TabulaX: Leveraging Large Language Models for Multi-Class Table Transformations

Arash Dargahi Nobari
dargahi@ualberta.ca
University of Alberta
Edmonton, AB, Canada

Davood Rafiei
drafiei@ualberta.ca
University of Alberta
Edmonton, AB, Canada

ABSTRACT

The integration of tabular data from diverse sources is often hindered by inconsistencies in formatting and representation, posing significant challenges for data analysts and personal digital assistants. Existing methods for automating tabular data transformations are limited in scope, often focusing on specific types of transformations or lacking interpretability. In this paper, we introduce TabulaX, a novel framework that leverages Large Language Models (LLMs) for multi-class column-level tabular transformations. TabulaX first classifies input columns into four transformation types—string-based, numerical, algorithmic, and general—and then applies tailored methods to generate human-interpretable transformation functions, such as numeric formulas or programming code. This approach enhances transparency and allows users to understand and modify the mappings. Through extensive experiments on real-world datasets from various domains, we demonstrate that TabulaX outperforms existing state-of-the-art approaches in terms of accuracy, supports a broader class of transformations, and generates interpretable transformations that can be efficiently applied.

KEYWORDS

Large Language Models, Heterogeneous Table Join, Data Integration, Data Transformation, Data Cleaning and Transformation

PVLDB Reference Format:
Arash Dargahi Nobari and Davood Rafiei. TabulaX: Leveraging Large Language Models for Multi-Class Table Transformations. PVLDB, 18(11): 3826 - 3839, 2025.
doi:10.14778/3749646.3749657

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/arashdn/TabulaX.

1 INTRODUCTION

The rapid growth of publicly available data and increased reliance on third-party sources have underscored the need for efficient methods to combine and transform data from diverse sources. This is crucial for a wide range of users, including data analysts, scientists, and personal digital assistants. However, significant challenges arise due to variations in formatting and presentation across these sources.

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749657

Table 1: Performance of SOTA models on KBWT dataset

	GPT-4o	DTT [8]	GXJoin [34]	AFJ [25]
F1-Score	0.510	0.254	0.083	0.093
Explainable	N	N	Y	N

Even within a single organization, data such as names, addresses, and financial records may exist in multiple inconsistent formats. Target tables may be stored in formats such as spreadsheets, relational databases, web tables, and comma-separated files—common across organizations, the web, and public government datasets [8, 31].

There has been a wide array of studies on automating the transformation of tabular data and ensuring consistency across diverse data sources [1, 8, 18, 34], yet significant challenges remain. Table 1 reports the performance, in terms of F1-score, of four state-of-the-art (SOTA) methods on KBWT [1], a dataset comprising tables extracted from a knowledge base (§5.1). Clearly, all SOTA models struggle and only GXJoin generates explainable transformations, while the others produce either an output string or a match decision, both of which lack interpretability. Figure 1 illustrates examples of source and target tables. In the first table pair, where individuals are identified by name in the source and username in the target, all SOTA models perform well. However, for the second pair, involving weight conversions between pounds and kilograms, and the third pair, linking company names with their CEOs, the models fail to produce meaningful results. While GPT-4o often produces correct outputs, reflected in its relatively high F1-Score in Table 1, it does not yield transformations that are scalable to large datasets.

The problem can be formulated as given a small set of matched rows between source and target tables, we want to identify mappings that transform source values into target values, enabling an equi-join between the two tables. This paper explores two key research questions in this context: (1) Can higher performance be achieved in transforming tables that exhibit a wide range of syntactic and semantic patterns (as illustrated in Figure 1)? (2) Can the transformation process be made explainable? Aligned with prior work in the literature [8, 14, 34, 53], this work primarily focuses on transformations applied to key or join columns. However, the proposed approach has the potential to be extended to include additional columns or more complex settings.

Challenges. There are a few critical challenges that must be addressed to tackle the problem. One major challenge is *the vast multi-class search space* due to the diverse nature of mismatches, as illustrated in Figure 1. These mismatches can range from string transformations, such as converting full names to usernames, to numeric conversions, like pounds to kilograms. In more complex

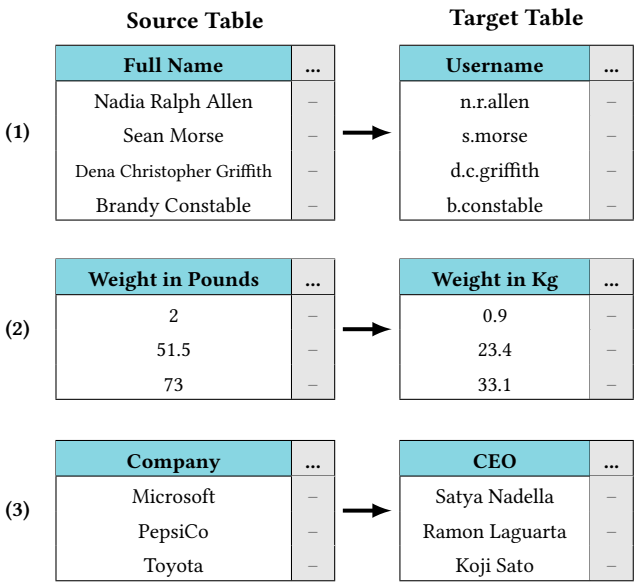

Figure 1: Example input tables with formatting mismatch

cases, transformations may even require external knowledge bases (for example, linking a company name to its CEO). An effective solution must not only handle all of these classes but also navigate the vast search space of potential transformations within each class to identify the most appropriate mappings. Another challenge arises from *the limited coverage of individual transformations*. A single transformation may only be applicable to a subset of the rows, as demonstrated in the first table in Figure 1, where entries containing middle names are transformed differently than those without. A comprehensive framework must identify multiple transformations and develop rules to apply the appropriate transformation based on observed data patterns. Additionally, *interpretability* is a crucial requirement for many application areas, particularly those involving sensitive or high-stakes data. Users need to understand and verify the logic behind each transformation to ensure it meets their requirements. Furthermore, the ability to adjust or refine these mappings fosters trust and transparency in data integration processes.

Limitations of existing approaches. Existing methods for addressing table transformations can be broadly categorized into (1) methods that learn a matching function for a given pair of values or records, and (2) methods that generate mapping rules between source and target tables [7, 8, 14, 34, 53]. The first group includes techniques focused on learning similarity functions [9, 25] or mapping functions [19, 46], as well as approaches for matching entity records that refer to the same real-world objects but differ in format or representation [2, 27, 52]. These approaches typically rely on machine learning models or (semi-)manually defined similarity metrics to rank and select the best candidate for each corresponding value. While effective for addressing formatting mismatches in joining tabular data, they lack the ability to generate explicit mappings to transform the source formatting into the target. This limits their flexibility in modifying transformations, may lead to non-interpretable functions, and restricts their use primarily to fuzzy joins. The second group of approaches focuses on generating mapping rules. Many of these studies are limited to string-based transformations and rely on an exhaustive search of the parameter space to find the appropriate mapping. Some methods are constrained to using a single transformation function per table [14, 15, 40, 53], while others depend entirely on retrieving functions from external sources [18, 20]. Also, some approaches employ pretrained language models, which return mappings that are not interpretable [8].

Our approach. We introduce TabulaX, a novel data integration framework that leverages Large Language Models (LLMs) for multi-class column-level tabular transformations. Similar to existing approaches, TabulaX transforms the entity or join column in the source table to its corresponding representation in the target. However, the mapping functions generated by our framework cover a much broader range of transformations, including textual, numeric, algorithmic, and those requiring external knowledge. A key feature of TabulaX lies in the interpretability and usability of its transformations, which are generated as numeric functions or programming language code, unlike some approaches in the literature that rely on their own domain specific languages [14, 15, 40]. Our experimental evaluation reveals that TabulaX outperforms existing state-of-the-art approaches in terms of accuracy, supports a broader class of transformations, and delivers mappings that can be efficiently applied to large-scale tables and datasets.

Contributions. Our contributions can be summarized as follows: (1) We propose TabulaX, a new framework for Multi-class Tabular Transformations, capable of handling a broad range of transformations—including textual, numeric, algorithmic, and those requiring external knowledge. (2) We introduce a classification mechanism that enables TabulaX to effectively apply tailored transformation strategies to diverse types of data mismatches. (3) We develop methods for generating human-interpretable transformation functions, such as numeric formulas and executable code, which improve transparency and allow users to understand, verify, and modify the mappings. (4) Through extensive experiments on real-world datasets from various domains, we demonstrate that TabulaX outperforms existing state-of-the-art approaches in terms of accuracy and supports a broader class of transformations. (5) We publicly release the TabulaX framework, source code, and datasets to support reproducibility and foster further research[1].

2 RELATED WORKS

We organize related work into three main areas: (1) example-driven table transformations, (2) the use of large language models for tabular data, and (3) techniques for integrating tabular datasets.

2.1 Example-Driven Table Transformations

This line of work, closely aligns with ours and has been an active area of research in recent years [7, 8, 14, 15, 40, 53]. Early approaches such as FlashFill [14] and BlinkFill [40] targeted spreadsheet data using substring-based transformations derived from user examples. Auto-join [53] improves robustness by employing predefined string transformation units and a recursive backtracking algorithm to partition and transform noisy input. CST [7] builds on this by

[1] https://github.com/arashdn/TabulaX

extracting common text fragments as transformation skeletons, applying row-level transformations, and ranking them by coverage. GXJoin [34] generalizes these transformation units, improving both coverage and noise tolerance.

While CST and GXJoin improve efficiency and noise handling, they remain limited to a narrow set of predefined substring-based operations and require long textual overlaps. To address these constraints, DTT [8] introduces a fine-tuned language model for example-driven transformations. This approach eliminates the need for predefined functions or pruning-based searches but relies on synthetic training data and produces transformations in a non-interpretable latent space. Our work addresses these problems by introducing a novel class-aware LLM-based framework capable of generating interpretable, human-readable transformation functions, bridging expressiveness, generalization, and transparency.

Some structural transformation methods, such as Foofah [22] and Explain-Da-V [39], focus on reshaping entire tables. While they target cell- and grid-level alignment for schema matching, our column-level mappings can complement these approaches by handling diverse value-level mismatches in integration tasks.

2.2 Large Language Models for Tabular Data

Early pretrained models such as T5 [36], BERT [11], and GPT-3 [4] have inspired the development of tabular-specific LLMs such as TaBERT [50], TURL [10], and TABBIE [21]. These encoder-based models are better suited for discriminative tasks like entity matching [2, 28] and QA [6, 21, 50], while models like RPT [42] and ByT5 [48] support generative tasks such as data to text [23, 35, 43]. Tabular LMs typically have less than 500M parameters and rely on fine-tuning for task adaptation. DTT [8], for instance, is built on fine-tuned ByT5 [48] for table joinability.

Recent advances in LLMs such as ChatGPT[2], Gemini[3], and Llama 3 [13] have enabled prompt-based methods that require minimal or no fine-tuning. Techniques such as self-consistency [45] and chain-of-thought prompting [47] improve LLM reasoning and performance by guiding generation through structured prompts. Some studies apply LLMs to tabular tasks using supervised fine-tuning [26, 51] and prompt-based approaches [41], emphasizing the importance of table serialization [8, 32, 41]. These approaches are complementary to ours, improving how LLMs process and reason about tabular data.

2.3 Tabular Data Integration

Research focused on linking and integrating structured datasets is closely related to our goals. A comprehensive overview of foundational methods is provided by Doan et al. [12], while more recent developments, particularly in the context of data lakes and open data integration, are reviewed by Miller et al. [30] and Khatiwada et al. [24]. Our work also connects with efforts to detect linkage points across datasets [16]. Notably, when schema elements (e.g., `dbpedia:stockSymbol` and `dbpedia:stockTicker`) are interpreted as cell values, their alignment can be viewed as a form of cell-level transformation. This perspective supports the idea that schema-level and value-level integration can be unified under a single transformation framework, which our method aims to achieve through class-aware, interpretable mappings.

[2]https://chatgpt.com
[3]https://gemini.google.com

3 MULTI-CLASS TRANSFORMATION DISCOVERY

In this section, we provide a detailed formulation of the problem, a discussion of our assumptions and observations, and a classification of transformations based on their functions.

Our aim is to transform the values in a source table column into their corresponding values in a target column, guided by a small set of provided examples. We assume that both the source values to be transformed and user-provided examples are available, which helps narrow the scope to concentrate specifically on data transformation tasks [7, 8]. When such examples are not provided by users, methods such as unequal joining [19, 25, 46] or token-based example generation [7, 53] can be employed to produce a set of examples, with the caveat that these automatically generated examples may include noise or invalid matches.

Let $S = \{s_1, s_2, \ldots\}$ denote a set of values in the source table, and $T = \{t_1, t_2, \ldots\}$ represent the corresponding values in the target, which may be partially or completely unavailable. For a small subset $S' \subset S$, let $E = \{(s_i, t_j) | s_i \in S'\}$ denote a set of examples where target values are provided to guide the transformation process. We want to find a transformation function f such that:

$$f : S \rightarrow T | \forall s_j \in S' ((s_j, f(s_j) \in E. \quad (1)$$

Ideally, the function learned from S' should be applicable to any value in S, rather than being restricted only to S'. The function is expected to be human-interpretable, meaning that its output should be generated in a manner supported by transparent patterns. Additionally, it should be implementable using commonly utilized programming languages in organizational settings. Moreover, given that many data lakes and tables sourced from third parties often lack schema information, we operate under the assumption that no metadata or schema is available for the framework to benefit. Accordingly, transformations will be extracted only using the values in the individual cells.

As an example, consider the middle row in Figure 1, where `S = {2, 51.5, 73}` and `T = {0.9, 23.4, 33.1}`. We provide the set of guiding examples as `E = {(2, 0.9), (51.5, 23.4)}`. The aim is to find a transformation function that maps `2` to `0.9`, and `51.5` to `23.4`, thereby identifying a pattern. For instance, the function $f(x) = \text{round}(0.453 \times x)$ can transform any source value in E into its corresponding target and is generalizable to all inputs in S.

As shown in Figure 1, different input tables require distinct classes of transformations. For example, the top table can be transformed using a sequence of string manipulation functions, such as substring and split, while the middle table relies on a numerical transformation (specifically, a linear function). Transformation of the last table is not possible without external knowledge that is not present in either the source or target.

Most existing approaches in the literature limit their scope when addressing this problem. Some methods focus only on string-based transformations [8, 14, 53], others only handle entity matching [28,

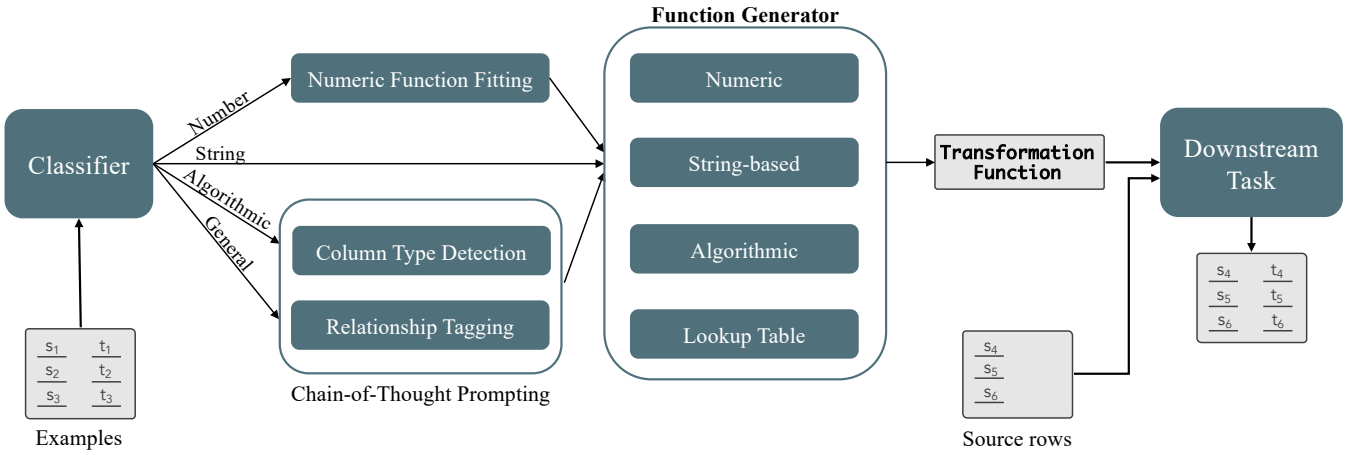

Figure 2: The architecture of our framework

52], and some retrieve mapping functions from large code repositories [18, 20]. One may argue that recent commercially available LLMs, such as GPT and Gemini, could address these challenges. To evaluate the performance of LLMs in this context, we conducted an experiment using the GPT model[4] on our benchmark consisting of diverse real-world tables. However, the out-of-the-box model struggled to recognize many textual patterns, and frequently produced hallucinated or irrelevant outputs in cases requiring external knowledge. Moreover, LLMs are not expected to handle numerical data well [5], and our experiments also confirmed that the model was unable to perform numerical transformations effectively.

To address this issue, we propose a framework that first classifies the input mappings. Based on the identified class, appropriate actions and methods are then selected to facilitate the transformation of the tables. We define four distinct classes of input transformations:

- **String-based**: This class includes input columns that can be transformed using a series of string manipulation functions commonly available in programming frameworks, such as substring, concatenation, split, case conversion (upper/lowercase), etc. An example of this class is shown in the top row in Figure 1.
- **Numerical**: This class encompasses input tables where both source and target values are rational numbers and exhibit an underlying mathematical relationship. An example is the conversion from pounds to kilograms illustrated in Figure 1. It is important to note that nominal numbers, such as postal codes or ISBNs, do not fall in this category.
- **Algorithmic**: This class includes input tables where an algorithm exists to transform source values into target values, but the transformation goes beyond numerical functions or sequences of string operations. Examples include transformation such as Gregorian to Hijri dates, characters to ASCII codes, or binary to hexadecimal numbers.
- **General**: This class includes any table transformations that require external knowledge sources or do not fit into the previous three categories. Examples include mapping a company name to its CEO, converting area codes to countries, or translating English words to French.

In the following sections, we will discuss the classification process and the specific actions required for each class of input tables.

4 APPROACH

Figure 2 depicts the architecture of TabulaX, our proposed framework for tabular data transformation and join. The first component is a classifier that assigns a transformation class from the aforementioned set of classes. Once classified, the tables are routed to the corresponding mapping modules, which generate the appropriate transformation functions. These transformations are then utilized by the downstream task component to produce end-to-end results. While this study focuses on unequal joins as the downstream task, the generated transformations can also support other applications, such as detecting anomalies or outliers by identifying deviations from the defined mappings, imputing missing values, etc. The remainder of this section details each component of the framework.

4.1 Input Data and Serialization

Multiple components in our approach interact with LLMs, and most LLMs accept input only as a sequence of tokens, hence a critical step is serializing the input tables into a linear sequence of text. This sequence is then tokenized and processed by the LLM. Most LLMs are trained on vast, diverse datasets from web pages and are familiar with common notations used to represent sets and pairs in mathematics and computer science. Leveraging this, we serialize the example pairs in a way that closely aligns with these conventions.

Specifically, each example is serialized as `("s" -> "t")`, where `s` and `t` represent the source and target values, respectively. To ensure generality and with the assumption that schema and metadata may not be available, all values are enclosed in quotation marks, regardless of their cell or column data types. We use the `->` token to indicate the direction from source to target, instead of a comma,

[4]We used the `gpt-4o-2024-05-13` model from the OpenAI API for this experiment

which is more commonly used for separating elements in pairs or tuples. This decision is supported by our observations that when a comma is used to separate tuple elements, LLMs, particularly GPT-4o, may fail to recognize the directionality from source to target, leading to erroneous predictions. Finally, the serialized examples are concatenated using commas to form the complete serialized list.

For instance, consider the first two rows from the bottom row in Figure 1 as the set of provided examples. This set will be serialized as:
`("Microsoft" -> "Satya Nadella"),( "PepsiCo" -> "Ramon Laguarta")`.
This serialization will be utilized throughout the framework whenever an LLM needs to process the example set. Since many LLMs impose a limit on input length, if the size of the example set exceeds the model's context window, a sampling step is applied during serialization to reduce the input length.

4.2 Classifier

Our experiments with various LLMs indicate that even large, widely-used commercial models such as GPT-4o struggle to effectively transform numeric values, recognize all textual patterns, and generate lookup tables without prompts that are tailored for each class. To address this, the first component of our framework is a classifier that determines the subsequent components to which the input tables will be directed. To achieve this, we use a general-purpose LLM. LLMs, trained on vast and diverse datasets, can handle a wide range of tasks by utilizing one or a few training examples (i.e., shots) [4, 38] in their prompts, known as in-context learning. Inspired by this idea, we developed a prompt template for classifying input types. The structure of the prompt and the given examples (shots) may significantly affect the quality of the classification, which led us to try 3-5 prompts before selecting one. Our classifier's LLM prompt[5] starts by defining the task, followed by a list of possible classes. Each class is then explained in detail, and representative examples are provided at the end.

Once the input class is identified, we propose a tailored method that best suits the transformation of values in that class. Recognizing that LLMs face challenges with numerical mappings, our framework avoids relying on LLMs for this class. It also uses Chain-of-Thought prompting [47] for advanced algorithmic and general classes to enhance the model's reasoning capabilities. In what follows, we will describe how transformations are generated for each class in more detail.

4.3 Numerical Transformations

When the input table is labeled as numerical, it is passed to the Numerical Function Fitting component, which is responsible for fitting a numerical function f_n (i.e., curve fitting) based on the input examples. The framework attempts to fit curves using the following types of functions:

- Linear: $f(x) = ax + b$,
- Polynomial: $f(x) = ax^2 + bx + c$,
- Exponential: $f(x) = a\ exp(bx)$,
- Rational: $f(x) = \frac{ax+b}{x+c}$.

The parameters a, b, and c are estimated during the curve fitting process using the Levenberg–Marquardt algorithm [37]. The framework then calculates the Mean Square Error (MSE) between the predicted and actual target values in the provided examples, and the function with the lowest MSE is selected as the mapping function, denoted as $f_n(x)$. The list of functions is not limited to those mentioned above; any function supported by the curve fitting algorithm can be used.

Once the best-fitting function $f_n(x)$ is selected, it is passed to the Numerical Function Generator component, which generates the corresponding transformation as a function in a programming language—specifically Python, in our setup.

4.4 String-Based Transformations

Table transformations using string primitives and regular expressions have been widely studied in the literature [7, 14, 34, 40, 53], often using a Programming-By-Example (PBE) approach. While any of these methods could be incorporated as our String-Based Function Generator, we propose a novel approach by leveraging LLMs, which outperform state-of-the-art techniques. Recent studies suggest that LLMs perform well in code generation [29, 33, 44], and our approach also utilizes LLMs to generate transformation functions for string-based inputs.

Specifically, the model is prompted to create a Python function that takes a string as input and reformats it to produce the expected output. In this case, the input examples are not presented using the set-like notation from Section 4.1. Instead, we format them as test cases: `Input:` s_i`, Expected output:` t_i, where s_i and t_i represent arbitrary source and target samples, respectively. This format is common on websites and forums, which are primary sources for common LLMs training data. As a result, LLMs better understand the task using this format, as demonstrated in our experiments, leading to improved code generation compared to the set-like notation.

Additionally, the model is instructed not to use any external library calls but is free to utilize any functions available in the standard Python library. This approach covers a broader range of string-based transformations compared to methods that rely on a limited set of predefined functions [7, 34, 53] or those using substring extraction via regular expressions [14, 15, 40]. Finally, the generated function is verified by a syntax checker before being used as the transformation function.

4.5 Algorithmic Transformations

Algorithmic transformations are more complex compared to numerical and string-based transformations. Our experiments indicate that applying the same approach used for string-based transformations is ineffective here, mainly due to the limitations of LLMs in handling complex reasoning. In many cases, the model fails to identify the correct transformation pattern and may hallucinate and generate code that does not align with the given examples.

Studies suggests that complex reasoning in LLMs can be improved through intermediate reasoning steps [47, 49], known as Chain-of-Thought (CoT) reasoning. Leveraging this approach, we break down the task of generating algorithmic transformations into

[5]The prompt template, along with all other prompting techniques and templates used in this study, are available in our publicly-available code repository.

two simpler steps: (1) *Relationship Tagging*, which identifies the conversion type or the relationship between the source and target, and (2) *Algorithmic Function Generator*, which produces a Python function to execute the transformation based on the identified relationship.

For tables requiring algorithmic transformations, the input is first processed by the relationship extractor, where the LLM is prompted to identify the relationship between the source and target columns. The relationship is formatted as `[type of source column] to [type of target column]`. For example, given the input `("2024/09/05" -> "1403/06/16"), ("1886/06/27" -> "1265/04/06")`, the model is expected to return `Gregorian date to Jalali (Solar Hijri) date`. This transformation is included in the training examples provided for in-context learning, alongside others, such as email to domain transformation. The detected relationship and input tables are then passed to the algorithmic function generator component, which is structured similarly to the string-based function generator. In this step, the model generates a Python function based on the identified relationship, using the same format and verification as the string-based component.

4.6 General Transformations

General transformations are the most complex task, as they often require external knowledge not present in the provided examples. We adopt a similar approach to the one used for algorithmic transformations, supported by Chain-of-Thought prompting, and break the problem into two subtasks. The first step is similar to the relationship tagging component in the algorithmic transformer, where the types of source and target columns are detected. The second step generates a transformation by leveraging a lookup table (i.e., a structured mapping between source and target values) as a bridge. There are multiple ways to obtain this table. One approach is to retrieve it from external repositories (e.g., structured databases, web tables, or open knowledge graphs), which requires extensive indexing and specialized retrieval algorithms [18–20]. In this work, however, we leverage LLMs to dynamically construct the lookup table, allowing for greater flexibility and adaptability.

When input tables are labeled for general transformation, they are passed to the *Column Type Detection* component, where the LLM is prompted to identify the type of the source and target columns, formatted as `[type of source column] to [type of target column]`. While the task shares some similarities with algorithmic transformer, the two training examples used in this context are selected to better match the general transformation scenario. For instance, given `Data: ("AGC" -> "United States"), ("YYZ" -> "Canada")`, the expected relationship is `Airport Code to Country` and for `Data: ("gallon" -> "liter"), ("inch" -> "centimeter")`, the relationship will be `Imperial Units to Metric Unit`.

As demonstrated in these examples, developing a deterministic algorithm for such cases is infeasible, as the transformations rely on external knowledge. Consequently, we leverage the LLM's learned knowledge to generate a lookup function that directly prompts the model to predict the target value for a given source value, based on the extracted column types and input examples.

Limitations of general transformations. While our task breakdown and other steps significantly improve the model's performance compared to state-of-the-art baselines, there are certain limitations. First, the external knowledge used by the model is inherently constrained by its training data. Despite the extensive training of modern LLMs on large datasets from web pages and public knowledge bases, they may still lack domain-specific knowledge, particularly in private sectors where similar data is not publicly available. Additionally, the transformation function in this category primarily relies on LLM predictions, which are not human-interpretable. Unlike other transformation types where the mapping function is generated based on a single prompt using a small set of example rows—independent of the source column size—general transformations may require an LLM call for each row, particularly if no batching strategy is applied. While this may raise scalability concerns, it is worth noting that these LLM calls are independent and can be executed in parallel to deal with performance bottlenecks.

Moreover, LLMs are susceptible to hallucinations, and unlike the numeric, string, or algorithmic transformers, where the output is an explicit algorithm that can be automatically or manually verified, hallucinations in this case may occur and remain undetected. In sensitive environments, one solution is to implement guardrails that verify the model's output using domain knowledge or compare it against available target values. This component could also be replaced with a retrieval-based approach to access relevant information from public knowledge bases or private data sources. For example, the system could be integrated with a table retrieval method that retrieves a bridge or cross table between the source and target [1, 19, 46], or by employing entity matching techniques [2, 27, 52]. These areas remain open for future works.

4.7 Example Walkthrough: Generating a Transformation

Figure 3 illustrates an end-to-end example of the framework in action. In this scenario, the source values represent Unicode code points of certain characters, while the target values correspond to their ASCII decimal representations. Four examples are provided to guide the transformation generation process. The examples are first passed to the classifier, which identifies the input type as an algorithmic transformation. Based on this classification, the framework invokes the algorithmic transformation process, beginning with the relationship tagging component. The model extracts the relationship as "Unicode code point to ASCII decimal value," which is then passed, along with the input examples, to the function generator component. The output of this step is a transformation expressed as human-interpretable Python code. This code is subsequently executed on the source rows to generate the predicted target values.

4.8 Integration with Downstream Tasks

The transformation functions generated by TabulaX can be directly applied to data analysis or used to support downstream tasks. Tabular transformations have been used for a variety of tasks, including auto-completion and auto-filling of spreadsheets [14, 40], predicting missing values, error correction [17], and joining tables [34, 53].

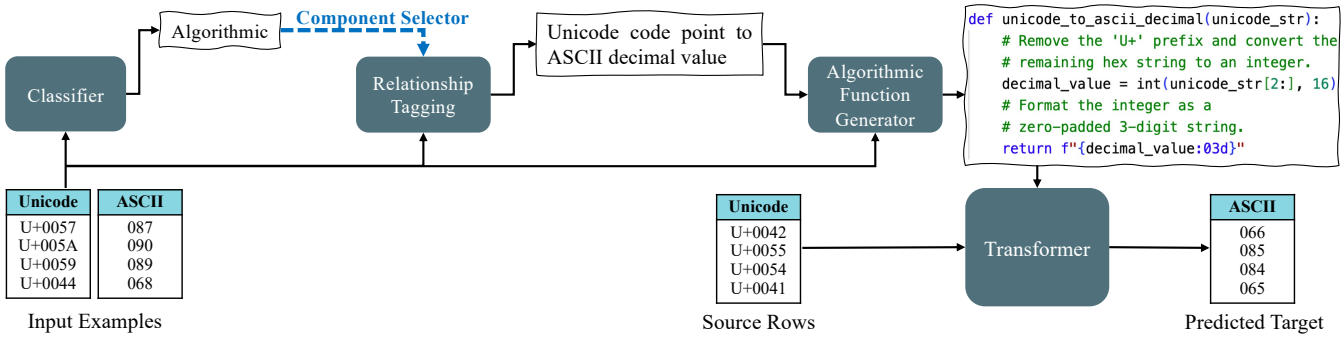

Figure 3: The steps of processing an arbitrary input table

In this section, we focus specifically on the task of joining heterogeneous tables, while the generalization of this framework for other downstream applications are left for future works.

Consider a scenario where a source table S and a target table T must be joined on semantically equivalent columns, guided by a set of examples mappings E. While these columns may differ in format, they represent the same real-world entities or concepts. TabulaX leverages E to learn a transformation function, which is applied to each entry $s_i \in S$ to produce a transformed value $\hat{t}_i$. This results in a candidate set $\hat{T}$, aligned with the format of T. Treating $\hat{T}$ as an intermediate column, an equality join with T can be used to complete the operation.

However, strict equality joins are sensitive to minor discrepancies between predicted and actual values. Unlike tasks such as auto-completion, where exact matches are essential, join operations can often tolerate minor mismatches. When a one-to-one relationship is expected (e.g., in primary-foreign key joins), the join can instead be performed by selecting the closest match in T for each $\hat{t}_i$, using string similarity metrics such as edit distance. Formally, the best match o_i for a source value s_i is given by:

$$o_i = \underset{t_j \in T}{\operatorname{argmin}}\ edit\_dist(\hat{t}_i, t_j). \tag{2}$$

This method can be generalized to handle one-to-many or many-to-many joins by defining upper and lower bounds on allowable edit distances. Edit-distance-based matching offers a transparent and targeted solution for handling small inconsistencies, though it is not intended as a full fuzzy matching algorithm.

For numeric transformations, edit distance is not appropriate. Instead, we use absolute numerical differences to measure proximity between predicted and actual values. This maintains consistency with the goal of interpretable and context-sensitive alignment while accounting for datatype-specific variation. Edit-distance-based matching is a simple yet effective technique, specifically used to address minor discrepancies between the predicted and target values, while maintaining transparency. It is not intended to perform a comprehensive fuzzy join but rather serves as a targeted approach to enhance matching accuracy in the presence of slight inconsistencies. Unlike the other classes, however, edit-distance-based matching is not applicable on numeric transformations; instead, we employ absolute numerical distance to assess the closeness of the transformed values to their targets.

5 EXPERIMENTS AND ANALYSIS

This section presents a comprehensive evaluation of our framework, examining its performance across different configurations and benchmarking it against established baselines.

5.1 Benchmark Datasets

To evaluate the performance of our proposed method and compare it with current state-of-the-art baselines, we employ four real-world datasets encompassing various input classes. Detailed descriptions of each dataset are provided below.

Web Tables Dataset (WT): Initially introduced by Zhu et al. [53] and subsequently utilized as a benchmark by Nobari et al. [7, 8], this dataset consists of 31 pairs of tables spanning 17 distinct topics. Each table averages 92.13 rows with an input length of approximately 31 characters. The tables were extracted from Google Fusion Tables by selecting those that appeared in the same query results but had different formats. This benchmark involves string-based transformations and includes natural noise and inconsistencies. Not all entities can be transformed using traditional primitive string-based transformations, making this dataset relatively challenging string-based input [7, 8].

Spreadsheet Dataset (SS): This dataset comprises 108 pairs of tables sourced from the Microsoft Excel product team and user help forums, focusing on users' data cleaning issues. The tables represent spreadsheet pages that convey the same information in different formats. It includes public benchmarks from FlashFill [14] and BlinkFill [40], and was featured in the 2016 Syntax-Guided Synthesis Competition (SyGuS-Comp) [3]. On average, each table contains 34.43 rows and has an input source length of 19 characters. Compared to the WT dataset, while SS dataset is also string-based, it features considerably less noise and inconsistency.

Table Transformation (TT): Introduced by He et al [18], this dataset was developed to support the task of retrieving data transformations from public code bases. It contains 230 pairs of tables, each averaging 8 rows, and covers a wide range of transformation types, including 102 algorithmic, 57 numeric, 68 string-based, and 3 general transformations. Compared to the WT and SS datasets, TT features more complex transformation logic, often involving numerical computations and algorithmic operations in addition to advanced string manipulations.

Knowledge Base Web Tables (KBWT): Introduced by Abedjan et al. [1], this dataset primarily consists of tables extracted from a Knowledge Base (KB), requiring semantic transformations and additional KB information. For our evaluation, we selected single-column tasks from this dataset, totaling 81 pairs of tables. Each table averages 113 rows with an input source length of 13 characters. This dataset differs notably from the WT and SS benchmarks, which focus mainly on textual transformations. Among the tables in the KBWT benchmark, three are numeric, four are algorithmic, and the rest are general transformations.

5.2 Experimental Setup and Evaluation Metrics

Our framework, TabulaX, follows an example-driven approach, taking n input examples to guide the transformation process. TabulaX is designed to allow any general-purpose LLM to serve as the inference model within the framework. We experiment with various numbers of input examples and multiple LLMs to provide a detailed analysis of our framework's performance. Where the value of n is not explicitly mentioned, it is set to 5 for the WT, SS, and KBWT datasets. For the TT dataset, n is set to 3 due to the relatively small number of rows in most of its tables. The default LLM used in the framework is OpenAI GPT-4o model, specifically, `gpt-4o-2024-05-13` version.

For each transformation class, we tested 3–5 prompt variants and selected the best-performing one for our experiments. In the string-based and algorithmic transformations, where the output domain is relatively constrained and the final task is code generation, we observed that changes in prompt had only a subtle effect on performance. In contrast, general transformations involve more diverse domains, and prompt variations had a somewhat greater impact. For this class, we selected the most general prompt that yielded the best overall results. Overall, performance remained stable across prompts that followed the guidelines described in Section 4, especially for larger models like GPT-4o.

The focus of this study is the heterogeneous join as the downstream task, and as described in Section 4.8, the prediction is obtained by calculating the edit distance between the predicted and target values. While this is the default approach in our framework, we also conduct some experiments using exact matching to showcase our model's generalizability to other downstream tasks where target values may be unavailable or edit distance may not be preferred.

Join performance is evaluated based on precision, recall, and F1-score, where precision measures the fraction of correct predictions that join with the target, recall represents the fraction of source rows that are correctly mapped, and F1-score is calculated by taking the harmonic mean of precision and recall. It is important to recognize that not all source rows may be mapped to a target value for reasons such as generating an invalid function, returning empty output, or runtime exception. In addition to those metrics, we also report the Average Edit Distance (AED) and Average Normalized Edit Distance (ANED), which indicates how much a prediction deviates from its target. ANED is normalized by the target length, facilitating comparisons between datasets and tables with different lengths. Each dataset's reported metrics are averaged across all tables within that dataset.

5.3 Performance of classification

Table 2 presents the performance of our classification module using three different LLMs: GPT-4o, GPT-4o-mini, and LLaMA 3.1 8B. In this table, support refers to the number of tables in each transformation class. To measure performance, we use common classification metrics: precision (P), recall (R), and F1-score (F). These metrics are calculated across four transformation classes: String, Numbers, Algorithmic, and General. The macro average is calculated as the simple average of the metrics across all classes, regardless of their support, while the micro average takes into account the weight of each class by considering the total number of instances. The goal is to assess how effectively each model can categorize input tables into the correct transformation class, which is crucial for directing them to the appropriate mapping components in our framework.

While the benchmark datasets provide source and target table pairs for evaluation, they do not include transformation type labels required for the classification task. To support this component, the transformation classes for all benchmark instances were manually annotated by the authors. During the annotation process, a conflict rate of approximately 8% was observed, primarily in distinguishing between algorithmic and string-based transformations. These disagreements were resolved through discussion and consensus to ensure consistency and accuracy in the labels.

As expected, GPT-4o outperforms the other models across all classes, achieving the highest macro and micro average scores. This superior performance can be attributed to its larger size and its training focused more on understanding complex tasks. All models achieve perfect recall for the numerical class, indicating that they reliably detect numerical transformations. However, GPT-4o still outperforms the others in precision, suggesting fewer misclassifications, such as incorrectly labeling nominal numbers (like zip codes) as numerical transformations or failing to recognize when only one side of the transformation involves numerical data. As a result, they achieve high recall, while their precision is not perfect.

Another important observation is the tendency of the models to misclassify certain String transformations as General or Algorithmic. This misclassification often occurs in cases requiring extensive edit operations or the addition of literal text. For example, when transforming usernames into email addresses by appending a domain name, these models sometimes label the task as General instead of String. They assume that adding a domain requires external knowledge rather than recognizing it as a fixed literal addition. This suggests that LLMS, especially the smaller ones, may struggle with more complex textual transformations that involve fixed patterns or extensive edits.

Overall, GPT-4o demonstrates sufficient capability for our classification tasks, emphasizing the importance of choosing a model complex enough to prevent misclassifications that could impact the subsequent mapping components in our framework. In certain domains, the transformation classes may already be established or can be determined by domain experts, eliminating the need for this classification component altogether.

5.4 Performance Compared to baseline models

In this section, we evaluate the performance of our framework, TabulaX, on the end-to-end task of heterogeneous or unequal table

Table 2: Classification performance of various LLMs across transformation classes

		GPT-4o			GPT-4o-mini			LLaMA 3.1		
Class	Support	P	R	F	P	R	F	P	R	F
String	207	0.92	0.85	**0.88**	0.89	0.74	0.81	0.72	0.91	0.80
Numbers	60	0.78	1.00	**0.88**	0.67	1.00	0.81	0.65	1.00	0.79
Algorithmic	105	0.84	0.70	**0.76**	0.88	0.56	0.69	0.88	0.20	0.33
General	76	0.78	0.96	**0.86**	0.59	0.92	0.72	0.53	0.49	0.51
Macro avg	448	0.83	0.88	**0.84**	0.76	0.81	0.75	0.69	0.65	0.61
Micro avg	448	0.86	0.85	**0.85**	0.81	0.77	0.76	0.71	0.68	0.64

joins. This task simulates a common scenario where source and target columns reside in different tables that need to be joined, but the values in these columns are formatted differently.

To benchmark our approach, we compare TabulaX against four state-of-the-art baselines: Deep Tabular Transformer (DTT) [8], GXJoin [34], Auto-FuzzyJoin (AFJ) [25], and Explain-Da-V [39]. DTT leverages language models to generate outputs directly from provided examples, focusing on string transformations. GXJoin extends the CST method [7] and exhaustively searches for general textual string-based transformations to facilitate table joinability. AFJ employs similarity functions to identify the most probable rows to join without generating explicit transformation functions. Explain-Da-V performs example-driven structural transformations by exploring a graph of intermediate table states with an A*-based heuristic search and type-aware operators, aiming to reshape source tables so they align with the target structure.

Table 3 summarizes the performance of our framework, TabulaX, and the baselines. As shown, TabulaX consistently achieves higher or comparable F1-scores across all datasets compared to these baselines. On datasets dominated by string-based transformations, such as WT and SS, the performance gap between TabulaX and DTT is relatively small, with DTT being the best-performing baseline in our setup. Specifically, TabulaX achieves an F1-score of 0.983 on the WT dataset, only marginally higher than DTT's 0.950. On the SS dataset, both methods perform almost identically, with F1-scores of 0.952. This similarity is expected since both models handle string transformations effectively, though TabulaX offers the added advantage of generating interpretable transformation functions. In contrast, DTT directly outputs target values without transparency, which can be a limitation in sensitive environments and scenarios requiring interpretability. The performance gap between TabulaX and the baselines increases on datasets involving more complex transformations. On the TT dataset, which includes a mix of string-based, numerical, and algorithmic transformations, TabulaX achieves an F1-score of 0.918, compared to 0.643 for DTT, 0.223 for GXJoin, and 0.219 for Explain-Da-V. The TDE approach [18] was proposed in the same work that introduced the TT dataset and reports a coverage of 72% on it, whereas our method achieves a recall of 92% on this benchmark. On the KBWT dataset, which contains a higher proportion of general transformations requiring external knowledge, TabulaX reaches an F1-score of 0.567, whereas DTT scores only 0.254. The gap highlights TabulaX's versatility and its ability to generalize across diverse data types, a feature not shared by the string-based baselines. Additionally, TabulaX's interpretable transformations are valuable in understanding and verifying the mappings, unlike DTT's black-box approach.

GXJoin, performs an exhaustive search in the string-based transformation space to generate interpretable transformations and is the best-performing state-of-the-art approach generating an interpretable mapping in our experiments. This large search space allows GXJoin to achieve high precision on all datasets, as it applies transformations without relying on edit distance-based matching. However, the recall is noticeably lower, indicating that it fails to transform a significant portion of the data. In contrast, TabulaX maintains a high recall by utilizing a broader set of transformation units and providing a clear mapping between input rows and transformations. Similar to other baselines, GXJoin is only limited to string-based transformations and is significantly outperformed by TabulaX and DTT on TT and KBWT datasets. Even without employing edit distance matching, as indicated in Table 4, TabulaX outperforms GXJoin due to its ability to handle a wider variety of transformations and its use of LLMs to interpret and apply complex patterns. GXJoin's exhaustive search methodology, while thorough, can lead to scalability issues. Moreover, while TabulaX provides a clear mapping between input rows and the applied transformations, GXJoin's exhaustive search does not offer such mapping, and it may apply multiple transformations without a straightforward way to trace back the exact steps for each row.

In terms of runtime, for a table with n rows, TabulaX requires only a single LLM call to generate transformations for string-based tables, two LLM calls (one for relationship tagging and another for transformation generation) for algorithmic tables, and $n + 1$ LLM calls for general tables when lookups rely on LLM calls. Costs can be reduced if bridge tables are accessible from alternative sources. In contrast, DTT, while effective primarily for string-based tables, requires $k \times n$ LLM calls, where k represents the number of self-consistency trials (set to 5 in the experiments). Both TabulaX (in the worst case) and DTT have a linear growth in the LLM calls with respect to input size, and their LLM calls can be executed in parallel if resources allow. On the other hand, GXJoin has quadratic growth in runtime due to its exhaustive search methodology. This approach also has more constraints on parallel execution, making GXJoin less scalable compared to TabulaX for larger datasets.

AFJ relies on similarity functions to identify matching rows and does not generate explicit transformation functions, which limits its applicability in scenarios where understanding the transformation process is essential. Explain-Da-V, while capable of producing interpretable transformations, is primarily optimized for structural

Table 3: Performance compared to the baselines

	TabulaX			DTT			GXJoin			AFJ			Explain-Da-V		
Dataset	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
WT	0.985	0.980	**0.983**	0.951	0.950	0.950	0.953	0.754	0.777	0.935	0.672	0.708	0.161	0.161	0.161
SS	0.955	0.949	**0.952**	0.954	0.952	**0.952**	0.997	0.787	0.810	0.943	0.662	0.691	0.163	0.163	0.163
TT	0.919	0.918	**0.918**	0.644	0.643	0.643	0.983	0.202	0.223	0.591	0.223	0.251	0.219	0.219	0.219
KBWT	0.722	0.532	**0.567**	0.276	0.248	0.254	0.962	0.081	0.083	0.749	0.067	0.093	0.038	0.038	0.038

table reshaping. Its support for value-level transformations is limited, which hinders its performance in our setup. As a result, both methods underperform compared to TabulaX across all datasets, with particularly poor results on more complex benchmarks such as TT and KBWT.

Overall, our results demonstrate that TabulaX not only outperforms baselines on complex datasets but also provides interpretable transformation functions, making it a superior choice for heterogeneous joins across a wide range of tables.

5.5 Impact of Matching Strategies

Table 4 presents a detailed comparison of the framework's performance under two different matching scenarios: edit-distance-based matching (left panel) and exact matching (right panel). The former matching strategy is more suited for applications where a set of target values is available, such as in unequal table joins, where small variations between the generated and expected values can be tolerated. The latter, exact matching, is more appropriate for tasks like missing data imputation, where precise outputs are required. The top panel of the table reflects results when using the LLM-based classification model (GPT-4o in this set of experiments), while the bottom panel provides performance metrics when using the ground truth classification, referred to as the "Golden" classifier. This comparison allows us to assess the impact of the classification on the overall performance of the framework.

An interesting observation is that the model performance on datasets consisting of string-based transformations (Web Tables and Spreadsheet) is slightly better when the classification is done via the LLM-based model compared to the golden classifier. For instance, on the Web Tables dataset using edit-distance matching, the F1-score with LLM-based classification is 0.983, marginally higher than the 0.973 achieved with the golden classifier. This counterintuitive performance can be attributed to the LLM-based classifier occasionally mislabeling some difficult string-based transformations as general transformations. In these challenging cases, treating them as general transformations allows the model to generate the target values directly by leveraging the LLM's generative capabilities, without being constrained to produce a specific transformation function covering all input rows. While this approach can lead to better target predictions and higher recall, it sacrifices interpretability since the general transformations are not explicitly defined. In contrast, the golden classifier strictly labels these cases as string-based transformations, requiring the model to generate explicit transformation functions, which may be more difficult for complex patterns.

Moreover, when exact matching is employed (right panel of the table), the model's precision increases to nearly perfect levels across all datasets and classification methods. This increase is expected, as outputs are only accepted if they exactly match the expected targets, significantly decreasing the possibility of false matches. However, this strict criterion leads to a reduction in recall, as the model misses rows where there are even minor discrepancies between the generated and expected values, such as differences in capitalization, punctuation, or minor formatting variations. In contrast, using edit-distance-based matching (left panel) allows minor differences between the predicted and target values, thereby considerably increasing recall and overall F1-score across all datasets. The decrease in precision with edit-distance matching is minimal compared to the considerable increase in recall, resulting in a substantial boost in F1-score, which is a consistent trend across all datasets. Consequently, edit-distance matching emerges as a better choice when target values are available, as it enhances the model's overall performance by providing a better trade-off between precision and recall.

This performance increase is more noticeable on string-based datasets, where minor inconsistencies are common. Specifically, on the Web Tables dataset under LLM-based classification, the F1-score improves from 0.816 with exact matching to 0.983 with edit-distance matching, representing a substantial increase of approximately 20.5%. This demonstrates that allowing for slight variations in matching can significantly enhance the model's ability to correctly map source to target values in real-world, noisy datasets with inherent inconsistencies. On the other hand, on the TT and KBWT datasets, which include more algorithmic and general transformations and may rely on external knowledge, the improvement in F1-score when using edit-distance matching is less pronounced. For example, on the KBWT dataset, the F1-score under LLM-based classification improves from 0.512 with exact matching to 0.567 with edit-distance matching, an increase of approximately 10.7%. This indicates that for general transformations, minor discrepancies are less probable to occur when the model is not constrained to a string-based transformation to cover all rows.

5.6 Class-Wise Performance and Error Analysis

In this section, we analyze the model's performance across each transformation class and highlight specific challenges encountered in some cases. This evaluation helps identify patterns in model performance and limitations that may inform future improvements. Table 5 provides an overview of the results for each class, with support indicating the number of tables in each category.

In the string transformation class, the framework demonstrates strong performance. When edit-distance matching is applied, only a few cases are missed, primarily due to noisy data or substantial

Table 4: The framework performance under various classification and matching methods

Classification	Dataset	Edit Distance Matching					Exact Matching				
		P	R	F1	AED	ANED	P	R	F1	AED	ANED
LLM	Web Tables (WT)	0.985	0.980	**0.983**	1.501	0.068	1.000	0.729	**0.816**	5.497	0.269
	SpreadSheet (SS)	0.955	0.949	**0.952**	1.944	0.094	1.000	0.818	**0.828**	4.879	0.180
	Table Transformation (TT)	0.904	0.895	0.898	2.678	0.219	0.966	0.736	**0.756**	4.252	0.340
	KBWT	0.722	0.532	0.567	7.203	0.398	0.975	0.435	0.512	7.139	0.387
Golden	Web Tables (WT)	0.975	0.971	0.973	2.223	0.094	1.000	0.690	0.770	6.362	0.308
	SpreadSheet (SS)	0.953	0.949	0.951	2.132	0.104	1.000	0.798	0.809	5.270	0.202
	Table Transformation (TT)	0.919	0.918	**0.918**	2.466	0.219	0.992	0.733	**0.756**	3.915	0.344
	KBWT	0.722	0.545	**0.580**	7.139	0.387	0.975	0.445	**0.523**	7.674	0.516

Table 5: Performance Metrics per Transformation Class

Class	Support	Edit Distance Matching					Exact Matching				
		P	R	F1	AED	ANED	P	R	F1	AED	ANED
String	207	0.957	0.953	0.955	1.795	0.120	1.000	0.775	0.802	4.144	0.223
Numbers	60	0.971	0.971	0.971	-	-	0.971	0.971	0.971	-	-
Algorithmic	105	0.874	0.865	0.865	3.342	0.263	1.000	0.577	0.604	5.488	0.422
General	76	0.702	0.530	0.567	6.926	0.365	0.974	0.419	0.507	7.564	0.506

variations in formatting that were not adequately represented in the provided examples. Edit-distance matching allows for minor discrepancies, improving recall, though it occasionally results in missed rows when format variations exceed the threshold for an approximate match. When exact matching is employed, the model's recall decreases in the string class. This drop is expected, as exact matching is more stringent and does not allow for minor discrepancies or transformations that partially transform the source into target. For instance, in a dataset containing information about state governors formatted as `Name; (birth date – death date)` in the source and `Name - (in office years)` in the target, the model successfully maps names but the date transformation is not feasible via string operations, which will adversely affect the performance when exact matching is employed.

Moreover, in cases where provided examples are not sufficiently representative the model may generate a transformation function that fails to generalize. Consider top row in Figure 1 as an example, if the examples only include names without middle names, the model might generate a transformation that concatenates the first letter of the first name with the last name, failing to account for middle names when they appear. This indicates the limitations of random sample selector used in our experiments and highlights the need for a more strategic approach to example selection, which is an area for future improvement.

For numeric transformations, the model performs exceptionally well across all tables, achieving near-perfect precision and recall. This success can be attributed to the straightforward nature of numeric relationships, where mathematical functions can precisely capture the transformation patterns. The few cases where the model did not perform well were mostly outliers or instances where no meaningful relationship existed between the input and output values. In the algorithmic transformation class, while the overall performance remains strong, we observe a slight drop in both precision and recall. The primary challenge here is the complexity of certain algorithms. For example, transforming Gregorian dates to Hijri dates involves a sophisticated algorithm that the model partially captures. While it correctly identifies the relationship among source and target, the generated code may is incomplete, leading to some missed transformations.

In the general transformation class, although the model significantly outperforms the baseline, the performance metrics are lower compared to the other classes. This transformation class is the most complex, as it heavily relies on the LLM's knowledge to produce accurate mappings, introducing several limitations. In some cases, the model has difficulty accurately identifying the types of the source and target columns. For instance, when the source is a U.S. Patent ID and the target is the patent name, the model may misinterpret the data, resulting in incorrect or nonsensical outputs. Even when the model correctly determines the column types, certain transformations in this class demand external knowledge beyond the model's pre-trained data. When tasked with mapping SWIFT codes to bank names, the model correctly recognizes the column types but often produces inaccurate or hallucinated outputs, indicating a limitation in knowledge availability.

Another common issue arises in one-to-many and many-to-many relationships, where multiple valid mappings exist for each source item. For example, in tables linking movies to their casts, a single movie may have numerous associated actors, and the model may generate a correct output that does not match the target, which is considered incorrect. These cases are not rare and contribute notably to the lower performance in the general class. Addressing

Table 6: Performance of the framework with Smaller and Open Source LLMs

Model	Dataset	Edit Distance Matching					Exact Matching				
		P	R	F1	AED	ANED	P	R	F1	AED	ANED
GPT-4o	Web Tables (WT)	0.975	0.971	**0.973**	2.223	0.094	1.000	0.690	**0.770**	6.362	0.308
	SpreadSheet (SS)	0.953	0.949	0.951	2.132	0.104	1.000	0.798	**0.809**	5.270	0.202
	Table Transformation (TT)	0.919	0.918	**0.918**	2.466	0.219	0.992	0.733	**0.756**	3.915	0.344
	KBWT	0.722	0.545	**0.580**	7.139	0.387	0.975	0.445	**0.523**	7.674	0.516
GPT-4o.m	Web Tables (WT)	0.924	0.920	0.922	3.752	0.142	0.983	0.626	0.685	8.214	0.372
	SpreadSheet (SS)	0.979	0.976	**0.977**	1.763	0.072	1.000	0.802	0.806	4.730	0.198
	Table Transformation (TT)	0.906	0.904	0.905	5.325	0.273	0.990	0.671	0.695	5.002	0.423
	KBWT	0.659	0.483	0.519	7.534	0.451	0.939	0.373	0.440	8.430	0.591
LLaMA	Web Tables (WT)	0.822	0.820	0.821	6.837	0.275	0.994	0.437	0.491	11.298	0.568
	SpreadSheet (SS)	0.915	0.912	0.913	3.551	0.235	0.981	0.389	0.398	8.942	0.611
	Table Transformation (TT)	0.815	0.811	0.812	7.422	0.505	0.992	0.452	0.465	7.531	0.717
	KBWT	0.546	0.317	0.358	9.288	0.597	0.856	0.222	0.278	9.423	0.751

these complex mappings effectively remains a challenge and is kept for future work.

5.7 Evaluation of Different Model Sizes and Open-Source Alternatives

To evaluate how the size and type of language models affect our framework's performance, we experimented with different commercial and open-source models of varying sizes. Table 6 summarizes the results of these experiments. In this table, GPT-4o refers to the `gpt-4o-2024-05-13` model, one of the leading state-of-the-art commercial models. GPT-4o.m denotes the smaller and more cost-effective `gpt-4o-mini-2024-07-18` model. The LLaMA model represents the `Llama-3.1-8B-Instruct`, an open-source model with 8 billion parameters executable on a single GPU.

As anticipated, the largest model, GPT-4o, generally outperforms the smaller models in terms of precision, recall, and F1-score, due to its advanced architecture and larger parameter count, especially when exact matching is used. However, for smaller models, edit-distance-based matching significantly mitigates performance gaps. This method accommodates minor variations in output, helping smaller models perform competitively. Notably, the gap between F1-scores using edit-distance versus exact matching grows as model size decreases. For instance, on the WT dataset, this difference is approximately 0.20 for GPT-4o, 0.24 for GPT-4o-mini, and 0.33 for LLaMA 3.1. The benefit of edit-distance matching even allows the GPT-4o-mini model to outperform GPT-4o on the SS dataset, underscoring its value in compensating for smaller models' limitations.

Interestingly, the smallest model tested, LLaMA 3.1, demonstrates baseline-comparable or superior performance across datasets with edit-distance matching. Although there is a noticeable performance drop compared to the GPT models, the gap is not substantial in the Spreadsheet dataset. This outcome implies that for less complex tasks or in scenarios where computational resources are limited, smaller open-source models like LLaMA can be a viable alternative. However, on more complex datasets like KBWT, which require handling intricate transformations and may depend on external knowledge, the larger GPT-4o model maintains a clear advantage. Additionally, when exact matching is necessary—such as in applications requiring high accuracy without tolerance for errors—the larger models demonstrate superior performance.

6 CONCLUSION AND FUTURE WORKS

In this paper, we introduced TabulaX, a novel framework that leverages LLMs for Multi-class Table Transformations. TabulaX addresses the challenges of integrating a wide range of heterogeneous tables with mismatched formats by classifying input data into distinct transformation classes—string-based, numerical, algorithmic, and general—and applying appropriate methods tailored to each class. Our approach generates human-interpretable transformation functions in the form of numeric formulas and programming code, enhancing transparency and allowing users to understand and modify the mappings as needed. Through extensive experiments on real-world datasets from various domains, we demonstrated that TabulaX outperforms existing state-of-the-art approaches in terms of accuracy and supports a broader class of transformations.

As possible future work, one direction is improving the handling of general transformations that require external knowledge; retrieval-based methods or domain-specific knowledge bases could be integrated to access additional information not present in the input data. Refinements to the classification mechanism could also be explored, potentially using more advanced models or leveraging metadata when available, to increase the accuracy of class detection. Additionally, extensions to support multi-column transformations, handling complex relational mappings, and further evaluations on downstream tasks such as anomaly detection, data imputation, and entity matching are among other potential future directions.

ACKNOWLEDGMENTS

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] Ziawasch Abedjan, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. 2016. DataXFormer: A robust transformation discovery system. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 1134–1145.

[2] Mehdi Akbarian Rastaghi, Ehsan Kamalloo, and Davood Rafiei. 2022. Probing the Robustness of Pre-trained Language Models for Entity Matching. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 3786–3790.

[3] Rajeev Alur, Dana Fisman, Rishabh Singh, and Armando Solar-Lezama. 2016. SyGuS-Comp 2016: Results and Analysis. *Electronic Proceedings in Theoretical Computer Science* 229 (Nov 2016), 178–202.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901.

[5] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A Survey on Evaluation of Large Language Models. *ACM Trans. Intell. Syst. Technol.* 15, 3, Article 39 (March 2024), 45 pages.

[6] Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Wang, and William W Cohen. 2020. Open question answering over tables and text. *arXiv preprint arXiv:2010.10439* (2020).

[7] Arash Dargahi Nobari and Davood Rafiei. 2022. Efficiently Transforming Tables for Joinability. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1649–1661.

[8] Arash Dargahi Nobari and Davood Rafiei. 2024. DTT: An Example-Driven Tabular Transformer for Joinability by Leveraging Large Language Models. *Proc. ACM Manag. Data (SIGMOD)* 2, 1, Article 24 (March 2024), 24 pages.

[9] Dong Deng, Guoliang Li, Shuang Hao, Jiannan Wang, and Jianhua Feng. 2014. MassJoin: A mapreduce-based method for scalable string similarity joins. In *2014 IEEE 30th International Conference on Data Engineering*. 340–351.

[10] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. 2020. TURL: Table Understanding through Representation Learning. *Proc. VLDB Endow.* 14, 3 (2020), 307–319.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*. Association for Computational Linguistics, 4171–4186.

[12] AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of data integration*. Elsevier.

[13] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[14] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets Using Input-Output Examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Austin, Texas, USA) *(POPL '11)*. Association for Computing Machinery, New York, NY, USA, 317–330.

[15] Sumit Gulwani, William R. Harris, and Rishabh Singh. 2012. Spreadsheet Data Manipulation Using Examples. *Commun. ACM* 55, 8 (Aug. 2012), 97–105.

[16] Oktie Hassanzadeh, Ken Q. Pu, Soheil Hassas Yeganeh, Renée J. Miller, Lucian Popa, Mauricio A. Hernández, and Howard Ho. 2013. Discovering linkage points over web data. *Proc. VLDB Endow.* 6, 6 (April 2013), 445–456.

[17] Jian He, Enzo Veltri, Donatello Santoro, Guoliang Li, Giansalvatore Mecca, Paolo Papotti, and Nan Tang. 2016. Interactive and deterministic data cleaning. In *Proceedings of the 2016 International Conference on Management of Data*. 893–907.

[18] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek Narasayya, and Surajit Chaudhuri. 2018. Transform-data-by-example (TDE): an extensible search engine for data transformations. *Proc. VLDB Endow.* 11, 10 (2018), 1165–1177.

[19] Yeye He, Kris Ganjam, and Xu Chu. 2015. SEMA-JOIN: Joining Semantically-Related Tables Using Big Table Corpora. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1358–1369.

[20] Yeye He, Kris Ganjam, Kukjin Lee, Yue Wang, Vivek Narasayya, Surajit Chaudhuri, Xu Chu, and Yudian Zheng. 2018. Transform-Data-by-Example (TDE): Extensible Data Transformation in Excel. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 1785–1788.

[21] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. TABBIE: Pretrained Representations of Tabular Data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 3446–3456.

[22] Zhongjun Jin, Michael R. Anderson, Michael Cafarella, and H. V. Jagadish. 2017. Foofah: Transforming Data By Example. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 683–698.

[23] Mihir Kale and Abhinav Rastogi. 2020. Text-to-Text Pre-Training for Data-to-Text Tasks. In *Proceedings of the 13th International Conference on Natural Language Generation*. Association for Computational Linguistics, Dublin, Ireland, 97–102. https://aclanthology.org/2020.inlg-1.14

[24] Aamod Khatiwada, Roee Shraga, Wolfgang Gatterbauer, and Renée J Miller. 2022. Integrating data lake tables. *Proceedings of the VLDB Endowment* 16, 4 (2022), 932–945.

[25] Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. 2021. Auto-FuzzyJoin: Auto-Program Fuzzy Similarity Joins Without Labeled Examples. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD/PODS '21)*. Association for Computing Machinery, New York, NY, USA, 1064–1076.

[26] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-GPT: Table Fine-tuned GPT for Diverse Table Tasks. *Proc. ACM Manag. Data* 2, 3, Article 176 (May 2024), 28 pages.

[27] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (sep 2020), 50–60.

[28] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (oct 2020), 50–60. https://doi.org/10.14778/3421424.3421431

[29] Mingxing Liu, Junfeng Wang, Tao Lin, Quan Ma, Zhiyang Fang, and Yanqun Wu. 2024. An Empirical Study of the Code Generation of Safety-Critical Software Using LLMs. *Applied Sciences* 14, 3 (2024).

[30] Renée J Miller. 2018. Open data integration. *Proceedings of the VLDB Endowment* 11, 12 (2018), 2130–2139.

[31] Renée J Miller, Fatemeh Nargesian, Erkang Zhu, Christina Christodoulakis, Ken Q Pu, and Periklis Andritsos. 2018. Making Open Data Transparent: Data Discovery on Open Data. *IEEE Data Eng. Bull.* 41, 2 (2018).

[32] Md Mahadi Hasan Nahid and Davood Rafiei. 2024. NormTab: Improving Symbolic Reasoning in LLMs Through Tabular Data Normalization. *arXiv preprint arXiv:2406.17961* (2024).

[33] Mohamed Nejjar, Luca Zacharias, Fabian Stiehle, and Ingo Weber. 2023. LLMs for science: Usage for code generation and data analysis. *Journal of Software: Evolution and Process* (2023), e2723.

[34] Soroush Omidvartehrani, Arash Dargahi Nobari, and Davood Rafiei. 2024. GXJoin: Generalized Cell Transformations for Explainable Joinability. In *Advances in Databases and Information Systems*. Springer Nature Switzerland, Cham, 123–137.

[35] Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A Controlled Table-To-Text Generation Dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 1173–1186. https://doi.org/10.18653/v1/2020.emnlp-main.89

[36] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research* 21 (2020), 140:1–140:67.

[37] Ananth Ranganathan. 2004. The levenberg-marquardt algorithm. *Tutoral on LM algorithm* 11, 1 (2004), 101–110.

[38] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. 2024. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927* (2024).

[39] Roee Shraga and Renée J Miller. 2023. Explaining dataset changes for semantic data versioning with explain-da-v. *Proc. VLDB Endow.* 16, 6 (2023).

[40] Rishabh Singh. 2016. BlinkFill: Semi-Supervised Programming by Example for Syntactic String Transformations. *Proc. VLDB Endow.* 9, 10 (June 2016), 816–827.

[41] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table Meets LLM: Can Large Language Models Understand Structured Table Data? A Benchmark and Empirical Study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining* (Merida, Mexico) *(WSDM '24)*. Association for Computing Machinery, New York, NY, USA, 645–654.

[42] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Sam Madden, and Mourad Ouzzani. 2021. RPT: Relational Pre-Trained Transformer is Almost All You Need towards Democratizing Data Preparation. *Proc. VLDB Endow.* 14, 8 (2021), 1254–1261.

[43] James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Halevy. 2021. From natural language processing to neural databases. In *Proc. VLDB Endow.*, Vol. 14. VLDB Endowment, 1033–1039.

[44] Jianxun Wang and Yixiang Chen. 2023. A Review on Code Generation with LLMs: Application and Evaluation. In *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*. 284–289.

[45] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).

[46] Yue Wang and Yeye He. 2017. Synthesizing Mapping Relationships Using Table Corpus. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1117–1132.

[47] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*, Vol. 35. Curran Associates, Inc., 24824–24837.

[48] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a Token-Free Future with Pre-trained Byte-to-Byte Models. *Transactions of the Association for Computational Linguistics* 10 (03 2022), 291–306.

[49] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems*, Vol. 36. Curran Associates, Inc., 11809–11822.

[50] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 8413–8426.

[51] Tianshu Zhang, Xiang Yue, Yifei Li, and Huan Sun. 2023. Tablellama: Towards open large generalist models for tables. *arXiv preprint arXiv:2311.09206* (2023).

[52] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-End Fuzzy Entity-Matching Using Pre-Trained Deep Models and Transfer Learning. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 2413–2424.

[53] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-join: Joining tables by leveraging transformations. *Proc. VLDB Endow.* 10, 10 (2017), 1034–1045.