EcoTable: Cost-effective Table Integration in Data Lakes for Natural Language Queries

Yuhui Wang
Beijing Institute of Technology
Beijing, China
wangyuhui@bit.edu.cn

Jinqi Liu
Beijing Institute of Technology
Beijing, China
liujinqi@bit.edu.cn

Chengliang Chai*
Beijing Institute of Technology
Beijing, China
ccl@bit.edu.cn

Hangyu Zhao
Beijing Institute of Technology
Beijing, China
zhyuo@bit.edu.cn

Yuhao Deng
Beijing Institute of Technology
Beijing, China
dyh18@bit.edu.cn

Yuyu Luo
The Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China
yuyuluo@hkust-gz.edu.cn

Xin Tang
University of Wisconsin–Madison
Beijing, China
alwintang@apache.org

Ye Yuan
Beijing Institute of Technology
Beijing, China
yuan-ye@bit.edu.cn

Guoren Wang
Beijing Institute of Technology
Beijing, China
wanggr@bit.edu.cn

Fengjin Wang
Kuaishou Technology
Beijing, China
wangfengjin@kuaishou.com

Lei Cao
Massachusetts Institute of Technology
Cambridge, MA, USA
lcao@csail.mit.edu

ABSTRACT

The diverse formats of CSV and Parquet files in data lakes pose a significant challenge to traditional ETL, which relies on data engineers to pre-define a target database schema and build a complex pipeline for data integration. Moreover, with this approach, the integrated data often cannot support various analytical needs, as the predefined schema does not necessarily satisfy the table format or join relationships required to answer unforeseen queries. To address this, we propose `EcoTable` , the first natural language-based data integration framework. Given a set of user-specified natural language queries, `EcoTable` automatically integrates the tables into a form that adequately supports the corresponding SQL queries. `EcoTable` achieves this by leveraging the semantic understanding and complex reasoning capabilities of Large Language Models (LLMs). Moreover, `EcoTable` addresses the scalability and cost issues introduced by expensive LLM inferences with a set of novel ideas.

`EcoTable` first introduces a graph to represent the overall search space, where nodes represent tables and edges carry weights indicating join likelihood produced by a lightweight deep learning model. On top of this graph data structure, `EcoTable` designs three components to achieve our goal: (1) the table identification layer aims to identify relevant tables via a two-stage schema linking based on user queries; (2) the graph-based validation layer aims to discover significant join paths, including necessary data transformations and bridging tables, by modeling the problem as Steiner tree searches; and (3) the table transformation layer generates transformation code to implement the joins using LLMs. We construct 4 real-world benchmark datasets with more than 200 queries. Extensive experiments demonstrate that `EcoTable` outperforms the state-of-the-art baselines, increasing accuracy by more than 30% and cutting LLM invocation costs by 5 times.

*Chengliang Chai is the corresponding author.

PVLDB Reference Format:
Yuhui Wang, Jinqi Liu, Chengliang Chai, Hangyu Zhao, Yuhao Deng, Yuyu Luo, Xin Tang, Ye Yuan, Guoren Wang, Fengjin Wang, and Lei Cao. EcoTable: Cost-effective Table Integration in Data Lakes for Natural Language Queries. PVLDB, 19(11): 2936 - 2949, 2026.
doi:10.14778/3836663.3836664

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/yuhuiwang02/EcoTable.

1 INTRODUCTION

Motivation. Data lakes store large collections of files (e.g., in CSV or Parquet formats) that do not fit neatly into relational tables, offering immense potential for data-driven decision making [3, 18]. However, deriving values from such a lightly structured data lake is notoriously difficult. The predominant approach relies on a traditional Extract-Transform-Load (ETL) process, where data engineers first *manually* define a target schema and then develop

Proceedings of the VLDB Endowment, Vol. 19, No. 11 ISSN 2150-8097.
doi:10.14778/3836663.3836664

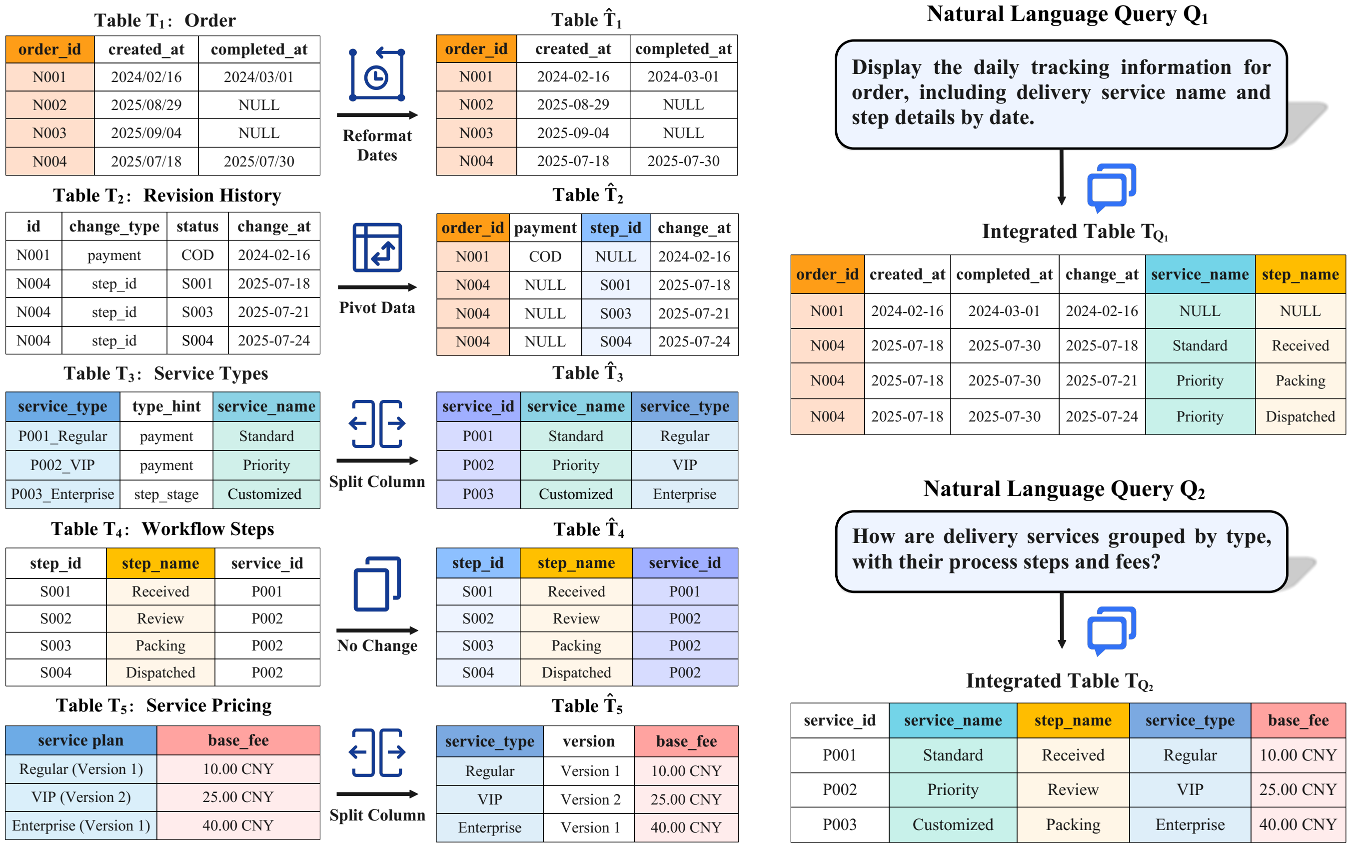

Figure 1: An Example of Query-driven Table Integration in Data Lakes.

a complex pipeline to integrate data into a data warehouse [44, 59]. This is challenging. First, it is rather difficult to pre-define an appropriate database schema that meets the diverse analytical needs of an application. Second, even if the schema were defined correctly, it is labor intensive to convert the data lake tables into the desired relational table format.

Next, we use a real example to better illustrate this.

Example 1.1. As shown in Figure 1, consider 5 tables (i.e., T_1: `Order`, T_2: `Revision History`, T_3: `Service Types`, T_4: `Workflow Steps`, and T_5: `Service Pricing`) sampled from a real-world E-Commerce data lake. A data engineer might naturally define a database schema over these tables, where T_1.`order_id` joins with T_2.`id`. However, the pre-defined schema does not necessarily satisfy the analytical need of users. For instance, suppose a user wants to check order status and thus issues a query Q_1, *"Display the daily tracking information for the order, including the name of the delivery service and workflow step details by date."* This query involves three tables (i.e., T_1: `Order`, T_3: `Service Types`, and T_4: `Workflow Steps`). However, in the initial database schema, it is impossible to join these tables to answer this query because T_1 lacks explicit join conditions with either T_3 or T_4. Furthermore, integrating tables requires significant manual efforts. To join T_1, T_3 and T_4 for Q_1, a data engineer must: (1) bridge T_1 and T_4 via T_2 with the join between T_2 and T_4 on the shared `step_id` attribute; and (2) transform T_3 by splitting `service_type` to enable join with T_4 on `service_id`. This process is complex, as it necessitates incorporating additional tables into join path and transforming tables. Moreover, the user might want to know more about service fees and thus submit another query Q_2: *"How are delivery services grouped by type, with their process steps and fees?"*. To answer Q_2, the data engineer must integrate T_5: `Service Pricing` with T_3 and T_4 – a new data integration requirement. The key step is to split the `service_plan` column in T_5 (e.g., "Regular (Version 1)") into distinct `service_type` and `version` columns. This prepares T_5 to be joinable with T_3 on `service_type` column.

This example shows that different queries have different requirements on the database schema. Therefore, pre-defining a database schema to meet users' analytical needs tends to be impractical. Instead, data integration has to be tailored to user queries.

Targeting this problem, we propose to build an automatic table integration system, called `EcoTable` . With this system, users only have to provide a set of natural language (NL) queries to intuitively describe their analysis intention. The system will automatically discover a database schema that adequately supports these queries and then perform data integration accordingly.

Existing work has proposed techniques to automate data integration such as schema matching [19, 33, 37], entity resolution [17, 34, 41, 46, 53, 54], data transformation [38, 39, 62], discovery of joinable columns [10, 21], etc. However, all of these approaches rely on a pre-defined target schema. They do not solve the problem of automatically discovering an appropriate database schema based on NL queries and tables. On the other hand, a line of research [25, 32, 58, 59] focuses on transforming tables, e.g., spreadsheets or web tables, into relational formats. However, because the relational schema these methods automatically discover is blind to user queries, it tends to be ineffective in supporting the analytical needs of users, as shown in the above example.

The semantic understanding and reasoning ability of Large Language Models (LLMs) make it possible for us to achieve the above goal. First, existing Text-to-SQL research has shown that LLMs excel at identifying the mapping between NL queries and columns in the tables through schema linking [4, 8, 48, 52]. Moreover, with its reasoning ability [28, 55, 57, 60], an LLM is able to produce a plan that combines multiple operations to collaboratively solve the data transformation problem.

Challenge. An intuitive method that leverages LLMs to solve this problem is to ask LLMs to (1) identify all tables that are relevant to the NL queries provided by users; (2) check whether every pair of these discovered tables is joinable, thus finding a join path for each query; (3) finally, perform data transformation and integration accordingly. However, routinely invoking LLMs to explore all possible join relationships is prohibitively expensive. This is because the number of potential join paths grows exponentially with the number of tables. Therefore, the key challenge lies in how to minimize the number of LLM calls while still achieving high-quality integration to reliably answer user queries.

Our Proposal. To address this challenge, we propose `EcoTable`, a cost-effective query-driven table integration system. The key idea is to combine smaller deep learning models and LLMs to efficiently identify and validate potential join paths. At a high level, deep learning models give prior probabilities of possible join relationships between tables. Then, to minimize LLM invocations, `EcoTable` prioritizes the pairs of tables for the LLMs to validate that are (1) most likely to be joinable and (2) required to answer user queries.

`EcoTable` models the search space as a fully connected graph where each node corresponds to a table, and each edge (i.e., the join relationship) connects two tables with a weight indicating the join likelihood initialized with a deep learning model. In this way, the problem of finding a *valid join path* for each NL query is converted to identifying a connected subgraph that covers the tables involved in the query. `EcoTable` consists of three key components that efficiently discover valid join paths and perform data transformation on top of the graph.

Table Identification Layer identifies query-related tables (i.e., nodes) from all data lake tables via a two-stage schema linking method that first leverages a lightweight deep learning model for coarse-grained filtering, followed by using LLMs for fine-grained verification. This step effectively reduces the search space to a subset of nodes in the above graph, ensuring that subsequent steps primarily focus on these relevant tables.

Graph-based Validation Layer. Next, `EcoTable` discovers valid join paths to integrate these tables. As observed in the above example, joining these tables may require additional bridging tables, which is challenging to discover among a potentially large number of tables. To address this, we model the problem as a Steiner tree search problem, where the goal is to find the minimal-weight tree, i.e., a tree composed of edges with the highest join likelihood, which connects all relevant tables for each query, potentially via a set of bridging tables.

More specifically, `EcoTable` gathers edges from Steiner trees w.r.t. various queries and submits them to LLMs for batch verification. This opens the opportunity to leverage shared edges among queries to reduce LLM costs. Then `EcoTable` updates the edge weights of the graph based on verified results and iteratively searches the Steiner trees until the necessary edges of all queries have been validated. In this way, `EcoTable` selectively applies LLMs to validate the most promising join relationships and progressively discovers valid join paths with minimal LLM usage.

Table Transformation Layer. `EcoTable` then applies the necessary data transformation operations to perform joins and integrate tables. `EcoTable` uses LLMs to automatically generate transformation code for each valid join path, guided by a ReAct-style reasoning process that alternates between transformation and join validation. To reduce latency, `EcoTable` further employs a graph-based parallelization strategy, which identifies parallelizable transformations by modeling the problem as an edge coloring task on the validated join paths.

In summary, we make the following contributions.

(1) We propose `EcoTable`, to the best of our knowledge, the first natural language driven system for table integration that minimizes LLM invocations while adequately supporting user queries.

(2) We design a graph-based validation layer that models table relationships as a weighted graph and identifies valid join paths using a Steiner tree–based optimization.

(3) We introduce a table transformation layer that employs a ReAct-style reasoning loop to automatically generate transformation code for joinable tables and a graph-based parallelization strategy to further reduce latency and LLM invocation cost.

(4) We construct 4 real-world benchmark datasets covering multiple domains with more than 200 natural language queries. Comprehensive experiments show that `EcoTable` surpasses state-of-the-art baselines by achieving over 30% accuracy improvement and reducing LLM invocation costs by 5 times.

2 PROBLEM DEFINITION

Given a data lake including a large number of tables (e.g., CSV or parquet files) and a set of user-specified natural language queries, we aim to integrate subsets of tables related to these queries such that each query could be well addressed by an integrated table. Due to the heterogeneity of the data lake tables, the integration process might be rather complicated, including transformations and joins over multiple tables. Formally, we define the problem as follows.

Definition 2.1 (Query-Driven Data Integration). Given N tables in a data lake $\mathcal{T} = \{T_1, T_2, \ldots, T_N\}$ and a set of M natural language queries $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_M\}$, we aim to construct a set of transformed tables $\mathcal{T}^* = \{\hat{T}_1, \hat{T}_2, \ldots, \hat{T}_n\}$ from $\mathcal{T}$ to form a global join graph $\mathcal{G}^* = (\mathcal{T}^*, E^*)$, such that each query $Q_k \in \mathcal{Q}$ can be successfully answered by querying a connected subgraph $\mathcal{G}_k^* = (\mathcal{T}_k^*, E_k^*)$ of $\mathcal{G}^*$, where $\mathcal{T}^* = \bigcup_{k=1}^{M} \mathcal{T}_k^*$ and $E^* = \bigcup_{k=1}^{M} E_k^*, k \in [1, M]$. That is, each connected subgraph $\mathcal{G}_k^*$ corresponds to an integrated table sufficient for answering query Q_k and the table is obtained

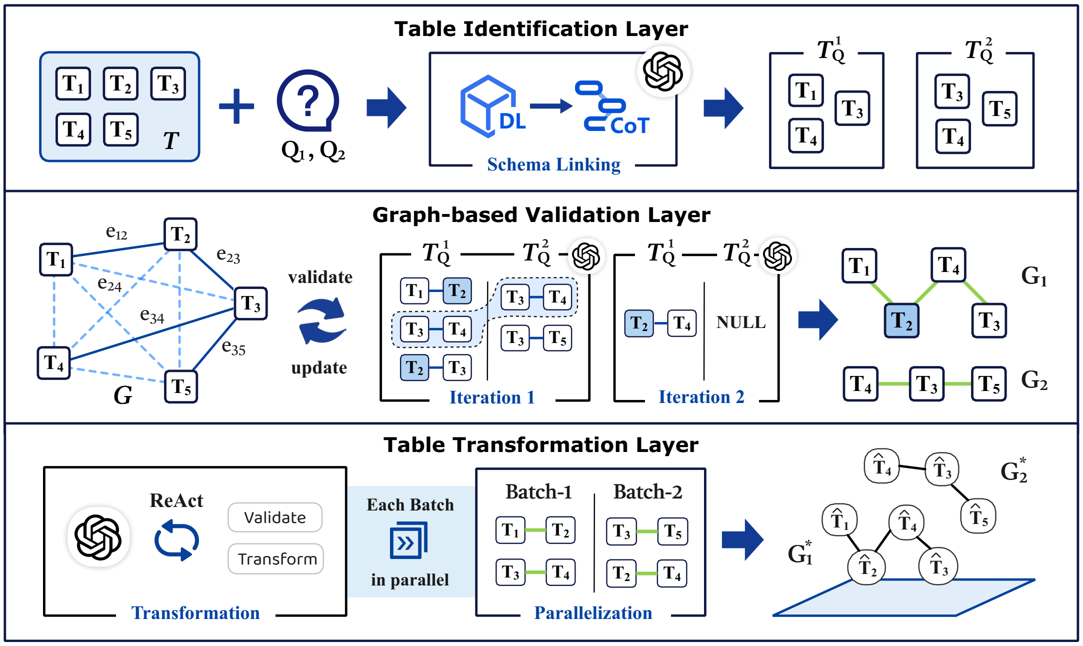

Figure 2: The Overall Framework of EcoTable.

by integrating tables in $\mathcal{T}_k^*$. The nodes and edges are defined as follows:

Nodes: For each query Q_k, the node set $\mathcal{T}_k^* = \{\hat{T}_k^1, \hat{T}_k^2, \ldots\}$ contains the transformed tables required to answer the query. Each $\hat{T}_k^i \in \mathcal{T}_k^*$ is derived from a source table $T_k^i \in \mathcal{T}_k \subseteq \mathcal{T}$ through the transformation ϕ_k^i, i.e., $\hat{T}_k^i = \phi_k^i(T_k^i)$. In this paper, $\phi_k^i(\cdot)$ includes operations such as reformatting, splitting, and pivoting.

Edges: For any edge $\hat{e}_k^{ij} \in E_k^*$, it connects two tables $\hat{T}_k^i, \hat{T}_k^j \in \mathcal{T}_k^*$ if they can be joined to answer Q_k. Note that $\mathcal{G}_k^*$ is connected, corresponding to the join path, but $\mathcal{G}^*$ might not be when different queries are associated with totally different tables.

Example 2.2. Based on Example 1.1 and Figure 1, the mapping to the formal graph $\mathcal{G}^*$ is as follows:

- Query Q_1 is addressed by subgraph $\mathcal{G}_1^*$, with nodes $\mathcal{T}_1^* = \{\hat{T}_1, \hat{T}_2, \hat{T}_3, \hat{T}_4\}$ and edges $E_1^* = \{\hat{e}_1^{12}, \hat{e}_1^{24}, \hat{e}_1^{34}\}$. Each $\hat{T}_i$ is the result of the transformation ϕ_1^i (e.g., date reformatting for T_1, table pivoting for T_2, and column splitting for T_3).
- Query Q_2 is addressed by subgraph $\mathcal{G}_2^*$, which involves nodes $\mathcal{T}_2^* = \{\hat{T}_3, \hat{T}_4, \hat{T}_5\}$ and edges $E_2^* = \{\hat{e}_2^{12}, \hat{e}_2^{13}\}$. Here, $\hat{T}_5$ is the result of the transformation ϕ_2^3 (e.g., column splitting for T_5).
- The global join graph $\mathcal{G}^* = (\mathcal{T}^*, E^*)$ is defined by the unions $\mathcal{T}^* = \bigcup_{k=1}^{M} \mathcal{T}_k^*$ and $E^* = \bigcup_{k=1}^{M} E_k^*$, forming a unified structure that supports all analytical requirements.

Table Transformation resolves structural and semantic discrepancies in data lakes.We focus on three essential operators:
<u>*Normalize*</u> standardizes data formats for consistency, e.g., converting the `created_at` date in Table T_1 from "2024/02/16" to the ISO standard "2024-02-16" in Table $\hat{T}_1$.
<u>*Split*</u> decomposes composite columns into atomic ones, e.g., splitting the `service_type` column in Table T_5 (e.g., "Regular (Version 1)") into separate `service_type` and `version` columns in Table $\hat{T}_5$.
<u>*Pivot*</u> restructures tables by transposing unique values into new columns, e.g., applying *Pivot* to T_2 on the `change_type` column generates new column headers `payment` and `step_id` in Table $\hat{T}_2$.
Join Discovery identifies semantically equivalent columns across tables to form valid join paths. Existing approaches include:
<u>*Deep learning methods*</u> (e.g., DeepJoin [10] and OmniMatch [24]) encode column profiles into vector representations and predict the join likelihood between columns using PLMs. For example, T_2.`change_type` and T_3.`type_hint` might be considered as joinable because PLM-based methods focus more on token overlaps or string similarities. In reality, they refer to different things: `change_type` is a revision category, whereas `type_hint` is an interface tag (e.g., `step_stage`) denoting the record's current workflow stage, so they should not be joined.
<u>*LLM-based methods*</u> leverage fine-tuning [33] or prompt engineering [37] to understand table semantics and validate join relationships. These methods demonstrate strong few-shot learning capabilities and can handle complex semantic matching scenarios that are challenging for traditional approaches. By reasoning over column descriptions and data patterns, LLMs achieve robust performance even on unseen schema or ambiguous relationships.

Given $\mathcal{T}$ and $\mathcal{Q}$, a straightforward way to construct $\mathcal{G}^*$ is that for each query Q_k, we enumerate a large number of possible table combinations from $\mathcal{T}$ and identify the combination that can satisfy the query, which is prohibitively expensive if we have to call LLMs to verify whether each combination is valid. Therefore, we propose the EcoTable framework to largely reduce the LLMs costs.

3 OVERVIEW

As illustrated in Sec. 1, `EcoTable` formulates the query-driven table integration task as a join path identification and validation problem over a graph, ensuring a balance between integration effectiveness and LLM invocation cost. Specifically, `EcoTable` first constructs a fully connected graph $\mathcal{G} = (\mathcal{T}, E)$. Then, `EcoTable` leverages the following three layers to integrate tables for correctly answering the given queries. As shown in Figure 2, the first table identification layer takes the queries as input, and outputs the table subsets relevant to these queries. Second, the graph-based validation layer takes these tables as input, and outputs valid join paths through searching within the graph $\mathcal{G}$, so as to connect these relevant tables. Third, the table transformation layer takes as input these identified join paths, applies necessary data transformation operations to actually join each two tables and outputs the integrated tables.

3.1 Table Identification Layer

In Figure 2, for example, given two queries Q_1 and Q_2, the module applies schema linking to obtain the corresponding relevant table sets $\mathcal{T}_Q^1 = \{T_1, T_3, T_4\}$ and $\mathcal{T}_Q^2 = \{T_3, T_4, T_5\}$, a well-studied process [4, 27, 43]. To strike a balance between accuracy and LLMs cost for identifying these tables, we adopt a hybrid approach of combining LLMs and deep learning models. Specifically, we first train a deep learning model M_s to predict the relevance between an NL query and a table. This model takes as input the concatenation of the query and the table's column names (i.e., its schema), embeds them and outputs a relevance score. Tables with high scores are then verified by LLMs (see Sec. 4.1 for details).

3.2 Graph-based Validation Layer

As shown in Figure 2, given the identified table subsets $\mathcal{T}_Q^1$ and $\mathcal{T}_Q^2$, this module finds valid join paths $\mathcal{G}_1$ and $\mathcal{G}_2$ that connect them. As outlined in Sec. 1, identifying such join paths is challenging because: (1) making relevant tables joinable often requires appropriate data transformations or added bridge tables, and (2) the search space is large to identify the valid join path.

To address this, we formulate it as a Steiner tree search problem based on the fully connected graph $\mathcal{G}$. Initially, `EcoTable` uses a deep learning model M_J to estimate the likelihood of joining pairs of tables and assign these values as edge weights in the graph. After that, for each Q_k, `EcoTable` attempts to search a corresponding Steiner tree from $\mathcal{G}$, where (1) the nodes cover the tables in $\mathcal{T}_Q^k$ and (2) the edges have the maximum overall joinable probability. Naturally, this Steiner tree represents the most promising join path in which the tables in $\mathcal{T}_Q^k$ could be joined to answer Q_k (e.g., for $\mathcal{T}_Q^1 = \{T_1, T_3, T_4\}$, a Steiner tree introduces T_2 as a bridge with edges $\{e_{12}, e_{23}, e_{34}\}$, as shown in Figure 2).

However, deep learning is typically not accurate enough, so we collect all edges from the Steiner trees corresponding to different queries and employ LLMs for further verification. This process leverages the fact that multiple queries may share common edges, thereby saving LLMs costs (e.g., edge e_{34} appears in both $\mathcal{T}_Q^1$ and $\mathcal{T}_Q^2$'s Steiner trees and requires only LLM validation once). Afterwards, we update the graph by removing edges that are deemed unjoinable by LLMs, while assigning a weight of one to those edges that are confirmed as joinable. We then iteratively recompute the Steiner trees using the updated graph and iterate the validation process until for every query, we either find a join path that makes the query executable or conclude that no such path exists due to insufficient connectivity (see Section 4.2 for details).

3.3 Table Transformation Layer

Finally, in Figure 2, starting from the valid join paths $\mathcal{G}_1$ and $\mathcal{G}_2$, the module applies data transformation operations to get $\mathcal{G}_1^*$ and $\mathcal{G}_2^*$, the integrated tables used to answer queries Q_1 and Q_2. To be specific, for each valid edge, `EcoTable` uses a transformation modular component M_T to automatically produce executable code, defined in a formal transformation grammar, for the two tables connected by the edge. To improve accuracy, LLMs verify each transformation using a ReAct-style reasoning process. For all these edges, a straightforward method is to batch multiple tables and ask LLMs to transform them together. However, this would degrade the accuracy due to complex chain dependencies within these tables, which is difficult for LLMs to understand. An alternative is to sequentially process each edge. However, this would incur high latency due to the iterative process of ReAct. To tackle this issue, `EcoTable` further introduces a graph-based parallelization strategy to identify independent transformations that can be executed concurrently based on the edge-coloring framework. The key observation is that two edges can be transformed in parallel if they do not share the same table. For example, in $\mathcal{G}_1$, transformations for e_{12} (T_1-T_2) and e_{34} (T_3-T_4) can be processed in parallel (see Section 5 for details).

4 GRAPH-BASED TABLE INTEGRATION

In this section, we introduce in details how `EcoTable` identifies query-related tables (Section 4.1), searches valid join paths (Section 4.2) and leverages LLMs to verify promising join relationships to produce finally integrated tables (Section 4.2).

4.1 Table Identification Layer

Stage 1 (PLM Table-level Selection). We fine-tune a lightweight-compact pre-trained language model, M_S, as a cross-encoder to compute relevance scores between a query and a table schema. Unlike bi-encoder setups that embed queries and schemas separately, M_S encodes them together in one sequence, allowing full-attention interactions between the query and schema metadata. As a modular component, M_S can be instantiated flexibly with various pre-trained backbones; in our implementation, we specifically support RoBERTa [36], RoBERTa-Large [36], and DeBERTa-v3-Large [20]. Subsequently, for each query, i.e., $Q_k \in \mathcal{Q}$, we choose the tables with large relevance scores as candidates, denoted as $\mathcal{T}_{\text{cand}}^k$. To be specific, we train the model on inputs of the form "*query* [TAB] *table_name* [COL] *col*_1 [COL] *col*_2 [COL], ...". The training process optimizes the model to predict binary labels $y_i \in \{0, 1\}$ by minimizing cross-entropy loss:

$$\mathcal{L} = -\left[y_i \log s_i + (1 - y_i)\log(1 - s_i)\right] \tag{1}$$

where s_i denotes the relevance score predicted by the model.

Overall, this step retrieves a sufficient number of tables that may answer the queries, i.e., ensuring a high recall. However, exploring the join pairs among this relatively large number of tables remains computationally expensive.

Stage 2 (LLMs Refinement). To address this, we employ LLMs to refine those candidate tables. Given $\mathcal{T}^k_{\text{cand}}$, `EcoTable` performs a fine-grained verification using the CoT reasoning capability of LLMs. Specifically, `EcoTable` prompts the LLMs to reason through the following steps:

Query Understanding. We utilize the LLMs as a query decomposer following a one-shot prompt setting. Given a natural language query Q, the LLMs are guided to split it into a sequence of sub-questions $q_1, q_2, \ldots, q_n$, effectively simplifying the complex request into atomic semantic components such as entities and attributes. For example, for Q_1 in Figure 1, LLMs identify key entities as *"order"* and *"delivery service"*, attributes such as *"order_id"* and *"service_name"*. This decomposition captures what the query focuses on.

Schema Reasoning and Alignment. For $\mathcal{T}^k_{\text{cand}}$, the LLMs use Chain-of-Thought to evaluate each candidate table step by step. Specifically, for each decomposed semantic component, the LLM tries to find a matching column in the table. It explicitly explains its reasoning, such as why a column matches a sub-question or why a table is missing the required information. If a candidate table cannot match any necessary components, the LLM rejects it. For example, for Q_1, LLMs reason that the entity *"order"* from the query is represented by the table T_1: `Order`, while its attribute *"order_id"* corresponds to the column `order_id` in T_1. In this way, LLMs reliably identify the final verified table set $\mathcal{T}^k_Q \subseteq \mathcal{T}^k_{\text{cand}}$, which contains the tables that are necessary to answer the query.

4.2 Graph-based Validation Layer

Given the fully-connected global join graph $\mathcal{G} = (\mathcal{T}, E)$, each edge $e_{ij} \in E$ represents a potential join relationship between tables T_i and T_j, associated with a weight that represents the joinable probability $p(e_{ij}) \in [0,1]$ computed by M_J. Given queries $Q = \{Q_1, Q_2, ..., Q_M\}$, `EcoTable` discover a collection of query-specific join paths $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_M\}$, where each $\mathcal{G}_k = (\mathcal{T}_k, E_k)$ connects tables in $\mathcal{T}^k_Q$ required to answer Q_k. Formally, this join path discovery problem can be formulated as:

$$\begin{aligned} \max_{\{x^k_{ij}\}} \quad & \sum_{k=1}^{M} \prod_{e_{ij} \in E} p(e_{ij})^{x^k_{ij}} \\ \text{s.t.} \quad & \forall k,\ \forall T_i \in \mathcal{T}^k_Q : \sum_{e_{ij} \in \delta(T_i)} x^k_{ij} \geq 1, \\ & \forall k,\ \forall S \subset \mathcal{T}'_k,\ S \neq \emptyset,\ S \neq \mathcal{T}'_k : \sum_{e_{ij} \in \phi(S)} x^k_{ij} \geq 1. \end{aligned} \tag{2}$$

where $x^k_{ij} \in \{0,1\}$ is a binary decision variable indicating whether edge e_{ij} is included in the join path of query Q_k. $\delta(T_i)$ denotes the set of edges corresponding to node T_i. $\mathcal{T}'_k = \{\ T_i | \exists T_j, x^k_{ij} = 1\}$, representing the set of tables (including the bridging tables) selected in the join path for query Q_k. For a subset $S \subseteq \mathcal{T}'_k$, $\phi(S)$ denotes the cut set of S (i.e., the edges connecting S to its complement $\mathcal{T}'_k \setminus S$), defined as $\phi(S) = \{e_{ij} \in E \mid T_i \in S, T_j \in \mathcal{T}'_k \setminus S\}$. The first constraint ensures that the join path includes all tables in $\mathcal{T}^k_Q$ for each query Q_k, while the second enforces connectivity. This optimization maximizes the sum of the overall joinability of each query, which is computed by multiplying the join likelihoods of

Algorithm 1: Graph-based Validation Algorithm

Input: Global weighted join graph $\mathcal{G} = (\mathcal{T}, E)$, relevant tables for each query $\{\mathcal{T}^1_Q, \mathcal{T}^2_Q, ..., \mathcal{T}^M_Q\}$.
Output: Validated join paths $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_M\}$.

$E^+ \leftarrow \emptyset,\ E^- \leftarrow \emptyset$;
while $\neg \bigcup_1^k E_k \subseteq E^+ \wedge \text{HasTree}(\mathcal{G}, \{\mathcal{T}^1_Q, ..., \mathcal{T}^M_Q\})$ **do**
  // Step 1: Graph update
  $\mathcal{G}_u \leftarrow \text{UpdateGraph}(\mathcal{G}, E^+, E^-)$;
  // Steps 2: LLMs validation
  $E_b \leftarrow \emptyset$;
  for $k \in [1..M]$ **do**
    $\mathcal{G}_k, \mathcal{T}_k, E_k = \text{SteinerTree}(\mathcal{G}_u, \mathcal{T}^k_Q)$;
    $E_b \leftarrow E_b \cup E_k$;
  $E_{new} \leftarrow E_b \setminus (E^+ \cup E^-)$;
  for *each* $e_{ij} \in E_{new}$ **do**
    if $\text{LLMValidate}(e_{ij}) = \textit{VALID}$ **then**
      $E^+ = E^+ \cup \{e_{ij}\}$;
    else
      $E^- = E^- \cup \{e_{ij}\}$;
for $k \in [1..M]$ **do**
  $\mathcal{G}_k = (\mathcal{T}_k, E^+ \cap E_k)$;
return $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_M\}$;

edges in the query's join path, i.e., $\prod_{e_{ij} \in E} p(e_{ij})^{x^k_{ij}}$. In this way, we prioritize the edges that are likely to be joined to answer queries for LLMs to validate.

Pairwise Joinability Estimation. Following [35], `EcoTable` adopts a deep learning model M_J to compute $p(e_{ij})$. This model serves as an efficient but approximate estimator of joinability, providing a prior probability for each edge. Specifically, given two columns, M_J takes input table features (e.g., metadata and column embeddings) and outputs join probabilities in (0,1]. Designed as a modular component, M_J allows for flexible instantiation. In this work, we utilize three models: Auto-BI [35], DeepJoin [10], and OmniMatch [24]. Formally, let $C \in T_i$ and $C' \in T_j$ denote two columns in tables T_i and T_j. $M_J(C_i, C_j) \in (0,1]$ predicts the probability that C and C' could be joined. Because the joinability between two tables often depends on the presence of at least one pair of aligned columns, we use the maximum pairwise similarity as a joinability score and assign this as the weight, i.e., $p(e_{ij}) = \max_{C \in T_i, C' \in T_j} M_J(C, C')$.

Iterative Validation with LLMs. Based on the weighted join graph $\mathcal{G}$, `EcoTable` utilizes the typical algorithm [23] with an approximation ratio of 2 (i.e., the KMB algorithm) to efficiently identify a Steiner tree for each query Q_k. Then `EcoTable` performs iterative validation with LLMs to refine the accuracy of these discovered Steiner trees. Specifically, at the beginning of each iteration, `EcoTable` selects all unverified edges from the Steiner tree of each query. These edges are submitted to LLMs to validate join relationship. `EcoTable` updates the weight of the join graph $\mathcal{G} = (\mathcal{T}, E)$ accordingly as follows:

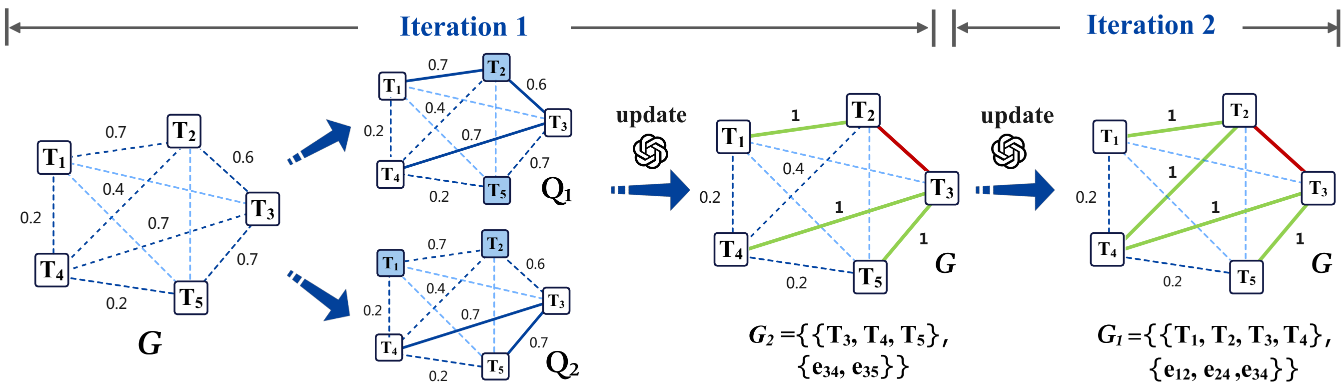

Figure 3: An Example of Join Path Search and Validation.

- If an edge e_{ij} passes validation, its corresponding weight is set to 1, ensuring that EcoTable will only validate one edge at most once.
- If e_{ij} is invalid, the edge is removed from the graph and will not be considered in later iterations.

For the updated graph, EcoTable recomputes a maximum Steiner tree for each query Q_k and checks the unverified edges in these new trees. This process repeats until all edges of the obtained trees pass the verification, or no more feasible Steiner trees are available.

Remark. Since multiple queries often involve overlapping table pairs, EcoTable caches all validated results globally. If an edge has been validated once—either as valid or invalid—the result is reused directly in the Steiner tree searches of other queries, eliminating redundant LLM calls throughout the query set $\mathcal{Q}$.

Algorithm Overview. Algorithm 1 summarizes the whole process of the graph-based validation layer. EcoTable initializes two edge sets E^+ and E^- to record valid and invalid joins verified before (line 1). In each iteration (line 2), EcoTable updates the join graph $\mathcal{G}$ by removing invalid edges and setting the weight of valid ones as 1 (line 4). It then searches for a new Steiner tree for each query over the updated graph (line 7-8). Next, the algorithm collects and deduplicates all unverified edges from these new Steiner trees into a candidate set E_{new} (line 9-10). Each edge is checked by LLMs. EcoTable adds valid edges to E^+ and invalid ones to E^- (line 11-15). It then uses the updated graph to recompute new Steiner trees for all queries until all edges pass the validation or no feasible tree remains. Finally, the algorithm outputs the validated join paths (line 16- 18).

Remark. Because LLM-based validation is prone to hallucinations, EcoTable further leverages majority voting for two types of critical edges: (1) edges that appear in Steiner trees corresponding to multiple queries within the same iteration, and (2) edges whose deletion would prevent constructing a new Steiner tree for a query in the subsequent iteration. EcoTable makes three independent LLM calls for each such edge and then adopts the majority decision.

Example 4.1. As shown in Figure 3, at the beginning, EcoTable constructs the initial weighted graph $\mathcal{G} = (\mathcal{T}, E)$, where the edge weights represent the joinability score estimated by model M_J. For each query, EcoTable first searches a Steiner tree that connects all query-relevant tables with the maximum total joinability score. The initial Steiner tree for Q_1 includes edges $\{e_{12}, e_{23}, e_{34}\}$, while for Q_2, it covers edges $\{e_{34}, e_{35}\}$. In the first iteration, EcoTable validates all unverified edges of both trees using LLM reasoning. Since e_{34} appears in both trees, EcoTable makes three independent LLM calls for it. Following the majority decision, e_{34} is confirmed as valid. Simultaneously, e_{12} and e_{35} are verified, while e_{23} is rejected and removed. All results are stored in the global cache. The graph $\mathcal{G}$ is updated accordingly: valid edges are set to weight 1, while invalid edges such as e_{23} are removed. At this point, Q_2 already has a validated join path: $\mathcal{G}_2 = \{\{T_3, T_4, T_5\}, \{e_{34}, e_{35}\}\}$. In the second iteration, EcoTable recomputes the Steiner tree for Q_1 on the updated graph. The new candidate tree contains edges $\{e_{12}, e_{24}, e_{34}\}$. Since e_{12} and e_{34} are already cached as valid, only e_{24} is newly verified. LLMs confirm e_{24} as valid, completing the join path for Q_1: $\mathcal{G}_1 = \{\{T_1, T_2, T_3, T_4\}, \{e_{12}, e_{24}, e_{34}\}\}$.

5 TABLE TRANSFORMATION LAYER

The table transformation layer (1) automatically generates the transformation code that enables table joins (Sec. 5.1), and (2) executes these transformations efficiently through a graph-based parallel strategy (Sec. 5.2).

5.1 Transformation Execution

For each edge $e_k^{ij} \in E_k$, EcoTable uses a transformation module M_T to produce executable code to join the two columns in T_k^i and T_k^j specified by the LLM in the last layer. Specifically, M_T identifies noise that interferes with equi-joins between the two tables and generates transformation code to fix these inconsistencies, thereby enabling an equi-join. As a modular component, M_T can be instantiated flexibly with various methods, such as DTT [6], TabulaX [39], or our proposed ReAct-style LLM agent. At a high level, the advantage of our approach is that it enables the agent to autonomously decide both when to invoke transformation operations and which specific ones are appropriate. Before proceeding, we must first give formal definitions of the grammars for several required operations.

Transformation Grammar. As shown in Table 1, we define 4 significant operators, organized into two functional categories:

Table 1: Transformation Grammar in EcoTable .

Category	Operator	Operator parameters
Validate	ValidateJoin	source_col, target_col
Transform	Normalize [16]	input_col, target_format
	Split [50]	split_col, delimiter
	Pivot [49]	reshape_type, target_cols

Validate (O_{val}). Given the two columns corresponding to the tables, the `ValidateJoin` operator prompts the LLM to determine whether a join between them is feasible. To support this decision, we supply the join containment ratio as a quantitative signal: Containment$(A, B) = \frac{|A \cap B|}{|A|}$ By comparing this ratio before and after each transformation, the agent checks whether the latest operation resolves inconsistencies. The procedure terminates and returns "joined" if the LLM concludes that no additional improvements can be made (for instance, when the ratio reaches 1), or when the maximum number of iterations is reached. Otherwise, the agent proceeds by applying further suitable operators from O_{trans}.

Transform (O_{trans}). We formally define the set of transformation operators used to support table joins as O_{trans} = {`Normalize`, `Split`, `Pivot`}. In particular, the `Normalize` operator allows the agent to generate distinct transformation functions for different formats within a single column (e.g., standardizing ISO strings and Unix timestamps together). The `Split` operator breaks a string column into two atomic columns based on a given delimiter (e.g., "_"), which facilitates accurate matching when dealing with composite keys. Lastly, the `Pivot` operator restructures the table so that schemas are aligned for downstream join operations.

Iterative ReAct Workflow. To summarize, the agent follows an iterative "Reason-Act-Observe" loop to join the tables. In each iteration: (1) The agent analyzes whether the tables need transformations based on the observation from previous iterations as well as current joinability score. If so, it selects the appropriate operators from the grammar. (2) The agent executes this code to apply specific transformations to the tables. (3) The environment captures the execution results and feedbacks the updated state. This loop persists until either the LLM determines as joinable or the maximum iteration limit is reached. Notably, this workflow supports multi-step transformation chains. For example, the agent can first invoke `Pivot` to align the table's overall structure. It can then apply `Normalize` to resolve any remaining cell-level inconsistencies.

5.2 Parallelization Strategy

We obtain the validated Steiner trees $\{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_M\}$ in Sec. 4, where each $\mathcal{G}_k = (\mathcal{T}_k, E_k)$. These trees are merged to construct the composite graph $\mathcal{G}_v = (\mathcal{T}_v, E_v)$, where $\mathcal{T}_v = \bigcup_{k=1}^{M} \mathcal{T}_k$ and $E_v = \bigcup_{k=1}^{M} E_k$. A straightforward approach is to input the entire graph $\mathcal{G}_v$ to the ReAct-style agent and ask it to generate transformation code for every edge in that graph at once. However, this would yield suboptimal accuracy. The reason is that the tree structure of join relationships introduces complex sequential dependencies that make it difficult for LLMs to reason about all transformations

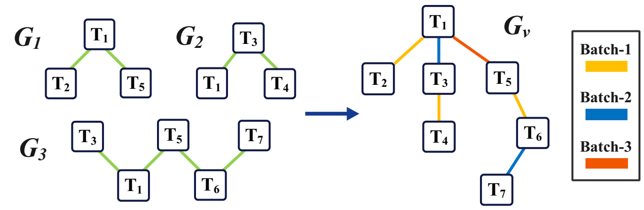

Figure 4: Parallelization Strategy

Algorithm 2: Parallel Transformation Algorithm

Input: Validated composite graph $\mathcal{G}_v = (\mathcal{T}_v, E_v)$.
Output: Final integrated join path $\mathcal{G}^* = (\mathcal{T}^*, E^*)$.

// Step 1: Edge batching
BatchList $\leftarrow$ Batch($\mathcal{G}_v$);
$\mathcal{T}^* \leftarrow \mathcal{T}_v$;
$E^* \leftarrow \emptyset$;
// Step 2: Execute batches in parallel
for *each batch* $I_t \in$ BatchList **do**
  for *each edge* $e_{ij} \in I_t$ **do in parallel**
    $(\hat{T}_i, \hat{T}_j, \hat{e}_{ij}) \leftarrow$ ReAct_Transform$(\mathcal{T}^*, e_{ij})$;
    $\mathcal{T}^* \leftarrow (\mathcal{T}^* \setminus \{T_i, T_j\}) \cup \{\hat{T}_i, \hat{T}_j\}$;
    $E^* \leftarrow E^* \cup \{\hat{e}_{ij}\}$;
return $\mathcal{G}^* = (\mathcal{T}^*, E^*)$;

coherently in a single pass. However, sequentially processing each edge would incur significant latency. To address this, we propose an iterative parallel processing strategy. In each iteration, we identify independent edges—those sharing no common nodes—within E_v and submit them to LLMs for parallel transformation. This approach ensures that transformations for independent edges can be reasoned about concurrently without cognitive interference. The process iterates until all edges in the join path have been processed, effectively balancing transformation quality with computational efficiency.

Maximizing Average Parallelism. We partition the edge set E_v into C disjoint batches $\{I_1, I_2, ..., I_C\}$, where each batch $I_t, t \in [1, C]$ contains a set of independent edges that can be transformed in parallel. Then, our goal is to maximize the average parallelism of batches, which is defined as the mean number of transformations executed per batch, i.e., $\frac{1}{C}\sum_{t=1}^{C}|I_t|$.

Formally, the optimization objective could be formulated as:

$$\begin{aligned} &\max_{\{I_t\}} \quad \frac{1}{C}\sum_{t=1}^{C}|I_t| \\ &\text{s.t.} \quad \bigcup_{t=1}^{C} I_t = E_v, \\ &\qquad \forall\, t \neq t' : \; I_t \cap I_{t'} = \emptyset, \\ &\qquad \forall\, e_{ij}, e_{uv} \in I_t : \; \{T_i, T_j\} \cap \{T_u, T_v\} = \emptyset. \end{aligned} \tag{3}$$

The second constraint ensures that the edges do not overlap between batches. The third constraint ensures that no two edges in one batch share the same table.

Parallel Transformation Algorithm. In Equation 3, our goal is to choose as many independent edges as possible per batch, which corresponds to minimizing the total number of batches. This can be regarded as the classical edge-coloring problem [22, 51], which assigns colors to edges such that no two adjacent edges share the same color, equivalent to our problem of finding independent edges. The goal is to minimize the number of colors used, which corresponds to the number of batches in our setting. `EcoTable` uses the typical edge coloring algorithm, i.e., Vizing [51] to identify a near-optimal list of batches (line 2, as shown in Algorithm 2). For each batch I_t, `EcoTable` executes all transformations in parallel using the ReAct reasoning loop, updates transformed tables, and records validated edges (line 3 - 10). In the end, all transformed tables and validated edges form the final integrated join path (line 11).

Next, we provide an example to demonstrate how `EcoTable` organizes transformations across multiple queries into parallel batches.

Example 5.1. As shown in Figure 4, we merge the Steiner tree $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$ into the composite graph $\mathcal{G}_v$. We then apply the Vizing algorithm to color the edges of $\mathcal{G}_v$, which produces three batches of independent edges, i.e., Batch-1 = $\{e_{12}, e_{34}, e_{56}\}$, Batch-2 = $\{e_{13}, e_{67}\}$, and Batch-3 = $\{e_{15}\}$. Each batch runs simultaneously.

6 EVALUATION

In this section, we first detail our benchmark construction process (Sec. 6.1), followed by extensive experiments to compare `EcoTable` against 6 baselines in both effectiveness and efficiency, focusing on answering the following research questions (RQs):

RQ1 (Sec. 6.2): How does `EcoTable` perform in discovering the correct join paths compared to baseline approaches?

RQ2 (Sec. 6.3): Can `EcoTable` improve the accuracy of end-to-end SQL execution when combined with an Text-to-SQL system?

RQ3 (Sec. 6.4): How do different PLM backbones in the table identification layer affect the retrieval of relevant tables?

RQ4 (Sec. 6.5): How does the graph-based validation layer perform in estimating joinability and balancing accuracy with cost?

RQ5 (Sec. 6.6): How effective is the ReAct-style method in generating accurate table transformations?

6.1 Experimental Settings

Datasets. We evaluate the performance of `EcoTable` across five benchmark datasets, categorized two groups: four real-world industrial benchmarks and a large-scale synthetic data lake. The statistics of these datasets are summarized in Table 2.

(i) DBT Benchmarks. We derive four datasets from well-known DBT datasets for table integration and transformation [7]. Specifically, (1) `Ad` has 69 tables, 126 queries, focusing on spend, impressions, and clicks, with 3.65 joins/query. (2) `Engagement` includes 56 tables, 38 queries on user engagement data, averaging 2.76 joins/query. (3) `Business` comprises 90 tables, 28 financial and sales data queries, with 4.33 joins/query. (4) `Platform` involves 35 tables, 24 queries on engineering activities, averaging 2.67 joins/query. The number of columns ranges from 2 to 389, and the number of rows ranges from 1 to 4,033 rows, highlighting integration complexity. These benchmarks include both equi-joins and fuzzy-joins, along with the required table transformations to answer the queries. The construction details of these benchmarks are in the technical report [56].

(ii) `NYC Data Lake`. To further test the scalability of `EcoTable` and its robustness, we introduce a large-scale benchmark constructed from NYC Open Data [5]. Unlike the DBT benchmarks, this dataset is generated to simulate a massive, messy data lake environment. It consists of 1,214 tables and 800 queries, featuring different types of noise and a large search space designed to challenge table identification and join path discovery algorithms. The benchmark includes 248 equi-joins, 204 fuzzy joins, and 2,440 table transformations, making it significantly more difficult than the DBT benchmarks. The detailed generation process for this benchmark is provided in the technical report [56].

Baselines. We compare `EcoTable` with various baselines, including existing methods and variants of our own approach.

(1) `Auto-BI` [35] identifies a join path using deep learning models given multiple tables. For fair comparison, we apply our table identification layer to capture query-relevant tables. `Auto-BI` then optimizes the join path by targeting these relevant tables.

(2) `Auto-Prep` [25] jointly predicts transformations and join paths with dual deep-learning models. It also considers the entire data lake tables rather than focusing on tables specific to an NL query. To ensure a fair comparison, we utilize our table identification layer to identify the tables relevant to the query. Subsequently, `Auto-Prep` performs searches for transformations and joins within these relevant tables.

(3) `LLMs-Only-EcoTable (LOE)` relies exclusively on LLMs for the entire pipeline, performing table identification, join path discovery, and transformation code generation without graph-based optimization.

(4) `Search-with-Greedy-Algorithm (SGA)` is a variation of our method that incrementally selects join edges with the highest joinability scores until all relevant tables are connected.

(5) `Without-ReAct-Reasoning (WRR)` is another variation that removes the ReAct-style loop, utilizing LLMs to produce the final transformation code directly in a single step.

(6) `EcoTable` is our full-fledged solution. It uses DeBERTa-v3-Large for M_S, DeepJoin for M_J, and the ReAct-style agent for M_T.

Metrics. We evaluate the join path accuracy, success rate for Text-to-SQL, average cost and average execution time across all the datasets. (1) Join Path Accuracy. We take F1 score as the evaluation metric. Let $\mathcal{G}_k^{gt} = (\mathcal{T}_k^{gt}, E_k^{gt})$ and $\mathcal{G}_k^{pred} = (\mathcal{T}_k^{pred}, E_k^{pred})$ denote the ground-truth and predicted join paths for query Q_k, respectively. We employ LLMs to judge whether the two edges represent the same join relation. Then, the precision and recall are computed as $\text{Pre}_k = \frac{|E_k^{pred} \cap E_k^{gt}|}{|E_k^{pred}|}$ and $\text{Rec}_k = \frac{|E_k^{pred} \cap E_k^{gt}|}{|E_k^{gt}|}$ respectively. The F1 score for Q_k is $F1_k = \frac{2 \times \text{Pre}_k \times \text{Rec}_k}{\text{Pre}_k + \text{Rec}_k}$. The overall accuracy is $\overline{F1} = \frac{1}{M} \sum_{k=1}^{M} F1_k$, where M is the total number of queries. (2) Success Rate for Text-to-SQL. We follow [28] to adopt the success rate (SR) metric, which is commonly used in Text-to-SQL evaluation [28] to measure the fraction of queries that are correctly answered. (3) Transformation F1-score. Following [6], we evaluate transformation quality by comparing the system-generated join cell pairs with the ground-truth pairs . (4) Average Cost. For each query Q_k, let C_k represent the total expense incurred by all LLM calls, which is calculated based on the API pricing [1, 40] calculated considering token usage. The average cost is determined as $\overline{C} = \frac{1}{M} \sum_{k=1}^{M} C_k$. (5)

Table 2: Statistics of Datasets.

Dataset	#Tables	#Queries	Max/Min/Avg. Cols	Max/Min/Avg. Rows	Avg. Tables/Joins	Equi	Fuzzy	Transf.
Ad [11]	69	126	86 / 3 / 19.51	2630 / 1 / 99.51	4.65 / 3.65	65	17	17
Engagement [13]	56	38	48 / 3 / 14.11	100 / 1 / 20.32	3.76 / 2.76	28	10	10
Business [12]	90	28	389 / 2 / 48.66	4033 / 1 / 100.48	5.33 / 4.33	13	7	7
Platform [15]	35	24	15 / 3 / 7.43	200 / 1 / 38.29	3.67 / 2.67	11	8	8
NYC Data Lake [14]	1214	800	219 / 1 / 19.67	480 / 1 / 183.58	5.04 / 4.14	248	204	2440

Cost-Aware Utility score (CAU). This score combines the F1 score and the normalized cost efficiency using a weight factor α to balance their importance: $CAU = \alpha \cdot F1 + (1-\alpha) \cdot \left(1 - \frac{Cost - Cost_{min}}{Cost_{max} - Cost_{min}}\right)$. We set $\alpha = 0.8$ in this paper.

6.2 Comparison with Baselines (RQ1)

6.2.1 Join Path Accuracy w.r.t. All Queries. Figure 5 reports the average F1 scores in join path accuracy with baselines. These methods are ranked as follows : EcoTable ≈ LOE ≈ SGA > WRR > Auto-Prep > Auto-BI.

EcoTable and LOE achieve the highest F1 scores among all baselines. LOE performs well because it leverages the powerful reasoning capabilities of LLMs to perform end-to-end join path discovery. EcoTable is competitive to LOE but incurs significantly lower costs. This is achieved by integrating a lightweight deep learning model with a Steiner tree formulation to identify a highly probable join path, which is then validated by LLMs solely on this path, thus reducing unnecessary LLMs calls while preserving high quality. Empirically, SGA and EcoTable achieve similar F1 because SGA also uses LLMs to validate edges but incurs higher costs because of the suboptimal edge selection strategy. Compared to WRR, EcoTable attains higher accuracy. because the ReAct mechanism effectively enables LLMs to detect and fix local reasoning errors during the transformation generation process.

EcoTable performs better than Auto-BI, for instance, on Ad, EcoTable achieves an F1 score of 79.2%, significantly outperforming Auto-BI (21.5%). This is due to (1) Auto-BI failing to consider table transformations, leading to a failure in joining two tables, and (2) Auto-BI depending solely on a deep model to estimate pairwise joinability. In contrast, EcoTable incorporates both table transformations and join path discovery using LLMs.

Notably, on NYC dataset, EcoTable and SGA outperform LOE, which is different from the observations in other benchmarks. The F1-score of LOE drops to 52% because its initial schema linking step introduces many query-irrelevant tables that significantly reduce precision. In contrast, EcoTable maintains a high F1 score of 65% by using a deep learning model to filter irrelevant tables, which ensures high recall while preventing LLM performance degradation due to the large number of noisy tables. Consequently, EcoTable proves more robust than end-to-end LLM methods when dealing with large-scale data lakes containing noisy data.

6.2.2 Average Cost. Figure 6 reports the average cost compared with baselines. EcoTable demonstrates a significant advantage in cost-efficiency compared to all baseline methods.

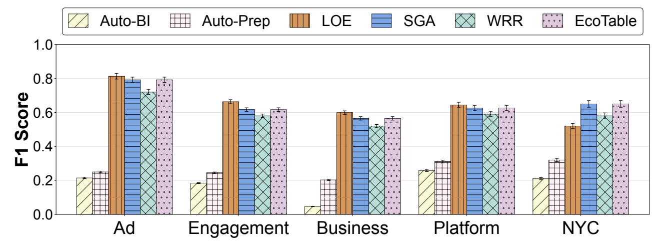

Figure 5: Comparison of Join Path Accuracy (F1 Score).

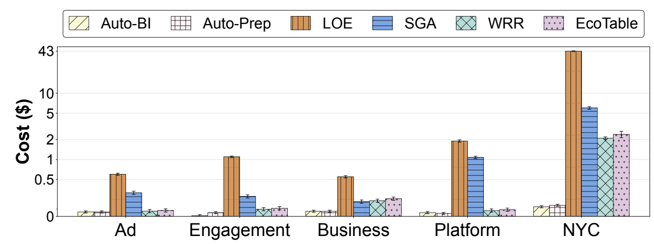

Figure 6: Comparison of Token Cost per Query.

EcoTable has substantially lower cost than LOE. On average for all datasets, EcoTable reduces token cost by about 5.3× compared to the greedy baseline SGA. This is because EcoTable leverages the Steiner-tree to achieve a global optimal edge selection, i.e., always identifying the most likely join paths for user-specified queries to validate. EcoTable also remains cheaper than LOE, which relies on end-to-end LLM exploration of the entire join space. Auto-BI and Auto-Prep use fewer costs because they only use LLMs for relevant table identification. However, they are not accurate enough, as illustrated in Sec. 6.2.1. EcoTable is slightly more expensive than WRR because the ReAct reasoning loop in EcoTable involves multiple iterative LLMs calls per transformation. However, the transformation code generated in a single LLM call often produces erroneous code due to a lack of execution feedback and step-by-step reasoning, resulting in a low F1 score of WRR. On the NYC dataset, the cost of EcoTable rises because the join validation algorithm may exhaustively traverse edges to find a valid path in a large search space. To improve scalability, Section 6.4 presents a variant of EcoTable with a sophisticated edge pruning strategy tailored for massive data lakes.

6.3 Integration with Text-to-SQL Systems (RQ2)

We evaluate how our method affects the SQL execution accuracy when integrated with a Text-to-SQL model. We integrate our method

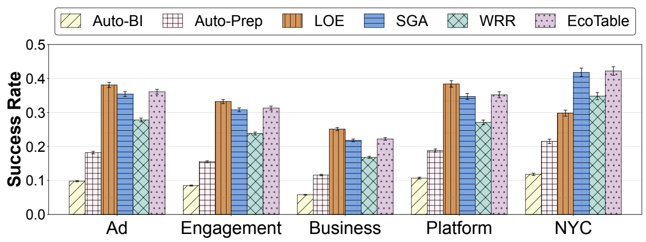

Figure 7: End-to-end SQL Success Rate.

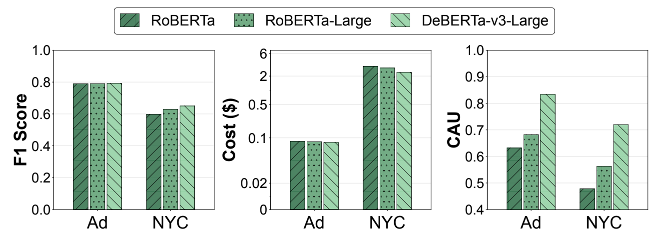

Figure 8: Evaluation of Table Identification Layer.

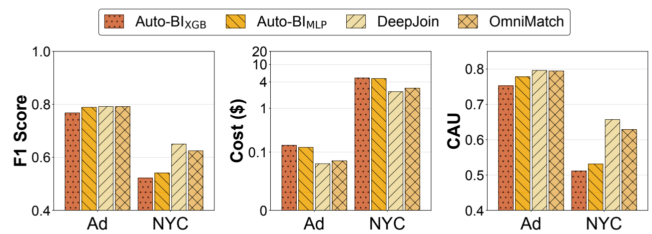

Figure 9: Evaluation of Graph-based Validation Layer.

with variant Text-to-SQL methods, including ReFoRCE [8], MAC-SQL [52], RSL-SQL [4] and Spider-Agent [28]. Due to space limitation, we only illustrate the results on the execution accuracy when integrated with ReFoRCE and put other results into our technical report [56]. Furthermore, we compare `EcoTable` with the following baselines:

(1) `Auto-BI + ReFoRCE (B+R)` combines the join paths identified by `Auto-BI` with ReFoRCE.
(2) `Auto-Prep + ReFoRCE (P+R)` merges the join paths detected by `Auto-Prep` with ReFoRCE.
(3) `LOE + ReFoRCE (L+R)` integrates the join paths found by `LOE` with ReFoRCE.
(4) `SGA + ReFoRCE (S+R)` incorporates the join paths generated by `SGA` with ReFoRCE.
(5) `WRR + ReFoRCE (W+R)` leverages the join paths produced by `WRR` with ReFoRCE.
(6) `EcoTable + ReFoRCE (Eco+R)` uses `EcoTable` to obtain verified join paths for ReFoRCE.

In Figure 7, our approach `Eco+R` achieves a competitive success rate compared to `S+R` and `L+R` across most datasets, but with significantly lower costs as illustrated in Sec. 6.2.2. `Eco+R` outperforms both `B+R` and `P+R` because both methods adopt a deep learning model to estimate pairwise joinability without LLMs validation, thus introducing incorrect join relationships to the Text-to-SQL model. `Eco+R` outperforms `W+R` because its ReAct-based transformations effectively resolve data inconsistencies, providing clean tables that enable the Text-to-SQL model to generate executable and correct answers for the query. Empirically, `L+R` and `Eco+R` achieve similar success rates on DBT benchmarks. However, on the NYC dataset, the performance of `L+R` drops because it introduces many query-irrelevant tables, which confuses the Text-to-SQL model and leads to incorrect join logic in the generated SQL.

6.4 Eval. of Table Identification Layer (RQ3)

We evaluate how different PLMs used in the table identification layer affect the results by comparing multiple state-of-the-art models. In particular, we evaluate `RoBERTa` [36], `RoBERTa-large` [36], and `DeBERTa-v3-Large` [20]. As shown in Figure 8, These three models achieve similar accuracy and cost although `DeBERTa-v3-Large` uses a larger model, because we subsequently leverage the LLM to verify the results, which demonstrates that `EcoTable` is robust on the model used for schema linking.

6.5 Eval. of Graph-based Validation Layer (RQ4)

We conduct a comprehensive evaluation of the Graph-based Validation Layer to validate the effectiveness of our joinability estimation module. Specifically, we compare `EcoTable` against the following methods:

(1) $\texttt{Auto-BI}_{\texttt{MLP}}$ [35] takes a pair of columns, encodes them with 24 handcrafted features and feeds them into a four-layer MLP to produce a probability.
(2) $\texttt{Auto-BI}_{\texttt{XGB}}$ [35] relies on the same set of 24 manually engineered features, but performs classification using a Gradient Boosting Decision Tree estimators.
(3) `OmniMatch` [24] encodes each table column using statistical features and contextual embeddings learned with a Relational Graph Convolutional Network, which aggregates information from similar columns to refine their vectors and then computes distances between these vectors as a joinability probability.
(4) `DeepJoin` [10] converts table columns into textual sequences. It then applies a pre-trained MPNet model to obtain dense vector embeddings and assesses joinability by computing the cosine similarity between them.

We can observe from Figure 9 that `DeepJoin` and `OmniMatch` achieve a higher accuracy than `Auto-BI` because it leverages the pre-trained language model to compute the joinability, while Auto-BI just uses the hand-crafted features. `Auto-BI` also costs more because its low accuracy causes the LLM to check more edges.

6.6 Eval. of Table Transformation Layer (RQ5)

To evaluate the accuracy of our ReAct-style method in the table transformation layer, we conduct a standalone evaluation comparing it against two automated baselines, i.e., `DTT` [6] and `TabulaX` [39] using the Transformation F1-score.

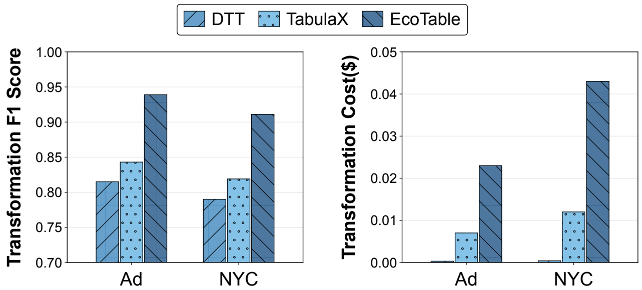

Figure 10: Evaluation of Table Transformation Layer.

As shown in Figure 10, EcoTable achieves a higher Transformation F1-score than both baselines because the ReAct-style reasoning dynamically composes atomic operators to resolve complex inconsistencies, whereas one-shot approaches like DTT and TabulaX struggle with unpredictable structural noise.

7 RELATED WORKS

Text-to-SQL. Modern Text-to-SQL methods primarily follow two paradigms: fine-tuning, which adapts models to specific database schemas [29, 30, 45] and LLM-based prompting, which leverages in-context reasoning without parameter updates [26, 43, 61].As more challenging benchmarks such as Spider 2.0 [28] and BIRD [31] emerge, recent work increasingly focuses on strengthening reasoning. For instance, RSL-SQL [4] targets the semantic gap through schema linking, while MAC-SQL [52] and ReFoRCE [8] employ multi-agent interaction and iterative refinement to address ambiguous queries. Despite these advances, existing approaches assume that the databases that perfectly support user queries exist apriori, while our work targets lightly structured data lakes where no database schema is available beforehand.

Joinable Table Discovery. Discovering joinable tables in data lakes is essential for data integration. Its key component is to compute the joinability between two tables, which is also a significant step in our graph-based validation layer. To be specific, early approaches such as Josie [63] and LSH Ensemble [65] rely on set-level overlap between column values to measure the joinability. Pexeso [9] and DeepJoin [10] leverage semantic embeddings from pre-trained language models to capture fuzzy joinability. OmniMatch [24] encodes each table column using statistical features and contextual embeddings learned with a Relational Graph Convolutional Network, which aggregates information from similar columns to refine their vectors and then computes distances between these vectors as a joinability probability. Besides, schema matching methods [2, 37, 42, 47] could also be utilized to compute the joinability based on the matching scores between two columns. In contrast to these methods, which use a table to identify joinable tables in a data lake, we focus on processing natural language queries to integrate multiple tables within a data lake.

Table Transformation. Table transformation methods fall into two main categories: traditional methods [6, 58, 64] and LLM-based methods [33, 39]. For the former, early approaches like Auto-Join [64] and Auto-Pipeline [59] rely on predefined operators and structural metadata to automate join discovery and pipeline construction, but are constrained by fixed operator sets. DTT [6] instead fine-tunes the PLM in an example-driven way to reformat data values. Among LLM-based methods, TableGPT [33] improves performance on table-related tasks via instruction tuning, while TabulaX [39] classifies columns into four transformation types and applies chain-of-thought reasoning to generate interpretable functions. Unlike prior methods, we directly transform two given tables into a unified format that directly supports equi-joins for the specific join needs of each query.

8 CONCLUSION

We introduced EcoTable, a cost-effective framework for integrating data lake tables via natural language queries. EcoTable uses a graph-based framework, combining lightweight models with LLMs to efficiently narrow the search space and minimize costly LLM calls through prioritized validation. Experiments show EcoTable achieves high integration accuracy but with lower costs.

Limitation. Our main focus is on queries with acyclic join graphs. We also recognize that a table pair in real-world data lakes might have several join relationships. In this case, maximizing joint joinability might identify a PK-FK join, which is not the right one for the given query. Although it is not perfect, maximizing joint joinability is still a reasonable way to identify from a large search space a path candidate that is very likely to successfully answer the query. Otherwise, the search cost will be prohibitive. To alleviate this issue, the identified path is only treated as a candidate. The LLM then checks the join relationships along the selected path, and if an edge is identified as non-joinable, our algorithm iteratively searches for the next candidate path. This iterative verification reduces the probability of producing an invalid path. However, it cannot completely eliminate this issue. An incorrect path may still be returned when it consists of valid PK-FK joins that all pass the LLM verification. Our error analysis shows that such errors account for only 2.7%–7.3% of all errors across the five evaluated datasets. In future work, we plan to compute the joinability score as the average of the top-k pairwise similarities instead of only the maximum pairwise similarity, and further verify whether the whole candidate path, rather than only individual join relationships, can answer the query.

ACKNOWLEDGMENTS

Chengliang Chai is supported by the NSFC (U25B2019, 62472031, 62577010), the National Key Research and Development Program of China (2024YFC3308200), Beijing Nova Program, and Huawei. Yuhao Deng is supported by the NSFC (624B2023). Yuyu Luo is supported by the NSF of China (62402409) and Youth S&T Talent Support Programme of Guangdong Provincial Association for Science and Technology (SKXRC2025461). Ye Yuan is supported by the NSFC (Grant Nos. 62225203, 62532001), Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China(Grant No. JYB2025XDXM108), Beijing Natural Science Foundation(Grant No. L241010). Guoren Wang is supported by the NSFC (62427808, U2001211), and the Liaoning Revitalization Talents Program (XLYC2204005). Lei Cao is supported by the NSF (DBI-2327954) and Amazon Research Awards.

REFERENCES

[1] Anthropic. n.d.. Anthropic API Pricing. https://www.anthropic.com/pricing. [n.d.] Website, accessed Nov. 1, 2025.

[2] David Aumueller, Hong Hai Do, Sabine Massmann, and Erhard Rahm. 2005. Schema and ontology matching with COMA++. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, Fatma Özcan (Ed.). ACM, 906–908. doi:10.1145/1066157.1066283

[3] Sarah Azzabi, Zakiya Alfughi, and Abdelkader Ouda. 2024. Data Lakes: A Survey of Concepts and Architectures. *Comput.* 13, 7 (2024), 183. doi:10.3390/COMPUTERS13070183

[4] Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. RSL-SQL: Robust Schema Linking in Text-to-SQL Generation. *CoRR* abs/2411.00073 (2024). arXiv:2411.00073 doi:10.48550/ARXIV.2411.00073

[5] City of New York. n.d.. NYC Open Data. https://opendata.cityofnewyork.us/. Accessed: 2026-03-01.

[6] 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.

[7] dbt Labs. 2025. dbt. https://www.getdbt.com/ Website, accessed Nov. 1, 2025.

[8] Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. 2025. ReFoRCE: A Text-to-SQL Agent with Self-Refinement, Format Restriction, and Column Exploration. *CoRR* abs/2502.00675 (2025). arXiv:2502.00675 doi:10.48550/ARXIV.2502.00675

[9] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 456–467.

[10] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2023. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. *Proc. VLDB Endow.* 16, 10 (2023), 2458–2470. doi:10.14778/3603581.3603587

[11] EcoTable Contributors. 2025. EcoTable: Ad. https://github.com/yuhuiwang02/EcoTable/tree/main/ad. GitHub repository, accessed Nov. 1, 2025.

[12] EcoTable Contributors. 2025. EcoTable: Business. https://github.com/yuhuiwang02/EcoTable/tree/main/business. GitHub repository, accessed Nov. 1, 2025.

[13] EcoTable Contributors. 2025. EcoTable: Engagement. https://github.com/yuhuiwang02/EcoTable/tree/main/engagement. GitHub repository, accessed Nov. 1, 2025.

[14] EcoTable Contributors. 2025. EcoTable: NYC. https://github.com/yuhuiwang02/EcoTable/tree/main/NYC. GitHub repository, accessed Nov. 1, 2025.

[15] EcoTable Contributors. 2025. EcoTable: Platform. https://github.com/yuhuiwang02/EcoTable/tree/main/platform. GitHub repository, accessed Nov. 1, 2025.

[16] Meihao Fan, Ju Fan, Nan Tang, Lei Cao, Guoliang Li, and Xiaoyong Du. 2025. AutoPrep: Natural Language Question-Aware Data Preparation with a Multi-Agent Framework. *Proc. VLDB Endow.* 18, 10 (2025), 3504–3517. doi:10.14778/3748191.3748211

[17] Meihao Fan, Xiaoyue Han, Ju Fan, Chengliang Chai, Nan Tang, Guoliang Li, and Xiaoyong Du. 2024. Cost-Effective In-Context Learning for Entity Resolution: A Design Space Exploration. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*. IEEE, 3696–3709. doi:10.1109/ICDE60146.2024.00284

[18] Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. 2023. Data lakes: A survey of functions and systems. *IEEE Transactions on Knowledge and Data Engineering* 35, 12 (2023), 12571–12590.

[19] Benjamin Hättasch, Michael Truong-Ngoc, Andreas Schmidt, and Carsten Binnig. 2020. It's AI Match: A Two-Step Approach for Schema Matching Using Embeddings. In *AIDB@VLDB 2020, 2nd International Workshop on Applied AI for Database Systems and Applications, Held with VLDB 2020, Monday, August 31, 2020, Online Event / Tokyo, Japan*, Bingsheng He, Berthold Reinwald, and Yingjun Wu (Eds.). https://drive.google.com/file/d/1xCGhHfghJR3DLuA-9o-XQXBS8M1_8a4P/view?usp=sharing

[20] Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. DeBERTaV3: Improving DeBERTa using ELECTRA-Style Pre-Training with Gradient-Disentangled Embedding Sharing. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=sE7-XhLxHA

[21] Aamod Khatiwada, Roee Shraga, and Renée J. Miller. 2026. Fuzzy Integration of Data Lake Tables. In *Proceedings 29th International Conference on Extending Database Technology, EDBT 2026, Tampere, Finland, March 24-27, 2026*, Wolfgang Lehner, Vanessa Braganholo, Kostas Stefanidis, Zheying Zhang, Alexander Krause, and João Felipe Nicolaci Pimentel (Eds.). OpenProceedings.org, 96–102. doi:10.48786/EDBT.2026.08

[22] Dénes König. 1916. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Math. Ann.* 77, 4 (1916), 453–465.

[23] Lawrence Kou, George Markowsky, and Leonard Berman. 1981. A fast algorithm for Steiner trees. *Acta informatica* 15, 2 (1981), 141–145.

[24] Christos Koutras, Jiani Zhang, Xiao Qin, Chuan Lei, Vassilis N. Ioannidis, Christos Faloutsos, George Karypis, and Asterios Katsifodimos. 2025. OmniMatch: Joinability Discovery in Data Products. *Proc. VLDB Endow.* 18, 11 (2025), 4588–4601. doi:10.14778/3749646.3749715

[25] Eugenie Lai, Yeye He, and Surajit Chaudhuri. 2025. Auto-Prep: Holistic Prediction of Data Preparation Steps for Self-Service Business Intelligence. *Proc. VLDB Endow.* 18, 7 (2025), 2212–2225. https://www.vldb.org/pvldb/vol18/p2212-he.pdf

[26] Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. MCS-SQL: Leveraging Multiple Prompts and Multiple-Choice Selection For Text-to-SQL Generation. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (Eds.). Association for Computational Linguistics, 337–353. https://aclanthology.org/2025.coling-main.24/

[27] Jihyung Lee, Jin-Seop Lee, Jaehoon Lee, YunSeok Choi, and Jee-Hyong Lee. 2025. DCG-SQL: Enhancing In-Context Learning for Text-to-SQL with Deep Contextual Schema Link Graph. (2025), 15397–15412. doi:10.18653/V1/2025.ACL-LONG.748

[28] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. https://openreview.net/forum?id=XmProj9cPs

[29] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 13067–13075.

[30] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–28.

[31] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems* 36 (2023), 42330–42357.

[32] Peng Li, Yeye He, Cong Yan, Yue Wang, and Surajit Chaudhuri. 2023. Auto-Tables: Synthesizing Multi-Step Transformations to Relationalize Tables without Using Examples. *Proc. VLDB Endow.* 16, 11 (2023), 3391–3403. doi:10.14778/3611479.3611534

[33] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2024. Table-gpt: Table fine-tuned gpt for diverse table tasks. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–28.

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

[35] Yiming Lin, Yeye He, and Surajit Chaudhuri. 2023. Auto-BI: Automatically Build BI-Models Leveraging Local Join Prediction and Global Schema Graph. *Proc. VLDB Endow.* 16, 10 (2023), 2578–2590. doi:10.14778/3603581.3603596

[36] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[37] Yurong Liu, Eduardo Peña, Aécio S. R. Santos, Eden Wu, and Juliana Freire. 2025. Magneto: Combining Small and Large Language Models for Schema Matching. *Proc. VLDB Endow.* 18, 8 (2025), 2681–2694. doi:10.14778/3742728.3742757

[38] Arash Dargahi Nobari and Davood Rafiei. 2022. Efficiently Transforming Tables for Joinability. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 1649–1661. doi:10.1109/ICDE53745.2022.00169

[39] Arash Dargahi Nobari and Davood Rafiei. 2025. TabulaX: Leveraging Large Language Models for Multi-Class Table Transformations. *Proc. VLDB Endow.* 18, 11 (2025), 3826–3839. https://www.vldb.org/pvldb/vol18/p3826-nobari.pdf

[40] OpenAI. n.d.. OpenAI API Pricing. https://openai.com/api/pricing. [n.d.] Website, accessed Nov. 1, 2025.

[41] George Papadakis, Ekaterini Ioannou, Emanouil Thanos, and Themis Palpanas. 2021. *The Four Generations of Entity Resolution.* Morgan & Claypool Publishers. doi:10.2200/S01067ED1V01Y202012DTM064

[42] Marcel Parciak, Brecht Vandevoort, Frank Neven, Liesbet M. Peeters, and Stijn Vansummeren. 2025. LLM-Matcher: A Name-Based Schema Matching Tool using Large Language Models. In *Companion of the 2025 International Conference on Management of Data, SIGMOD/PODS 2025, Berlin, Germany, June 22-27, 2025*, Volker Markl, Joseph M. Hellerstein, and Azza Abouzied (Eds.). ACM, 203–206. doi:10.1145/3722212.3725112

[43] Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems* 36 (2023), 36339–36348.

[44] Mattia Di Profio, Mingjun Zhong, Yaji Sripada, and Marcel Jaspars. 2025. FlowETL: An Autonomous Example-Driven Pipeline for Data Engineering. *CoRR* abs/2507.23118 (2025). arXiv:2507.23118 doi:10.48550/ARXIV.2507.23118

[45] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 9895–9901. doi:10.18653/V1/2021.EMNLP-MAIN.779

[46] Nima Shahbazi, Jin Wang, Zhengjie Miao, and Nikita Bhutani. 2024. Fairness-Aware Data Preparation for Entity Matching. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*. IEEE, 3476–3489. doi:10.1109/ICDE60146.2024.00268

[47] Roee Shraga, Avigdor Gal, and Haggai Roitman. 2020. ADnEV: Cross-Domain Schema Matching using Deep Similarity Matrix Adjustment and Evaluation. *Proc. VLDB Endow.* 13, 9 (2020), 1401–1415. doi:10.14778/3397230.3397237

[48] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755* (2024).

[49] The pandas development team. n.d.. Pivot in Python Pandas. https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.pivot.html. Accessed: 2026-02-28.

[50] The pandas development team. n.d.. Python Series Str.split. https://pandas.pydata.org/docs/reference/api/pandas.Series.str.split.html. Accessed: 2026-02-28.

[51] Vadim G Vizing. 1965. The chromatic class of a multigraph. *Cybernetics* 1, 3 (1965), 32–41.

[52] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (Eds.). Association for Computational Linguistics, 540–557. https://aclanthology.org/2025.coling-main.36/

[53] Jin Wang and Yuliang Li. 2022. Minun: evaluating counterfactual explanations for entity matching. In *DEEM '22: Proceedings of the Sixth Workshop on Data Management for End-To-End Machine Learning Philadelphia, PA, USA, 12 June 2022*, Matthias Boehm, Paroma Varma, and Doris Xin (Eds.). ACM, 7:1–7:11. doi:10.1145/3533028.3533304

[54] Jin Wang, Yuliang Li, and Wataru Hirota. 2021. Machamp: A Generalized Entity Matching Benchmark. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong (Eds.). ACM, 4633–4642. doi:10.1145/3459637.3482008

[55] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 2609–2634. doi:10.18653/V1/2023.ACL-LONG.147

[56] Yuhui Wang, Jinqi Liu, Chengliang Chai, Hangyu Zhao, Yuhao Deng, Yuyu Luo, Xin Tang, Ye Yuan, Guoren Wang, Fengjin Wang, and Lei Cao. 2026. EcoTable: Cost-effective Table Integration in Data Lakes for Natural Language Queries. *arXiv preprint arXiv:2606.26613* (2026). https://arxiv.org/abs/2606.26613

[57] Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024. Chain-of-Table: Evolving Tables in the Reasoning Chain for Table Understanding. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=4L0xnS4GQM

[58] Cong Yan and Yeye He. 2020. Auto-Suggest: Learning-to-Recommend Data Preparation Steps Using Data Science Notebooks. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1539–1554. doi:10.1145/3318464.3389738

[59] Junwen Yang, Yeye He, and Surajit Chaudhuri. 2021. Auto-Pipeline: Synthesize Data Pipelines By-Target Using Reinforcement Learning and Search. *Proc. VLDB Endow.* 14, 11 (2021), 2563–2575. doi:10.14778/3476249.3476303

[60] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=WE_vluYUL-X

[61] Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. ACT-SQL: In-Context Learning for Text-to-SQL with Automatically-Generated Chain-of-Thought. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023 (Findings of ACL, Vol. EMNLP 2023)*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 3501–3532. doi:10.18653/V1/2023.FINDINGS-EMNLP.227

[62] Yong Zhang, Jiacheng Wu, Jin Wang, and Chunxiao Xing. 2020. A Transformation-Based Framework for KNN Set Similarity Search. *IEEE Trans. Knowl. Data Eng.* 32, 3 (2020), 409–423. doi:10.1109/TKDE.2018.2886189

[63] 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.

[64] 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.

[65] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *Proc. VLDB Endow.* 9, 12 (2016), 1185–1196. doi:10.14778/2994509.2994534