Auto-Prep: Holistic Prediction of Data Preparation Steps for Self-Service Business Intelligence

Eugenie Y. Lai
MIT
eylai@mit.edu

Yeye He
Microsoft Research
yeyehe@microsoft.com

Surajit Chaudhuri
Microsoft Research
surajitc@microsoft.com

ABSTRACT

Business Intelligence (BI) plays a critical role in empowering modern enterprises to make informed data-driven decisions, and has grown into a billion-dollar business. Self-service BI tools like Power BI and Tableau have democratized the "dashboarding" phase of BI, by offering user-friendly, drag-and-drop interfaces that are tailored to non-technical enterprise users. However, despite these advances, we observe that the "data preparation" phase of BI continues to be a key pain point for BI users today.

In this work, we systematically study around 2K real BI projects harvested from public sources, focusing on the data-preparation phase of the BI workflows. We observe that users often have to program both (1) data transformation steps and (2) table joins steps, before their raw data can be ready for dashboarding and analysis. A careful study of the BI workflows reveals that transformation and join steps are often intertwined in the same BI project, such that considering both holistically is crucial to accurately predict these steps. Leveraging this observation, we develop an Auto-Prep system to holistically predict transformations and joins, using a principled graph-based algorithm inspired by Steiner-tree, with provable quality guarantees. Extensive evaluations using real BI projects suggest that Auto-Prep can correctly predict over 70% transformation and join steps, significantly more accurate than existing algorithms as well as language-models such as GPT-4.

PVLDB Reference Format:
Eugenie Y. Lai, Yeye He, and Surajit Chaudhuri. Auto-Prep: Holistic Prediction of Data Preparation Steps for Self-Service Business Intelligence. PVLDB, 14(7): 2212 - 2225, 2025.
doi:10.14778/3734839.3734856

1 INTRODUCTION

Business Intelligence (BI) plays a crucial role in modern enterprises, by empowering non-technical enterprise users to make informed data-driven decisions, and has grown into a billion-dollar business.

Over the past decades, "self-service" BI tools like Power BI and Tableau have democratized BI [4, 5], by providing intuitive drag-and-drop interfaces, to enable even non-technical users to create engaging BI dashboards for interactive data analysis.

Data preparation remains a pain point in BI. Despite the advances in the "dash-boarding phase" of BI, which is a strength of today's BI tools, depicted as the final step (S3) in Figure 1, we observe that non-technical users continue to struggle with the "data preparation" phase, which in a typical BI workflow often needs to happen first before dashboards can be built and analysis can be performed, like depicted as step (S1) and (S2) in the figure.

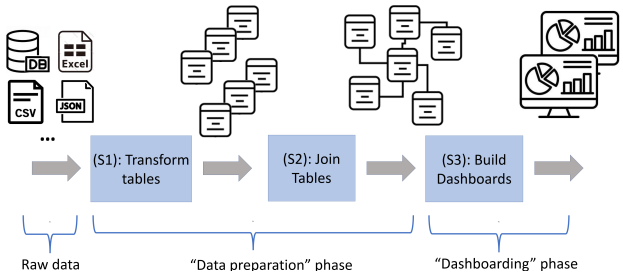


Figure 1: Typical workflows of end-to-end BI. (S1) Raw data is first transformed; (S2) the transformed tables are then joined to create a BI model; (S3) finally, users use the BI model to create dashboards. We in this work focus on (S1) "transform" and (S2) "join", in the "data preparation" phase of BI.

Specifically, starting from raw data (which can be database tables, or flat files like CSV and Excel, etc.), users typically need to go through two steps in the data preparation phase, which are (S1) Transform: or convert data files into proper tables, by relationalizing data [35, 62] and standardizing values [44, 46]; and then (S2) Join: or link transformed tables via logical join relationships [3, 32], similar to how primary keys / foreign keys (PKs/FKs) are specified in databases, so that inter-linked analysis (e.g., cross-filtering) between multiple tables becomes possible on dashboards [6, 20].

Both of these data preparation steps, (S1) transform and (S2) join, are common in practice. In a sample of 1837 real BI projects, we found 89% have multiple tables that would require joins, and 43% require transformations. We use an example simplified from real BI projects, to illustrate the two types of data-preparation steps.

Example 1. Consider our BI expert Alex who collected Table 1-4, and now wants to build BI dashboards to visualize how birth rates and economic statistics changed in different countries over time.

For a BI expert and from a dimensional-modeling perspective [38, 57, 69], it would be obvious that `Fertility` and `Economics` are what is known as "*fact tables*" that contain key numerical measures of interest (e.g., fertility rates and economic readings), while "`Date`" and "`Country`" are "*dimension tables*", whose primary keys ("`Year`" and "`Country`") can join with corresponding foreign-key attributes in the fact tables, so that users can slice/dice to perform analysis over these dimensional attributes (e.g., to answer questions like "in which decade and which continent is fertility the highest?")

However, as is often the case in real-world BI, these raw input tables are not yet ready for analysis in their current forms, because:

Proceedings of the VLDB Endowment, Vol. 14, No. 7 ISSN 2150-8097.
doi:10.14778/3734839.3734856

Table 1: Fertility (fact)

Country	2010	2011	2012
Poland	1.38	1.3	1.3
Chile	1.86	1.84	1.83
Morocco	2.58	2.65	2.71
Turkey	2.1	2.08	2.06

Table 2: Date (dim)

Year	IsLeap	Days
2010	No	365
2011	No	365
2012	Yes	366

Table 3: Economics (fact)

Line-ID	Code	Metric	Value
CHL-CY2010	CHL	GDP	12756
CHL-CY2011	CHL	GDP	14637
CHL-CY2012	CHL	GDP	15397
CHL-CY2010	CHL	CPI	100
CHL-CY2011	CHL	CPI	103.3
CHL-CY2012	CHL	CPI	106.4
CHL-CY2010	CHL	Payroll	55.16
CHL-CY2011	CHL	Payroll	57.01
CHL-CY2012	CHL	Payroll	57.43

Table 4: Country (dim)

Code	POL	CHL	TUR	MAR
Country	Poland	Chile	Turkey	Morocco
Continent	Europe	S. America	Europe	Africa
Developed	Yes	No	Yes	No

(1) The year values ("2010", "2011", etc.) are presented as column headers in the "Fertility" table (Table 1), in a cross-tabulated format, which makes it hard to join these years with the "Year" column in the dimension-table "Date" (Table 2) for analysis;
(2) Even though both "Fertility" (Table 1) and "Economics" (Table 3) need to join with the dimension-table "Country" (Table 4) for analysis, countries in the "Country" table are not shaped in the vertical direction to make such joins possible;
(3) The table "Economics" (Table 3) has a "Line-ID" column using the country-code-then-year format ("CHL-CY2010"), which needs to be transformed (e.g., into "2010"), before it can equi-join with the "Year" column in "Date" (Table 2);
(4) Finally, the fact table "Economics" (Table 3) contains different types of economic readings (GPD, CPI, Payroll, etc.), which however are stored as key-value pairs in the column "Metric" and "Value", that should move into separate columns for analysis;

As a seasoned BI expert, Alex identified these necessary steps, often using clues based on what needs to "join" to decide what transformations may be needed, before any analysis can be performed. She then went on to program these steps, using her favorite tools (which can be Python Pandas, R, etc.). Specifically, she would:
(1) Invoke the "Unpivot" transformation on "Fertility" in Table 1, so that the years become a new column in the transformed version of "Fertility" in Table 5 (marked in yellow), which becomes joinable with the "Year" column in the dimension table "Date", also marked in yellow in Table 7;
(2) Invoke the "Transpose" transformation on "Country" in Table 4, to produce a transformed version of "Country" in Table 8, which becomes joinable with both the transformed "Fertility" in Table 5, through the "Country" column (marked in green), and also "Economics" in Table 6, through the "Code" column (marked in purple);
(3) Invoke a string transformation "Split" on the "Line-ID" column of "Economics" in Table 3, which splits the field using delimiter "-", takes the second component, and then uses "Substring" to extract the last 4 characters to produce a new "Year" column in Table 6 (marked in yellow), which becomes joinable with the "Year" column of "Date" in Table 7 (also marked in yellow).
(4) Invoke the "Pivot" transformation on "Economics" in Table 3, to produce Table 6, which now has different metrics (GDP, CPI, etc.) as separate columns for analysis;

After Alex programmed the transformations above, she would have completed the transformation stage of BI (stage S1 in Figure 1). Alex can then join/link the transformed tables in Table 5-8, which

Table 5: Fertility (fact)

Country	Year	Fertility
Chile	2010	1.86
Chile	2011	1.84
Chile	2012	1.83
Turkey	2010	2.30
Turkey	2011	2.21
Turkey	2012	2.34

Table 6: Economics (fact)

Year	Code	GDP	CPI	Payroll
2010	CHL	12756	100	55.16
2011	CHL	14637	103.3	57.01
2012	CHL	15397	106.4	57.43
2010	TUR	71022	100	93.29
2011	TUR	78102	102.3	98.10
2012	TUR	82019	104.1	99.38

Table 7: Date (dim)

Year	IsLeap	Days
2010	No	365
2011	No	365
2012	Yes	366

Table 8: Country (dim)

Code	Country	Continent	Developed
POL	Poland	Europe	Yes
CHL	Chile	S. America	No
TUR	Turkey	Europe	Yes
MAR	Morocco	Africa	No

are effectively PK/FK joins as indicated by their colors, to complete the join stage (S2 in Figure 1). These logical join relationships create what is known as a "BI model" [3, 32], from which dashboards can then be easily built using drag-and-drop in tools like Tableau and Power BI (reaching stage S3 of Figure 1). Note that in this process, data formats across tables have been automatically standardized, so that tables can be linked and enriched using other tables. □

While our BI professional Alex has the expertise to program all these transformation and join steps, it is clearly challenging for non-technical users in self-service BI-tools like Tableau and Power BI, who typically are neither DBAs nor BI professionals. This challenge is evidenced by large numbers of user questions on Power BI and Tableau forum [15, 23], where users are often stuck and raise various questions about the data preparation process (e.g., how to transform a particular table [12, 13, 21, 22], and how best to join two related tables [14, 16, 24, 25], etc.).

Prior art: predictions for transform and join only. We are not the first to recognize the need to predict transformations or joins — while there is extensive prior research, existing methods predominantly predict either only transformations, or only joins, as two *separate and standalone* problems. For example, prior work on join predictions, such as PK/FK joins [41, 51, 64, 73, 89], all assumes that input tables are already properly transformed, and therefore does not consider transformations that need to happen before joins.

Similarly, while there is existing research on predicting transformations [35, 39, 59, 62, 86], existing approaches all operate on *individual tables* that predict one table at a time, without taking into account signals across tables, and how transformations interact with joinability in a global sense.

When testing existing join-only and transform-only prediction methods on real BI projects, we find that they perform poorly on complex multi-table BI projects, because transformation and join are not considered in a holistic manner.

Auto-Prep: "holistic" prediction across tables and steps. Our insights from inspecting real BI projects reveal that, interestingly, (1) Considering join and transform jointly can allow the two to "help each other", in finding correct joins and transforms; (2) Considering transforms across multiple tables in the same BI project also allows these tables to "help each other" in finding the most likely transforms. We argue that by modeling end-to-end data preparation (both transform and join) holistically and across all tables, and by focusing on the set of most common and important transforms in the context of BI that we will precisely define using a DSL, predictions for both transform and join can be made more accurately, as we illustrate in the following example.

Example 2. We revisit Table 1-4 in Example 1 again. Looking at "Fertility" (Table 1) alone, we may not be sure if an "Unpivot"

is necessary to transform the column headers "2010/2011/2012" into column values. However, inspecting other related tables in the same BI project, such as "Date" (Table 2) with a column "Year" containing value "2010/2011/2012", should give us clues that "2010/2011/2012" are data values, and an "Unpivot" is needed for Table 1 (to produce Table 5), so that the two tables can join.

Similarly, looking at the "Country" table (Table 4) in isolation, we may not be sure if "Transpose" or "Unpivot" would be required, but the presence of the "Country" column in Table 1 and the "Code" column in Table 3 gives strong indication that Table 4 is oriented in the row direction, and a "Transpose" is needed to turn its rows into columns (producing Table 8), which can then become joinable with both Table 1 and Table 3 for analysis.

As a final example, in the "Economics" table (Table 3), the need to perform "Split" and "Substring" to extract year values like "2010/2011", is obvious only in the presence of the "Year" column in "Date" (Table 2), for the two to join in downstream analysis. □

As we can see from these examples, holistic reasoning across transform/join, and for all tables in the same project, helps us accurately predict both transforms and joins, which is a unique opportunity overlooked by existing methods that tend to consider transform and join as separate problems.

While the intuition for holistic prediction is clear, the technical challenges lie in principled modeling of disparate types of transformation steps (Unpivot, Transpose, Pivot, Split, Substring, etc.) that are common in BI transformations, together with join steps. In this work, we develop Auto-Prep that builds on a new graph-based formulation, which seamlessly integrates diverse classes of transforms and joins in a unified search graph, with probabilistic interpretations of the transform/join steps as weighted edges on the graph. We develop new graph algorithms inspired by Steiner-tree, to solve the resulting optimization problem both efficiently, and with provable quality guarantees.

We evaluate Auto-Prep using real BI projects, treating user-programmed transforms and joins as ground truth. Our results suggest that Auto-Prep can accurately predict over 70% of the data preparation steps, substantially better than alternative methods.

2 RELATED WORK

Business intelligence (BI). Business intelligence (BI) plays a crucial role in modern enterprises, with leading vendors such as Tableau [34] and Power-BI [33] offering "self-service" BI tools that are highly popular, especially among non-technical users [4, 5], by enabling them to build visualization dashboards using simple drag-and-drop [65]. BI is closely related to research topics such as data warehousing [38, 57, 85], OLAP [45, 74, 81], and ETL [76, 78, 80].

Join-only prediction methods. Join is a core operation in data management, with a large body of work studying join predictions, which, however, do not consider the need for transformations.

For example, the canonical join prediction problem studies the detection of PK/FK joins in classical database settings, with many influential methods developed in the literature, such as MC-FK [89], Fast-FK [41], HoPF [51], ML-FK [73], BI-Join [64], etc. All of these methods consider a classical database scenario, where tables loaded into databases are already properly transformed/cleaned a priori (e.g., by ETL engineers), where additional transformations are not necessary. However, this is insufficient for the real BI workflows we study, where users often have to start from raw data tables that need to be transformed first, before they are ready for analysis. There exist other types of joins beyond equi-joins, such as semantic joins [42, 47, 70] and fuzzy joins [61, 75, 91], which we consider as out of scope for this work.

Transformation-only prediction methods. There is also extensive research on transformation predictions, which, however, do not consider joins that need to happen afterwards. Furthermore, existing transformation-prediction methods make predictions *one table at a time*, overlooking the rich interactions between tables that can provide signals for what transformations may be needed. For instance, transformations are predicted based on user-provided examples [35, 39, 43, 44, 46, 48, 52, 53, 77], inferred data patterns [27, 54], characteristics of input tables [59, 65, 86], or relationships between input/output tables [36, 79, 82, 88, 90, 93], but most focusing on one single table at a time. We show that on complex BI projects, such methods tend to produce inaccurate predictions.

In this work, we demonstrate how existing transform and join predictions can be seamlessly integrated and optimized holistically, for accurate predictions at a global level.

3 PRELIMINARIES

3.1 Real-world BI projects

To study real-world BI workflows end-to-end, we use 1837 real BI projects created using the popular Power BI tool (in the .pbix format), crawled from public sources. Our inspection suggests that these BI projects cover diverse application domains, including financial reporting, inventory management, sales analytics, etc.

For each crawled BI project, we programmatically extract the following: (1) a set of raw input tables, (2) user-programmed data transformation steps on each table (the ground-truth transformations we want to predict), and (3) user-specified join relationships on the transformed tables (the ground-truth joins we want to predict). Example 1 shows a simplified version of such a BI project, as well as what we extract from the BI project. Detailed statistics of the BI projects we collected will be discussed in Section 8.

3.2 Transformations in BI

In this work, we consider 6 common transformation operators listed in Table 9 that are essential for BI analysis, which cover two broad categories: (1) table-reshaping transformations, and (2) string transformations. These transformations are both common and crucial to enable multi-table BI analysis, as illustrated in Example 1.

Table-reshaping transformations. Reshaping transformations such as Unpivot [29], Transpose [26], Pivot [9], change the basic shape/structure of an input table, which are frequently used to convert raw data into a suitable relational form for analysis. Variants of these reshaping transformations have been studied in prior work (e.g., [35, 52, 62, 86]), and we will give a short overview of these transformations below.

Unpivot. The unpivot operator collapses a set of selected columns (specified by a "selected_columns" parameter, shown in the first line of Table 9) into one single column, while keeping the remaining columns unchanged. For instance, in Example 1, the column headers "2010/2011/2012" in Table 1 need be to unpivoted, in order to

Table 9: DSL of transformation operators in Auto-Prep.

Category	Operator	Operator parameters	Operator description (example in parenthesis)
Table-reshaping	Unpivot [29]	start_column, end_column	Collapse homogeneous columns into one column (e.g., unpivot Table 1 to produce Table 5)
	Pivot [9]	pivot_column, value_column	Lift values in a column into column headers (e.g., pivot Table 3 to produce Table 6)
	Transpose [26]	none	Convert rows into columns and columns into rows (e.g., transpose Table 4 to produce Table 8)
String-transform	Split [18]	delimiter, select_pos	Split a string using delimiters (e.g., split to extract years from Table 3 to produce Table 6)
	Concatenate [17]	array of strings or columns	Concatenate two strings and produce an output string
	Substring [19]	column, start, length	Extract substrings by positions (e.g., substring to extract years from Table 3 to produce Table 6)
	No-op	none	No transformation is required for a table (e.g., Table 2 requires no transformation for the BI project)

produce Table 5, which becomes joinable with Table 7, and is more amenable to analysis (e.g., for range-filters or aggregation).

Unpivot is a common operator that is available in Python Pandas [29] and R [30] for programmers and data scientists, as well as in Power BI [28] and Tableau [31] for less technical BI users.

Pivot. The pivot operator is the inverse operator of unpivot, which lifts the values in a column into column headers. In Example 1, values in the "`Metric`" column of Table 3 ("`GDP/CPI/Payroll`") need to be pivoted into separate columns for relational analysis, in accordance with principles such as the First Normal Form [56]. Like unpivot, this is a common operator in Python Pandas [9], R [10], Power BI [8], and Tableau [11].

Transpose. The transpose operator turns rows to columns and vice versa. In Example 1, transpose is necessary to convert Table 4 to Table 8, so that joins become possible for cross-table analysis.

String transformations. Unlike reshaping transformations, string transformations operate on values in the same row and do not alter the shape of the input table. We give an overview of these operators (Table 9) in the interest of space and refer the reader to [35, 39, 44, 46, 52] for more details.

Split. The split operator uses a "`delimiter`" parameter to break an input string into segments, from which one segment is selected using a "`select_pos`" parameter, to produce an output string.

Concatenate. The concatenate operator is the reverse of split, which assembles multiple strings into one output string.

Substring. The substring operator selects a sub-part of a string column, based on a "`start_pos`" and "`length`" parameter.

String transformation operators often need to be composed, to produce a desired output [44, 46]. For instance, in Example 1, the "`Line-ID`" column in Table 3 is first split using `delimiter = "-"`, to produce an array of segments, `{[CY2010, CHL], [CY2011, CHL], ...}`, from which the first segment is selected (`select_pos = "1"`) to produce `{CY2010, CY2011, CY2012}`. Then a substring operator, with parameters `start_pos = 2` and `length = 4`, is used to produce values `{2010, 2011, 2012}`, which become the "`Year`" column in Table 6 that is joinable with Table 5 and Table 7.

No-op. Finally, many tables in BI projects may require no transformations (e.g., Table 2). It is crucial that our predictions do not "over-trigger", and should correctly predict "no-op" on such tables.

Additional transformations. We focus on the transformation in Table 9, which are key to enable cross-table analysis in real BI projects.[1] We note that our approach is general and extensible, where new transformation operators can easily "plug-in". We give details of additional transformations and extensibility in [1].

3.3 Join relationships in BI

Real BI projects often come with multiple tables (89% of BI projects we study have more than one table). With multiple tables in one BI project, it is crucial to create accurate join relationships across tables, to enable cross-table analysis. Popular BI tools all offer intuitive GUI tools to help users define join relationships between tables, as documented in tutorials for Power-BI [3] and Tableau [32].

[1]We consider operations that usually require humans in the loop to verify/approve, such as error detections and data cleaning (e.g., [49, 50, 66, 67, 83, 87]) as out of scope.

From a database perspective, these join relationships in BI are often PK/FK joins, whose detection has been extensively studied [41, 51, 64, 73, 89]. However, existing PK/FK methods are designed for the classical database setting where input tables are assumed to be properly transformed already, which is insufficient for the end-to-end BI scenario we study.

4 PROBLEM DEFINITION

We now define our high-level *Most-Probable BI-Prep (MPBP)* problem, as finding the most likely data preparation (join and transform) steps, for a given set of input tables, such that the joint probability of these (join and transform) steps is maximized. We will state the high-level MPBP problem below first, before making it a concrete graph optimization problem (in Section 7).

Definition 1. [Most-Probable BI-Prep (MPBP)]. Given a set of n input tables $\mathcal{T} = \{T_i | i \in [n]\}$, and a space of transformation operators $\mathbf{O} = \{$no-op, `transpose`, `unpivot`, `pivot`, `split`, `concatenate`, `substring`, $\ldots\}$, where each operator $O \in \mathbf{O}$ can be parameterized as $O(P)$, using parameter P drawn from a space of parameters $\mathcal{P}$. For each input table $T_i \in \mathcal{T}$, let $S_i = (O_1(P_1), O_2(P_2), \ldots)$ be a candidate sequence of appropriately parameterized transformations on T_i, and let $p(S_i|T_i)$ be the probability of transformation S_i given the input table T_i. Denote by $S_i(T_i)$ the transformed version of T_i, $\mathcal{S} = \{S_i | i \in [n]\}$ the set of all transformations for each $T_i \in \mathcal{T}$, and by $\mathcal{S}(\mathcal{T}) = \{S_i(T_i) | i \in [n]\}$ the set of all transformed tables. Finally, let $J(\mathcal{S}(\mathcal{T}))$ be the candidate joins found on the transformed $\mathcal{S}(\mathcal{T})$.

The *Most-Probable BI-Prep (MPBP)* problem is to find the optimal set of transformations $\mathcal{S}^*$, such that the overall transformation probability, written as $p(\mathcal{S}|\mathcal{T}) = \prod_{i \in [n]} p(S_i|T_i)$, and the join probability of $\mathcal{S}(\mathcal{T})$, written as $p(J(\mathcal{S}(\mathcal{T})))$, are jointly maximized:

$$\mathcal{S}^* = \arg\max_{\mathcal{S}} \; p(\mathcal{S}|\mathcal{T}) \cdot p(J(\mathcal{S}(\mathcal{T}))) \tag{1}$$

$$= \arg\max_{\mathcal{S}} \left(\prod_{S_i \in \mathcal{S}} p(S_i|T_i) \right) \cdot p(J(\mathcal{S}(\mathcal{T}))) \tag{2}$$

□

Note that both the term $p(\mathcal{S}|\mathcal{T})$, to estimate the probability of transformations, and the term $p(J(\mathcal{S}(\mathcal{T})))$, to estimate the probability of joins, have been studied separately in the literature (e.g., [52, 62, 86] for transformations, and [41, 51, 64, 73] for joins). We will describe how we enhance and integrate these estimations in our holistic graph formulation in Section 6.

The main challenge of MPBP lies in the need to *jointly optimize* both transforms and joins across all tables, so that the predicted transformations $\mathcal{S}$ not only have high transformation probabilities, but also the transformed tables $\mathcal{S}(\mathcal{T})$ should have high join probabilities. This is important, because greedily finding the most likely transform/join in isolation often leads to sub-optimal solutions, as we illustrate in the following example.

Example 3. We revisit Example 1, and use T_1 to T_4 to denote Table 1 to Table 4 for short. Recall that the desired transformations for T_1 to T_4, denoted as S_1 to S_4, are:

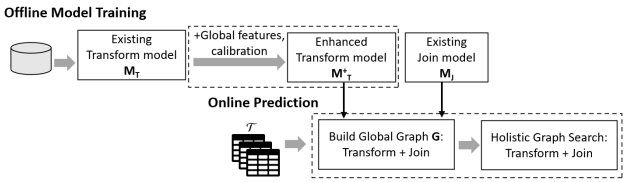


Figure 2: Architecture overview of Auto-Prep.

S_1 = (Unpivot("2010", "2012"))
S_2 = (no-op)
S_3 = (Pivot("Metric", "Value"), Substring(Split ("Line-ID", "-")[1]), 2, 4)
S_4 = (Transpose())

Let $\mathcal{S}^* = \{S_1, S_2, S_3, S_4\}$ be a candidate solution, and let the transformation probability of $p(S_1|T_1)$, $p(S_2|T_2)$, $p(S_3|T_3)$, $p(S_4|T_4)$ be 0.8, 0.8, 0.8, and 0.4, respectively. The overall transformation probability $p(\mathcal{S}^*|\mathcal{T})$ is then $(0.8)^3 \cdot 0.4$. Furthermore, on the transformed tables $\mathcal{S}^*(\mathcal{T})$, let the join probability of the resulting joins (color-coded in Table 5 to Table 8), be $p(J(\mathcal{S}^*(\mathcal{T}))) = 0.9$. The overall objective function in Equation (1) is then $(0.8)^3 \cdot 0.4 \cdot 0.9 = 0.18$.

Now suppose there is an alternative transformation for T_4, S'_4 = (no-op) (or not transposing T_4), which has a slightly higher probability $p(S'_4|T_4) = 0.6$. If we were to optimize transformations alone, independent of the joins, we may find $\mathcal{S}^- = \{S_1, S_2, S_3, S'_4\}$ to have better transformation probability, $p(\mathcal{S}^-|\mathcal{T}) = 0.8^3 \cdot 0.6$, which is greater than $p(\mathcal{S}^*|\mathcal{T}) = (0.8)^3 \cdot 0.4$.

However, when we bring the "goodness" of joins into the picture, we find the join probability $p(J(\mathcal{S}^-(\mathcal{T}))) = 0.5$ to be considerably inferior to $p(J(\mathcal{S}^*(\mathcal{T}))) = 0.9$, because not transforming T_4 (no-op) leads to worse overall joins, when the original T_4 cannot join with any other table. Overall, because $p(J(\mathcal{S}^*(\mathcal{T}))) \cdot p(\mathcal{S}^*|\mathcal{T}) = (0.8)^3 \cdot 0.4 \cdot 0.9 = 0.18$ is better than $p(J(\mathcal{S}^-(\mathcal{T}))) \cdot p(\mathcal{S}^-|\mathcal{T}) = (0.8)^3 \cdot 0.6 \cdot 0.5 = 0.15$, the MPBP formulation in Definition 1 would select $\mathcal{S}^*$, which are the desired transformations in Example 1. □

Since we are searching transform/join candidates across all n tables simultaneously, it is easy to see that the search space of MPBP can become intractable quickly. This is because for each input table, with 7 operators, and hundreds of possible parameters for each operator, only considering one-step transformations already yields a search space of at least $(7 \cdot 100)^n$, which is before we even consider multi-step transformations, and different join paths.

In the following, we describe how to solve MPBP using principled graph optimizations, both efficiently and with quality guarantees.

5 AUTO-PREP: OVERVIEW

In this section, we give an overview of how we solve MPBP using Auto-Prep. The architecture of our system is shown in Figure 2, which at a high level has (1) an offline model training step, and (2) an online prediction step using graph-based optimizations.

In the offline model training stage, shown in the top half of Figure 2, we leverage existing join prediction models [64, 73], denoted as M_J in the figure, as well as existing transformation prediction models M_T designed for single tables (e.g. Pivot [86], Unpivot [86], String-Transform [92]). Using all input tables holistically, we enhance the transformation classifiers using global-level features, which we denote as M^+_T, to better estimate $p(\mathcal{S}|\mathcal{T})$ in Equation (1). We will highlight the key intuition in the offline part in Section 6 .

The online prediction phase, shown in the lower half of Figure 2, is what we consider the core contributions of Auto-Prep, where we holistically consider all transformation and join candidates across all tables defined in MPBP (Definition 1). Specifically, for a given set of input tables $\mathcal{T}$, we construct a global search graph G, where we model each (original or transformed) table as a vertex, and all transformation or join candidates as edges, using probabilities from M^+_T and M_J models as edge-weights. We show that MPBP can then be equivalently solved on the graph, using a new graph algorithm we develop. We describe the core algorithm in Section 7 in detail.

It should be noted that our Auto-Prep framework is general and extensible, as we can extend and "plug-in" new transformation operators (together with their model M_T), while still utilizing the same graph algorithm, which is a salient feature of Auto-Prep.

6 OFFLINE MODEL TRAINING

We now describe the offline part of Auto-Prep (the upper half of Figure 2). Note that while it is an integral part of our system (and we create useful enhancements to existing transformation models), we do not consider these to be core contributions of this work. We will therefore only highlight the key intuition here for completeness, leaving more details to the technical report [1].

6.1 Transformation models M_T

Recall that in MPBP of Definition 1, we need to estimate the transformation probability $p(\mathcal{S}|\mathcal{T}) = \prod_{i \in [n]} p(S_i|T_i)$ in Equation (1). Given that each S_i is a sequence of transformations $S_i = \{O_1, O_2, O_3, \ldots\}$[2], the probability $p(S_i|T_i)$ can in turn be estimated as:

$$p(S_i|T_i) = p(O_1|T_i) \cdot p(O_2|O_1(T_i)) \cdot p(O_3|O_2(O_1((T_i))) \ldots \quad (3)$$

where each $p(O|T)$ denotes the estimated probability of transformation O given table T.

Note that the problem of estimating $p(O|T)$ has been studied in program synthesis, where transformations need to be predicted based on a table T, often using machine learning classification models [86, 92], which we write as M_T.

In Auto-Prep, we build on top of these existing models M_T, and use the same $p(O|T)$ abstraction to estimate the probability of each transformation O. Additionally, we perform enhancements to M_T, and create M^+_T, after observing that we operate on *a set* of tables $\mathcal{T}$ in the BI setting (as opposed to one individual table), which presents opportunities for leveraging global signals from all tables in $\mathcal{T}$. We leave details of these M^+_T models to our technical report [1] in the interest of space, since we do not consider them our core contributions in Auto-Prep.

We use boosted decision trees [40] to train our transformation models M^+_T, whose outputs are calibrated to true probabilities, or $p(O|T) \in [0, 1]$, using calibration techniques in machine learning [2]. This makes sure that $p(O|T)$ from different transformation operators are true probabilities comparable on the same scale.

6.2 Join models M_J

Recall that for a candidate set of transformations $\mathcal{S}$, the objective function of MPBP in Equation (1) has another term $p(J(\mathcal{S}(\mathcal{T})))$, which estimates the "goodness" (probability) of all joins, for a set of transformed tables $\mathcal{S}(\mathcal{T})$ that are induced by $\mathcal{S}$.

[2]We write $O_1(P_1), O_2(P_2)$ in the original notation of Definition 1, simply as O_1, O_2, omitting the parameters P_1, P_2 for simplicity, when the context is clear.

Since join prediction has been extensively studied in the literature [41, 51, 64, 73, 89], we use a standard abstraction for join probabilities as follows. Let M_J be a join model (e.g., a regression model), that predicts the join probability between two columns (C, C'), as $M_J(C, C') \in [0, 1]$. We then use $\tilde{p}(T, T')$ to denote the join probability between any two given tables (T, T'), defined as:

$$\tilde{p}(T, T') = \max_{C \in T, C' \in T'} M_J(C, C') \quad (4)$$

which estimates the join probability between (T, T'), as the maximum join probability between any two columns $C \in T, C' \in T'$.

We use probability calibration [2, 72] to ensure that $\tilde{p}(T, T') \in [0, 1]$, and are true probabilities, where 1 indicates full confidence that T, T' should join, 0 indicates full confidence that they should not join, while 0.5 is right on the decision boundary. Given that $\tilde{p}(T, T')$ is true probability, we use $\tilde{p}(T, T') > 0.5$ as the cutoff for whether two tables should join, similar to prior work [64].

As a final normalization step, since we want to compare the goodness of joins using between tables predicted to join (i.e., any table-pairs with $\tilde{p}(T, T') < 0.5$ would not join and therefore should not contribute to the joinability score), we use a normalized version of $\tilde{p}(T, T')$, written as $p(T, T')$:

$$p(T, T') = \max(\tilde{p}(T, T'), 0.5) \quad (5)$$

which is lower-bounded at 0.5 (the decision-boundary for joins), to normalize the effect of non-joinable table-pairs[3]. We will see how $p(T, T')$ is used as edge-weights in our graph formulation next.

7 ONLINE GLOBAL GRAPH SEARCH

We are now ready to describe the core online phase of Auto-Prep, which is our graph-based formulation and its graph algorithms.

7.1 Graph representations

In this section, we will introduce how we represent the entire search space, including transformation steps and join candidates across all tables, using a graph in a unified manner.

Transformation trees. We will start by representing all possible transformations that originate from a single input table, $T \in \mathcal{T}$, using a tree we call *transformation-tree*, denoted by $G(T)$.

Definition 2. [Transformation tree $G(T)$]. Let $T \in \mathcal{T}$ be an input table, $O \in \boldsymbol{O}$ be a predicted transformation on T, with probability $p(O|T)$. Let $O(T)$ be a transformed version of T after applying O, and $O'(O(T))$ in turn be a transformed version of $O(T)$ after applying O', etc., up to m-level deep.

We construct a weighted tree $G(T) = (V(T), E(T))$ to represent all possible transformations from T as follows. We first represent the input table T as a vertex $v(T)$, which is the root of $G(T)$.

We represent each of T's transformed descendants $O(T), \forall O \in \boldsymbol{O}$, also as a vertex, written as $v(O(T))$. We connect $v(O(T))$ and $v(T)$ with an edge $e(O)$ to represent the transformation O on T, where the edge weight $w(e(O))$ is the transformation probability $P(O|T)$. This forms the first level of nodes in $G(T)$, or $m = 1$.

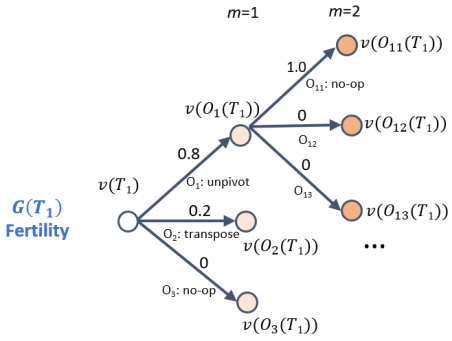


Figure 3: Transformation tree $G(T_1)$ to represent all possible transformations on the `Fertility` table T_1 (Table 1). Edge weights represent transformation probabilities (from M_T).

Similarly and recursively, each transformed descendant of $O(T)$ using transformation $O' \in \boldsymbol{O}$, written as $O'(O(T))$, is also represented as a vertex $v(O'(O(T)))$, which is connected to $v(O(T))$ with an edge, whose edge weight is set similarly as $P(O'|O(T))$. This forms additional levels of nodes in $G(T)$, for $m > 1$.

The resulting $G(T) = (V(T), E(T))$ is our *transformation tree* for one input table T, up to m levels (we use $m = 2$ in this work). □

We illustrate transformation trees $G(T)$ with an example below.

Example 4. We revisit Example 3, and focus on the "`Fertility`" table in Table 1, which we refer to as T_1. We explain how to represent all possible candidate transformations on this table T_1, using a transformation tree $G(T_1)$ shown in Figure 3.

We first represent the "`Fertility`" table T_1 as a root node $v(T_1)$ in the figure. Suppose the candidate transformations on T_1 include O_1 = `Unpivot`("2010", "2012"), O_2 = `Transpose`(), O_3 = `no-op`, etc., where O_1 is the desired transformation for T_1, to produce $O_1(T_1)$ that corresponds to the transformed table in Table 5.

We represent each transformation O_i as an edge from $v(T_1)$, weighted by transformation probability $p(O_i|T_1)$, and each table resulting from a transformation O_i as a vertex $v(O_i(T_1))$ in $G(T_1)$. This forms the first level of nodes in the tree ($m = 1$). For example, for O_1 = `Unpivot`, with predicted probability $p(O_1|T_1) = 0.8$, then the edge weight between $v(T_1)$ and $v(O_1(T_1))$ is set to $p(O_1|T_1) = 0.8$, as shown in Figure 3. The same is true for O_2, O_3, etc.

Note that each transformed table $O_i(T_1)$ may be transformed again, which are represented as a second level of edges in the transformation tree ($m = 2$). For example, we may predict O_{11}, O_{12}, O_{13} on the transformed table $O_1(T_1)$, with probability 1.0, 0, and 0, respectively, like represented in the figure. □

Global search graph. We can now connect all transformation trees $G(T_i), \forall T_i \in \mathcal{T}$, to represent the global search graph $G(\mathcal{T})$ for both transformations and joins, defined as follows.

Definition 3. [Global search graph $G(\mathcal{T})$]. Let $G(T_i) = (V(T_i), E(T_i))$ be the transformation tree for each $T_i \in \mathcal{T}$ in Definition 2. The set of transformation trees, $\{G(T_i)|T_i \in \mathcal{T}\}$, induces a *global search graph* $G(\mathcal{T}) = (V(\mathcal{T}), E(\mathcal{T}))$, where the vertex set $V(\mathcal{T}) = \bigcup_{T_i \in \mathcal{T}} V(T_i)$ is simply the union of vertices in $G(T_i)$, whereas the edge set $E(\mathcal{T}) = E_T(\mathcal{T}) \cup E_J(\mathcal{T})$ consists of two types edges, transformation-edges $E_T(\mathcal{T})$ and join-edges $E_J(\mathcal{T})$:

[3]The reason to lower-bound $p(T, T')$ at 0.5 in Equation (5), is to ensure that "bad" (non-joinable) candidates have equalized effects. For example, given two join candidates, $\tilde{p}(T, T') = 0.2$ and $\tilde{p}(T, T'') = 0.1$, the former should not contribute more to the overall objective-function than the latter (even though the score of the former is better), as neither will be selected to join in the final solution and, therefore, should be treated equally. Note that in the normalized $p(T, T')$, both will have the identical score 0.5, normalizing the score difference of such non-joinable candidates.

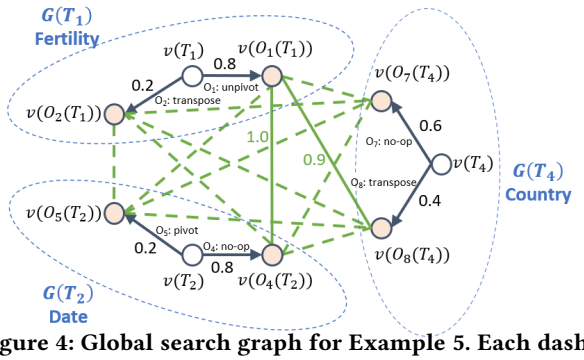


Figure 4: Global search graph for Example 5. Each dashed-circle corresponds to a transformation tree $G(T_i)$, whose black edges represent possible transformations for table T_i. Join candidates are represented as green edges between transformation trees, where joinable tables are represented by solid edges (with $w(e) > 0.5$), and non-joinable "placeholder edges" are represented as dashed green edges (with $w(e) = 0.5$).

(1) *Transformation-edges* $E_T(\mathcal{T})$, defined as $E_T(\mathcal{T}) = \bigcup_{T_i \in \mathcal{T}} E(T_i)$, is simply the union of all transformation-edges $E(T_i)$ in $G(T_i)$, where each edge $e \in E_T(\mathcal{T})$ represents a transformation, whose edge weight is its transformation probability $w(e) = p(O|T)$, based on the transformation model M_T trained offline (Section 6.1);

(2) *Join-edges* $E_J(\mathcal{T})$, written as $E_J(\mathcal{T}) = \{e(v, v') | v \in V_{\text{leaf}}(T_i), v' \in V_{\text{leaf}}(T_j), i \neq j\}$, represent all candidate joins between tables, where each edge $e(v, v') \in E_J(\mathcal{T})$ stands for a candidate join between two tables represented by leaf vertices (v, v')[4], from two different transformation trees $G(T_i)$ and $G(T_j)$. The edge weight of each join-edge $e(v, v')$ is the normalized join probability $w(e) = p(v, v')$ from the join model M_J (Equation (5) in Section 6.2). □

The pseudo-code for building global search graphs can be found in [1]. We illustrate the global search graph using an example.

Example 5. We continue with Example 4, and use Figure 4 to represent the global search graph $G(\mathcal{T})$ for the tables in Example 3. For simplicity, we show only the first level of transformations for each transformation tree $G(T_1), G(T_2), G(T_4)$, marked in dashed-circles. We omit table T_3 from Example 3 here (for T_3 requires two-level transformations that is too big to show on the figure).

Note that $G(T_1)$ in $G(\mathcal{T})$ directly corresponds to the transformation tree for T_1 in Figure 3 (Example 4), where O_1 is the desired "`Unpivot`" transformation for T_1 with probability $p(O_1|T_1) = 0.8$, O_2 is the second-ranked "`Transpose`" with probability $p(O_2|T_1) = 0.2$, etc. We omit additional transformations like O_3, and the second level of transformations like O_{11}, O_{12} from Figure 3, to simplify this illustration and avoid clutter.

We construct $G(T_2)$ and $G(T_4)$ similarly, which are shown in their respective circles. Note that all black edges in the graph are now the transformation-edges $E_T(\mathcal{T})$ in Definition 3.

Finally, we construct join-edges $E_J(\mathcal{T})$ in $G(\mathcal{T})$, between any two vertices (v, v') at the leaf-level of two different transformation trees $G(T_i)$ and $G(T_j)$, represented as green edges. The edge-weight of each (v, v') corresponds to the joinability of the two tables (v, v'), computed by invoking the join models M_J (Section 6.2).

In the figure, we use solid green edges to represent join candidates $e = (v, v')$ that are predicted as joinable, whose edge weights $w(e) = p(v, v') \geq 0.5$. For example, $(v(O_1(T_1)), v(O_4(T_2)))$ and $(v(O_1(T_1)), v(O_8(T_4)))$ are clearly joinable, with edge-weights 1.0 and 0.9, respectively. The remaining non-joinable candidates are represented as dashed green edges, which are "placeholder join edges" for non-joinable candidates, whose edge weights $w(e)$ are all set to the constant 0.5 (Equation 5).[5] □

7.2 Solve MPBP using global search graph

Using the global search graph $G(\mathcal{T})$, we now describe how the MPBP problem in Definition 1 can be solved using the new graph algorithms we design in this work.

Recall that in MPBP, given input tables $\mathcal{T}$, we need to select both (1) transformations $\mathcal{S}$, and (2) joins between the transformed tables $\mathcal{S}(\mathcal{T})$, denoted by $J(\mathcal{S}(\mathcal{T}))$, so that both the "goodness" (probability) of the transformations $p(\mathcal{S}|\mathcal{T})$, and the "goodness" (probability) of the joins $p(J(\mathcal{S}(\mathcal{T})))$, are jointly maximized in the objective function in Equation (1).

Valid solutions to MPBP on search graph $\mathbf{G}(\mathcal{T})$. We first show how a valid solution to MPBP would look like on $G(\mathcal{T})$.

First, the transformations selected in $\mathcal{S}$ for MPBP can naturally be represented as transformation-edges on $G(\mathcal{T})$. Since we are only performing exactly one sequence of transformations for each table T_i, then in each transformation tree $G(T_i)$, the selected transformations will form a unique "path", from the root of $G(T_i)$, denoted by $v(T_i)$ representing the original input table T_i, to a unique leaf-vertex in $G(T_i)$, denoted by $v(L_i) \in V_{\text{leaf}}(G(T_i))$, representing a transformed table L_i that originates from T_i. The unique path induced by $v(T_i)$ in $G(T_i)$, written as $path(v(T_i) \rightarrow v(L_i))$, naturally corresponds to the sequence of transformations selected for T_i. The union of such paths across all $G(T_i)$, is then the overall selected transformations, $\mathcal{S} = \{path(v(T_i) \rightarrow v(L_i)) | i \in [n]\}$

Similarly, the joins selected in $J(\mathcal{S}(\mathcal{T}))$ can be represented as join-edges in the search graph $G(\mathcal{T})$. Because the selected joins $J(\mathcal{S}(\mathcal{T}))$ need to be based on the selected transformations $\mathcal{S}$, the join-edges need to be between leaf vertices representing transformed tables $\{v(L_i) | i \in [n]\}$ on the search graph $G(\mathcal{T})$.

Since join relationships in BI settings often follow star/snowflake-schema [57, 64], where given n input tables, $(n-1)$ join-edges need to exist to "connect" all n tables, we therefore use the $(n-1)$ most confident join-edges that can connect all n tables, as our selected joins in $J(\mathcal{S}(\mathcal{T}))$.

A sub-graph in $G(\mathcal{T})$ with these required transformation-edges $\mathcal{S}$, and join-edges $J(\mathcal{S}(\mathcal{T}))$, is then a valid solution to MPBP, defined as follows.

Definition 4. [Valid solutions to MPBP on search graph $G(\mathcal{T})$.] Let $G(\mathcal{T})$ be the global search graph induced by transformation trees $\{G(T_i) | T_i \in \mathcal{T}\}$ in Definition 3. Let $v(L_i)$ be a unique leaf-vertex selected in transformation tree $G(T_i)$, for all $i \in [n]$.

A *valid solution to MPBP* on $G(\mathcal{T})$ is an edge set $E = E_T \cup E_J$, where $E_T = \{e | e \in path(v(T_i) \rightarrow v(L_i)), i \in [n]\}$ is the union of transformation-edges on the paths between root $v(T_i)$ and leaf $v(L_i), \forall i \in [n]$, and $E_J = \{(v(L_i), v(L_j))\}, |E_J| = (n-1)$, are $(n-1)$ join-edges between vertices in $\mathcal{L} = \{v(L_i) | i \in [n]\}$, that form a spanning tree to connect n transformed tables represented by $\mathcal{L}$. □

[4]Note that "`no-op`" is also a predicted transformation, therefore leaf vertices can naturally represent both transformed tables, as well as un-transformed original tables.

[5]This normalizes scores for all non-joining edges (since none of them will be selected in the final solution), to equalize their effect on the objective function (Section 6.2).

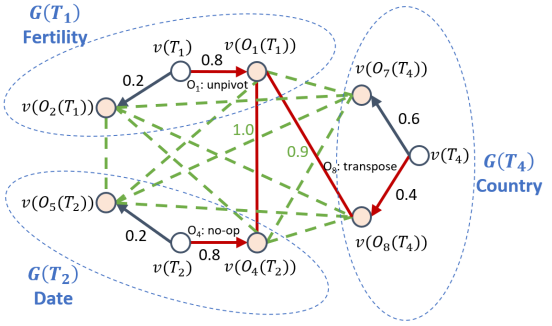


Figure 5: An optimal solution for the search graph in Figure 4, that corresponds to the solution $\mathcal{S}^*$ in Example 3.

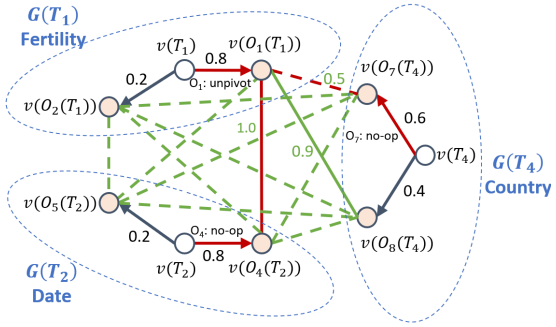


Figure 6: An inferior solution for the search graph in Figure 4, that corresponds to the solution $\mathcal{S}^-$ in Example 3.

We use an example to show valid solutions on search graphs.

Example 6. Figure 5 and Figure 6 show two valid solutions to MPBP, on the search graph $G(\mathcal{T})$ from Example 5. The selected transformation-edges and join-edges are all marked in red.

In Figure 5, the selected transformation-edges are: O_1 (unpivot) on T_1, O_4 (no-op) on T_2, and O_8 (transpose) on T_4, which are the same set of desired transformations in Example 3. Between the three transformed tables $v(O_1(T_1))$, $v(O_4(T_2))$ and $v(O_8(T_4))$, two join-edges, $(v(O_1(T_1)), v(O_4(T_2)))$ and $(v(O_1(T_1)), v(O_8(T_4)))$, are marked in red, indicating that they are selected to join (which would correspond to the yellow and green join path on transformed tables in Table 5-Table 8, respectively).

Intuitively, we can see that the red edges correspond to a valid solution to MPBP, because in each transformation tree $G(T_1)$, $G(T_2)$ and $G(T_4)$, we select one and exactly one transformation path, from the root of the transformation tree, to a unique transformed table (leaf vertex), which are $v(O_1(T_1))$, $v(O_4(T_2))$ and $v(O_8(T_4))$, respectively. Furthermore, we select exactly 2 join-edges to form a spanning tree that connects $v(O_1(T_1))$, $v(O_4(T_2))$ and $v(O_8(T_4))$.

In Figure 6, we show another valid solution, that selects a different set of transformations in red: O_1 (unpivot) for T_1, O_4 (no-op) for T_2, and O_7 (no-op) for T_4. The selected 2 join-edges are similarly marked in red. This is still a valid solution, because a unique transformation path is selected in each transformation tree $G(T_i)$, and 2 join-edges are selected to span all transformed tables. □

Goodness of solutions on search graph $\mathcal{G}(\mathcal{T})$. Given that there exist many valid solutions (exponential in the number of tables), to evaluate the relative merits of these solutions, we resort to the objective function of MPBP in Equation (1), which has two components, $p(\mathcal{S}|\mathcal{T})$ for the goodness of transformations, and $p(J(\mathcal{S}(\mathcal{T})))$ for the goodness of joins.

The first term for transformations, $p(\mathcal{S}|\mathcal{T})$, can be expanded as $\prod_{S_i \in \mathcal{S}} p(S_i|T_i)$, where each $S_i = (O_{i1}, O_{i2}, \ldots)$ is the sequence of transformations selected for table T_i. This can further be expanded in terms of individual transformations O_{ij} as:

$$p(\mathcal{S}|\mathcal{T}) = \prod_{S_i \in \mathcal{S}} p(S_i|T_i) = \prod_{O_{ij} \in S_i, S_i \in \mathcal{S}} p(O_{ij}) \tag{6}$$

On the search graph $G(\mathcal{T})$, the term in Equation (6) is effectively the cross-product of probabilities (edge weights) of all selected transformation $p(O_{ij})$ in $\mathcal{S}$, which naturally measures the collective likelihood of all selected transformations in $\mathcal{S}$.

Given a valid solution $E = E_T \cup E_J$ on $G(\mathcal{T})$ in Definition 4, where E_T is the selected transformation-edges, it can be written as:

$$p(\mathcal{S}|\mathcal{T}) = \prod_{e \in E_T} w(e) \tag{7}$$

Similarly, the goodness of joins in $p(J(\mathcal{S}(\mathcal{T})))$, is also the cross-product of join-probability of all selected join-edges in E_J:

$$p(J(\mathcal{S}(\mathcal{T}))) = \prod_{e \in E_J} w(e) \tag{8}$$

The overall objective function $p(\mathcal{S}|\mathcal{T}) \cdot p(J(\mathcal{S}(\mathcal{T})))$ in Equation (1), can now be equivalently written as the product of Equation (7) and (8), where the optimal edge-set E^* on $G(\mathcal{T})$ is:

$$E^* = \arg\max_{E=E_T \cup E_J} \prod_{e \in E_T} w(e) \prod_{e \in E_J} w(e) \tag{9}$$

We can now define the search problem on graph $G(\mathcal{T})$, that is the graph-equivalent of MPBP as follows.

Definition 5. [Most-Probable BI-Prep on Graph (MPBP-G)]. Let $G(\mathcal{T})$ be the global search graph induced by transformation trees $\{G(T_i) \,|T_i \in \mathcal{T}\}$. The *MPBP on Graph (MPBP-G)* problem, is to select a set of edges $E = E_T \cup E_J$ from $G(\mathcal{T})$, such that E is a valid solution on $G(\mathcal{T})$ as defined in Definition 4, while the objective function in Equation (9) is maximized. □

Example 7. We revisit Figure 5 and Figure 6 from Example 6 to illustrate Definition 5. The set of edges selected in Figure 5, denoted by E^*, corresponds to the optimal solution $\mathcal{S}^*$ in Example 3. There are 3 transformation-edges in E^*, O_1 (unpivot) for T_1, O_4 (no-op) for T_2, and O_8 (transpose) for T_4, with probability 0.8, 0.8, and 0.4, respectively, whose collective transformation probability is then $(0.8)^2 \cdot 0.4 = 0.256$ (Equation (7)). It can be shown that in order to connect the transformed tables $v(O_1(T_1))$, $v(O_4(T_2))$ and $v(O_8(T_4))$ and make it a valid solution, the best join-edges are $(v(O_1(T_1)), v(O_4(T_2)))$ and $(v(O_1(T_1)), v(O_8(T_4)))$ like indicated in Figure 5, whose overall join probability is $1.0 \cdot 0.9 = 0.9$ (Equation (8)). The overall objective function in Equation (9) is then $0.256 \cdot 0.9 = 0.23$.

In comparison, the set of edges selected in Figure 6, denoted by E^-, has transformation-edges O_1, O_4, and O_7, with probability 0.8, 0.8, and 0.6, respectively. The overall transformation probability is then $(0.8)^2 \cdot 0.6 = 0.384$. To connect the transformed tables $v(O_1(T_1))$, $v(O_4(T_2))$ and $v(O_7(T_4))$ using join-edges and make it a valid solution, the best possible join-edges are $(v(O_1(T_1)), v(O_4(T_2)))$ and $(v(O_1(T_1)), v(O_7(T_4)))$, whose overall join probability is $1.0 \cdot 0.5 = 0.5$, leading to an overall score of $0.384 \cdot 0.5 = 0.19$, making this E^- inferior to the solution E^* in Figure 5.

Note that if we were to only consider transformations, E^- would have a better transformation probability (0.384) than E^* (0.256), which however is an inferior solution when both transformations and joins are considered, like shown above. This demonstrates the importance of holistic optimization in Auto-Prep. □

Theorem 1. The MPBP-G problem in Definition 5 is NP-complete.

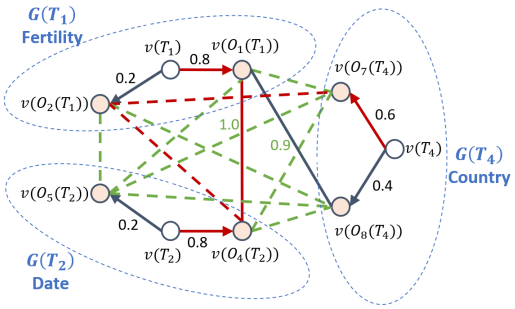


Figure 7: An invalid solution E^x to MPBP-G, that is nevertheless a valid Steiner tree w.r.t terminals $R = \{v(T_i)|i \in [n]\}$.

We show the hardness of MPBP-G using a reduction from Exact Cover by 3-Sets (X3C) [55], a proof of which can be found in [1].

Intuitively, we can also see that finding the optimal edges $E = E_T \cup E_J$ is hard — given m possible multi-step transformations (easily in the thousands when each transformation can be parameterized differently), and n input tables, for E_T alone there are m^n options to choose from.[6] The selection of optimal join-edges E_J on top of E_T further compounds the complexity, and makes the problem hard.

7.3 Solving MPBP-G leveraging Steiner-tree

We now describe our graph algorithm for MPBP-G, that can solve MPBP-G both efficiently and with provable quality guarantees.

Observing that the cross-products of probabilities (edge weights) in Equation (9) are not amenable to graph optimization, our first step is to apply logarithmic transformation on edge weights $w(e)$:

$$\overline{w}(e) = -\log(w(e)) \tag{10}$$

This allows us to rewrite the objective function in Equation (9) as:

$$E^* = \arg\max_{E=E_T \cup E_J} \prod_{e \in E_T} w(e) \prod_{e \in E_J} w(e) \tag{11}$$

$$= \arg\min_{E=E_T \cup E_J} \sum_{e \in E_T} \overline{w}(e) + \sum_{e \in E_J} \overline{w}(e) \tag{12}$$

Using the new edge weights $\overline{w}(e)$, we can instead minimize the *sum* of edge weights in the selected edge-set E, which is amenable to graph-based optimizations. More specifically, our MPBP-G problem in Definition 5, with the objective function in Equation (12), now draws a close parallel to the *Minimum Steiner Tree (MST)* problem [55, 58] in graph theory.

Recall that given a weighted graph $G = (V, E)$, and a subset of vertices $R \subseteq V$ known as the terminals, a *valid Steiner tree* is a tree in G that spans (or contains) all vertices in R. A *minimum Steiner tree* is the valid Steiner tree with minimal total edge weights [55, 58].

In the MPBP-G problem, if we let the union of all root vertices $v(T_i)$ in transformation trees $G(T_i)$ be the terminals R in the Steiner-tree problem, or $R = \{v(T_i)|i \in [n]\}$, we prove that the following connections between Steiner tree and MPBP-G hold:

Proposition 1. A valid solution to MPBP-G in Definition 4 must be a valid Steiner tree on the same graph $G(\mathcal{T})$, that spans the terminals defined by the root vertices of all transformation trees, $R = \{v(T_i)|i \in [n]\}$. □

A proof of Proposition 1 can be found in [1]. Intuitively, to see why a valid solution to MPBP-G must be a valid Steiner tree for the given R, observe that for a valid solution to MPBP-G in Definition 4, $E = E_T \cup E_J$, because E_T would connect each root $v(T_i)$ with a leaf-vertex $v(L_i)$ via a unique path, and the union of the leaf vertices $v(L_i)$, written as $\mathcal{L} = \{v(L_i)|i \in [n]\}$, would in turn be connected by $(n-1)$ edges in E_J (which is the minimum number of edges to connect $\mathcal{L}$). The resulting $E = E_T \cup E_J$ therefore connects all terminals in R and is a valid Steiner tree.

Note that in Proposition 1, we only state $E = E_T \cup E_J$ being a valid tree, with no statement about the minimality of such a tree.

Example 8. Figure 5 and 6 show two valid solutions to MPBP-G (red edges). Both can be verified as valid Steiner trees, as the selected edges form trees that connect all terminal vertices in $R = \{v(T_i)|i \in [n]\}$, or the root vertices of all transformation trees. □

The reverse of Proposition 1, however, is not always true, which means that not all valid Steiner trees on $G(\mathcal{T})$ are valid solutions to MPBP-G in Definition 4, which we illustrate using an example.

Example 9. Figure 7 shows an example, where selected edges marked in red can be verified as a valid Steiner tree, since the red edges form a tree that connects all $R = \{v(T_1), v(T_2), v(T_4)\}$. However, it is not a valid solution to MPBP-G in Definition 4, because the vertex $v(O_2(T_1))$ is incident on selected join-edges (red dashed lines in the figure), effectively used to join, but does not have corresponding transformation-edge (namely, $v(T_1) \rightarrow v(O_2(T_1))$), to create this transformed table first, which therefore is not a valid solution to MPBP-G (Definition 4). □

What this means is that we could not hope to solve MPBP-G by directly invoking Minimum Steiner Tree (MST) on $G(\mathcal{T})$, because solutions to MST may not be valid solutions to MPBP-G. However, we prove the following two key properties of MPBP-G, which will allow us to construct principled algorithms to solve MPBP-G.

Proposition 2. Any valid solution to MPBP-G in Definition 4 has exactly $(m \cdot n)$ transformation-edges and $(n-1)$ join edges, where m is the depth of transformation trees (Definition 2), and n is the number of input tables in MPBP-G. □

A proof of Proposition 2 can be found in [1]. To see why it holds, recall that valid solutions to MPBP-G requires a unique transformation-path (with exactly m edges) for each transformation tree, which translates to $(m \cdot n)$ transformation-edges for n transformation trees. Since a valid solution to MPBP-G also requires $(n-1)$ join edges to connect n input tables, Proposition 2 then follows.

Example 10. As intuitive examples for Proposition 2, we revisit the two valid solutions to MPBP-G in Figure 5 and 6. Given $n = 3$ input tables, and tree-depth set to $m = 1$, both solutions have exactly $(m \cdot n) = 3$ transformation edges, and $(n-1) = 2$ join edges. □

Proposition 3. A valid Steiner tree on the search graph $G(\mathcal{T})$ of MPBP-G that connects all terminals $R = \{v(T_i)|i \in [n]\}$, has at least $(m \cdot n) + (n-1)$ edges. Furthermore, any valid Steiner tree with exactly $(m \cdot n) + (n-1)$ edges is a valid solution to MPBP-G. □

A proof of this can be found in [1], where the key idea is that for a valid Steiner tree to connect all terminals in $R = \{v(T_i)|i \in [n]\}$, at least one path of length m from the root $v(T_i)$ to a leaf vertex $v(L_i)$ in each transformation tree $G(T_i)$ is required, which together

[6] Note that figures like Figure 4 are simplified for illustration, which show only 1-step transformations, and without considering parameter options for the same operator O.

with at least $(n-1)$ join edges to connect n transformation trees, leads to at least $(m \cdot n) + (n-1)$ edges.

Example 11. Continue with Example 10, where $n = 3$ and $m = 1$, we can see that both trees marked in red in Figure 5 and 6 are valid Steiner trees with 5 edges, while the tree in Figure 7 is also a valid Steiner tree with 6 edges. All three cases indeed have at least $(m \cdot n) + (n-1) = (1 \cdot 3) + (3-1) = 5$ edges. Furthermore, both valid Steiner trees with exactly $(m \cdot n) + (n-1) = 5$ edges in Figure 5 and 6, are indeed valid solutions to MPBP-G (Example 6). □

Using Proposition 1-3, we now describe our Algorithm 1, which builds on top of MST that can provably solve MPBP-G.

Algorithm 1: Solve MPBP-G on global search graph $G(\mathcal{T})$

Input: $G(\mathcal{T})$ (global search graph)
Output: Edge set E in $G(\mathcal{T})$ (solution to MPBP-G)

1 **foreach** $G(T_i) \in G(\mathcal{T})$ **do**
2 $\quad v(L_i) = \arg\min_{v(L_i) \in V_{leaf}(T_i)} \left(\sum_{e \in path(v(T_i), v(L_i)))} \overline{w}(e) \right)$
3 $\quad \overline{w}(L_i) = \sum_{e \in \text{path}(v(T_i), v(L_i)))} \overline{w}(e)$
4 $S_b = \bigcup_{i \in [n]} \{e | e \in path(v(T_i), v(L_i)))\}$
5 $S_b = S_b \bigcup \{e | e \in \text{spanning-tree}(\{v(L_i) | i \in [n]\})\}$
6 $p = \sum_{i \in [n]} \overline{w}(L_i) + (n-1)(-\log(0.5))$
7 **foreach** $e \in G(\mathcal{T})$ **do**
8 $\quad$ update e edge weight as $\overline{w}(e) + 2p$
9 $S_s \leftarrow$ solve MST on $G(\mathcal{T})$ `// using Kou's algorithm [58]`
10 $W_s = \sum_{e \in S_s} \overline{w}(e) + 2p \cdot |S_s|$
11 $W_b = p + 2p \cdot |S_b|$
12 return the edge set S_s if $W_s < W_b$, otherwise S_b;

In Algorithm 1, we start by iterating through each transformation tree $G(T_i)$, where in each $G(T_i)$, we select the leaf vertex $v(L_i)$ with the best transformation probabilities (sum of edge weights) on the path between root $v(T_i)$ and $v(L_i)$ (Line 2). Note that since we are using negative log-probabilities $\overline{w}(e)$ as edge weights (Equation (10)), where smaller is better, $v(L_i)$ is selected using arg min. Assign $\overline{w}(L_i)$ as the sum of the edge weights on each best path (Line 3).

We know at this point that there is one valid solution S_b to MPBP-G, that simply selects all transformation-edges on each $path(v(T_i), v(L_i))$ (Line 4), as well as any spanning-tree that connects $\{v(L_i) | i \in [n]\}$ (Line 5), whose objective function is at least as good as $p = \sum_{i \in [n]} \overline{w}(L_i) + (n-1)(-\log(0.5))$ (Line 6), where the second term comes from the fact that the join score of S_b from any spanning-tree will be no worse than $(n-1)(-\log(0.5))$ (since each non-joinable "placeholder join edge" has a worst-case score of $-\log(0.5)$).

Next, we add $2p$ as an edge-weight "penalty term" to every edge $e \in G(\mathcal{T})$ (Line 8). We then solve the minimum Steiner tree (MST) problem on the resulting graph $G(\mathcal{T})$ using the classical Kou's algorithm [58] to find a Steiner tree S_s (Line 9), whose total edge-weight is W_s (Line 10). We also calculate the total edge-weight of W_b for S_b including the $2p$ penalty term (Line 11). Finally, we return the better of S_s and S_b, based on their cost W_s and W_b, as our solution.

We prove that Algorithm 1 not only finds a valid solution to MPBP-G, but also has the following approximation guarantee.

Theorem 2. Algorithm 1 produces a valid solution to MPBP-G in polynomial time. Furthermore, this solution is within a $(2 - (2/n))$ factor of the optimal, where $n \geq 2$ is the number of input tables.

A proof of Theorem 2 can be found in [1], which follows from Proposition 1-3. We prove the approximation ratio and polynomial complexity, using arguments from Kou's algorithm [58].

Optimistic vs. Precise Mode of Auto-Prep. So far we focused on formulating and solving Auto-Prep as a graph search problem, assuming that the edge weights (transformation probabilities) are fixed. In practice, since we use global features across multiple tables to estimate transformation probabilities (Section 6.1), the probability of a transformation (edge-weight) can depend on whether certain transformations are actually selected on other tables, which can therefore evolve. We design two variants of Auto-Prep to handle this, one "optimisitc" and one "precise".

In the "optimistic" mode of Auto-Prep, for each table T, we take an optimistic view and use the best possible feature configuration from other tables $\mathcal{T} \setminus T$ to estimate its transformation probabilities, which become *fixed* edge-weights on the search graph $G(\mathcal{T})$, on which we can then directly invoke Algorithm 1 once to solve.

In the "precise" mode of Auto-Prep, we iteratively invoke Algorithm 1, where in each iteration we use the bounds from the best solutions found so far, to prune away infeasible graph edges (transformations) and update edge weights, so that Algorithm 1 can be invoked on the updated graph again until convergence. This tends to yield better solutions but is more expensive. We give the pseudo-code of the precise variant in [1] in the interest of space.

8 EXPERIMENTAL EVALUATION

8.1 Evaluation Setup

8.1.1 Benchmark.

We sampled 1837 real BI projects for evaluation, where 510 projects are set aside as test cases that are never looked at, and the rest are used as training data to calibrate our offline classification models (Section 6). We extract user-programmed transformations and joins from the real BI projects as ground truth, which algorithms will need to predict correctly (Section 3.1). The distribution of transformations in our dataset is the following: unpivot: 28.2%; pivot: 3.3%; transpose: 6.7%; string-transform: 5.1%; no-op: 56.7%.

8.1.2 Metrics.

In our problem, since both transformations and joins need to be predicted, we evaluate quality for both transformations and joins.

Transformation quality. We compare the predicted transformations of different methods against the ground truth (user-programmed transformations), where both the operator, parameters, and order of the transformations (Table 9) need to be predicted exactly correctly (we do not assign partial credit for inexact matches). We use the standard precision, recall, and F1-score metrics, where precision $P_{\text{transform}} = \frac{\text{num-of-correct-predicted-transforms}}{\text{num-of-total-predicted-transforms}}$, recall $R_{\text{transform}} = \frac{\text{num-of-correct-predicted-transforms}}{\text{num-of-total-true-transforms}}$, and F1-score $F_{\text{transform}} = \frac{2P_{\text{transform}}R_{\text{transform}}}{P_{\text{transform}}+R_{\text{transform}}}$ is the harmonic mean of precision and recall.

Join quality. We compare the predicted joins against the ground truth joins programmed by users. Precision P_{join}, recall R_{join}, and F1-score F_{join} are defined similarly.

8.1.3 Methods compared.

While we propose to holistically predict both transformations and joins, existing methods predominately predict either only transformations, or only joins (reviewed in Section 2). As a result, for this evaluation, we test against 3 classes of baselines:

(1) Transformation-only methods: We compare with the following transformation-only methods, for transformation quality. Our goal here is to show that considering only transformation (without considering joins holistically), misses the opportunity to predict accurately given the intertwined nature of the two.

- **Single-Operator-Predict (SOP) [86]**. This method uses machine learning models to predict single transformation operators, based on the characteristics of input tables. This is the basis of our transformation-classifiers (Section 6) used in Auto-Prep. We refer to this as Single-Operator-Predict (SOP) to differentiate with the next baseline below that considers multiple operators.
- **Multi-Operator-Predict (MOP) [62]**. This is a state-of-the art method that predicts transformations from multiple operators, with a focus on reshaping tables. We refer to this as Multi-Operator-Predict (MOP).
- **GPT-transformation (GPT-t)**. Since language models like GPT have shown diverse capabilities, including program synthesis [37], we use a recent version of GPT-4[7] [71] and in-context-few-shot learning, as another baseline for transformation predictions. We extensively test combinations of different prompt strategies, including few-shot examples, chain-of-thought [84], and dynamic RAG to retrieve most relevant examples [60], in order to optimize the quality of the GPT-based method.
- **Fine-tuned-GPT-transformation (FT-GPT-t)**. In addition to invoking vanilla GPT-4, we also used the OpenAI API [7] to fine-tune GPT-4 models, with the same data available to Auto-Prep, in an attempt to optimize prediction quality. We fine-tuned GPT both only for transform (FT-GPT-t), and also end-to-end for both transform and join (referred to as FT-GPT).

(2) Join-only methods: We compare with 2 representative methods from the literature for join predictions.

- **BI-join (BI-j) [64]**. BI-Join is a recent method designed to predict joins in BI settings, and has been shown to be competitive on a range of join benchmarks [64].
- **GPT-join (GPT-j)**. Since GPT is capable of understanding tables [63, 68], we again use GPT-4 and few-shot learning, as another baseline for join predictions. We refer to this as GPT-j.

(3) Transformation + join methods: Given the 4 transformation-only and 2 join-only baselines mentioned above, we further compare with a third class of baselines, which sequentially invoke the 4 transformation-only methods, followed by the 2 join-only methods, for a total of $4 \times 2 = 8$ methods (which are **SOP + BI-j**, **MOP + BI-j**, **GPT-t + BI-j**, **FT-GPT-t + BI-j**, **SOP + GPT-j**, **MOP + GPT-j**, **GPT-t + GPT-j**, and **FT-GPT-t + GPT-j**). These methods are directly comparable to Auto-Prep (except that Auto-Prep holistically optimizes across transforms/joins, whereas these baselines simply invoke join predictions after transformations sequentially).

We compare all baselines above, with two variants of *our method*:

- **Auto-Prep-Optimistic (AP-O)**. This is a version of our algorithm that directly invokes Algorithm 1 once to solve MPBP-G, based on optimistic estimates of transformation probabilities.
- **Auto-Prep-Precise (AP-P)**. This is the precise variant of our algorithm that iteratively invokes Algorithm 1 based on precise estimates of probabilities, which tends to give better solutions, but is more expensive than AP-O. We set the search depth m as 2 consistently for both AP methods across all experiments.

Table 10: Transformation quality comparison.

Method	AP-P	AP-O	SOP	MOP	GPT-t	FT-GPT-t
$P_{\text{transform}}$	**0.785**	0.757	0.231	0.339	0.411	0.296
$R_{\text{transform}}$	**0.741**	0.740	0.644	0.571	0.646	0.258
$F_{\text{transform}}$	**0.762**	0.748	0.340	0.426	0.502	0.275

8.2 Experimental results

8.2.1 Quality Comparison: Transformation.

Table 10 shows a comparison of transformation quality between Auto-Prep (AP) and the existing transformation methods. Overall, both AP-P and AP-O significantly outperform existing transformation baselines[8], correctly predicting more than 70% transformations, with AP-P performing noticeably better than AP-O, as it uses more precise estimates of probabilities via iterative graph computation.

Recall that our task of transformation prediction has a large search space (with millions of possible operator and parameter combinations), where a random guess has an exceedingly small chance (less than 1%) of being correct, such that GPT-t based on GPT-4 also produces low precision.

Specialized algorithms like SOP and MOP are more competitive, but are still substantially less accurate than AP, which holistically optimize transformations/joins in a principled manner.

Note that for GPT-t, we extensively experimented with 6 prompting strategies to optimize the prompt for GPT, including using few-shot vs. zero-shot, using COT [84] vs. no-COT, and the use of RAG [60] vs. no-RAG. We report the best prompt for GPT-t, which uses few-shot with COT, where few-shot examples provide demonstrations, with COT further enhancing structured reasoning. Desite extensive optimizations, GPT-t still lags behind especially in terms of precision. Details of our results in testing different prompt strategies can be found in [1] in the interest of space.

8.2.2 Quality Comparison: Join.

Table 11 shows a comparison of the join quality, where we compare AP-P and AP-O, against both join-only baselines (BI-j and GPT-j), and transform+join baselines (6 methods). Overall, both AP-P and AP-O considerably outperform all baselines, showing the benefit of our holistic graph optimizations.

Among join-only baselines, the specialized BI-j is substantially better than GPT-j, which is not surprising as it directly leverages numeric features tailored to the join problem (e.g., column overlap in Jaccard similarity and containment).

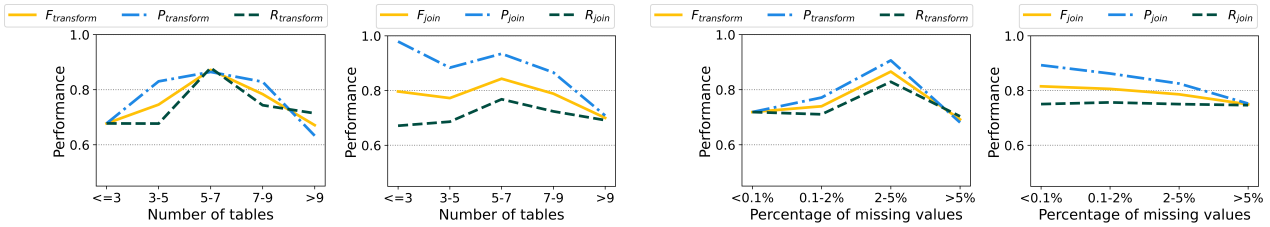
The transformation + join baselines (e.g., SOP+BI-j and MOP+BI-j) outperform join-only methods (BI-j) in join quality, underlining the need to predict transformations together with joins. However, these baselines are still inferior to AP methods, as simply invoking transformation-predictions followed by join-predictions is still sub-optimal to principled graph optimizations.

[7] GPT-4-0613, accessed from Azure OpenAI in March 2024.

[8] Note that we omit "transformation+join" baselines here, since their transformation-predictions are identical to those of transformation-only methods (for in these baselines we always invoke transformation-predictions before join-predictions).

Table 11: Join quality comparison.

Method	Auto-Prep		Join-only		Transform + Join								End-to-end
	AP-P	AP-O	BI-j	GPT-j	SOP+BI-j	MOP+BI-j	GPT-t+BI-j	FT-GPT-t+BI-j	SOP+GPT-j	MOP+GPT-j	GPT-t+GPT-j	FT-GPT-t+GPT-j	FT-GPT
P_{join}	0.806	0.806	0.737	0.274	0.767	0.769	0.762	**0.853**	0.222	0.244	0.221	0.270	0.518
R_{join}	**0.734**	0.714	0.671	0.350	0.631	0.600	0.626	0.588	0.271	0.299	0.225	0.311	0.470
F_{join}	**0.769**	0.758	0.702	0.307	0.705	0.674	0.687	0.696	0.244	0.269	0.223	0.289	0.493



(a) Transformation quality. (b) Join quality. (c) Transformation quality. (d) Join quality.

Figure 8: Sensitivity analysis. (a, b): Varying # of tables per BI project. (c, d): Varying % of missing values per BI project.

Table 12: Sensitivity of quality to varying m (tree depth).

Depth	1	2	3	Depth	1	2	3
$P_{\text{transform}}$	0.788	0.785	0.787	P_{join}	0.805	0.806	0.804
$R_{\text{transform}}$	0.742	0.741	0.734	R_{join}	0.714	0.734	0.712
$F_{\text{transform}}$	0.765	0.762	0.760	F_{join}	0.756	0.769	0.756

Table 13: Ablation studies: Transformation quality.

	Full (AP-P)	No holistic graph search	No classifier	No calibration
$P_{\text{transform}}$	**0.785**	0.645	0.146	0.736
$R_{\text{transform}}$	**0.741**	0.707	0.697	0.717
$F_{\text{transform}}$	**0.762**	0.674	0.242	0.727

8.2.3 *Sensitivity Analysis.*

We perform sensitivity analysis to understand how the performance of Auto-Prep would change with different parameters.

Sensitivity to the number of tables. BI projects come with varying numbers of input tables, often reflecting their inherent complexity for data preparation. Figure 8a shows the transformation prediction quality, on BI projects with different numbers of input tables. As the number of tables increases, the transformation quality improves initially (likely because having more tables in a project provides more global signals for transformation predictions), which then decreases on challenging projects with more than 9 tables. Figure 8b shows a similar analysis for join quality, which reveals a more consistent trend as we vary the number of input tables.

Sensitivity to the amount of missing values. Another factor that can influence prediction quality is the amount of missing/empty cells in tables, so we group BI projects based on the average percentage of missing cells in tables, and analyze the corresponding prediction quality. In Figure 8d, we see that the join quality slips slightly as the fraction of missing values increases, while in Figure 8c, we observe that the transformation quality can go up slightly with a moderate amount of missing values, which decreases when the amount of missing values increases.

Sensitivity to the depth of search tree m. Recall that in Section 7.1, we use a parameter m to control the depth of the search graph, which is a hyper-parameter to determine the complexity of the transformation sequences we search in the algorithm. In our main experiments m is always set to 2, in Table 12 we vary the search depth m from 1 to 3, and report the resulting quality. As can be seen from the table, our join and transformation quality do not change significantly with different m, which is desirable.

8.2.4 *Ablation Studies.*

We perform ablation studies to understand the importance of various components, which are reported in Table 13 and 14.

No holistic graph search. To show the benefit of our graph-based optimization (Section 7), we replace our principled graph-search algorithm with a greedy heuristic, which greedily picks candidates based on the highest transformation/join probabilities. We observe that both transformation and join quality drop noticeably.

Table 14: Ablation studies: Join quality.

	Full (AP-P)	No holistic graph search	No classifier	No calibration
P_{join}	**0.806**	0.801	0.746	0.798
R_{join}	**0.734**	0.692	0.416	0.703
F_{join}	**0.769**	0.742	0.534	0.748

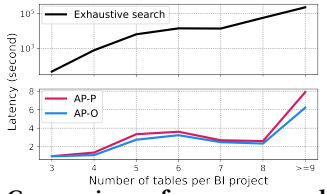


Figure 9: Comparison of average search latency.

No classifiers. To understand the importance of our classification models, we replace our M_T^+ models (Section 6) using M_T models without global-level features and calibrated scores, leading to a substantial decrease in prediction quality.

No calibration. To study the benefit of calibrating probability scores from classifiers (Section 6), we remove the calibration step, and observe a small drop in result quality.

8.2.5 *Efficiency Comparison.*

We perform an efficiency comparison to understand the search and end-to-end latency of the system.

Search latency. Figure 9 shows the average latency of our graph algorithm based on Steiner-tree, and exhaustive search, bucketized by the number of input tables in each BI project (x-axis).

The average latency of our search is less than 4 seconds with less than 8 tables in a BI project, which grows to 8 seconds on large BI projects with over 9 tables. In comparison, exhaustive search takes over an hour on BI projects with 5 tables (since the search space grows exponentially in the number of input tables), showing the importance of our principled graph search.

Additional results. Additional experimental results, such as additional quality results, sensitivity analysis, and end-to-end latency, can be found in [1] in the interest of space.

9 CONCLUSIONS

Observing the need to predict both transformations and joins holistically in real BI projects, we propose a new BI-prep problem that has been overlooked thus far, and develop new graph algorithms for holistic optimizations. Future directions include exploring holistic predictions in scenarios beyond BI (e.g., in ML and ETL workflows), and further improving predictions on large and complex BI projects.

REFERENCES

[1] [n.d.]. Auto-Prep: full technical report. https://arxiv.org/abs/2504.11627.

[2] [n.d.]. Calibration of probabilities in sklearn. https://scikit-learn.org/stable/modules/calibration.html.

[3] [n.d.]. Create and manage join relationships in Power BI. https://learn.microsoft.com/en-us/power-bi/transform-model/desktop-create-and-manage-relationships.

[4] [n.d.]. Gartner: the Future of Self-Service Is Customer-Led Automation. https://www.gartner.com/en/newsroom/press-releases/2019-05-28-gartner-says-the-future-of-self-service-is-customer-l. Accessed: 2024-05-11.

[5] [n.d.]. IDC: U.S. Business Intelligence and Analytics Platforms.

[6] [n.d.]. Model relationships in Power BI. https://learn.microsoft.com/en-us/power-bi/transform-model/desktop-relationships-understand.

[7] [n.d.]. OpenAI Fine-tuning. https://platform.openai.com/docs/guides/fine-tuning. Accessed: 2025-01.

[8] [n.d.]. Pivot in Power BI. https://learn.microsoft.com/en-us/power-query/pivot-columns.

[9] [n.d.]. Pivot in Python Pandas. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html.

[10] [n.d.]. Pivot in R. https://tidyr.tidyverse.org/articles/pivot.html.

[11] [n.d.]. Pivot in Tableau. https://help.tableau.com/current/pro/desktop/en-us/pivot.htm.

[12] [n.d.]. Power BI forum question: Help with transform data. https://community.fabric.microsoft.com/t5/Power-Query/Help-with-transforming-data/m-p/3699752.

[13] [n.d.]. Power BI forum question: How to Transforming data as indicated. https://community.fabric.microsoft.com/t5/Power-Query/Transforming-data-as-indicated/m-p/3794251.

[14] [n.d.]. Power BI forum question: Join after transform. https://community.fabric.microsoft.com/t5/Power-Query/Join-all-sheets-in-one-excel-after-transforming-them/m-p/3155786.

[15] [n.d.]. Power BI forum question: over 10000 questions for a search on "Transform data". https://community.fabric.microsoft.com/t5/forums/searchpage/tab/message?filter=location&q=transformdata.

[16] [n.d.]. Power BI forum question: Tables Join after transform. https://community.fabric.microsoft.com/t5/Desktop/Tables-Join/m-p/631121.

[17] [n.d.]. Python concatenate. https://pandas.pydata.org/docs/reference/api/pandas.concat.html.

[18] [n.d.]. Python split. https://pandas.pydata.org/docs/reference/api/pandas.Series.str.split.html.

[19] [n.d.]. Python substring. https://pandas.pydata.org/docs/reference/api/pandas.Series.str.slice.html.

[20] [n.d.]. Relationships: Asking questions across multiple related tables. https://www.tableau.com/blog/relationships-asking-questions-across-multiple-related-tables.

[21] [n.d.]. Tableau forum question: How to pivot or transform. https://community.tableau.com/s/question/0D58b0000CFihDyCQJ/how-to-pivot-or-transform-in-tableau.

[22] [n.d.]. Tableau forum question: How to transform this table. https://community.tableau.com/s/question/0D54T00000C6BsmSAF/how-to-transform-data.

[23] [n.d.]. Tableau Forum question: over 3000 questions for a search on "Transform". https://community.tableau.com/s/global-search/@uri#q=transform&t=All.

[24] [n.d.]. Tableau forum question: Transform table and join. https://community.tableau.com/s/question/0D54T00000C5tE4SAJ/transform-table-and-join.

[25] [n.d.]. Tableau forum question: Transform table and join. https://community.tableau.com/s/question/0D58b0000ADNKzZCQX/im-trying-to-find-a-way-to-define-a-relationship-between-two-excel-sheets-where-the-common-field-may-extend-across-multiple-fields.

[26] [n.d.]. Transpose in Python Pandas. https://pandas.pydata.org/docs/reference/api/pandas.transpose.html.

[27] [n.d.]. Trifacta: Standardize Using Patterns. (Retrieved in 07/2023). https://docs.trifacta.com/display/DP/Standardize+Using+Patterns.

[28] [n.d.]. Unpivot in Power BI. https://support.microsoft.com/en-us/office/unpivot-columns-power-query-0f7bad4b-9ea1-49c1-9d95-f588221c7098.

[29] [n.d.]. Unpivot in Python Pandas (melt). https://pandas.pydata.org/docs/reference/api/pandas.melt.html.

[30] [n.d.]. Unpivot in R (melt). https://rdatatable.gitlab.io/data.table/reference/melt.data.table.html.

[31] [n.d.]. Unpivot in Tableau. https://community.tableau.com/s/question/0D54T00000C6ndgSAB/how-to-unpivot-data.

[32] [n.d.]. Use Relationships for Multi-table Data Analysis. https://help.tableau.com/current/server/en-us/datasource_multitable_normalized.htm.

[33] Retrieved in 2023-01. Power BI. https://powerbi.microsoft.com/en-us/.

[34] Retrieved in 2023-01. Tableau. https://www.tableau.com/.

[35] Daniel W Barowy, Sumit Gulwani, Ted Hart, and Benjamin Zorn. 2015. FlashRelate: extracting relational data from semi-structured spreadsheets using examples. *ACM SIGPLAN Notices* 50, 6 (2015), 218–228.

[36] Rohan Bavishi, Caroline Lemieux, Roy Fox, Koushik Sen, and Ion Stoica. 2019. AutoPandas: neural-backed generators for program synthesis. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 1–27.

[37] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher 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. arXiv:2005.14165 [cs.CL]

[38] Surajit Chaudhuri, Umeshwar Dayal, and Vivek Narasayya. 2011. An overview of business intelligence technology. *Commun. ACM* 54, 8 (2011), 88–98.

[39] Ran Chen, Di Weng, Yanwei Huang, Xinhuan Shu, Jiayi Zhou, Guodao Sun, and Yingcai Wu. 2022. Rigel: Transforming tabular data by declarative mapping. *IEEE Transactions on Visualization and Computer Graphics* 29, 1 (2022), 128–138.

[40] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[41] Zhimin Chen, Vivek Narasayya, and Surajit Chaudhuri. 2014. Fast foreign-key detection in Microsoft SQL server PowerPivot for Excel. *Proceedings of the VLDB Endowment* 7, 13 (2014), 1417–1428.

[42] Arash Dargahi Nobari and Davood Rafiei. 2024. Dtt: An example-driven tabular transformer for joinability by leveraging large language models. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–24.

[43] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. 2017. Robustfill: Neural program learning under noisy i/o. In *International conference on machine learning*. PMLR, 990–998.

[44] Sumit Gulwani, William R Harris, and Rishabh Singh. 2012. Spreadsheet data manipulation using examples. *Commun. ACM* 55, 8 (2012), 97–105.

[45] Jiawei Han. 1998. OLAP mining: An integration of OLAP with data mining. In *Data Mining and Reverse Engineering: Searching for semantics. IFIP TC2 WG2. 6 IFIP Seventh Conference on Database Semantics (DS-7) 7–10 October 1997, Leysin, Switzerland*. Springer, 3–20.

[46] 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. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1165–1177.

[47] Yeye He, Kris Ganjam, and Xu Chu. 2015. Sema-join: joining semantically-related tables using big table corpora. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1358–1369.

[48] 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*. 1785–1788.

[49] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data*. 829–846.

[50] Zhipeng Huang and Yeye He. 2018. Auto-detect: Data-driven error detection in tables. In *Proceedings of the 2018 International Conference on Management of Data*. 1377–1392.

[51] Lan Jiang and Felix Naumann. 2020. Holistic primary key and foreign key detection. *Journal of Intelligent Information Systems* 54, 3 (2020), 439–461.

[52] Zhongjun Jin, Michael R Anderson, Michael Cafarella, and HV Jagadish. 2017. Foofah: Transforming data by example. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 683–698.

[53] Zhongjun Jin, Michael Cafarella, HV Jagadish, Sean Kandel, Michael Minar, and Joseph M Hellerstein. 2018. CLX: Towards verifiable PBE data transformation. *arXiv preprint arXiv:1803.00701* (2018).

[54] Zhongjun Jin, Yeye He, and Surajit Chauduri. 2020. Auto-transform: learning-to-transform by patterns. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2368–2381.

[55] Richard M Karp. 2010. *Reducibility among combinatorial problems*. Springer.

[56] William Kent. 1983. A simple guide to five normal forms in relational database theory. *Commun. ACM* 26, 2 (1983), 120–125.

[57] Ralph Kimball and Margy Ross. 2011. *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons.

[58] Lawrence Kou, George Markowsky, and Leonard Berman. 1981. A fast algorithm for Steiner trees. *Acta informatica* 15 (1981), 141–145.

[59] Doris Jung-Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang, Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A Hearst, et al. 2021. Lux: always-on visualization recommendations for exploratory dataframe workflows. *arXiv preprint arXiv:2105.00121* (2021).

[60] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[61] 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*. 1064–1076.

[62] Peng Li, Yeye He, Cong Yan, Yue Wang, and Surajit Chaudhuri. 2023. Auto-Tables: Synthesizing Multi-Step Transformations to Relationalize Tables without Using Examples. arXiv:2307.14565 [cs.DB]

[63] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2023. Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv preprint arXiv:2310.09263* (2023).

[64] Yiming Lin, Yeye He, and Surajit Chaudhuri. 2023. Auto-BI: Automatically Build BI-Models Leveraging Local Join Prediction and Global Schema Graph. arXiv:2306.12515 [cs.DB]

[65] Jock Mackinlay, Pat Hanrahan, and Chris Stolte. 2007. Show me: Automatic presentation for visual analysis. *IEEE transactions on visualization and computer graphics* 13, 6 (2007), 1137–1144.

[66] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective error correction via a unified context representation and transfer learning. *Proceedings of the VLDB Endowment* 13, 12 (2020), 1948–1961.

[67] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data*. 865–882.

[68] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can foundation models wrangle your data? *arXiv preprint arXiv:2205.09911* (2022).

[69] Solomon Negash and Paul Gray. 2008. Business intelligence. *Handbook on decision support systems 2* (2008), 175–193.

[70] Arash Dargahi Nobari and Davood Rafiei. 2024. TabulaX: Leveraging Large Language Models for Multi-Class Table Transformations. *arXiv preprint arXiv:2411.17110* (2024).

[71] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[72] John Platt et al. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers* 10, 3 (1999), 61–74.

[73] Alexandra Rostin, Oliver Albrecht, Jana Bauckmann, Felix Naumann, and Ulf Leser. 2009. A machine learning approach to foreign key discovery.. In *WebDB*.

[74] Arie Shoshani. 1997. OLAP and statistical databases: Similarities and differences. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. 185–196.

[75] Yasin N Silva, Walid G Aref, and Mohamed H Ali. 2010. The similarity join database operator. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. IEEE, 892–903.

[76] Alkis Simitsis, Panos Vassiliadis, and Timos Sellis. 2005. Optimizing ETL processes in data warehouses. In *21st International Conference on Data Engineering (ICDE'05)*. Ieee, 564–575.

[77] Rishabh Singh. 2016. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *Proceedings of the VLDB Endowment* 9, 10 (2016), 816–827.

[78] Manel Souibgui, Faten Atigui, Saloua Zammali, Samira Cherfi, and Sadok Ben Yahia. 2019. Data quality in ETL process: A preliminary study. *Procedia Computer Science* 159 (2019), 676–687.

[79] Quoc Trung Tran, Chee-Yong Chan, and Srinivasan Parthasarathy. 2009. Query by output. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 535–548.

[80] Panos Vassiliadis. 2009. A survey of extract–transform–load technology. *International Journal of Data Warehousing and Mining (IJDWM)* 5, 3 (2009), 1–27.

[81] Panos Vassiliadis and Timos Sellis. 1999. A survey of logical models for OLAP databases. *ACM Sigmod Record* 28, 4 (1999), 64–69.

[82] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing highly expressive SQL queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 452–466.

[83] Pei Wang and Yeye He. 2019. Uni-detect: A unified approach to automated error detection in tables. In *Proceedings of the 2019 International Conference on Management of Data*. 811–828.

[84] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[85] Jennifer Widom. 1995. Research problems in data warehousing. In *Proceedings of the fourth international conference on Information and knowledge management*. 25–30.

[86] Cong Yan and Yeye He. 2020. Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks. In *International Conference on Management of Data (SIGMOD)*. ACM, 1539–1554. https://www.microsoft.com/en-us/research/publication/auto-suggest-learning-to-recommend-data-preparation-steps-using-data-science-notebooks/

[87] Jing Nathan Yan, Oliver Schulte, MoHan Zhang, Jiannan Wang, and Reynold Cheng. 2020. Scoded: Statistical constraint oriented data error detection. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 845–860.

[88] Junwen Yang, Yeye He, and Surajit Chaudhuri. 2021. Auto-pipeline: synthesizing complex data pipelines by-target using reinforcement learning and search. *arXiv preprint arXiv:2106.13861* (2021).

[89] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M Procopiuc, and Divesh Srivastava. 2010. On multi-column foreign key discovery. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 805–814.

[90] Sai Zhang and Yuyin Sun. 2013. Automatically synthesizing sql queries from input-output examples. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 224–234.

[91] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*. 847–864.

[92] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1034–1045.

[93] Moshé M Zloof. 1975. Query-by-example: the invocation and definition of tables and forms. In *Proceedings of the 1st international conference on very large data bases*. 1–24.