficient to identify the transformation-based column relatedness.

To synthesize data transformation, Foofah [1] follows the programming-by-example (PBE) paradigm to create a search space using operators (e.g., drop and split) and to search the space using A* heuristic. AutoJoin [2] and Auto-Pipeline [4] extends the PBE paradigm to allow the output provided by the user not necessarily being aligned with the input table and requiring table reshaping operations (e.g., group by). Recently, Explain-Da-V [3] is introduced to explain changes between two given dataset versions. It handles a variety of data transformations involving multiple data types. DATALORE is different from these methods: (1) DATALORE does not restrict transformations between two tables, rather supporting a sequence of data transformations involving multiple related tables; (2) DATALORE leverages the power of LLMs to support more data transformation functions; and (3) DATALORE is equipped with the optimization techniques around LLMs and a polynomial-time complexity greedy algorithm in table selection refinement to be more effective and efficient.

A similar line of research revolves around lineage tracing. SMOKE [14] provides fast lineage capture and lineage query processing by integrating the lineage capture logic into physical database operators and storing lineage in efficient representations. LIMA [15] provides an efficient multi-level lineage tracing and reuse, as well as lineage deduplication for loops and functions for ML systems. Vizier [16] is a multi-modal data science platform built on top of a single incremental workflow platform with built-in support for versioning, provenance, error & tracking, and data cleaning. However, they do not focus on synthesizing and explaining the missing data transformations (backward tracing) like DATALORE.

Large language models (LLMs) or foundation models have been leveraged in data cleaning and integration tasks [17]. The evaluations show that LLMs (e.g., text-davinci-002) generalize and achieve SoTA performance without being trained for these data tasks. In this work, we investigate the applicability of LLMs to a more challenging task, data transformation. Our findings show that LLMs can effectively alleviate the heavy-lifting in designing comprehensive techniques to resolve and explain changes between dataset versions.

III. Preliminaries

A data repository is composed of a set of tables $\mathcal{T}$ and each table $T \in \mathcal{T}$ consists of a set of rows (i.e., tuples) $T_R = \{r_1, ...r_m\}$ and columns (i.e., attributes) $T_C = \{c_1,...,c_n\}$. Similar to [3], we define each row as $r_i = \langle r_{i0}, ...r_{in} \rangle$ such that r_{i0} is the row identifier and r_{ij} ($j \neq 0$) is a value assigned to the column c_j in the tuple r_i.

Problem statement. Given a data repository $\mathcal{T}$, a base table $T \in \mathcal{T}$ and an augmented table $T' \in \mathcal{T}$, our goal is to find a subset of tables $\mathcal{T}_{rel} \subset \mathcal{T}$ and a set of data transformation functions $\mathcal{F}$ such that $T' = \mathcal{F}(T, \mathcal{T}_{rel})$. In this work, we assume that tuples in T and T' with the same identifier (r_{i0} from T and r'_{i0} from T') represent the same entity.

Data transformation functions. In this work, we aim to cover data transformations commonly used in ML data preprocessing pipelines [18]. Hence DataLore primarily focuses on *numeric*, *textual*, *categorical* and mixed data types. In addition, DataLore also supports *one-to-one*, *one-to-many* and *many-to-one* column transformations. Due to space limit, Table I summarizes most supported data transformations.

TABLE I: Data Transformation Functions in DataLore

Data Type	Columns (1:1, 1:m, m:1)	Transformations
Numeric-Numeric	1:1	log, sqrt, exp, polynomial, aggregation reciprocal, round, float_to_int, normalization z-score_normalization, SMOTENC, etc.
	m:1	polynomial_regression, inter_relation, etc.
Textual-Textual	1:1	lower, upper, lemmatization remove_punc, remove_url, str_len substring, extract_by_regex, etc.
	1:m	split
	m:1	merge
Categorical-Categorical	1:1	category_reduce, category_convert one-hot-encoding, etc.
Textual-Numeric	1:1	frequency_encoding pattern_ext, embedding, etc.
Numeric-Categorical	1:1	binning

Prompting code-generating LLMs. LLMs are "conditional model" where the model output is conditioned on the input. Prompting engineering, found to be particularly useful for larger models, refers to the design of input prompt or instruction to elicit more specific, relevant and accurate responses from these models. LLMs such as GPT-4[2] are trained and tuned on a vast corpus of text from the internet and various sources including code snippets from different programming languages. The exposure to a wide range of code examples allows LLMs to learn the syntax, structure and logic of the programming languages. Similar to other text generation tasks, the code generation in LLMs works by tokenizing the input prompt, processing it through an LLM's context window, and generating a probability distribution over the vocabulary of tokens resulting in a executable code snippet. Hence the quality of the generated code can be heavily influenced by the prompt design.

[2]https://openai.com/research/gpt-4

IV. DataLore System

In this section, we first briefly describe the major components in DataLore, including related table discovery, LLM-based data transformation generation and table selection refinement (Figure 2). Note that all components are offline processing.

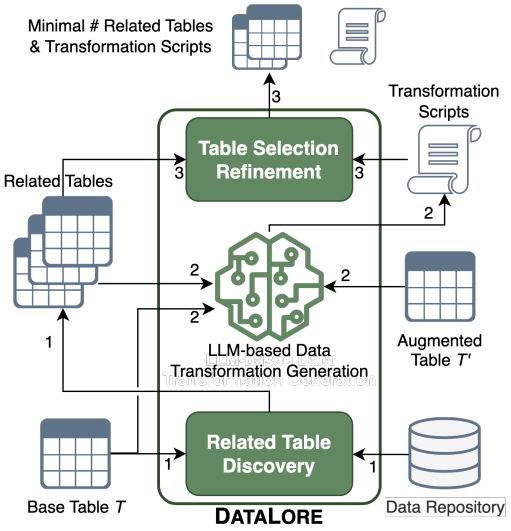


Fig. 2: DataLore overview.

Related table discovery takes as input the given base table T and data repository $\mathcal{T}$. It employs a variety of techniques to find related tables that can join with the base table in different ways, including equi-join, semantic-join and transformation-based join. Compared to equi-join, the semantic join [19], [20] focuses on cells with similar meanings, so as to handle data with misspellings and discrepancy in formats/terminologies.

LLM-based transformation generation first joins all related tables $\mathcal{T}_{rel}$ with the base table to produce a unified table T_u. It then consumes the unified table T_u along with the augmented table T'. To use the LLM more effectively, only a subset of selected column pairs are passed to the LLM to generate a possible transformation F. If the LLM-synthesized F can transform the column from T_u to its correspondence, then F is considered to be valid.

Table selection refinement analyzes all generated data transformations between column pairs in a global manner, in order to identify a subset of related tables and their associated data transformations that are concise and highly-accurate to cover all columns in the augmented table T'. We reduce the problem to a weighted set cover problem and exploit a

greedy algorithm to find the minimal number of related tables involved in the transformation from T to T'.

A. Related Table Discovery

Given a base table T, DATALORE automatically discovers a set of tables $\mathcal{T}_{rel}$ that are related to T from the data repository $\mathcal{T}$. Combining traditional data discovery methods, representation learning-based methods and transformation-based methods, DATALORE effectively and efficiently discovers related tables from the data repository.

Traditional table discovery. Given a base table T, DATALORE first uses traditional equi-join that relies on string equality comparisons to perform joins. Specifically, DATALORE exploits JOISE [10] for finding equi-join related tables, as it has been shown to be effective and efficient in dealing with large tables in data lakes.

Embedding-based table discovery. While equi-join works well in curated data repository, it often falls short in scenarios where data is less curated. Hence DATALORE utilizes TURL [21] to generate the table representations including both schema and data instances, as it has been shown to be effective on various table-related tasks. By computing the cosine similarity between table representations, DATALORE finds tables in the data repository that can be fuzzy joined using similarity predicates. To handle large data repositories, DATALORE further leverages HNSW [22] to index these table representations (i.e., embeddings) for fast cosine similarity lookup. This allows us to efficiently and effectively identify semantically related tables.

Transformation-based table discovery. Finally, we introduce a transformation-based table discovery technique to select related tables that require syntactic transformations to enable equi-join with the base table. Inspired by AutoJoin [2], DATALORE uses local q-grams as an indicator to measure the table relatedness. Intuitively, if we encounter unique 1-to-1 q-gram matches from two tables, they are unlikely coincidence but the result of certain relationships. This method has been shown effective and scalable with web and enterprise datasets [2]. Note that DATALORE does not generate any transformations at this stage.

Using the above described table discovery methods, DATALORE identifies a set of high-quality related tables $\mathcal{T}_{rel}$ potentially involved in generating the augmented table T' with a sequence of data transformations. These tables along with the base table T and the augmented table T' are then fed into DATALORE's LLM-based data transformation generation.

B. LLM-based Data Transformation Generation

DATALORE at its core is powered by an LLM (e.g., GPT-4 or Code Llama[3]) to generate data transformations. Code-generating LLMs are trained using a vast amount of specialized datasets including code repositories, technical forums, coding platforms, documentation of various products and general web data. To generate the desired code snippet, the prompt should be precise and specific about task and/or behavior of the program. For example, one can ask an LLM to "*Write a Python 3 function to convert an input string to lowercase*." As compared to a typical text-to-code generation task, prompting an LLM to generate an appropriate data transformation is challenging since the desired transformation is unknown (expected as a part of the response) and cannot be included or described in the prompt. Also the data transformation space is huge due to the diversity of data sources, types, and formats.

1) Prompt Design: We address the above challenges in DATALORE's prompt design that primarily consists of three elements - instruction, context, and data.

Task instructions. We define the exact input/output format and requirements for data transformation to reduce the hallucinations from the LLM. As demonstrated in Figure 3, the input table consists of one or more columns selected from the unified table T_u and the target list represents a selected column from the augmented table T'. The instruction also specifies the desired Python function signature including the types of the arguments it takes and the type of the value it returns.

Instruction

You are given a table enclosed in triple quotes and serialized in JSON format. You are also given a target list enclosed in quadruple quotes. You should generate a function such that it takes as input the given table and produces the given target list, while the i-th row in the table should correspond to the i-th row in the target list after applying the function on the given table. You can assume that the input table has been deserialized into a pandas DataFrame, and the target list has been deserialized into a pandas Series before being passed to your function. When generating the function, pay extra attention to spaces in text. The result should be only Python code starting with ```python and closed by'''', containing the generated function. In-line comments are allowed. There should NOT BE any other characters in the result.

Fig. 3: Data transformation task instruction.

Context. LLMs have shown remarkable performance in various natural language tasks with their ability to perform few-shot learning [23], where they can adapt to new tasks. Figure 4 shows a demonstration of rounding numbers. In DATALORE, we provide a pool of examples based on the data transformation functions listed in Table I as in-context learning (ICL) prompts, and dynamically select the appropriate example from the pool based on the given input data (Section IV-B2).

In-Context Example

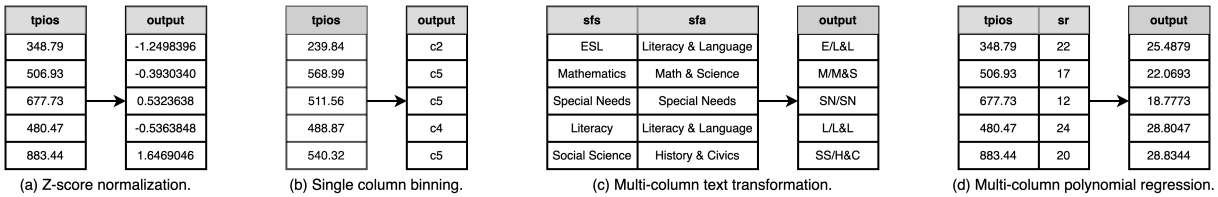
I have two numeric columns. The first column has values [136.5, 314.2, 87.6, 100.0] and the second column has values [137, 314, 88, 100]. One possible data transformation between these two columns is rounding numbers. Below is the corresponding Python code.

```
def round_num(nums):
    rounded_nums = [round(n) for n in nums]
    return rounded_numbers
```

Fig. 4: Data transformation in-context prompt.

Input data. To help the LLM generate the function and to better work with LLM token limits, we sample the cell values from the source columns and provide the corresponding transformed values from a target column. A low sampling rate reduces the input length, therefore reducing the cost of using the LLM. A higher sampling rate provides more demonstrations to the LLM, potentially resulting in a higher accuracy.

[3]https://ai.meta.com/blog/code-llama-large-language-model-coding/



tpios	output
348.79	-1.2498396
506.93	-0.3930340
677.73	0.5323638
480.47	-0.5363848
883.44	1.6469046

(a) Z-score normalization.

tpios	output
239.84	c2
568.99	c5
511.56	c5
488.87	c4
540.32	c5

(b) Single column binning.

sfs	sfa	output
ESL	Literacy & Language	E/L&L
Mathematics	Math & Science	M/M&S
Special Needs	Special Needs	SN/SN
Literacy	Literacy & Language	L/L&L
Social Science	History & Civics	SS/H&C

(c) Multi-column text transformation.

tpios	sr	output
348.79	22	25.4879
506.93	17	22.0693
677.73	12	18.7773
480.47	24	28.8047
883.44	20	28.8344

(d) Multi-column polynomial regression.

Fig. 5: Sample inputs to LLM-based data transformation generation.

Once the transformation function is generated, DATALORE executes it using the complete data to validate its correctness.

In Figures 5-9, we provide 4 examples of prompting GPT-4 to infer and generate python functions that implement the operators described in Table I. In these example, we construct each prompt with the "system message" and 5 demonstrating transformation value pairs as the "user message". The LLM generates correct Python code and the correct transformation logic. Specifically, for single column z-score normalization, multi-column text transformation and multi-column polynomial regression, the generated transformation logic are identical to the ground truth. The binning transformation is partially correct due to a small number of samples used in the prompt.

```
def func(df):
    # subtract the mean and divide by the standard
     deviation
    standardized = (df["tpios"] - df["tpios"].mean()) /
        df["tpios"].std()
    return standardized
```

Fig. 6: Single column z-score normalization.

```
def func(df):
    bins = [0, 299.99, 499.99, 599.99, 699.99,
        float('inf')]
    labels = ['c1', 'c2', 'c3', 'c4', 'c5']
    return pd.cut(df['tpios'], bins=bins, labels=labels)
```

Fig. 7: Single column binning.

```
def func(df):
    return df['sfs'].apply(
        lambda x: ''.join([word[0] for word in
        x.split(' ')])).add('/').add(df['sfa'].apply(
            lambda x: x.replace(' ', '&')[0:3])
    )
```

Fig. 8: Multi-column text transformation.

```
def func(df):
    return (df["tpios"] / 10) + df["sr"]]
```

Fig. 9: Multi-column polynomial regression.

2) Optimizations in DATALORE*:* **Candidate pruning.** As described in Section III, DATALORE joins the base table with related tables to generate the unified table T_u. DATALORE then prompts the LLM to generate various transformation functions from columns in the unified table T_u to the columns in the augmented table T'. Since the source and target columns have to be specified in a prompt, DATALORE needs to generate a set of candidate pairs. An example candidate pair can be $(\{T_u.c_i\}, T'.c_j)$ or $(\{T_u.c_i, T_u.c_j\}, T'.c_k)$. The number of candidate pairs grows exponentially with respect to the number of columns in both T_u and T' due to the *many-to-one* operations.

To use the LLM more effectively, we design a pruning strategy to reduce the number of candidate pairs by filtering out pairs that do not have compatible data or semantic types. For example, a column of float type in T_u is less likely to be transformed into a text column in T'. Therefore these two columns should not form a candidate pair. Furthermore, we also prune the candidate pairs with semantically different source and target columns. To this end, we leverage the column profiling techniques from Sherlock [24] with a similarity threshold for filtering. DATALORE constructs the prompts for the remaining candidate pairs and use them to generate transformation functions in Python through the LLM. DATALORE then validates the correctness of the generated function by executing it over the complete source columns and comparing the results against the target column. Following Explain-da-V [3], we compute a *validity* score = $|F_i(T_u.c_j) \cap T'.c_k|/|T'.c_k|$, indicating the proportion of the data by which can be transformed using the function F_i. Note that the validity score is computed in a tuple-based manner over value-pairs. The rationale is that it is not always possible to find one perfect transformation function to transform all tuples in one column/table to the ones in the other table due to incorrect, corrupted, incorrectly formatted, or incomplete data within a table.

In-context example selection. Given the search space for possible transformations is huge, we design an in-context example selection strategy that picks one appropriate demonstration from the example pool based on the given candidate pair. Here we use the data type of columns in a candidate pair to match with the supported data types of transformation functions listed in Table I. For example, if the data type of two columns in a candidate pair is categorical, DATALORE chooses either category_reduce or category_convert as the in-context example in the prompt, which improves the LLM's performance as confirmed in Section V.

Discussions. To further reduce the cost of using LLMs without sacrificing high validity, we can leverage column profiles (e.g., min, max and $variance$) as contextual information to supplement a low sampling rate. In the binning example shown in Figure 7, the LLM can benefit from knowing such as min, max and $variance$ to better infer the binning strategy. Generating a complex transformation function can be challenging to LLMs. Also as shown in recent studies [25], LLMs' reasoning ability can be greatly improved by decomposing a complex task into sub-tasks. A promising direction is to design operator-specific prompting strategies involving reasoning the intermediate steps to break down the task complexity for LLMs.

C. Table Selection Refinement

The LLM-based data transformation generation may result in many valid transformations, which could be redundant to reproduce the augmented table T', leading to unnecessary costs to the ML data pipeline. Hence table selection refinement of DATALORE aims to choose the minimum number of tables from $\mathcal{T}_{rel}$ such that all columns in T' can be covered by the transformations among these tables and the base table T.

To solve this problem, we reduce the well-known *weighted set cover problem* to our table selection refinement problem. Intuitively, a universe $U = \{T'_C \in T'\}$ represents all columns in the augmented table T'. A family $\mathcal{S}$ of subsets of $\mathcal{T}_{rel}$ consists of all possible subsets of tables in $\mathcal{T}_{rel}$. Lastly, the weight of each set $s \in \mathcal{S}$ is denoted by $w(s)$, in which the weight represents the *validity* of data transformations associated with the tables in s. Therefore, the goal is to find a collection $I = \{s_1,...,s_k\}$ such that (1) all elements in U are covered by I and (2) the sum of the weights $\sum_{s_i \in I} w(s_i)$ of the collection I is minimized. Due to space limit, we omit the proof of problem reduction in this paper.

Note that the weighted set cover problem is NP-complete. In this work, we adopt a polynomial-time complexity greedy algorithm [26], which has been shown that the total weight of the returned collection I is at most [1+ln(n/OPT)]OPT+1.

V. INITIAL EXPERIMENT RESULTS

A. Experimental Setup

Datasets. We use Semantic Data Versioning Benchmark (SDVB) from [3] and Auto-Pipeline Benchmark (APB) [4]. Note that we keep pipelines involving more than one table with a join. We further revise them to introduce additional transformations to ensure that both datasets cover all 40 different types of transformation functions in Table I.

Baseline. Our DATALORE is compared against Explain-Da-V (EDV) [3], a state-of-the-art solution that generates data transformations to explain changes between two given dataset versions. Generating transformations in both DATALORE and EDV have an exponential worst case time complexity, so we apply a 60 second timeout for both methods following the default in EDV. Additionally, we limit the maximum number of columns involved in a multi-column transformation to be 3 in DATALORE.

Metrics. Since both DATALORE and EDV can return transformations that do not have a validity score of 1.0, we report the average validity score of all transformations included in the datasets. In other words, we use the augment table T' as the ground truth and evaluate whether the outputs (related tables and data transformation functions) from DATALORE can successfully reproduce the augmented table T'. Note for a fair comparison, we provide the base table as T and the augmented table as T', but not the intermediate tables involved in a sequence of data transformations. We evaluate the effectiveness of related table discovery method separately as EDV does not support it.

TABLE II: Performance - success rate with 3 ICL examples

Dataset	Semantic Data Versioning Benchmark (SDVB)					
Methods	Overall	N-N (32%)	T-T (13%)	C-C (39%)	T-N (8%)	N-C (8%)
DATALORE	0.870	0.856	0.855	0.863	0.952	0.970
EDV	0.751	0.775	0.684	0.755	0.762	0.636
Dataset	Auto-Pipeline Benchmark (APB)					
Methods	Overall	N-N (40%)	T-T (5%)	C-C (20%)	T-N (15%)	N-C (20%)
DATALORE	0.832	0.821	0.857	0.843	0.857	0.829
EDV	0.634	0.682	0.571	0.607	0.610	0.600

B. Experimental Results

Main results. Table II reports the results between DATALORE and EDV over two benchmarks with a result breakdown to each transformation category associated with their respective percentages (e.g., 32% of test cases in SDVB benchmark belong to *numeric-to-numeric* transformation). Overall, DATALORE consistently outperforms EDV in all categories given its ability to cope with a variety of data types (numeric, textual and categorical). We also observe a larger performance margin over the APB dataset, as only DATALORE support transformations with a join. Looking at different transformation categories, DATALORE does particularly well in *textual-to-textual* and *textual-to-numerical* transformations compared to EDV. DATALORE still have a room for improvement in terms of *numeric-to-numeric* and *numeric-to-categorical* transformations, due to their complexity.

Effectiveness of related table discovery. In this ablation study, we use randomly select (without replacement) 20 tables from each version-set in the SDVB dataset to create a common data repository with 100 tables. Then we randomly select (without replacement) 5 tables again from each version-set as source tables. The goal is to validate, given a source table, if DATALORE's related table discovery can find all 20 related tables from the common data repository. DATALORE achieves 0.85 in *precision*@20, where DATALORE missed few related tables, given limited amount of information indicating potential relatedness to the source table.

Effectiveness of in-context learning examples. We varied the number of examples from 0 to 5 and observed that 3 examples (reported in Table II) provide the best trade-off between performance and cost (# of input tokens to the LLM). Using 3 examples boosts DATALORE's overall performance by approximately 11% compared to the zero-shot setting. However, the performance gain of using 5 examples becomes marginal ($<$2%) against using 3 examples.

VI. CONCLUSION

In this paper, we present DATALORE, a novel system that leverages LLMs to effectively generate a variety of data transformations, without suffering from the time-consuming and error-prone search process to examine all possible transformations. DATALORE also employs data discovery techniques to identify high-quality relevant tables from a data repository and introduces a refinement process to choose the minimum number of tables and the associated data transformation scripts required to reproduce the given augmented table.

REFERENCES

[1] Z. Jin, M. R. Anderson, M. Cafarella, and H. V. Jagadish, "Foofah: Transforming data by example," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, p. 683–698.

[2] E. Zhu, Y. He, and S. Chaudhuri, "Auto-join: Joining tables by leveraging transformations," *Proc. VLDB Endow.*, vol. 10, no. 10, p. 1034–1045, 2017.

[3] R. Shraga and R. J. Miller, "Explaining dataset changes for semantic data versioning with explain-da-v," *Proc. VLDB Endow.*, vol. 16, no. 6, p. 1587–1600, 2023.

[4] J. Yang, Y. He, and S. Chaudhuri, "Auto-pipeline: Synthesizing complex data pipelines by-target using reinforcement learning and search," *Proc. VLDB Endow.*, vol. 14, no. 11, p. 2563–2575, 2021.

[5] S. Gulwani, "Programming by examples - and its applications in data wrangling," in *Dependable Software Systems Engineering*, 2016, vol. 45, pp. 137–158.

[6] R. C. Fernandez, A. J. Elmore, M. J. Franklin, S. Krishnan, and C. Tan, "How large language models will disrupt data management," *Proc. VLDB Endow.*, vol. 16, no. 11, p. 3302–3309, 2023.

[7] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker, "Aurum: A data discovery system," in *34th IEEE International Conference on Data Engineering*, 2018, pp. 1001–1012.

[8] S. Castelo, R. Rampin, A. Santos, A. Bessa, F. Chirigati, and J. Freire, "Auctus: A dataset search engine for data discovery and augmentation," *Proc. VLDB Endow.*, vol. 14, no. 12, p. 2791–2794, 2021.

[9] G. Fan, J. Wang, Y. Li, and R. J. Miller, "Table discovery in data lakes: State-of-the-art and future directions," in *Companion of the 2023 International Conference on Management of Data*, 2023, p. 69–75.

[10] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller, "Josie: Overlap set similarity search for finding joinable tables in data lakes," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, p. 847–864.

[11] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller, "Lsh ensemble: Internet-scale domain search," *Proc. VLDB Endow.*, vol. 9, no. 12, p. 1185–1196, 2016.

[12] A. Khatiwada, G. Fan, R. Shraga, Z. Chen, W. Gatterbauer, R. J. Miller, and M. Riedewald, "Santos: Relationship-based semantic table union search," *Proc. ACM Manag. Data*, vol. 1, no. 1, 2023.

[13] X. Hu, S. Wang, X. Qin, C. Lei, Z. Shen, C. Faloutsos, A. Katsifodimos, G. Karypis, L. Wen, and P. S. Yu, "Automatic table union search with tabular representation learning," in *Findings of the Association for Computational Linguistics: ACL 2023*, Jul. 2023, pp. 3786–3800.

[14] F. Psallidas and E. Wu, "Smoke: Fine-grained lineage at interactive speed," *Proc. VLDB Endow.*, vol. 11, no. 6, pp. 719–732, 2018.

[15] A. Phani, B. Rath, and M. Boehm, "LIMA: fine-grained lineage tracing and reuse in machine learning systems," in *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, 2021, pp. 1426–1439.

[16] O. Kennedy, B. Glavic, J. Freire, and M. Brachmann, "The right tool for the job: Data-centric workflows in vizier," *IEEE Data Eng. Bull.*, vol. 45, no. 3, pp. 129–144, 2022.

[17] A. Narayan, I. Chami, L. Orr, and C. Ré, "Can foundation models wrangle your data?" *Proc. VLDB Endow.*, vol. 16, no. 4, p. 738–746, 2022.

[18] V. Shah, J. Lacanlale, P. Kumar, K. Yang, and A. Kumar, "Towards benchmarking feature type inference for automl platforms," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, p. 1584–1596.

[19] Y. Dong, K. Takeoka, C. Xiao, and M. Oyamada, "Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach," in *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, 2021, pp. 456–467.

[20] Y. Dong, C. Xiao, T. Nozawa, M. Enomoto, and M. Oyamada, "Deepjoin: Joinable table discovery with pre-trained language models," *Proc. VLDB Endow.*, vol. 16, no. 10, p. 2458–2470, 2023.

[21] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu, "TURL: table understanding through representation learning," *Proc. VLDB Endow.*, vol. 14, no. 3, pp. 307–319, 2020.

[22] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *CoRR*, vol. abs/1603.09320, 2016.

[23] T. B. Brown, B. Mann, N. Ryder *et al.*, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020.

[24] M. Hulsebos, K. Hu, M. Bakker, E. Zgraggen, A. Satyanarayan, T. Kraska, Ç. Demiralp, and C. Hidalgo, "Sherlock: A deep learning approach to semantic data type detection," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 1500–1508.

[25] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *NeurIPS*, 2022.

[26] N. E. Young, "Greedy set-cover algorithms," in *Encyclopedia of Algorithms*, 2016, pp. 886–889.