A Neural Approach to Program Synthesis for Tabular Transformation by Example

YOSHIUMI UJIBASHI\textsuperscript{1}, ATSUHIRO TAKASU\textsuperscript{2}, (Member, IEEE)

\textsuperscript{1}The Graduate University for Advanced Studies, SOKENDAI, Shonan Village, Hayama, Kanagawa, Japan
\textsuperscript{2}Fujitsu Ltd, 4–1–1 Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa, Japan

Corresponding author: Yoshifumi Ujibashi (e-mail: ujibashi@nii.ac.jp).

ABSTRACT Data transformation is a laborious and time-consuming task for analysts to properly prepare the data for analysis. Programming by example (PBE) is a technology that makes this troublesome task easier for data analysts by automatically generating programs for data transformation. In the past, most PBE research has been based on search algorithms, but with the recent success of machine learning (ML), PBE research using ML has emerged. For example, RobustFill, an ML-based PBE for string transformations using long-short-term memory (LSTM) as the sequential encoder-decoder model, has been proposed. However, there is still no ML-based PBE research for tabular transformations, which are frequently used in data analysis. This paper proposes an ML-based PBE for tabular transformations. First, we address tabular transformations, which are more complex and data-intensive than string transformations, and propose a new ML-based PBE based on the state-of-the-art Transformer sequential encoder-decoder model. To the best of our knowledge, this is the first ML-based PBE for tabular transformations. Next, we propose two decoding methods, multistep beam search and Program Validation (PV)-Beam Search, which are optimized for program generation and so generate correct programs with higher accuracy. The evaluation results show that the Transformer-based PBE achieves much higher performance than LSTM-based PBE when applied to tabular transformations and furthermore, the Transformer-based model with the proposed decoding method outperforms the conventional PBE using the search-based method.

INDEX TERMS Automatic Programming, Beam Search, LSTM, Neural Networks, Programming by Example, Program Synthesis, Tabular Data, Transformer

I. INTRODUCTION

The integration of data from a variety of data sources into a unified format is a time-consuming and labor-intensive task for engineers or domain specialists. This data integration task has long been a high hurdle in data analysis and data utilization. An example of such a data integration task is described below. Suppose a data analyst wants to utilize the semi-structured tabular data shown in Table 1. This data needs to be stored in a database so that it can then be analyzed. However, since Table 1 was created without considering such later data analysis, it needs to be converted into a relational form that can be stored in a database such as Table 2. This task requires various transformations, which are difficult and troublesome for data analysts who do not have expertise in data transformation.

To solve this hurdle in data utilization, various approaches have been adopted. Programming by example (PBE) is one of these approaches. PBE is a type of program synthesis that has been studied for a long time. Program synthesis is a technology that generates programs according to the specification conditions given by the user. PBE generates programs that satisfy the input-output examples given by the user as specification conditions.

As pointed out in [2], many data integration tasks involve string transformations and table layout transformations. (The transformation in the previous example is also a combination of both types of transformations.) Such a transformation is called tabular transformation in this paper.
Foofah [1] is a PBE study of such tabular transformation. It is built around techniques for solving search problems, i.e., finding a program that converts given input table to the output one. Such a search approach usually involves complex search algorithms and pruning rules, and requires considerable engineering effort in their design, development, modifications, and possible extensions.

Recently, many PBE studies based on Machine Learning (ML) approaches have emerged greatly motivated by their easiness to expand their transformation scope by simply expanding the training dataset. Previous works [3], [4] propose ML-based PBE by leveraging long-short-term memory (LSTM), which is an encoder-decoder model mainly used for natural language processing and time series processing. Their results show that an ML-based PBE can have a competitive accuracy to the conventional non-ML-based PBE, and furthermore, can achieve noise robustness that a non-ML-based PBE do not have. However, these studies are on PBE for string transformation, not tabular transformation.

In this paper, we propose a new ML approach to achieve PBE for tabular transformation, which is required in data analysis and data integration scenarios. The contributions of this paper are as follows:

- We adopt the Transformer-based model to the ML model for PBE. Fast training by parallel processing and expressiveness by large capacity make it possible to handle table transformations, which are more complex and data intensive than string transformations.
- We propose decoding methods called multistep beam search, which is suitable for finding a consistent program by running beam search multiple times, and PV (Program Validation)-beam search, which realizes efficient beam search by searching only the hypothesis space that is valid as a program.
- We implemented the Transformer-based model with proposed decoding methods and evaluated the performance in experiments. We empirically show that the Transformer-based model outperforms the LSTM-based model. Furthermore, the Transformer-based model with our proposed decoding methods achieved better performance compared to the conventional search-based method which has been the most advanced study of PBE for tabular transformations.

The remainder of this paper is organized as follows. We formally define the problem in Section II and explore related work in Section III. We present our proposal in Section IV and examine our proposal’s performance via a set of benchmarks used in previous work and evaluate our ML model in Section V. Finally, we conclude the paper in Section VI.

II. PROBLEM FORMULATION

We now define formally the PBE problem discussed in this paper and the domain-specific language used for tabular data transformation.

A. THE PROBLEM

Given a pair of input-output tabular data \((T_i, T_o)\), the requirement is to synthesize a tabular transformation program \(P\) that satisfies \(T_o = P(T_i)\). The program \(P\) is composed of a sequence of tabular transformation operations \((P_1, P_2, \ldots, P_n)\). Each operation \(P_i\) is a unit of tabular data transforms. The program \(P\) is synthesized by chaining operations that satisfy the following expression.

\[ T_1 = P_1(T_i), \quad T_2 = P_2(T_1), \quad \ldots \quad T_o = P_n(T_{n-1}) \]

B. PROGRAM COMPONENTS

Each operation is parameterized by arguments that specify, for example, column(s) to be manipulated or delimiters for dividing cells. We use the operators and their arguments listed in Table 3 as the domain-specific language in this work. Our operation list is very similar to that of “Foofah” [1], with the differences highlighted in Table 3.

For example, we can convert the input Table 1 to output Table 2 by the following sequence of the operators:

- **Split("": 1)**: split the column 1 with the delimiter “:”, which converts Table 1 to Table 4.
- **Delete(1)**: delete rows including a empty cell at the column 1, which converts Table 4 to Table 5.
- **Fill(0)**: fill empty cells by the value above at the column 0, which converts Table 5 to Table 6.
- **UnFold()**: unfold the table, which converts Table 6 to Table 2.

III. RELATED WORK

A. SEARCH-BASED PBE

PBE has been applied to a variety of domains including the data integration. FlashFill [5] is a typical example of research that applies PBE to solve data integration tasks. It handles data integration, especially string transformation in Excel cells, and allows users to convert strings simply by giving examples of what they want to convert, without using complicated Excel macros. FlashFill searches the program
TABLE 3. Domain-specific language for tabular data transformation (differences from the original Fooah operator is flagged in bold)

| Operator                          | Description                                                                                     |
|-----------------------------------|-------------------------------------------------------------------------------------------------|
| MoveToEnd(pos)                    | Moves a column to the last position (replaces Fooah’s original Move operator)                 |
| MoveFromEnd(pos)                  | Moves the last column to another position (replaces Fooah’s original Move operator)             |
| Drop(pos)                         | Drops a column                                                                                       |
| Copy(pos)                         | Duplicates a column to the following position                                                    |
| CopyToDest(pos, pos)              | Duplicates a column to another position (newly added)                                              |
| Fill(pos)                         | Fills empty cells in a column with the one above                                                    |
| Split delim regex, pos            | Splits a column into two columns at the occurrence of a delimiter or the match of a regex       |
| MergeToOne delim, pos             | Merges two or more columns after a position by a delimiter                                         |
| Divide(predicate, pos)            | Divides a column into two columns whereby one satisfies the predicate and the other does not     |
| Extract(regex, pos)               | Extracts the first match at a column with addition of the matched column (newly added)           |
| Fold(pos)                         | Folds all the columns after the specified column into one column                                  |
| Unfold()                          | Expands the folded columns of the table                                                           |
| FoldHeader(pos)                   | Folds the column names and the values of all the columns after the specified position            |
| UnFoldHeader(pos)                 | Transform the folded columns into a relational form whereby the column names are placed in the table header |
| Delete(pos)                       | Deletes rows which have an empty cell at the specified column                                     |
| DeleteEmptyCols                   | Deletes columns where all cells are empty                                                          |
| Transpose()                       | Transposes rows and columns of the table                                                          |
| Wrap(pos)                         | Concatenates rows which have the same values at the specified column                              |
| WrapToOne()                       | Concatenates all rows to one row of the table                                                     |
| WrapEveryKRows(k)                 | Concatenates every k row of the table into one row                                                 |

**Examples:**

- **Operator:** MoveToEnd
  - **Description:** Moves a column to the last position.

- **Operator:** CopyToDest
  - **Description:** Duplicates a column to another position.

**Table 4.** The second column of Table 1 is split by ‘:’.

| Name   | Tel    | Fax    |
|--------|--------|--------|
| Alice  | (03)7345-3850 | (03)7001-1400 |
| Bob    | (045)873-9639 | (045)873-8762 |
| Carol  | (06)2340-0987 | (06)2340-6701 |

**Table 5.** Rows that are blank in the second column are deleted from Table 4.

| Name   | Tel    | Fax    |
|--------|--------|--------|
| Alice  | (03)7345-3850 |                  |
| Bob    | (045)873-9639 |                  |
| Carol  | (06)2340-0987 |                  |

**Table 6.** Fill each blank cell in the first column of Table 5 by the cell above.

| Name   | Tel    | Fax    |
|--------|--------|--------|
| Alice  | (03)7345-3850 | (03)7001-1400 |
| Bob    | (045)873-9639 | (045)873-8762 |
| Carol  | (06)2340-0987 | (06)2340-6701 |

space for a program that matches the example efficiently using a pruning technique called as Version Space Algebra.

In addition, ProgFromEx [6], Fooah [1] and FlashRelate [7] are PBE studies that include layout transformation among the data integration tasks. They also use search algorithms for program generation. For example, Fooah uses the A* algorithm, a graph search algorithm, to make the search process more efficient. Thus, traditional PBE studies were developed by devising search algorithms to make the search more efficient.

**B. ML-BASED PBE**

Since the neural methods are expected to extend a model by simply increasing the number of training data without the need for laborious technical development, research on realizing PBE using neural methods has attracted much attention in recent years [3, 8, 9, 10]. RobustFill [3] is a fundamental study for PBE using neural methods and is used as a basis by many other studies. It achieves a string transformation PBE by leveraging LSTM model, which is a well-known sequential encoder-decoder model. However, to the best of our knowledge, no neural method for table transformation PBE has been proposed and implemented.

**C. BEAM SEARCH**

Beam search has been studied for a long time, especially in recent years in natural-language-processing (NLP) neural models [11] [12] [13] [14] [15]. Iterative beam search [15] focuses on generating diverse sequences for tasks such as...
dialog response generation and is an example of research on improving beam search to make it suitable for applications.

In this paper, we propose variants of beam search, namely multistep beam search and PV-beam search, which are beam search methods optimized for program generation.

IV. OUR APPROACH

A. OVERVIEW

![System overview in our approach](Image)

Fig. 1 gives an overview of the system architecture used in our approach. The processing in our system can be split into a training phase and an inference phase.

In the training phase, the neural model is trained using training datasets, which are composed of pairs of input-output tables and the corresponding programs.

In the inference phase, given an example of an input-output table, the previously trained model generates a candidate program, and eventually provides a final program consistent with the example.

B. TABULAR DATA LINEARIZATION

We linearize a tabular data into a serialized structure in order to feed the tabular data into the sequential encoder-decoder model. Fig. 2 shows an example of this processing. First, each string in each cell of the input table is separated into character-level tokens with special characters `<eoc>` and `<eol>` that represent the boundaries of columns and rows respectively and linearized to a sequence of tokens.

In addition, in the Transformer-based model (see IV-D), we concatenate the input table and output table by introducing the special token `<sep>` to represent the separation point between the two tables.

C. PROGRAM LINEARIZATION

A program is also linearized into a sequence of operation tokens. An operation token is constructed from the Operators, its numeric parameters and its non-numeric parameters listed in Table 3 according to the tokenization rules. We have the following two options for tokenization rules.

a: Separated Token Rule

A combination of operator and non-numeric parameter are represented by one token and numeric parameters are represented by one token individually in separated token rule.

For example, Extract("\w+", 1) is represented by two tokens. The first token is Extract("\w+", pos) and the second one is the numeric parameter "1".

The vocabulary size of operation tokens in separated token rule is 100 including the special tokens. Separated token rule is the default rule in our experimental settings.

b: Combined Token Rule

A combination of operator, non-numeric parameter and numeric parameter is represented by one token in combined token rule.

For example, Extract("\w+", 1) is represented by one token.

The vocabulary size of operation tokens in combined token rule is 2148 including the special tokens.

D. THE TRANSFORMER-BASED ENCODER-DECODER MODEL

"Transformer" is a state-of-the-art ML model for the NLP tasks [16]. The architecture is based on attention mechanisms and achieves a high performance by high-level parallelization and short training times. This is realized by dispensing with recurrent and convolutional processing entirely. Fig. 3 illustrates our Transformer-based model.
In order to feed a pair of input-output tables and a program into the Transformer model, they are linearized to token sequences as described in IV-B and IV-C, respectively. Let a token sequence of input-output tables be \( t \) and a token sequence of program be \( o \).

The Transformer embedding layers embed \( t \) and \( o \) and append the positional encoding \( PE \).

\[
\begin{align*}
x_0 &= \text{EncoderEmbedding}(t) + PE \\
y_0 &= \text{DecoderEmbedding}(o) + PE
\end{align*}
\]

Let \( x_i = (x_0^i, x_1^i, \cdots, x_n^i) \) be the output of the \( i^{th} \) layer of the Transformer encoder, where \( x_j^i \in \mathbb{R}^{d_{emb}} \) and \( d_{emb} \) is the embedding dimension size. Similarly, let \( y_i = (y_0^i, y_1^i, \cdots, y_m^i) \) be the output of the \( i^{th} \) layer of the Transformer decoder, where \( y_j^i \in \mathbb{R}^{d_{emb}} \). The Transformer encoder and decoder are structured as layers, where the sub-layers process the computations described in the following expressions.

\[
\begin{align*}
x_i &= \text{TransformerEncoderLayer}_{i^{th}}(x_{i-1}) \\
y_i &= \text{TransformerDecoderLayer}_{i^{th}}(y_{i-1}, x_i)
\end{align*}
\]

When all computations in the encoder and decoder sub-layers have terminated, the output from the last sublayer of decoder \( y_{last} \) is provided. The linear and softmax layers at the last part of the Transformer-based model (see Fig. 3) compute the program probabilities from \( y_{last} \).

For a set \( T_0 \) of token indexes for the input-output tables and the token sequence length \( n \) of the linearized input-output tables, we denote a token sequence of the input-output tables as an integer vector \( t = (t_0, t_1, \cdots, t_n) \) where \( t_i \in T_0 \).

For a set \( O \) of token indexes for the program and the token sequence length \( m \) of the sequence the program, we denote a token sequence of a program as an integer vector \( o = (o_0, o_1, \cdots, o_m) \) where \( o_i \in O \).

Now, program probabilities can be represented as a sequence \( p = (p_0, p_1, \cdots, p_m) \), where \( p_i \in \mathbb{R}^{|O|} \). The variable \( p_i = P(z \mid o_{<i}, t) \) represents the conditional probability distribution of the discrete random variable \( z \in O \) at position \( i \) when \( (o_0, o_1, \cdots, o_{i-1}) \) and \( t \) are given.

The left part of Fig. 4 shows the decoding process during the training phase. The program probabilities \( p \) are output after inputting a sequence of tokens for input-output tables \( t \) into the encoder layer and a sequence of tokens for the program (shifted by 1 token \( (<sos>) \)) into the decoder layer.

We can compute the cross-entropy loss from \( p \) and (the one-hot vector of) \( o \), and then proceed to training with the aim of reducing the loss value.

The right part of Fig. 4 shows the decoding process during the inference phase. In this phase, beam search processes the decoding via autoregression and generates the candidates of the programs. The basic beam search processing is as follows.

In the first step, given a token \( <sos> \) at the decoder layer and the token sequence \( t \) at the encoder layer, the decoder outputs the conditional probability \( p_0 = P(z \mid t) \) at position \( t = 0 \). Using beam search with the beam size of 1 gives the token \( z_0 \in O \) having the highest probability to give to the decoder at the next position.

In the next step, given the token sequence \( t \) at the encoder layer tokens \( <sos> \) and \( z_0 \) at the decoder layer, the decoder outputs the conditional probability \( p_1 = P(z \mid z_0, t) \) at position \( t = 1 \).

Again, beam search determines the token \( z_1 \in O \) having the highest score given by the equation 1 with respect to the tokens \( <sos>, z_0, \) and \( z_1 \) to give to the decoder at the next position. In this way, the token \( z_1 \) is found having the highest score at each position \( t \).

\[
\text{score} = \log(\prod_{i=0}^{t} P(z \mid z_{<i}, t))
= \sum_{i=0}^{t} \log(P(z \mid z_{<i}, t)).
\]

The beam search terminates when the end-of-sequence token \( <eos> \) has the highest score at positions \( t < m \), finally obtaining one candidate \( z = (z_0, z_1, \cdots, z_{t-1}) \).

The procedure of beam search with a beam size of \( K > 0 \) is described in Algorithm 1.

The beam search maintains \( K \) number of candidates \( H_{i}^{\text{rand}} \) of token sequences (each token sequence is called as hypothesis \( h \)) at each position \( t \). At each position \( t \), each hypothesis \( h \) in the previous candidates \( H_{i-1}^{\text{rand}} \) is concatenated by each token \( z \in O \) (line 8 in Algorithm 1), and the scores for concatenated hypotheses \( h \) are given by equation 1 (line 9 in Algorithm 1).

The hypotheses are ranked by the score and the top-\( K \) hypotheses are kept in the candidates \( H_{i}^{\text{rand}} \) (line 10-22 in Algorithm 1). When \( <eos> \) token is selected as a candidate, the \( K \) value is reduced by one and the candidate is placed in the “final-candidate” list \( H_{final} \) (line 16-18 in Algorithm 1).

When the beam size \( K \) becomes zero, the search stops with a final candidate list containing \( K \) number of candidates (line 19-20 in Algorithm 1).
Finally, we can get the program by selecting a hypothesis from the final candidate list and the hypothesis into the program form. Not all of the programs in the candidates are consistent with the input-output tables.

Therefore, in order to select a consistent program, each program is picked out of the final candidate list one-by-one and checked whether each program is consistent with the input-output tables.

Once the consistency check on a program is successful, it’s the solution for the input-output tables. If no successful program is found from the final candidate list, it is regarded as a failure of finding the solution for the input-output tables.

E. DECODING METHODS

Focusing on the fact that the PBE task is a search problem finding consistent programs, we propose two methods to enhance the basic beam search method for the program generation task in this section.

a: Multistep beam search

We focus on the intermediate tables created by transforming the input table into the output table. We assume that intermediate tables is closer to the output table than input table. Therefore, the beam search that originating from intermediate table is successful than from input table.

On this assumption, we designed multistep beam search which repeats beam search multiple times changing the origin of beam search into other intermediate tables and eventually finds the solution.

Algorithm 2 describes the procedure of the multistep beam search.

Algorithm 1 basic beam search

Algorithm 2 multistep beam search

In order to keep the candidates of the intermediate tables and efficiently handle them, we maintain a directed tree structure $T = (N', E')$ that comprises a set of nodes $N' = \{n_i | i \in \mathbb{Z}\}$ and a set of directed edges $E = \{e_i | i \in \mathbb{Z}\}$ (see Fig. 5).

This tree structure represents the space already explored by the multistep beam search. Each node $n_i$ represents a table and each directed edge $e_i$ represents an operation that transforms a table into another table.

$T$ is a tree structure that has the following features.

- The root node of $T$ is the node $n_{\text{input}}$ representing the given input table.
- A child node $n_{\text{child}}$ is the result of applying a directed edge $e_i$ to a parent node $n_{\text{parent}}$. We also denote this relation in the form $n_{\text{child}} = e_i(n_{\text{parent}})$.
- $\forall n_i \in N'$ is not identical to $n_{\text{output}}$.

First, the multistep beam search “picks up” a node $n_{\text{pick}}$ from $T$ that is an origin node of a beam search (see Line 3 in Algorithm 2). In the first step, the $n_{\text{input}}$ is picked up as the $n_{\text{pick}}$, because $T$ includes only the node $n_{\text{input}}$.

Next, it invokes a basic beam search with the origin node $n_{\text{pick}}$, the goal node $n_{\text{output}}$, namely the output table, and beam size $K$, returning the set of hypotheses $H = \cup_{t=0}^{K-1} H_t$ (see Line 4 in Algorithm 2).

For each hypothesis $H_t$, a sequence $S = (N', E')$ is generated, which contains the operation sequence $E' = (e_0', e_1', \ldots, e_{t-1}')$, where $e_i' = \langle <\text{eos}>, e_i \rangle$. Applying the operation sequence $E'$ to each $n_{\text{pick}}$, in order, we obtain a sequence $S = (N'', E'')$, such that $N'' = \{n_{\text{pick}}, n_1', n_2', \ldots, \text{null}, \text{null}\}$ and $E'' = \{e_0', e_1', \ldots, \text{null}, \text{null}\}$, where $n_1' = e_0(n_{\text{pick}})$, $n_2' = e_1(n_1')$, etc. (see Line 6 in Algorithm 2 and Fig. 5).
If the application of the operation to a node fails, all subsequent processing in this operation sequence will fail (denoted by null). The null entries are then deleted from \( \mathcal{N}^\prime \) and \( \mathcal{E}^\prime \).

If some \( n \in \mathcal{N}^\prime \) is identical to \( n_{\text{output}} \), the corresponding operation sequence from \( n_{\text{input}} \) to \( n_{\text{output}} \) is considered the desired program \( P \) (see Line 8 in Algorithm 2).

To merge the sequence \( S \) into the directed tree \( T \), the first node \( n_{\text{pick}} \) of \( S \) is grafted onto the \( n_{\text{pick}} \) of \( T \) (see Line 9 in Algorithm 2 and Fig. 7).

It then prunes the edge and node that reach the identical nodes among \( T \) and \( S \) that have longer paths from the root node \( n_{\text{input}} \) of \( T \) (see Line 10 in Algorithm 2 and Fig. 7).

This merging process is applied to all hypotheses, and then it proceeds to the next pick-up (see Line 3 in Algorithm 2).

Pick-up processing can be based on either of two strategies, namely a shortest-path strategy or a longest-path strategy. Using a shortest-path strategy, the node with the shortest path from \( n_{\text{input}} \) is picked up, whereas the node with the longest path is picked up under a longest-path strategy.

After a node is picked up, the node is marked as done and never picked up in the subsequent processing. This procedure is repeated until a program consistent with the given input-output table is generated or a timeout occurs.

**Algorithm 3 PV-beam search**

**Input:** \( \text{input\_table} \) \( \triangleright \) input table

**Input:** \( K \) \( \triangleright \) Beam size

**Output:** \( \mathcal{H}_{\text{final}} \) \( \triangleright \) Hypotheses obtained finally

1. \( \mathcal{H}^\text{cand}_{0} \leftarrow \{ h \} \) where \( h = (\text{<sos}> \}
2. \( \mathcal{H}_{\text{final}} \leftarrow \emptyset \)
3. for \( t \in \mathbb{N} \) in ascending order:
4. \( \mathcal{H}_{t} \leftarrow \emptyset \)
5. \( \mathcal{S}_{t} \leftarrow \emptyset \)
6. for \( h \in \mathcal{H}^\text{cand}_{t-1} \):
7. for \( z \in \mathcal{O} \):
8. \( h \leftarrow h \parallel z \) \( \triangleright \parallel \) concatenates tokens
9. \( s \leftarrow \text{score}(h) \) \( \triangleright \) score given by equation 1
10. \( \mathcal{H}_{t} \leftarrow \mathcal{H}_{t} \cup \{ h \} \)
11. \( \mathcal{S}_{t} \leftarrow \mathcal{S}_{t} \cup \{ s \} \)
12. sort \( \mathcal{S}_{t} \) by scores
13. sort \( \mathcal{H}_{t} \) by the same order to \( \mathcal{S}_{t} \)
14. \( \mathcal{H}^\text{cand}_{t} \leftarrow \emptyset \)
15. for \( h \in \mathcal{H}_{t} \) in order:
16. synthesize a program \( P \) from hypothesis \( h \)
17. \( \text{output\_table} = P(\text{input\_table}) \)
18. if \( \text{output\_table} \) is not a valid table:
19. continue
20. if the last token of \( h \) is \( \text{<eos>} \):
21. \( \mathcal{H}_{\text{final}} \leftarrow \mathcal{H}_{\text{final}} \cup \{ h \} \)
22. \( K \leftarrow K - 1 \)
23. if \( K = 0 \):
24. return \( \mathcal{H}_{\text{final}} \)
25. if \( |\mathcal{H}^\text{cand}_{t}| < K \):
26. \( \mathcal{H}^\text{cand}_{t} \leftarrow \mathcal{H}^\text{cand}_{t} \cup \{ h \} \)

**V. EXPERIMENTS**

**A. EXPERIMENT SETTINGS**

1) Metrics

There are two metrics for evaluating the performance of PBE systems [3], namely **consistency** and **generalization**.

**Consistency** measures whether any generated program is consistent with a user-given example. Here, the PBE system
would be regarded as successful in terms of the consistency test.

*Generalization* measures whether a program generated from a example is consistent with the original table from which the example is derived. If a program, which has been generated from the example, can transform the original table into the user’s intended table, the program would be regarded as generalizable in terms of the original table.

Consistency is the important metric and is used mainly to evaluate our methods in V-B. We define accuracy as the proportion of benchmark tests for which the PBE system is successful in terms of consistency.

Although generalization is also another important metric, with several past works [17] [18] [19] particularly focusing on it, we focus on the consistency in our evaluations as with the work [4].

2) Training Datasets

Preparing a large-scale training dataset for PBE ML model is quite difficult, because such a dataset is composed of many input-output pairs of tabular data with corresponding transformation programs. We therefore synthesized the training datasets as in previous works [3], [20], [21].

In order to construct the training data, we employed the random sampling and generation methodology shown in Fig. 8. An input table and a program are randomly synthesized with characters or operations or others that are randomly sampled in a uniform distribution.

![Example of synthesizing training data](image)

The input table is synthesized with ASCII characters, numbers, and punctuation characters. The size of the string in a cell, number of rows and columns of the table are also randomly chosen, up to a predefined maximum value. In our experiments, these maximum values for strings, rows, and columns were set as 10, 5, and 5, respectively.

Each program is a sequence of tokens generated from operators and its arguments randomly sampled from items given in Table 3. The length of each program is up to the predefined maximum size. The maximum size is parameterized from 3 to 8, and its default value is 6 in our experiments.

An output table is generated by transforming the synthesized input table using the synthesized program. Because a program might fail when applied to the input table, program synthesis is repeated until the synthesized program no longer fails when applied to the input table.

We prepared training datasets of various sizes (each item comprises an input-output table pair and the corresponding program): 1,024 (1K), 10,240 (10K), 1,024,000 (1M), 2,048,000 (2M), 4,096,000 (4M), and 10,240,000 (10M).

Each training dataset was partitioned, with 90% used for training and 10% for validation.

3) Benchmark Datasets

We evaluated our models using the evaluation benchmark proposed and used in Foofah [1]. This benchmark is composed of 250 tests gathered from previous works and constructed to evaluate tabular-data transformation tasks. Examples of this benchmark are shown in Fig. 9.

![Examples of benchmark data](image)

We divided the benchmark into two parts according to the size of the table. The “small” benchmark for each test involves tables with rows smaller than 5 and columns smaller than 5, whereas the remaining ones are referred as “large”. The “small” benchmark was composed of 73 tests, as opposed to 177 for the “large” benchmark.

4) Hyperparameters and Settings

The hyperparameters are given in Table 7.

![Hyperparameters and settings for our experiments](image)

The experimental results in various settings are listed in Table 8.

![Table of hyperparameters and settings](image)

[2] https://github.com/markjin1990/foofah_benchmarks
### TABLE 8. Experimental results for all parameter values. Values in parentheses are default values for the parameters. Unlisted values also imply the default.

| No. | Model type (Transformer-based) | Special symbol (+sep, +eoc, +eol) | Scan (horizontal) | Program length (6) | No. of samples (1M) | No. of layers (2) | Decoding method (basic beam search) | Beam width (100) | Program Tokenization (separated) | Percent accuracy on small data | Percent accuracy on large data |
|-----|--------------------------------|-----------------------------------|-------------------|-------------------|---------------------|-------------------|-----------------------------------|-----------------|-------------------------------|-------------------------------|-------------------------------|
| 1   | LSTM-based                     |                                   |                   |                   |                     |                   |                                   |                 |                               | 66.9                         | 10.7                           |
| 2   | 10K                            |                                   |                   |                   |                     |                   |                                   |                 |                               | 52.0                         | 6.2                             |
| 3   | 100K                           |                                   |                   |                   |                     |                   |                                   |                 |                               | 61.7                         | 18.0                           |
| 4   | 2M                             |                                   |                   |                   |                     |                   |                                   |                 |                               | 87.6                         | 33.8                           |
| 5   | 4M                             |                                   |                   |                   |                     |                   |                                   |                 |                               | 80.8                         | 31.7                           |
| 6   | 10M                            |                                   |                   |                   |                     |                   |                                   |                 |                               | 90.4                         | 34.4                           |
| 7   | 100K                           |                                   |                   |                   |                     |                   |                                   |                 |                               | 82.1                         | 36.7                           |
| 8   | 6                              |                                   |                   |                   |                     | 10K               |                                   |                 |                               | 52.0                         | 2.25                           |
| 9   | 100K                           |                                   |                   |                   |                     | 6                  |                                   |                 |                               | 65.7                         | 12.4                           |
| 10  | 6                              |                                   |                   |                   |                     |                   |                                   |                 |                               | 86.3                         | 36.1                           |
| 11  | 2M                             |                                   |                   |                   |                     |                   |                                   |                 |                               | 90.4                         | 34.4                           |
| 12  | 4M                             |                                   |                   |                   |                     |                   |                                   |                 |                               | 93.1                         | 41.8                           |
| 13  | 6                              |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 95.8                         | 42.3                           |
| 14  | 4                              |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 87.6                         | 37.8                           |
| 15  | 5                              |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 94.5                         | 36.7                           |
| 16  | 7                              |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 89.0                         | 37.8                           |
| 17  | 8                              |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 82.1                         | 37.8                           |
| 18  | 4                              |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 91.7                         | 35.5                           |
| 19  | 5                              |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 91.7                         | 40.1                           |
| 20  | 7                              |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 90.4                         | 41.2                           |
| 21  | 8                              |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 93.1                         | 41.8                           |
| 22  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 76.7                         | 22.5                           |
| 23  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 89.0                         | 32.2                           |
| 24  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 89.0                         | 35.5                           |
| 25  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 91.7                         | 43.5                           |
| 26  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 93.1                         | 46.8                           |
| 27  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 93.1                         | 51.4                           |
| 28  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 94.5                         | 50.2                           |
| 29  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 86.3                         | 40.6                           |
| 30  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 90.4                         | 37.2                           |
| 31  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 90.4                         | 40.6                           |
| 32  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 91.7                         | 43.5                           |
| 33  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 93.1                         | 46.8                           |
| 34  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 93.1                         | 51.4                           |
| 35  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 94.5                         | 50.2                           |
| 36  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 86.8                         | 29.9                           |
| 37  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 94.5                         | 41.8                           |
| 38  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 94.5                         | 45.1                           |
| 39  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 94.5                         | 49.1                           |
| 40  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 94.5                         | 50.8                           |
| 41  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 95.8                         | 53.1                           |
| 42  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 97.2                         | 51.4                           |
| 43  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 97.2                         | 51.4                           |
| 44  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 90.8                         | 24.2                           |
| 45  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 89.0                         | 38.4                           |
| 46  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 89.0                         | 44.6                           |
| 47  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 91.7                         | 57.6                           |
| 48  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 93.1                         | 62.7                           |
| 49  | 10M                            |                                   |                   |                   |                     | 10M               |                                   |                 |                               | 94.5                         | 64.4                           |

#### 5) Hardware and Software Settings

We trained our model and evaluated it using one GPU slot of an NVIDIA Tesla V100-PCIE GPU with 32 GB memory. An Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40 GHz with 132 GB memory was used to run the search-based PBE system (based on Foofah).

We developed and evaluated both types of neural models using the Pytorch library [22]3 and our Transformer-based model using the fairseq toolkit [23]4.

#### 6) Baselines

**a: The LSTM-based Baseline System**

We developed a baseline system using the LSTM model, which is one of well-known sequential encoder-decoder model, with reference to the Attention-A model in Robust-Fill [3]. Note that Robust-Fill is designed for the string PBE.

---

3https://pytorch.org/  
4https://github.com/pytorch/fairseq
We adopted the model as tabular transform PBE to compare with our neural model using tabular data linearization and program linearization described in IV-B and IV-C respectively. See Fig. 10 for overview of the model.

b: Search-based Baseline System
We implemented Foofah as a baseline search-based system to compare our neural model using the A*-graph search as the core search algorithm with the table-edit-distance batch as the heuristic function and the same pruning rules as with Foofah. Its operators are listed in Table 3 and slightly differ from Foofah.

B. ANALYSIS OF OUR NEURAL NETWORK MODELS
In this subsection, we present and analyze the experimental results from our proposed models and the baselines. All experimental results are listed in Table 8 and are referred to by Experiment number in the following discussions.

1) Comparing the LSTM-based Model with the Transformer-based Model
We compared a baseline (LSTM-based) model and our Transformer-based models with 2 and 6 layers in terms of the learning time required to train them. Table 9 gives the training times for each model.

Note that the LSTM-based model took much longer to train for one epoch in average, despite the much smaller number of parameters for the LSTM-based model than for the Transformer-based models.

Fig. 11 shows the learning curves for the models. The loss value for the Transformer model decreases rapidly and reaches a stable point of validation loss after about one day for the 6-layer model and about two days for the 2-layer model. These two models finish their training at this point. However, the loss curve for the LSTM-based model decreases very slowly and does not reach a stable point of validation loss, even after 14 days.

Experiment numbers 1, 4, and 10 in Table 8 give the accuracy for the benchmarks of the LSTM-based model and both Transformer-based models (2-layer and 6-layer). The Transformer-based models outperform the LSTM-based model for both large and small benchmarks.

The reason might be that the Transformer-based model has a greater capacity to learn a statistical distribution and capture the features of the training datasets much more quickly than LSTM-based model.

While the Transformer-based model might be better performed by being trained from a much larger variety of datasets as described in the following experiments, the LSTM-based model can not be trained any more due to the much longer training time.

| Model                      | Accuracy on small data | Accuracy on large data |
|----------------------------|------------------------|------------------------|
| LSTM-based                 | 69.8                   | 10.7                   |
| Transformer (2 layers)     | 87.6                   | 33.8                   |
| Transformer (6 layers)     | 86.3                   | 36.1                   |

2) Comparing Models Trained on Various Numbers of Samples
We examined the performance on training datasets of various sizes. The corresponding Experiment numbers are 2–13 in Table 8. Fig. 12 gives these experimental results as a bar plot.

We can note that, in general, as the number of training samples increases, the accuracy for both large and small datasets also increase.

A second tendency is that the 6-layer Transformer-based model shows higher accuracy with large-scale training data.
(1M, 2M, 4M, 10M) than the 2-layer model with a few exceptions. Because increasing the numbers in the training dataset enables the models to learn a larger variety of tabular data and achieve a more expressive ability, we can hypothesize that the model with the larger capacity (the 6-layer model) can learn expressive representations more effectively.

4) Comparing Various Beam Search Methods
We compare the accuracy of our proposed multistep beam search and PV-beam search with basic beam search and iterative beam search.

We implemented the iterative beam search with reference to the work [15] for comparing to our proposed decoding methods. The iterative beam search repeats the basic beam search iteratively avoiding searching the hypotheses seen in the previous iterations and eventually searches wider search space diversely.

We set the beam size from 1 to 1000 and a timeout to 30 seconds in our experiments. Each experiment aborts when the timeout occurs. In the experiments of the following sections, the combined token rule described in IV-C is employed as a program linearization method.

The experimental results here is given Experiment number 22-49 in Table 8 and Fig. 14 shows plots for these results.

The performance of the basic beam search improves by increasing the beam size because the possibility that correct hypotheses get to be removed decreases with the increased beam size.

The performance of the iterative beam search outperforms the basic beam search with the small beam size because the iterative beam search repeats the basic beam search and thereby searches the extensive search spaces. However, its performance does not better than the basic beam search with the beam size of over 50 because the second iteration of iterative beam search takes over 30 seconds to process due to the time-consuming process of checking whether each hypothesis has explored in the previous iteration.

The performance of the multistep beam search with the shortest path strategy is shown in the experimental results because we observed the performance for shortest path strategy is slightly better than or equals to that for longest path strategy in almost experimental settings. The performance of the multistep beam search betters according to the increase of beam size and achieves the best at the beam size of 500, eventually outperforming the iterative beam search and the basic beam search.

Since multistep beam search repeats the basic beam search changing the starting point of the search until it succeeds to find a consistent program, it always succeeds whenever the basic beam search does. The light-weight computational cost of it results in outperforming the basic beam search with every beam size, although the iterative beam search does not outperform the basic beam search with the beam size over 50.

The performance of the PV-beam search outperforms other beam search methods with beam size of over 50 in terms of the accuracy on the all data and achieves the best accuracy among other beam search methods.

5) Comparing the Proposed Model with Baselines
We show the experimental results of two baseline systems (LSTM-based system and search-based system) and the Transformer model with our proposed methods (multistep

3) Comparing Models for a Variety of Program Lengths
We evaluated the accuracy with respect to program length. The corresponding experimental results are Experiment numbers 14–21 in Table 8. Fig. 13 shows these experimental results in bar-plot form.

Although we observe no significant differences for the variety of program lengths, the accuracy for the 6-layer model is always higher than for the 2-layer model for the program length of over 6. This also indicates that the 6-layer model has sufficient capacity to train via datasets containing long programs (as was also found in Section V-B2).

We use 10M for the number of samples in training data and 6 for the program length of training data in the following experiments, because these values show nearly the best performances in V-B2 and V-B3.

5) Comparing the Proposed Model with Baselines
We show the experimental results of two baseline systems (LSTM-based system and search-based system) and the Transformer model with our proposed methods (multistep

FIGURE 12. Accuracy with respect to number of samples in training data

FIGURE 13. Accuracy with respect to program length in training data
beam search and PV-beam search) and compares the performance of them. The results are Experiment number 1, 40, 41, 47, 48, 50 in Table 8 and summarized in Table 11.

**TABLE 11.** Accuracy of baselines and proposed methods

| model             | decoding method | accuracy on small data | accuracy on large data |
|-------------------|----------------|------------------------|------------------------|
| LSTM              | -              | 69.8                   | 10.7                   |
| Transformer       | multistep (beam=100) | 94.5                  | 49.1                   |
| Transformer       | multistep (beam=500) | 95.8                  | 53.1                   |
| Transformer       | PV-beam (beam=100)  | 93.1                  | 62.7                   |
| Transformer       | PV-beam (beam=500)  | 94.5                  | 64.4                   |
| search-based      | -              | 73.9                   | 63.2                   |

The LSTM-based system shows the worst performance on both small and large data. As described in Section V-B1, this is mainly because LSTM-based system does not have enough expressiveness for tabular transformation and can not be trained by large-scale training data. Therefore, the transformer model outperforms the LSTM-based system considerably.

The search-based system, which is the best system ever developed for tabular transformation PBE system to the best of our knowledge, shows the performance independent on the size of data and achieves good performance even on large data. While the Transformer model have outperformed the search-based system on small data, it has been difficult for the transformer-based model to outperform the search-based system. However, the Transformer-based system with our proposed PV-beam search (beam=500) achieved the better accuracy even on large data than the search-based system due to the efficient program generation processing of PV-beam search.

Finally, we compare our Transformer-based systems and the baseline systems in terms of response performance. Fig. 15 shows the comparisons of the response time of Experiment number 1, 40, 47 and 50 from Table 8, namely two baseline systems and the Transformer models with proposed decoding methods with beam size of 100. The horizontal axis shows the response time from the start of the inference and the vertical axis shows the corresponding accuracy at that response time.

Each panel shows the Transformer models outperform the LSTM-based system on every benchmark data from start to the timeout (30 seconds).

The top panel shows that the Transformer-based models better the search-based system on small benchmark data in terms of accuracy from the start to the timeout.

The center panel shows that the Transformer-based model with PV-beam search outperforms the search-based system except for a few seconds from start and ends up with almost even eventually, and the Transformer-based model with multistep beam search has a better response than other systems for the first few seconds.

The bottom panel shows that the Transformer-based model with PV-beam search outperforms the search-based system from start to timeout. And it also shows the Transformer-based model with multistep beam search has a much better response than other systems.

We observed in our experiments the Transformer model with PV-beam search with beam size of 500 and 1000 (Experiment number 48 and 49) loses to the search-based method for a few seconds from the start time because of the high computational cost of the program validation process (line 16-18 in Algorithm 3). Therefore, we conclude that the Transformer-based model with PV-beam search by beam size 100 is the best system considering the balance of both the final accuracy and the response.

We can also say that Transformer-based model with PV-beam search with beam size of 500 would be better if the final accuracy is preferred. And the Transformer-based model with multistep beam search would be better in the situation that a quick response is more desirable.
VI. CONCLUSION AND FUTURE WORK

In this paper, we propose a new ML approach to realize PBE for tabular transformations, which are required in data analysis and data integration scenarios.

In order to deal with tabular transformations, which are complex and require a large amount of data, we proposed an ML-based PBE for tabular transformations using the Transformer model, which is faster and more expressive than the LSTM model (baseline) used in conventional ML-based PBE, for tabular transformations using the Transformer model, which is faster and more expressive than the LSTM model (baseline) used in conventional ML-based PBE research.

In addition, we proposed multistep beam search and the PV-Beam search, which optimize the conventional beam search for program generation, and implemented them on top of the Transformer model. As a result, we found that the PV-beam search far surpasses the accuracy and response performance of the baselines and the multistep beam search realizes much shorter response time than the search-based system.

As a result, our proposed systems achieve superior performance to baselines and provide a high-performance and scalable ML model for tabular transformation PBE, which enables users to perform table transformation easily and effortlessly.

As one of our future works, we are planning to study an embedding method for 2D tabular data using a sophisticated method as proposed by Tapas [24] which will improve the simple linearizing method in this paper.

REFERENCES

[1] Zhongjin Jin, Michael R Anderson, Michael Cafarella, and HV Jagadish. Foofah: Transforming data by example. In Proceedings of the 2017 ACM International Conference on Management of Data, pages 683–698, 2017.
[2] Vijayshankar Raman and Joseph M Hellerstein. Potter’s wheel: An interactive data cleaning system. In VLDB, volume 1, pages 381–390, 2001.
[3] Jacob Devlin, Jonathan Usatto, Surya Bhupatiraju, Rishabh Singh, Abdelrahman Mohamed, and Pushmeet Kohli. Roobustifl: Neural program learning under noisy i/o. In International conference on machine learning, pages 990–998. PMLR, 2017.
[4] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neural-symbolic program synthesis. arXiv preprint arXiv:1611.01855, 2016.
[5] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. ACM Sigplan Notices, 46(1):317–330, 2011.
[6] William R Harris and Sumit Gulwani. Spreadsheet table transformations from examples. ACM SIGPLAN Notices, 46(6):317–328, 2011.
[7] Daniel W Barowy, Sumit Gulwani, Ted Hart, and Benjamin Zorn. Flashrelate: extracting relational data from semi-structured spreadsheets using examples. ACM SIGPLAN Notices, 50(6):218–228, 2015.
[8] Rohan Bavishi, Caroline Lemieux, Roy Fox, Koushik Sen, and Ion Stoica. Autopandas: neural-backed generators for program synthesis. Proceedings of the ACM on Programming Languages, 3(OOPSLA):1–27, 2019.
[9] Ashwin Kalyan, Abhishek Mohta, Oleksandr Polozov, Dhruv Batra, Prateek Jain, and Sumit Gulwani. Neural-guided deductive search for real-time program synthesis from examples. In International Conference on Learning Representations, 2018.
[10] Rishabh Singh. Blinkfill: Semi-supervised programming by example for syntactic string transformations. Proceedings of the VLDB Endowment, 9(10):816–827, 2016.
[11] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. arXiv preprint arXiv:1409.3215, 2014.
[12] Sam Wiseman and Alexander M Rush. Sequence-to-sequence learning as beam-search optimization. arXiv preprint arXiv:1606.02960, 2016.
[13] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. arXiv preprint arXiv:1702.01806, 2017.
[14] Daphne Koller and Ramesh M. Rao. Discriminative models. arXiv preprint arXiv:1606.03532, 2017.
[15] Ilia Kulikov, Alexander H Miller, Kyunghyun Cho, and Jason Weston. Importance of search and evaluation strategies in neural dialogue modeling. arXiv preprint arXiv:1811.00907, 2018.
[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. arXiv preprint arXiv:1706.03762, 2017.
[17] Larissa Laich, Pavol Bielik, and Martin Vechev. Guiding program synthesis by learning to generate examples. In Proceedings of the 8th International Conference on Learning Representations (ICLR 2020). International Conference on Learning Representations (ICLR), 2020.
[18] Natarajan Nagarajan, Simmons Danny, Datha Naren, Jain Prateek, and Gulwani Sumit. Learning natural programs from a few examples in real-time. Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATs) 2019.
[19] Raza Mohammad and Gulwani Sumit. Disjunctive program synthesis: a robust approach to programming by example. 2018, Association for the Advancement of Artificial Intelligence.
[20] Richard Shin, Neel Kant, Kavi Gupta, Chris Bender, Brandon Trabucco, Rishabh Singh, and Dawn Song. Synthetic datasets for neural program synthesis. In International Conference on Learning Representations, 2019.

[21] Judith Clymo, Haik Manukian, Nathanad Fijalkow, Adria Gascón, and Brooks Paige. Data generation for neural programming by example. In Silvia Chiappa and Roberto Calandra, editors, Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics, volume 108 of Proceedings of Machine Learning Research, pages 3450–3459. PMLR, 26–28 Aug 2020.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., 2019.

[23] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In Proceedings of NAACL-HLT 2019: Demonstrations, 2019.

[24] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. arXiv preprint arXiv:2004.02349, 2020.

YOSHIFUMI UJIBASHI was born in 1973. He received his B.S. degree from Waseda University in 1997 and his M.S. degree from the University of Tokyo and joined Fujitsu Limited in 1999. He is currently a senior researcher at Fujitsu Ltd and a Ph.D. student at the Graduate University for Advance Strudies, SOKENDAI. His research interests are data engineering and machine learning. He is a member of IPSJ.

ATSUHIRO TAKASU received his B.E., M.E., and Dr. Eng. from the University of Tokyo in 1984, 1986, and 1989, respectively. He is a profes-

or at the National Institute of Informatics, Japan. His research interests are data engineering and data mining. He is a member of ACM, IEEE, IEICE, IPSJ and JSAI.

***