Auto-Tables: Synthesizing Multi-Step Transformations to Relationalize Tables without Using Examples

Literature Review

COMPUTER SCIENCE

CSP 520 – Data Integration, Warehousing and Provenance

Group - 11

Bavith Kumar Reddy Komtireddy - A20518408

Satya Nandikeswara rao Chinta – A20516065

Kareem Ahaan – A20513492

Under The Guidance of

Prof. Boris Glavic

Date: 11/30/2023

Illinois Institute of Technology
Chicago, Illinois, United States of America

Abstract

In traditional databases, tables are structured in a specific way: each row represents an entity (like a person or an item), and each column provides information about that entity (such as age or price). However, not all tables conform to this standard format, particularly those encountered in real-world situations like spreadsheets or on the web.

This review paper explores the structure of tables in traditional databases, where each row corresponds to a unique entity, and each column represents a specific attribute associated with that entity, like age or price. However, real-world tables, as encountered in spreadsheets and on the web, often deviate from this standard format.

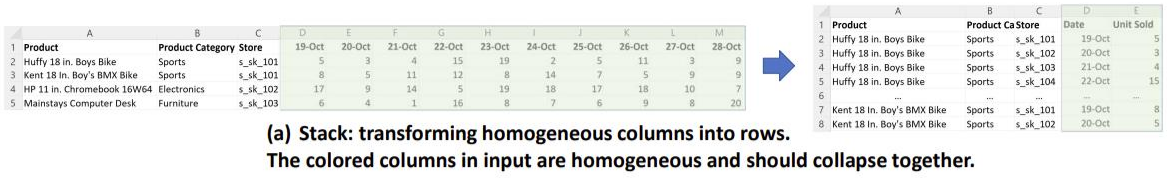
Our investigation reveals that over 30% of these unconventional tables do not adhere to standard relational rules. This non-conformity poses challenges for widely used database tools like SQL, requiring users to manually implement intricate transformations to make these tables compatible. The prevalence of inquiries on platforms such as Stack Overflow and Excel/Tableau forums attests to the complexity of this manual process.

To address these challenges, Auto-Tables is a system designed to automatically generate pipelines with multi-step transformations in programming languages like Python. These transformations convert non-standard tables into the standard relational format, eliminating the need for users to manually program these intricate changes. Extensive testing involving 244 real-world cases from user spreadsheets and forums demonstrate that Auto-Tables successfully transforms over 70% of these cases rapidly and without user intervention. This establishes Auto-Tables as an effective tool for both technical and non-technical users seeking to prepare their data for analysis.

Contents

1. Introduction

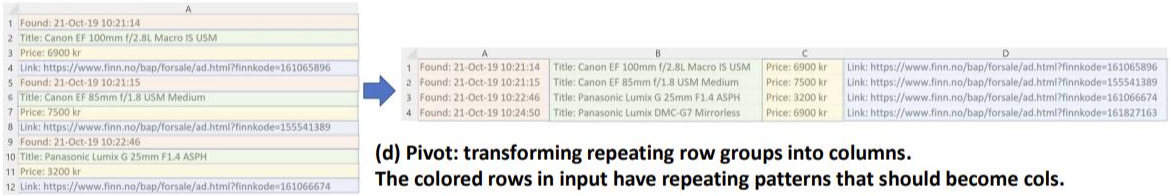
Modern data analytics tools, such as SQL and BI, rely on a standard format of relational tables. In this format, each row represents a specific "entity," and each column corresponds to an "attribute" containing consistent data values for those entities. However, a significant number of tables in real-world scenarios, like spreadsheets and web tables, deviate from this standard, making them challenging to query using SQL-based tools.



(a) Stack: transforming homogeneous columns into rows.
The colored columns in input are homogeneous and should collapse together.

(b) Wide-to-long: transforming repeating column groups into rows.
The colored col-groups in input have repeating patterns and should collapse.

(c) Transpose: transforming rows to columns and vice versa.
The colored rows in input have homogeneous content in the horizontal direction.



(d) Pivot: transforming repeating row groups into columns.
The colored rows in input have repeating patterns that should become cols.

Figure 1: Example input/output tables for 4 operators in AUTO-TABLES: (a) Stack, (b) Wide-to-long, (c) Transpose, (d) Pivot. The input-tables (on the left) are not relational and hard to query, which need to be transformed to produce corresponding output-tables (on the right) that are relational and easy to query. Observe that the color-coded, repeating row/column-groups are "visual" in nature, motivating a CNN-like architecture like used in computer vision for object-detection.

Year	1st highlighter	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
2020	United States 20,936.600	China 14,772.731	Japan 5,064.873	Germany 3,806.060	United Kingdom 2,707.744	India 2,622.984	France 2,603.004	Italy 1,886.445	Canada 1,643.408	South Korea 1,630.525
2015	United States 18,036.650	China 11,226.186	Japan 4,382.420	Germany 3,365.293	United Kingdom 2,863.304	France 2,420.163	India 2,088.155	Italy 1,825.820	Brazil 1,801.482	Canada 1,552.808

Race of mother	Number of births in 2016	% of all born	TFR (2016)	Number of births in 2017	% of all born	TFR (2017)	Number of births in 2018	% of all born	TFR (2018)	Number of births in 2019	% of all born	TFR (2019)
White	2,900,933	73.5%	1.77	2,812,267	72.9%	1.76	2,788,439	73.5%	1.75			
> NH White	2,056,332	52.1%	1.719	1,992,461	51.7%	1.666	1,956,413	51.6%	1.640	1,915,912	51.1%	1.611
Black	623,886	15.8%	1.90	626,027	16.2%	1.92	600,933	15.8%	1.87			

	Country (or area)	1970	1971	1972	1973	1974	1975	1976
1	Afghanistan *	1,749	1,831	1,596	1,733	2,156	2,367	2,556
2	Albania *	2,266	2,331	2,398	2,467	2,537	2,610	2,686
3	Algeria *	5,167	5,376	7,193	9,250	13,290	15,591	17,790

Unit		Goldsmith 1984[44]	Hopkins 1995/96[45]	Temin 2006[46]	Maddison 2007[47]	Milanovic 2007[48]	Bang 2008[49]	Scheidel/Friesen 2009[50]
Approx. year		14 AD	14 AD	100 AD	14 AD	14 AD	14 AD	150 AD
GDP (PPP) per capita in	Sesterces	HS 380	HS 225	HS 166	HS 380	HS 380	HS 229	HS 260
	Wheat equivalent	843 kg	491 kg	614 kg	843 kg	–	500 kg	680 kg
	1990 International Dollars	–	–	–	$570	$633	–	$620

Figure 2: Real Web tables from Wikipedia that are also non-relational, similar to the spreadsheet tables shown in Figure 1.

In our random sampling of user spreadsheets (in Excel) and web tables (from Wikipedia), approximately 30-50% of tables exhibit such issues. Real examples in Figures 1 and 2 illustrate

common problems. It's important to note that this problem is widespread due to the millions of such tables present in spreadsheets and on the web.

For example, Figure 1(a) shows a non-standard relational table on the left, where each green-marked column contains sales numbers for a single day, creating horizontal homogeneity. While this format is convenient for humans to observe day-to-day changes, it is challenging to analyze using SQL. Transforming this table, as shown on the right, simplifies it by consolidating the homogeneous columns into "Date" and "UnitsSold," making queries more manageable.

Other examples, like Figure 1(b) and Figure 1(c), demonstrate tables with repeating column groups and attributes represented in rows, respectively, requiring different transformation operators ("wide-to-long" and "transpose"). These transformations enhance the tables for SQL-based analysis. Figure 2 displays similar structural issues in web tables, emphasizing the need for transformations before effective querying.

Table 1: Auto-Tables DSL: table-restructuring operators and their parameters to "relationalize" tables. These operators are common and exist in many different languages, like Python Pandas and R, sometimes under different names.

DSL operator	Python Pandas equivalent	Operator parameters	Description (example in parenthesis)
stack	melt [18]	start_idx, end_idx	collapse homogeneous cols into rows (Fig. 1a)
wide-to-long	wide_to_long [22]	start_idx, end_idx, delim	collapse repeating col-groups into rows (Fig. 1b)
transpose	transpose [21]	-	convert rows to columns and vice versa (Fig. 1c)
pivot	pivot [19]	repeat_frequency	pivot repeating row-groups into cols (Fig. 1d)
explode	explode [16]	column_idx, delim	convert composite cells into atomic values
ffill	ffill [17]	start_idx, end_idx	fill structurally empty cells in tables
subtitles	copy, ffill, del	column_idx, row_filter	convert table subtitles into a column
none	-	-	no-op, the input table is already relational

To address these challenges, Table 1 outlines eight common transformation operators needed to make non-relational tables compatible with SQL-based analysis. These operators include "stack," "wide-to-long," "transpose," and "pivot," among others. Understanding and applying these operators are crucial for making diverse tables amenable to relational analysis.

Table 1 comprises two columns: one with the name of a "logical operator" used in different programming languages (like Python or R), and the second with the equivalent operator in the popular Python library called Pandas , widely used by developers and data scientists.

Even though functionalities similar to those in Table 1 exist in languages like R and Python, using them correctly is not easy for users. Here's why:

1. Users must visually spot structural issues in a table that make querying difficult, which isn't obvious to non-experts.
2. Users need to connect the identified pattern to the right operator in Table 1, a challenge since they may not be familiar with specific terms (like pivot or stack).
3. Users must set up the chosen operator correctly, using parameters tailored to the input table, like deciding which columns to collapse into rows. This is tough, even for developers who often need to consult complex API documentation.
4. Some tables require more than one transformation step, meaning users must repeat steps (1)-(3) multiple times.

Even technical users, like developers, struggle with these steps, as seen in numerous questions on forums like Stack Overflow. Figure 3 provides an example where a developer seeks help on Pandas operators, emphasizing the difficulty even for those with technical expertise.

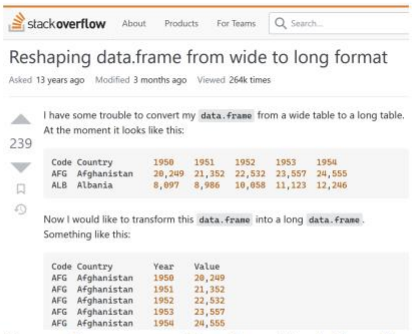


Figure 3: Example user question from StackOverflow, on how to restructure tables. Questions like this are common not only among technical users, but also non-technical users, as similar questions are commonly found on forums for Excel, Power-BI, and Tableau users too [6–9].

If technical users find it challenging to restructure tables, it's no surprise that non-technical enterprise users, dealing with tables in spreadsheets, face even greater difficulty. Similar questions on Excel and Tableau forums highlight user complaints about the complexity of data analysis without the necessary transformations.

The abundance of these questions underscores that restructuring tables is a common challenge for both technical and non-technical users.

Auto-Tables: Transforming Tables Without Examples. In this work, they introduced a new approach to automatically transform tables into a standard format using a set of operations listed in Table 1. Importantly, the approach doesn't require users to provide specific examples of what they want. The idea is that, given a table, the necessary steps to make it fit the standard format are usually unique and clear. This is because our transformations only "restructure" tables without changing the content, unlike other methods that alter the table content and require examples to show the desired outcome.

It's crucial not to ask users for examples because it would mean users have to describe an entirely new table, which can be a lot of work.

As humans, we can visually recognize patterns in rows and columns (like color-coded groups in Figure 1) to predict which operation to use. The question explored in this paper is whether an algorithm can "learn" to recognize these patterns by just looking at the input tables, similar to how computer-vision algorithms identify more complex objects in pictures.

Unlike hand-written rules for computer vision, our table patterns are data-dependent and subtle, making it hard to write rules for them. For example, imagine a table like the one in Figure 1(b) with three different groups of columns. Writing rules to differentiate them becomes challenging when the columns all have similar-looking numbers. Tests were conducted with a basic approach

using rules, and it didn't work well due to these subtle differences in the data. This led to the development of a learning-based method tailored to our task.

In computer vision, algorithms are usually trained using labeled data (like pictures of dogs labeled as dogs). Unfortunately, we don't have such labeled data for our task. To overcome this, created a unique self-training system that uses the inverse relationships between operations (e.g., the inverse of "stack" is "unstack") to automatically generate a large amount of training data. The process is described in Figure 6.

With the training data generated, built an Auto Tables system that can "learn to synthesize" table-restructuring transformations. Used a deep tabular model inspired by architectures popular in computer vision. Our approach is effective, solving over 70% of real-world test cases from user forums and spreadsheets with interactive sub-second response times.

2. Related Work

The use of examples in program synthesis has been widely studied, focusing on input/output transformations. One set of methods concentrates on "row-to-row" transformations, where each input row corresponds to a single output row. Examples include TDE and FlashFill. However, these differ from AutoTables' table-restructuring transformations, which involve operators listed in Table 1, capable of altering table structures. Another set of transformations deals with "table-to-table" operators, like Foofah, and SQL-by-example techniques such as PATSQL, QBO, and Scythe. While these consider some table-restructuring operators, they necessitate users to provide an example output table, which can be a substantial effort.

In the realm of computer vision, notable progress has been made in object detection using CNN architectures to extract vital visual features from images. Given the visual nature of our table transformation problem (as shown in Figure 1) and the resemblance between pixels in images and rows/columns in tables, forming two-dimensional rectangles, our model architecture draws inspiration from CNN architectures for object detection, tailored specifically for our table transformation task.

Various techniques have been proposed to represent tables using deep models, such as TaBERT, Tapas, and Turl. However, most of these concentrate on the natural-language aspects of tables, tailored to NL-related tasks like NL-to-SQL and entity-linking. These approaches are not suitable for our table-transformation task, which demands exploiting the structural homogeneity of tables, such as cell similarity in row/column directions.

In classical database research, schema design often involves normalizing or decomposing large tables into smaller ones to satisfy relational "normal forms" (3NF, BCNF, etc.), improving storage efficiency and preventing update anomalies. In contrast, our work centers on restructuring an input table to enhance query ease, focusing on single-table-to-single-table transformations. Consequently, our approach is both independent of and complementary to schema design; if necessary for storage in databases, our transformed table can undergo schema-design steps.

3. Preliminary and Problem

In this section, we will introduce the operators considered for restructuring tables and explain the synthesis problem.

3.1. Table-Restructuring Operators

Identified 8 table-restructuring operators in the DSL, outlined in Table 1. After analyzing real-world tables in spreadsheets and on the web, found that these operators cover the majority of scenarios required to make tables suitable for relational analysis. It's important to note that the synthesis framework, utilizing self-supervision for training, isn't bound to specific operator choices. This flexibility enables us to easily incorporate additional operators for new functionalities.

Operators and Parameters

1. Stack
    - Operator: Pandas Stack (also known as melt and unpivot in other contexts)
    - Function: Collapses contiguous blocks of homogeneous columns into two new columns.
    - Example: Figure 1(a) illustrates the transformation.
    - Parameters: Requires start_idx and end_idx, specifying the starting and ending column index of the homogeneous column group to be collapsed.
2. Wide-to-Long
    - Operator: Pandas Wide-to-Long (similar functionality in R)
    - Function: Collapses repeating column groups into rows.
    - Example: Figure 1(b) showcases the transformation.
    - Parameters: Requires start_idx, end_idx, and "delim" (delimiter used to split original column headers).
3. Transpose
    - Operator: Transposes rows to columns and columns to rows.
    - Example: Figure 1(c) demonstrates the transformation.
    - Parameters: No parameters are needed, as all rows and columns will be transposed.
4. Pivot
    - Operator: Converts rows to columns (with repeating groups).

- Example: Figure 1(d) displays the transformation.
- Parameters: Requires "repeat_frequency" to specify the frequency at which rows repeat in the input table.

Additional Operators

There are 4 more table-restructuring operators listed in Table 1:

- Explode : Converts columns with composite values into atomic values.
- Ffill : Fills values in structurally empty cells for better querying.
- Subtitle: Converts rows representing table subtitles into separate columns.
- None: For input tables already relational, no transformation is needed.

This comprehensive set of operators enables us to handle various table-restructuring scenarios, making tables more suitable for relational analysis.

3.2. Problem Statement

Let's delve into the matter at hand. We've got a table, and our goal is to modify it using specific operations like "stack," "transpose," and "pivot." These operations have precise methods of altering the table, and our objective is to determine the correct sequence of these operations, along with their specific settings, to create a more organized and user-friendly table.

Here's a detailed breakdown:

Task Overview: We begin with a table T and a set of operations (such as stack, transpose, pivot, etc.), each with its unique set of settings or parameters. The task involves finding a sequence of steps. When these steps are applied consecutively to the table T, it transforms into a more organized and relational format.

Key Challenge: The challenge lies in predicting the right operation and its exact settings for each step. This complexity arises from the multitude of possibilities. Even for a single step, there could be thousands of options. For instance, using "stack" on a table with 50 columns gives us 50x50 = 2500 potential parameter combinations for start_idx and end_idx. For a two-step transformation, the possibilities skyrocket into millions. A slight variation in parameters can result in an incorrect transformation.

Figure 4: An example input table (on the left) that requires two transformation steps to relationalize: (1) a "`transpose`" step to swap rows and columns, (2) a "`stack`" step to collapse homogeneous columns (C to H) into two new columns. The resulting output table (on the right) becomes substantially easier to query with SQL (e.g., to filter and aggregate).

Illustrative Example: Consider the input table *T* in Figure 4. The accurate transformation requires two steps: first, a "transpose" to interchange rows with columns, followed by a "stack" to merge similar columns. Any misstep in the order or using slightly different parameters, like changing the start_idx from "2015" to "2016," could yield an inaccurate outcome.

Before presenting solutions using a specific syntax, like the DSL (Domain Specific Language), these can be effortlessly translated into various languages such as Python Pandas or R for practical implementation. It's also essential to note that different-looking programs might convey the same meaning and can verify their equivalence based on a set of rules.

Significant Challenge: This task is intricate due to the vast number of possibilities, and even minor errors can lead to inaccurate transformations.

4. Auto Tables

Let's delve into the Auto-Tables system, a tool designed to learn and create transformations. An overview of its architecture, breaking down its key components.

4.1 Architecture Overview:

The overall architecture is showcased in Figure 5, illustrating two primary phases: offline training and online inference.

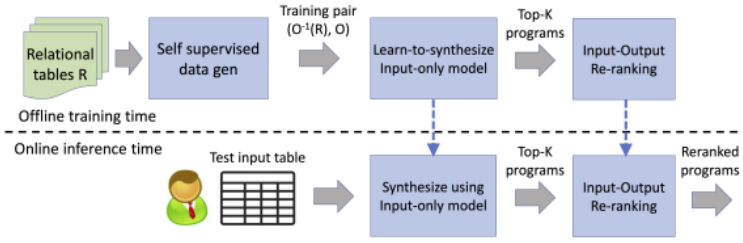


Figure 5: Architecture overview of AUTO-TABLES

Offline Training Time: During offline training, Auto-Tables relies on three main components:

1. Training Data Generation: This component processes extensive sets of relational tables (represented as *R*) to generate pairs of examples and labels.
2. Input-Only Synthesis Module: This module hones the skill of synthesis using the generated training data.
3. Input-Output Reranking Module: This module considers both the input table and the output table (resulting from the synthesized program) to pinpoint the most probable program.

Online Inference Time: The online inference process mirrors the offline steps. The last two blue boxes in the figure represent the models trained offline. When users provide an input table, it passes through the input-only synthesis model to identify the top-*k* candidate programs. These candidates then undergo re-ranking by the input-output model for the final predictions.

Now, let's delve into the details of each of these three modules.

4.2 Self-Supervised Training Data Generation

As previously discussed, patterns in input tables, like repeating column groups and row groups, provide cues for predicting necessary transformations. These patterns, being "visual," are comparable to what computer-vision algorithms handle.

However, unlike tasks in computer vision with abundant labeled data (such as ImageNet with image-label pairs), our synthesis task lacks existing labeled data. Manually labeling tables from scratch on a large scale would be costly and impractical.

To overcome this challenge, a unique self-supervision framework leveraging the inverse relationships between operators is proposed. This approach automatically generates substantial training data without the need for human labels.

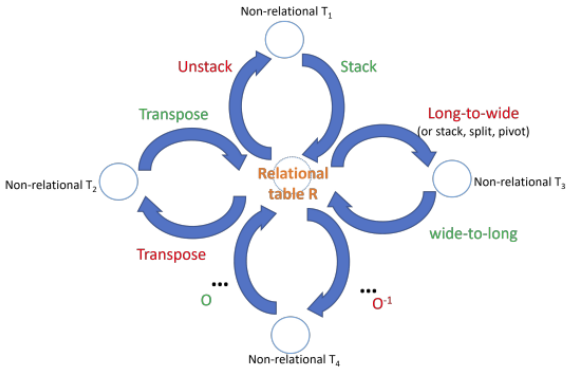


Figure 6: Leverage inverse operators to generate training data. In order to learn-to-synthesize operator O, we can start from any relational table R, apply its inverse operator O^{-1} to obtain $O^{-1}(R)$. Given $T = O^{-1}(R)$ as an input table, we know O must be its ground-truth transformation, because $O(O^{-1}(R)) = R$.

Algorithm 1: Auto-gen training examples

input : DSL operators **O**, large collections of relational tables **R**
output : Training table-label pairs: (T, O_p)

$E \leftarrow \{\}$
foreach O *in* **O** **do**
  foreach R *in* **R** **do**
    foreach R' *in Augment(R)* // Crop rows and columns
    do
      $p \leftarrow$ sample valid parameter from space $P(O)$
      $O_{p'}^{-1} \leftarrow$ construct the inverse of O_p
      $T \leftarrow O_{p'}^{-1}(R')$
      $E \leftarrow E \cup \{(T, O_p)\}$
return all training examples E

Here's how it works

Inverse Operators: For each operator O in the system, identify its inverse operator (or create a sequence of steps equivalent to its inverse), denoted as $O-1$.

Data Generation: To create a training example for an operator O (e.g., "stack"), we take any relational table R, apply the inverse $O-1$ (e.g., "unstack"), and generate a non-relational table $T = O-1(R)$. Given T as input, we know that O must be its correct transformation.

Scalability: Such examples can be generated at scale by sampling different relational tables R, various operators O, and different parameters for each O. This effectively addresses the lack of data issues in Auto-Tables.

Data Augmentation: We also employ a technique called data augmentation, commonly used in computer vision. This involves:

1. Cropping: Randomly sampling contiguous blocks of rows and columns in a relational table R to create a new table R'.
2. Shuffling: Randomly reordering the rows/columns in R to produce a new R'.

Starting with over 15,000 relational tables and creating around 20 augmented tables for each, enhancing the diversity of the training data, thereby improving the overall performance of the model. This mirrors the effectiveness observed in computer vision tasks that use augmented images.

4.3 Input-Only Synthesis

After accumulating a significant amount of training data in the (T,Op) format via self-supervision, let's explore our "input-only" model. This model is designed to take T as input and forecast an appropriate transformation Op.

4.3.1 Model Architecture

Our model draws inspiration from computer vision principles, intricately fashioned for the task of synthesizing transformations from tables. The architecture, depicted in Figure 7, encompasses four layers

1. Table Embedding Layers: These layers encode each cell in T into a vector, capturing semantic and syntactic features. Semantic features leverage the pre-trained Sentence-BERT, while syntactic features involve predefined attributes such as data types and string lengths. This results in a $n \times m \times 423$ tensor for an input table T with n rows and m columns.

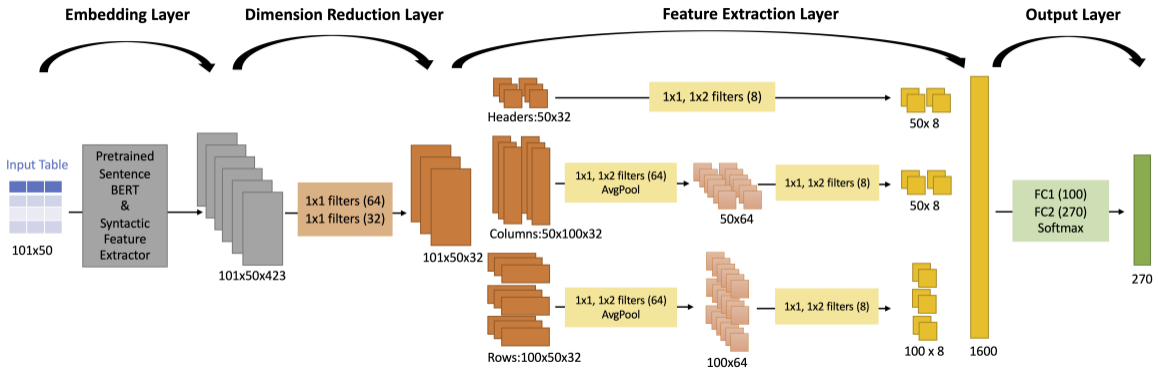


Figure 7: Input-only synthesis: model architecture.

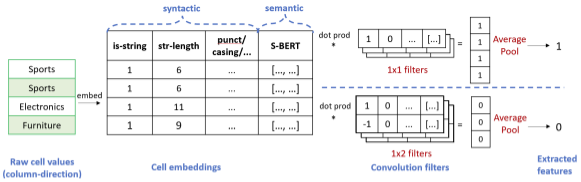
2. Dimension Reduction Layers: To handle the dimensionality of the initial representation, two convolution layers with 1 × 1 kernels are employed, reducing the tensor from 423 to 32 dimensions.

3. Feature Extraction Layers: Inspired by CNN but customized for tabular tasks, these layers focus on crucial signals:

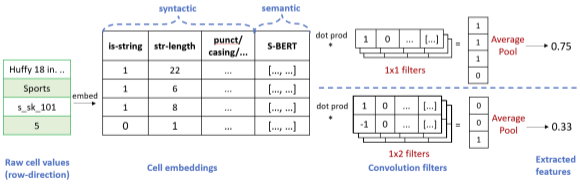
    - Identifying if values in rows or columns are "similar" enough to be "homogeneous."
    - Recognizing if entire rows or columns exhibit repeating patterns.

Utilizing 1x2 and 1x1 convolution filters followed by average pooling in both row- and column-directions, the goal is to effectively represent rows, columns, and headers.

Example: A simplified example illustrates how 1x1 and 1x2 filters extract meaningful information from input cells, aiding in recognizing homogeneity and patterns.



(a) Feature extraction for the input table in Figure 1(a), Column-B



(b) Feature extraction for the input table in Figure 1(a), Row-2

Figure 8: Example feature extraction using 1x1 and 1x2 filters

4. Output Layers: Comprising two fully connected layers followed by softmax classification, these layers generate an output vector encoding the predicted operator type and its parameters. With 8 possible operator types in our DSL, represented as an 8-dimensional one-hot vector, additional bits in the vector encapsulate parameters. The output vector of

270 dimensions facilitates simultaneous predictions of operator type and parameters for a given T.

Normalization into a probability distribution is achieved through standard softmax functions for each prediction vector.

This model architecture aligns with our goal of predicting suitable transformations based on input tables. Subsequent sections will delve into the intricacies of each layer for a comprehensive understanding.

4.3.2 Training and Inference

Training Time: Loss Function

To train our model (illustrated in Figure 7), employ a loss function. For a given input table T with the true operator O and corresponding parameters $P = (p1, p2, ...)$, the model predicts distributions $O\hat{}$ and $P\hat{} = (p\hat{}1, p\hat{}2, ...)$. The training loss on T sums the losses on all predictions, covering both the operator type and relevant parameters:

$$Loss(T) = L(O, \hat{O}) + \sum_{p_i \in P, \hat{p}_i \in \hat{P}} L(p_i, \hat{p}_i) \qquad (1)$$

Here, $L(y,y\hat{})$ represents cross-entropy loss [46] common in classification tasks, defined as:

$$L(y, \hat{y}) = -\sum_{i=1}^{n} y_i log(\hat{y}_i) \qquad (2)$$

With a large training dataset T generated through self-supervision, train the Auto-Tables model (H) by minimizing the overall training loss $\sum_{T \in T} Loss(T)$using gradient descent until convergence.

Inference Time: Synthesizing Transformations

During inference, given an input T, our trained model (H) produces probabilities for potential candidate steps OP instantiated with operator O and parameters $P = (p1, p2, ...)$, denoted as $Pr(OP|T)$:

$$Pr(O_P|T) = Pr(O) \cdot \prod_{p_i \in P} Pr(p_i) \qquad (3)$$

Using these predicted probabilities, determining the most likely transformation step O^*_P given T is straightforward:

$$O_P^* = \arg\max_{O,P} Pr(O_P|T) \qquad (4)$$

This yields the most likely one-step transformation for T. However, some tables may require multiple transformation steps.

To synthesize multi-step transformations, follow the predictions step by step until no suitable transformation is found. Initially, find the most likely transformation $O^1{}_p$ for T, apply it to T, and then use the resulting table as input for the next prediction, repeating this process iteratively. Continue until a "none" transformation is predicted, indicating the input table is already relational and needs no further transformation.

Algorithm 2: Multi-step pipeline synthesis by top-k search

input : Auto-Tables model H, input table T
output : Top-k predicted pipelines by probabilities: M_1, M_2 ... M_k

$Cands = [\,], M \leftarrow [\,], M.prob = 1$ // initialize
$B_{cur} \leftarrow [(T, M)]$
for $i = 1, 2, \ldots L$ **do**
  $B_{next} \leftarrow [\,]$
  foreach (T, M) *in* B_{cur} **do**
    $\hat{O_{p1}}, \hat{O_{p2}}., .\hat{O_{pk}} \leftarrow H(T)$ // top k predictions
    for $j = 1, 2, \ldots k$ **do**
      $T_{next} \leftarrow \hat{O_{pj}}(T), M_{next} \leftarrow M.append(\hat{O_{pj}})$
      $M_{next}.prob \leftarrow M.prob \times \hat{O_{pj}}.prob$
      if $\hat{O_{pj}} = none$ **then**
        $Cands.append(M_{next})$
      else
        $B_{next}.append((T_{next}, M_{next}))$
  sort B_{next} by $M.prob$, $B_{cur} \leftarrow B_{next}[:k]$
Sort $Cands$ by $M.prob$
return $Cands[:k]$

The algorithm above outlines a basic approach for multi-step synthesis, considering only the top-1 choice at each step. Generally, we need to consider the top-k choices at each step to find the most likely multi-step transformations overall. This is achieved through a general search procedure, such as beam search [46].

The algorithm starts with an empty pipeline M and the original input table T. At each iteration, it uses the model (H) on the top-k output tables from the last iteration to obtain the top k candidate operators. The predicted transformations are applied to expand each M with one additional predicted step. The probability of each expanded pipeline is computed as the product of the probabilities of its operators. If a predicted operator is "none," the pipeline is saved as a candidate. The algorithm keeps the current pipeline in the beam for further search if a terminal state is not reached. At the end of each iteration, it ranks all partial pipelines by probabilities and retains only the top k pipelines with the highest probability. The search terminates after a total of L steps, and the top-k pipelines with the highest probabilities are returned as output.

Analyzing Algorithm 2 through a concrete example in Example 3, let's revisit Example 1. Take a look at a table T illustrated on the left side of Figure 4. Using the trained model H, we make predictions for two potential transformations: $O1$, involving a "transpose" operation with a 50%

likelihood, resulting in an output table $O1(T)$, and $O2$, indicating a "stack" operation with specific parameters (start-idx = Col-B, end-idx=Col-E), also with a 50% chance, leading to $O2(T)$. Both of these initial transformations, $\{O1, O2\}$, are taken into account, and the exploration continues toward potential second steps.

Moving along the path of $O1$, where $O1(T)$ becomes the new input table, the top two anticipated steps are $O3$, a "stack" operation (start-idx = Col-C, end-idx=Col-E) with an 80% probability, and $O4$, a "none" transformation, with a 10% probability. Conversely, by following the path of $O2$ and using $O2(T)$ as the new input, we generate its top two predictions. This results in a total of 2×2 = 4 potential 2-step transformations. Then choose the top two with the highest probabilities to extend our search to 3 steps and so forth.

The resulting multi-step transformations are ordered based on their probabilities. Significantly, $\{O1, O3\}$ emerges as the most likely transformation, with a probability calculated as 0.5 * 0.8 = 0.4. This precisely corresponds to the desired transformation detailed in Example 1.

This approach ensures a thorough exploration of possibilities and provides a range of high-probability multi-step transformations. The process involves careful consideration of various candidate pipelines to arrive at the most likely transformations for a given input table.

The algorithm's effectiveness is demonstrated using a detailed example, showcasing the step-by-step selection of transformations in a realistic scenario. This comprehensive methodology ensures robust synthesis of multi-step transformations, aligning with the model's training and inference objectives.

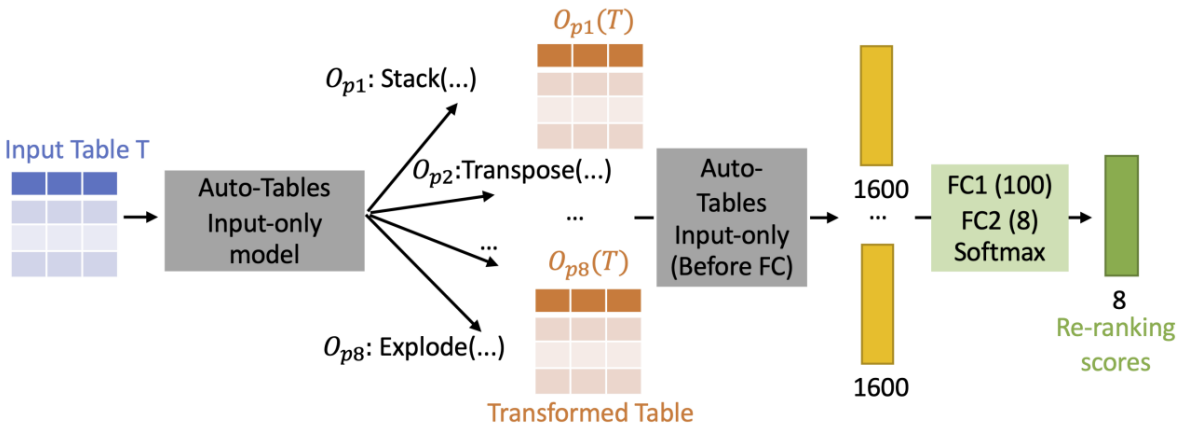


Figure 9: Input/output re-ranking: model architecture.

4.4 Input/Output Re-ranking

Until now, our synthesis model has solely focused on the input table T to forecast transformations M. However, at times, this approach falls short because insights from the output table, $M(T)$, can offer valuable signals. Let's clarify this with an example.

Example 4: In Example 3, relying solely on the input T in Figure 4, our model suggests both $O1$ "transpose" and $O2$ "stack" as potential choices, each with a probability of 0.5. However, "stack" received a higher ranking, and this might be inaccurate. Simply by examining T, "stack" appears plausible because T has numerous similar columns (Col-B to E), aligning with the expected pattern for "stack" as shown in Figure 1(a).

To refine the prediction between $O1$ and $O2$, execute both operations on T and examine the resulting outputs $O1(T)$ and $O2(T)$. A closer look at $O1(T)$ reveals similarities in values within the same columns, while $O2(T)$ (using "stack") blends values like "ES" and "MS" (from "GroupID") with integers in the same columns, which is suboptimal. Our model identifies and penalizes such less-than-ideal situations. Scrutinizing the outputs $O1(T)$ and $O2(T)$ enables us to accurately re-rank $O1$ as a more probable transformation than $O2$, a task challenging when the model relies only on T.

This realization prompts the development of an "input/output-based" re-ranking model (Figure 9). After the input-only synthesis model (Section 4.3) produces the top-k likely operators $\{Opi, i \in [k]\}$, the re-ranking model evaluates all output transformed tables $\{Opi(T), i \in [k]\}$ to generate a re-ranking score for each, indicating the operator's suitability based on the output transformed tables.

Following the approach of the input-only model, each transformed table undergoes conversion into a feature vector using table embedding, dimension reduction, and feature extraction layers. Leveraging the well-trained input-only model, reuse the architecture and weights of these layers. Subsequently, concatenate the feature vectors of all transformed tables and utilize fully connected layers, followed by a softmax function, to produce a k-dimensional vector as re-ranking scores.

For training, the re-ranking task is treated as a classification task, predicting the ground truth among the k transformed tables. The training loss is computed using cross-entropy loss, and the re-ranking model is trained using the same data generated from self-supervision in Section 4.2.

Table 2: Details of ATBench Benchmark

	Forum	Notebook	Excel	Web	Total
Single-Step	**23**	**75**	**65**	**55**	**218**
- transpose	0	11	11	6	28
- stack	10	20	2	24	56
- wtl	6	24	1	3	34
- explode	2	17	14	15	48
- ffill	0	0	11	7	18
- pivot	5	3	0	0	8
- subtitle	0	0	26	0	26
Multi-Step	**0**	**4**	**21**	**1**	**26**
Total	**23**	**79**	**86**	**56**	**244**

5. Experiment

Thorough assessments were conducted of various algorithms using real-world data, demonstrating that our method surpasses baseline techniques in both quality and efficiency. The labeled benchmark data employed for this evaluation is publicly accessible on GitHub, facilitating future research.

5.1. Experimental Setup

Our analysis is based on the ATBench benchmark, designed to evaluate the performance of our method in practical scenarios. This benchmark integrates real cases from online user forums, Jupyter notebooks, and actual spreadsheet and web tables.

- Forums: Utilized 23 questions from StackOverflow and Excel user forums, where users present input/output tables to illustrate their restructuring requirements.
- Notebooks: Our sample includes 79 table-restructuring steps from Jupyter Notebooks, with transformations implemented by data scientists serving as the ground truth.
- Excel+Web: A total of 56 real web tables and 86 spreadsheet tables are sampled, and the desired transformations are manually documented as the ground truth.

In total, ATBench encompasses 244 test cases, with 26 cases involving multi-step transformations.

Table 3: Quality comparison using Hit@k, on 244 test cases

Method	No-example methods				By-example methods			
	Auto-Tables	TaBERT	TURL	GPT-3.5-fs	FF	FR	SQ	SC
Hit @ 1	**0.570**	0.193	0.029	0.196	0.283	0.336	0	0
Hit @ 2	**0.697**	0.455	0.071	-	-	-	0	0
Hit @ 3	**0.75**	0.545	0.109	-	-	-	0	0
Upper-bound	-	-	-	-	0.471	0.545	0.369	0.369

Table 5: Ablation Studies of Auto-Tables

Method	Full	No Re-rank	No Re-rank &				
			No Aug	No Bert	No Syn	1x1 Only	5x5
Hit@1	**0.570**	0.508	0.463	0.467	0.504	0.471	0.480
Hit@2	**0.697**	0.652	0.582	0.627	0.648	0.607	0.594
Hit@3	**0.75**	0.730	0.656	0.693	0.676	0.652	0.660

using the standard $Hit@k$ metric [51], defined as:

$$Hit@k(T) = \sum_{i=1}^{k} \mathbf{1}(\hat{M}_i(T) = M_g(T))$$

Evaluation Metrics

Our evaluation focuses on assessing algorithm performance in terms of quality and efficiency.

- Quality: To determine the synthesis success rate compared the predicted transformations with ground-truth transformations. The average success rate across all test cases ($Hit@k$) is reported for k up to 3.

- Efficiency: The synthesis latency is reported using wall-clock time.

Methods Compared

Table 4: Synthesis latency per test case

Method	Auto-Tables	Foofah (excl. 110 timeout cases)	FlashRelate (excl. 91 timeout cases)
50 %tile	**0.127s**	0.287s + human effort	3.4s + human effort
90 %tile	**0.511s**	22.891s + human effort	57.16s + human effort
95 %tile	**0.685s**	39.188s + human effort	348.6s + human effort
Average	**0.224s**	5.996s + human effort	59.194s + human effort

Table 6: Sensitivity to different semantic embeddings.

Embedding methods	sentenceBERT	fastText	GloVe	No Semantic
Hit@1	0.508	0.529	0.525	0.467
Hit@2	0.652	0.656	0.676	0.627
Hit@3	0.730	0.734	0.734	0.734
Avg. latency per-case w/ this embedding	0.299s	0.052s	0.050s	0.026s

Our Auto-Tables method is compared with the following approaches:

- Auto-Tables: Our unique approach that doesn't require users to provide input/output examples. It is trained using a substantial dataset of input-table and transformation pairs.
- Foofah (FF): Synthesizes transformations based on input/output examples.
- Flash-Relate (FR): Another approach requiring input/output examples to synthesize table-level transformations.
- SQLSynthesizer (SQ): A SQL-by-example algorithm that synthesizes SQL queries based on input/output examples.
- Scythe (SC): Another SQL-by-example method.
- TaBERT: A table representation approach developed for natural language processing tasks.
- TURL: Another table representation approach for data integration tasks.
- GPT-3.5: A large language model pre-trained on text and code, used for comparison in a few-shot in-context learning scenario.

These evaluations are conducted on a Linux VM with specific hardware specifications, ensuring a comprehensive comparison of each method's effectiveness and efficiency in synthesizing transformations.

5.2 Experiment Results

Quality Comparison: In Table 3, compared Auto-Tables with other methods across 244 test cases. Methods fall into two categories: "No-example" (Auto-Tables, TaBERT, TURL) and "By-example" (Foofah, FlashRelate, SQLSynthesizer, Scythe). Auto-Tables excels, transforming 75%

of test cases in its top 3 without user-provided examples, a remarkable feat given the expansive search space in our tasks.

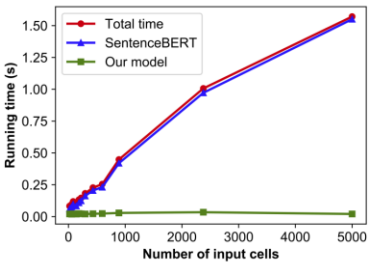


Figure 10: AUTO-TABLES latency analysis

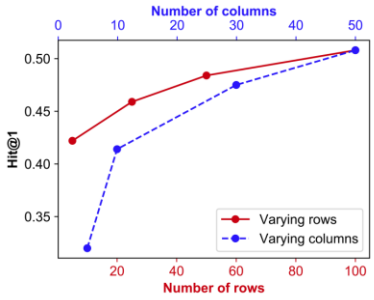


Figure 11: Vary input size

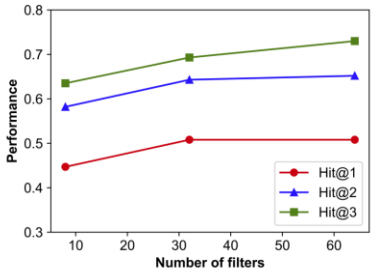


Figure 12: Vary number of filters

Compared to other "No-example" methods, Auto-Tables significantly outperforms TaBERT and TURL. Against "By-example" methods, Auto-Tables exhibits marked improvement, showcasing its aptness for table restructuring without user input.

Source-specific Results: Table 7 breaks down results by data sources (forums, notebooks, Excel+web), revealing consistent quality across diverse test cases, underscoring Auto-Tables' versatility.

Table 7: Quality comparisons by data sources

	Forum			Notebook			Excel			Web		
Method	**AT**	**TA**	**FR**	**AT**	**TA**	**FR**	**AT**	**TA**	**FR**	**AT**	**TA**	**FR**
Hit @ 1	**0.522**	0.217	0.043	**0.582**	0.241	0.278	**0.558**	0.174	0.5	**0.589**	0.143	0.286
Hit @ 2	**0.696**	0.478	-	**0.722**	0.557	-	**0.651**	0.442	-	**0.732**	0.321	-
Hit @ 3	**0.696**	0.565	-	**0.747**	0.62	-	**0.709**	0.547	-	**0.839**	0.429	-

Quality with Relational Tables: Auto-Tables adeptly identifies relational tables not needing transformations. Table 8 illustrates high quality, confirming its accuracy in predicting "none" for tables without transformations.

Table 8: Quality comparison using Hit@k, on 244 cases with non-relational input (requiring transformations), and 244 cases with relational input (not requiring transformations).

Method	Auto-Tables	TaBERT	TURL
Hit @ 1	**0.695**	0.258	0.175
Hit @ 2	**0.803**	0.594	0.387
Hit @ 3	**0.840**	0.699	0.444

Running Time: Table 4 compares average latencies among different methods. Auto-Tables boasts sub-second latency in almost all cases, a sharp contrast to longer times for Foofah and FlashRelate. Figure 10 depicts Auto-Tables' linear latency growth with increasing input cells, ensuring user-friendly and efficient performance.

Ablation Study: Insights from Table 5's ablation studies on Auto-Tables components:

- Re-Ranking Contribution: Integrating the re-ranking model significantly enhances Hit@1 and Hit@2.
- Data Augmentation Impact: Data augmentation is pivotal; disabling it markedly reduces Hit@k.
- Embeddings Importance: Both syntactic and semantic embeddings are crucial; removing them drastically reduces performance.
- Effectiveness of 1D Filters: The chosen 1x1 and 1x2 filters prove effective; altering them adversely affects performance.

Sensitivity Analysis:

- Varying Input Size: Adjusting the number of rows/columns affects quality until it plateaus at around 30 columns and 50 rows.
- Varying Number of Filters: Using 32 filters significantly enhances feature extraction, with diminishing returns.
- Varying Embedding Methods: Auto-Tables perform well with alternative embeddings (GloVe and fastText), showcasing versatility with lower latency.

Error Analysis: Analyzing errors in Table 9 and Table 10 reveals common operator-type mistakes between similar transformations, while accurate parameter predictions persist despite the extensive parameter space.

Table 9: Confusion matrix for single-step top-1 predictions.

True \ Pred	trans.	stack	wtl	explode	ffill	pivot	subtitle	none
trans.	14	10	1	1	0	0	0	2
stack	2	36	3	1	0	0	0	14
wtl	0	6	23	1	0	0	0	4
explode	0	0	0	32	0	0	0	16
ffill	0	1	0	1	10	0	0	6
pivot	0	0	0	0	0	8	0	0
subtitle	0	1	0	0	0	0	24	1

Table 10: Accuracy of operator parameter predictions

operator parameter	stack start-idx	stack end-idx	wtl start-idx	wtl end-idx	explode col-idx	ffill col-idx	pivot row-freq
Accuracy	0.889	1	0.957	1	0.969	1	0.875

6. Conclusion

The "Auto-Tables" technology represents a significant step forward in the world of table transformation, showcasing the potential of algorithms that can teach themselves. While the paper has strengths in its innovative approach and thorough experiments, addressing certain areas for improvement will enhance its overall contribution and provide a more solid understanding of what "Auto-Tables" can offer and where it might fall short.

1. Evaluation Metrics: The paper primarily uses Hit@k metrics to evaluate transformation quality. Incorporating additional metrics that consider precision, recall, or specific use-case scenarios would offer a more nuanced evaluation.
2. User-Friendliness and Practicality: The paper emphasizes the user-friendliness of Auto-Tables, but it could benefit from a discussion on potential challenges or limitations faced by users in real-world scenarios. Understanding the practical implications and potential user difficulties would add depth to the analysis.
3. Broader Implications: The discussion on future work could benefit from a broader exploration of how Auto-Tables might impact the fields of data science, automation, and human-computer interaction. Identifying challenges or ethical considerations would contribute to a more comprehensive research agenda moving forward.

In conclusion, refining these aspects will solidify the impact and understanding of "Auto-Tables" in the realm of table transformation and related fields.