

Synthesizing Tabular Transformations from Examples using Transformer-based Neural Networks

by

Yoshifumi UJIBASHI

Dissertation

submitted to the Department of Informatics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy



The Graduate University for Advanced Studies, SOKENDAI
September 2022

A dissertation submitted to Department of Informatics,
School of Multidisciplinary Sciences,
The Graduate University for Advanced Studies, SOKENDAI,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Advisory Committee

Prof. Atsuhiro TAKASU (Chair)	National Institute of Informatics, The Graduate University for Advanced Studies, SOKENDAI
Prof. Akiko AIZAWA	National Institute of Informatics, The Graduate University for Advanced Studies, SOKENDAI
Assoc.Prof. Norio KATAYAMA	National Institute of Informatics, The Graduate University for Advanced Studies, SOKENDAI
Assoc.Prof. Fuyuki ISHIKAWA	National Institute of Informatics, The Graduate University for Advanced Studies, SOKENDAI
Assist.Prof. Hiroyuki KATO	National Institute of Informatics, The Graduate University for Advanced Studies, SOKENDAI
Prof. Tatsuya AKUTSU	Kyoto University

Abstract

Data analysis with a large size of data from various public or enterprise data sources allows us to discover new knowledge or rules that no one has found so far. Data analysis thus has been recognized as a promising process for decision making by business people or policy making by governments in these days.

Data preparation is an essential process as the first step of data analysis. Data from various sources is often in a non-relational or unstructured form and is not able to be directly inserted into a downstream information system, like a database or visualization systems. Data analysts transform such raw data into a desired format that can be easily consumed by other systems. Such a data preparation task regularly involves reformatting data values, reconstructing layout of tables, looking up values with keys from other tables, and integrating multiple data sources, often requiring programming skills. Therefore, it is a laborious and time-consuming task for data analysts who have limited coding skills. Generally, data analysts spend more time for preparing data than analyzing it: the process takes up to 80% of data analyst's time.

Programming by example (PBE) is a technique that makes this troublesome task easier for data analysts by automatically generating programs for data transformation. PBE is one research field of program synthesis. In a program synthesis problem, a program synthesizer takes a program specification from users and then automatically generates a program according to the specification. Meanwhile, in a PBE problem, a program synthesizer takes an input-output example as a specification from users and then automatically generates a program consistent with the input-output example. The system with a PBE program synthesizer should solve the data preparation problem. It allows a user to synthesize a program through specifying a desired transformation by providing an input-output example. The user only needs to know how to describe the transformed data without knowing any particular transformation operation or programming code.

In past years, researchers have been studying PBE using techniques based on non-neural algorithms such like graph search algorithm or version space algebra. Many researchers started studying PBE using techniques based on ML approaches in recent years, inspired from many successes of machine learning (ML) or neural network models. One example of such study, RobustFill, has been proposed as a neural network model that generates a string transformation program. It employs an encoder-decoder model

that consists of two long-short-term-memory (LSTM) architectures. Once the model is trained via supervised training, it translates input-output strings into the corresponding transformation program.

Although RobustFill shows a feasibility of ML-based PBE system and possibility of its application to string transformation, it does not support both syntactic (string) and layout transformations. Syntactic transformations re-format each cell content of a table, whereas layout transforms re-construct the layout of a table. The transformations in data preparation are composed by combining these transforms. We call these transforms as *tabular transformation* collectively.

The goal of this dissertation is to realize an ML-based PBE for tabular transformations. This is the first ML-based PBE for tabular transformations to the best of our knowledge. Furthermore, our experiments show that our neural model outperforms the existing non-neural PBE system for tabular transformations, thus indicating that our neural approach to synthesizing tabular transformation programs is promising. Our contributions are as follows.

First, we propose a new ML-based PBE system for tabular transformations. The ML-based PBE system is an encoder-decoder model based on the Transformer neural network which is known as the state-of-the-art translation neural network. Since tabular transformations have more intricate data structures and complicated transformations than string transformations, it requires larger expressive power, therefore, a larger number of parameters for the neural network models. The LSTM network which is used in conventional ML-based PBE systems is difficult to have such large parameters, because the LSTM network spends much longer time to train its parameters due to its sequential processing feature of recurrent neural networks. Thus, the LSTM cannot have such an expressive power that is capable of learning tabular transformations in a practical training time. To address this shortage, we propose a Transformer-based model as an ML-based PBE system instead of the LSTM.

Next, we propose an embedding method that embeds two-dimensional tabular data into the Transformer neural network model. We introduce tabular positional encodings which encodes the positions of each location of the tabular data. The tabular positional encodings deal with the tabular data in the Transformer model properly. This method allows us to embed each row index, column index and the local position in a cell of input-output example tables to represent two(or more)-dimensional positions in the Transformer network. Our experiments show that the tabular positional encodings improve the performance of Transformer-based model through learning and capturing the structure of two-dimensional tabular data.

Finally, we propose two decoding methods, multistep beam search and Program Validation (PV)-Beam Search. Our Transformer-based model generates a sequence which corresponds to a program for input-output example tables from the Transformer decoder in the encoder-decoder model. The Transformer decoder firstly outputs how likely each program component occurs according to the provided input-output examples. Next, it generates the most likely programs from the likelihoods. Generally, the generations of the most likely programs from the likelihoods is performed by beam search. However, the beam search is not a technique designed for program generation, thereby causing an inefficient exploration of the program search space. Our proposed two variants of beam search are optimized for the program generation task, hence generating correct programs at higher probability than the original beam search. Our experimental results ensure that the Transformer-based model with our proposed decoding method outperforms the conventional PBE system for tabular transformations.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Atsuhiro TAKASU for supporting my research continuously. My dissertation will not be accomplished without his technical supports and continuous guidance and encouragements. I also extremely thank advisory committee, Prof. Akiko Aizawa, Prof. Norio KATAYAMA, Assoc. Prof. Fuyuki ISHIKAWA, Assist. Prof. Hiroyuki KATO, and Prof. Tatsuya AKUTSU for their insightful and constructive comments.

Moreover, I am specially grateful to secretary Akiko TAKENAKA, and all the members in Takasu Laboratory, including Takaya KAWAKATSU, LE Hong Van, and Panagiotis ANDRIOTIS. They gave me a lot of advices and things helpful in my research.

Furthermore, I am grateful to my managers and my colleagues belonging to Fujitsu Laboratories. They gave me an opportunity to pursue PhD and many supports and advices to help my continuous study whilst working.

Last but certainly not least, I would like to express my appreciation to my family for their gracious supports and encouragements through my life.

Contents

Abstract	i
Acknowledgements	iv
Contents	v
List of Figures	ix
List of Tables	xi
1 Introduction	1
1.1 Overview	1
1.1.1 Data Preparation	1
1.1.2 Challenges in Data Preparation	2
1.1.3 Programming by Example	4
1.2 Contributions	5
1.3 Outline	6
2 Problem Formulation	7
2.1 Problem Definition	7
2.2 Tabular Transformation	8
2.2.1 Transforms for Potter’s Wheel	8
2.2.1.1 Transform of Individual Data Values	9
2.2.1.2 Vertical Transforms: One Row to One Row Mappings	9
2.2.1.3 Horizontal Transforms 1	12
2.2.1.4 Horizontal Transforms 2: Many-to-Many Mappings	14
2.2.1.5 Summary of Expressive Power of Potter’s Wheel	16
2.2.2 Operations for Our Study	16
2.2.2.1 Operations Almost Similar to Those of Potter’s Wheel	16
2.2.2.2 Operations Deleted	18
2.2.2.3 Operations Added or Modified for Simplifying Program	18
2.2.2.4 Operations Added for Increasing Expressive Power	19
2.2.2.5 Summary of Expressive Power of Our Operations	19
2.2.3 A Tabular Transformation Example	20
2.3 Overview of ML-based PBE system	21
3 Related Work	22

3.1	Data Preparation	22
3.1.1	SchemaSQL	22
3.1.2	Potter’s Wheel	23
3.1.3	Wrangler and Recent Works	24
3.2	PBE	25
3.2.1	Search-Based PBE Studies	26
3.2.1.1	FlashFill	28
3.2.1.2	Foofah	28
3.2.2	ML-based PBE Studies	30
3.2.2.1	RobustFill	31
3.3	Encoder-Decoder Neural Networks	32
3.3.1	Long Short Term Memory (LSTM)	33
3.3.2	Transformer	34
3.3.3	Transformer-based model on Larger Data	37
3.3.4	Synthesizing Realistic Training Data	37
3.3.5	Training Objectives	38
3.4	Two-dimensional Tabular Data Embeddings	38
3.4.1	TAPAS	39
3.4.2	Vision Transformer	39
3.5	Beam Search	41
3.5.1	Iterative Beam Search	42
4	Transformer-based Neural Model for Tabular Data	43
4.1	Transformer-based Encoder-Decoder Model	43
4.2	Tabular Data Linearization	44
4.2.1	Scanning Direction	45
4.3	Program Linearization	45
4.3.1	Separated Token Rule	45
4.3.2	Combined Token Rule	46
4.4	Model Description	46
4.5	Transformer-based Model with Tabular Positional Encodings	49
4.5.1	Positional Encoding	49
4.5.2	Tabular Positional Encoding	50
4.5.2.1	Additive Tabular Positional Encoding (ATPE)	51
4.5.2.2	Concatenative Tabular Positional Encoding (CTPE)	52
4.5.3	Positional Encoding Implementations	52
4.5.3.1	Fixed Encoding	52
4.5.3.2	Learned Encoding	52
4.6	Experiments	53
4.6.1	Experimental Settings	53
4.6.1.1	Metrics	53
4.6.1.2	Training Datasets	53
4.6.1.3	Benchmark Datasets	54
4.6.1.4	Hyperparameters	55
	Common Parameters	55
	Parameters different depending on Experiments	56
4.6.1.5	Hardware and Software Settings	57

4.6.1.6	Baselines	57
	The LSTM-based Baseline System	57
	The Search-based Baseline System	59
4.6.2	Preliminary Experiments using a Variety of Training Data	59
4.6.2.1	Comparison of Linearization Methods	59
4.6.2.2	Comparison of Models Trained on Large Size Tables	61
4.6.2.3	Comparison of Models Trained on Various Numbers of Samples	62
4.6.2.4	Comparison of Models for a Variety of Program Lengths	64
4.6.3	Comparison of the LSTM-based Model and the Transformer-based Model	65
4.6.4	Comparison of Variations of Tabular Positional Embeddings and Baselines	67
4.7	Visualization of Attentions	68
4.7.1	Attention Weights	68
4.7.2	An Example of Attention	68
5	Decoding Methods for Program Space Exploration	72
5.1	Multistep beam search	72
5.2	PV-Beam Search	75
5.3	Experiments	75
5.3.1	Experimental Results	75
5.3.2	Comparison of Various Beam Search Methods	76
5.3.3	Comparison of the Proposed Beam Search Methods and Baselines	78
6	Transformer-based Model with TPE and PV-Beam Search	84
6.1	Experimental Evaluation	84
6.2	Towards Practical PBE Systems	87
6.2.1	Iterative PBE System	87
	Easiness of Creating and Giving Examples	89
	Consistent and Fast Program Synthesis	89
	Easiness of Validation	89
	Generalization of Program	89
	Resolving Failure Cases	90
6.2.2	Generalization	90
6.2.3	Resolving Failure Cases	93
6.2.3.1	Ambiguous Specification	93
6.2.3.2	Solvable Failure	94
6.2.3.3	Unsolvable Failure	95
6.2.4	Practical Data Preparation System	95
	Importing data	96
	Cleaning & Cleansing data	96
	Enriching Data	96
	Understanding data	96
7	Conclusion	98
7.1	Summary	98
7.2	Other Applications	99

7.3	Future Work	99
7.3.1	Directions for Study of Improving our model	100
	Large-Scale Tabular Data	100
	Synthesis of Training Data	100
	Reinforcement Learning	100
	Benchmark Datasets	100
7.3.2	Directions for Study of Managing the PBE system	100
	Extensibility	100
	Practical PBE System	101

Bibliography	102
---------------------	------------

List of Figures

1.1	CRISP-DM diagram	2
2.1	Merge and Split	10
2.2	Divide	11
2.3	Higher order schematic heterogeneities	12
2.4	Demotes by Formats, Fold and Split	12
2.5	Folding without demoting	13
2.6	A series of Two-column Fold	13
2.7	An Unfold transform	14
2.8	An Unfold transform without an explicit column name to align	15
2.9	System overversion of our approach	21
3.1	Three stock schemas	23
3.2	Trifacta interface for suggestion of transform	25
3.3	Trifacta interface for overview	25
3.4	FlashFill: an extension of Microsoft Excel	28
3.5	A* algorithm	29
3.6	Neural sub-network architecture for single example	31
3.7	Neural network architecture for multi-example pooling	32
3.8	LSTM network	33
3.9	Sequence to sequence model	34
3.10	Transformer model architecture	35
3.11	Multi-head attention	36
3.12	Question answering from a table	39
3.13	Vision Transformer	40
4.1	Our Transformer-based model	44
4.2	Linearization processing for tabular data	44
4.3	Two types of scanning directions	45
4.4	Decoding process in the encoder-decoder model	47
4.5	The Transformer-based model	50
4.6	Tabular positional encodings	51
4.7	An example of synthesizing training data	54
4.8	Examples of “small” benchmark data	54
4.9	Examples of “large” benchmark data	55
4.10	LSTM-based model	57
4.11	Accuracy with respect to linearization variations	60
4.12	Accuracy with respect to table size for the training data	62

4.13	Accuracy with respect to number of samples in training data	63
4.14	Accuracy with respect to program length in training data	65
4.15	Learning curves	66
4.16	Input-output example for exp0_33_3	69
4.17	Self attention in encoder for exp0_33_3	70
4.18	Cross attention across encoder and decoder for exp0_33_3	70
4.19	Self attention in decoder for exp0_33_3	71
5.1	Tree structure \mathcal{T} and sequence \mathcal{S}	74
5.2	Graft processing	74
5.3	Prune processing	75
5.4	Accuracy of various types of beam search with respect to beam width . .	79
5.5	Accuracy with respect to elapsed time with proposed decoding methods (beam size = 100)	81
5.6	Accuracy with respect to elapsed time with proposed decoding methods (beam size = 500)	83
6.1	Accuracy with respect to elapsed time with baselines and proposed methods	86
6.2	The desired transformation	87
6.3	Input-output example tables used in the first step	88
6.4	Input-output example tables used in the second step	88
6.5	Input-output tables of \mathcal{D}_1 (left) and \mathcal{D}_2 (right) in exp0_10 benchmark . .	91
6.6	Input-output tables of \mathcal{D}_3 (left) and \mathcal{D}_4 (right) in exp0_10 benchmark . .	91
6.7	Input-output tables of \mathcal{D}_5 in exp0_10 benchmark	91
6.8	Venn Diagram for a success case	94
6.9	Venn Diagram for a ambiguous specification case	94
6.10	Venn Diagram for a solvable failure case	95
6.11	Venn Diagram for a unsolvable failure case	95

List of Tables

1.1	An unstructured spreadsheet	3
1.2	A relational form of Table 1.1	3
2.1	Notations for Table 2.2	9
2.2	Definitions of Potter’s Wheel’s transforms	9
2.3	Operations for tabular transformation in our study	17
2.4	The table converted by splitting the second column of Table 1.1 with ‘:’	20
2.5	The table converted by deleting rows that are blank in the second column from Table 2.4	20
2.6	The table converted by filling each blank cell in the first column of Table 2.5 by the cell above	21
3.1	Table edit operators	30
4.1	Hyperparameters common for all experiments	55
4.2	Parameters varying on experiments	56
4.3	Experimental results of various linearization methods	59
4.4	Experimental results for models trained on large size of tables	61
4.5	Accuracy with respect to # of samples of training data	63
4.6	Accuracy with respect to program length in training data	64
4.7	Average time required to train each model in one epoch and the number of parameters for each model	65
4.8	Accuracy of LSTM-based model and Transformer-based models	66
4.9	Accuracy of the Transformer-based model with tabular positional encoding and baselines	67
5.1	Accuracy with respect to beam size for various decoding methods	77
5.2	Accuracy of baselines and proposed methods	80
6.1	Accuracy of baselines and proposed models	85
6.2	Generalization (our proposed model)	92
6.3	Generalization (search-based)	92

Chapter

1

Introduction

1.1 Overview

1.1.1 Data Preparation

Data integration from a variety of data sources into a unified format is a time-consuming and labor-intensive task for engineers or domain specialists. ETL (Extract, Transform, Load) has long been a popular approach tackling this challenge on data integration. It became a popular concept from the 1970s and is often used in data integration tasks. It is composed of three-phase processes where it firstly extracts wanted data from data sources, then transforms data from various formats into a common pre-defined format, and finally loads data into a data container like a data-warehouse or a data mart. Through this processes, ETL systems integrate data from distributed or heterogeneous sources into data with a unified format. Many products for ETL has been released including well-known representative products like PowerCenter [1] and DataStage [2].

On the other hand, data preparation, called data wrangling as well, has emerged in recent years. It is a technique to prepare data from raw data which may come from disparate data sources into a form that can readily be analysed for business or analytical purpose.

These two techniques, ETL and data preparation, are similar in functions where both techniques transform heterogeneous data from multiple sources into a unified format to integrate them. In contrast, they are different in target users and applications [3].

ETL tools are intended to be used by IT professionals like developers or IT staffs who have programming expertise to some extent. ETL users make data conversion processes not only using graphical user interfaces (GUIs) equipped in ETL tools, but also using glue languages like Perl, Python and shell script or declarative languages like SQLs. The users use GUIs for typical easy processing, and conversely coding for uncommon complicated processing.

Additionally, ETL is intended to be used to extract kind of well-structured data from data sources such as Enterprise Resources Planning (ERP) [4], Customer Relationship Management (CRM) [5], or Human Resource (HR) databases in an organization or an enterprise. Then, it load transformed data into a data warehouse or a data mart which generates any kind of regular reports.

Data preparation tools are intended to be used by non-IT professionals such as domain specialists, scientists, physicians or executives. Data preparation aims to allow such users to convert their data easily for loading it into Business Intelligent (BI) tools or statistical analytics tools, consequently enabling the users to analyze their data quickly in an agile and flexible manner.

Data preparation is on focus these days due to its feature of civilizing novice IT users to data analytics. It leads such users to obtain new cutting-edge ideas and insights from the knowledge, through utilizing a lot of domain knowledge of those users effectively. This is the reason why we focus on realizing a more efficient and effective data preparation technique in this dissertation.

1.1.2 Challenges in Data Preparation

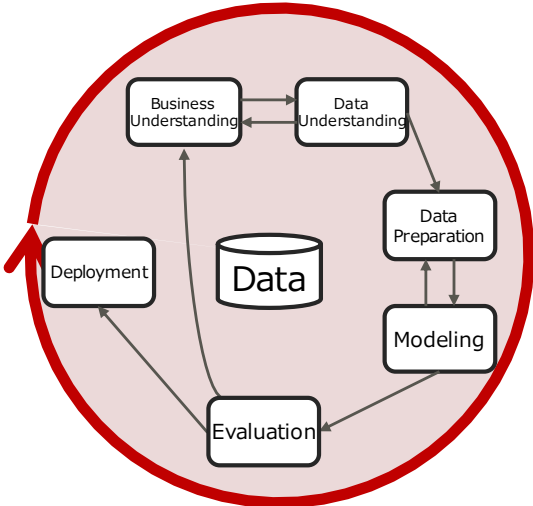


FIGURE 1.1: The process diagram showing relationships of phrases in CRISP-DM

CRISP-DM [6] diagram illustrated in Figure 1.1 represents data analysis processes. *Data preparation* is the pre-processing phase of *modeling* in the diagram. This phase includes a lot of time-consuming and cumbersome tasks for novice users, and thus is known to enforce analysts to spend their time as much as 50 to 80 percent of total time spent in data analysis [7]. Therefore, this data preparation task has long been a high hurdle in data analysis.

Here we illustrate an example of such transformations often seen in data preparation tasks. Suppose a data analyst wants to utilize a semi-structured tabular data shown in Table 1.1. This data needs to be stored in a database so that it can then be analyzed by downstream information systems or visualization systems. Since Table 1.1 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 1.2.

TABLE 1.1: An unstructured spreadsheet. This example is featured in Foofah [8]

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

TABLE 1.2: A relational form of Table 1.1. This example is featured in Foofah [8]

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

As pointed out in Gulwani et al. [9], transformations seen in many data preparation tasks are divided into two types: syntactic transformations and table layout transformations. Syntactic transformations convert the formats of cell contents, whereas layout transformations rearrange cell positions in the spreadsheet without changing the cell contents. The example here also consists of these transformations. We call such a transformation as *tabular transformation* in our study.

This task is difficult for users with little expertise to perform, because the task consists of a sequence of various kinds of transforms which is difficult for novice users to build in mind.

To eliminate this hurdle, various approaches have been studied. Potter’s Wheel [10] is an earlier work that develops an interactive tool for data cleaning, which are known as

data preparation in recent years. One of the main contributions of this work is defining transforms needed in data preparation tasks. The authors of Potter’s Wheel set their goal to designing transforms easy to specify graphically, flexible enough to be applied interactively, and at the same time powerful enough to express most common transforms. They choose a set of transforms to balance these goals. The set of transforms involves Format, Add, Drop, Copy, Merge, Split, Divide, Filter, Fold, and UnFold.

Data preparation tools such as OpenRefine [11], Trifacta [12], Tableau Prep [13], Paxata [14] and Alteryx [15] are proposed and developed in recent years, as open source software (OSS) [11] and as commercial products [12–15]. These tools equip many data-transformation functions which allow users to convert tabular data into a desired form viewing data as spreadsheet grids. A data preparation project Wrangler [16] designs its wrangler transformation language using the set of transforms proposed in Potter’s Wheel. To our knowledge, other data preparation tools basically equip a similar set of transforms to that of Potter’s Wheel.

However, even if users attempt to perform such tabular transformations using modern data preparation tools, they still find difficult to complete this task. Especially, syntactic transforms like Merge and Split include glue and splitter parameters often need regex-based parameters, and Filter is conditioned by predicate based on regular expressions as well. In addition, layout transforms like Fold and Unfold are simply difficult for users to follow how table cells move from sources to destinations as a result of the transform.

To address such a problem on the difficulty of successfully making tabular transformations, we propose using a Programming by Example (PBE) technique to automatically assemble tabular transformations from a user-provided input-output examples. We hope that the PBE system allows users to easily generate complicated tabular transformations.

1.1.3 Programming by Example

PBE is one type of program synthesis that has been studied for a long time [17]. Program synthesis is a technique that generates programs consistent with the specification conditions given by users. PBE takes an input-output example as a specification condition given by users. Thus, it allows users to automatically create a program by only providing input-output examples. It would be extremely helpful for users, since all they must know in performing data transformation is how to describe the desired output example.

Foofah [8] is a PBE study for generating a tabular transformation. It is built around techniques for solving search problems, that is, finding a program that converts a given input table to the output one from a huge program space. 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. Although Foofah is a representative study that suits to our problem domain that targets onto tabular transformation, there is more room to improve its methodology by a more modern approach such like machine learning for example.

Recently, many PBE studies based on Machine Learning (ML) approaches have emerged. Researchers are greatly motivated to these studies because their transformation scope can be easily expanded by simply enriching the training dataset. The previous work RobustFill [18] and the work by Parisotto et al. [19] propose ML-based PBE using an encoder-decoder model based on Long Short Term Memory (LSTM) neural model. Inspired from the success of LSTM in translation problem, RobustFill models their PBE problem as a translation one, where it translates an input-output example into the corresponding program. The experimental results show that an ML-based PBE has a competitive accuracy to the conventional non-neural PBE, and furthermore, achieves noise robustness that a non-neural PBE lacks. However, these studies aim to string transformation, not tabular transformation.

1.2 Contributions

In this dissertation, we propose a new ML approach to achieving PBE for tabular transformation, which is required in data preparation scenarios. The contributions of this dissertation are as follows:

- We propose a new ML-based PBE system for tabular transformations. We adopt a Transformer architecture upon an encoder-decoder model as an ML model for PBE. The model generates programs from a user-given input-output example in analogy with translation between two different languages as in translation from Japanese to English. Our experiments show that our Transformer-based model extremely outperforms the conventional LSTM-based model in performance of benchmark datasets for tabular transformations.
- We propose an embedding method to deal with two-dimensional tabular data upon neural network models. We implemented two types of tabular positional encoding, named as ATPE and CTPE, and two variants of the embedding layer, namely sinusoidal and learned. These methods allow us to embed each row index, column index

and the local position of input-output example tables to represent two(or more)-dimensional positions in the Transformer network. Our experiments indicate that the tabular positional encoding improves the performance of Transformer-based model through learning and capturing the structure of two-dimensional tabular data.

- We propose two types of 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. Our proposed two variants of beam search are optimized for the program generation task. Hence, it generates correct programs at higher probability than the original beam search. Our experimental results ensure that the Transformer-based model with proposed decoding method outperforms that with original beam search.
- We implemented the Transformer-based model with both proposed methods described above and evaluated the performance in experiments. Notably, our three proposals above can be applied to our model at the same time and improve their performance accumulatively. We empirically show that our Transformer-based model with our proposed decoding methods and tabular positional encoding markedly achieved better performance compared to the conventional search-based (non-neural) method which has been the most advanced study of PBE for tabular transformations.

1.3 Outline

The remainder of this dissertation is organized as follows.

We formally define the problem to solve in this dissertation in Chapter 2, and explore the literature related to this dissertation in Chapter 3. In Chapter 4, we present the Transformer-based model that generates a tabular transformation for input-output example tables, and next propose the embedding methods enabling the neural network models to handle two-dimensional tabular data. This chapter is mostly based on our work [20] and [21]. And then, in Chapter 5, we propose the decoding methods optimized for program generation, and it thereby generates programs efficiently. This chapter is mostly based on our work [20]. In Chapter 6, we evaluate our model with all of our contributions and discuss about practical PBE. Chapter 7 finally concludes this dissertation and presents directions for future work.

Chapter 2

Problem Formulation

2.1 Problem Definition

We now formally define the PBE problem discussed in this dissertation. We can define the PBE problem for tabular transformation formally as below.

Definition 2.1. (PBE Problem). Let e_i be an example tabular data extracted from \mathcal{R}_i , which is a raw tabular data to be transformed, and e_o be the tabular data the user wants to be transformed from e_i . Given a user-provided pair of examples $\mathcal{E} = (e_i, e_o)$, synthesize a program \mathcal{P} such that $e_o = \mathcal{P}(e_i)$.

Here, a raw tabular data \mathcal{R}_i is a table with two-dimensional grids, and notably includes schematic information like column names as well as data.

A program \mathcal{P} consists of a sequence of operations for tabular transformations, each of which transforms a tabular data into another tabular data. We do not introduce control structures like conditionals and loops to the program, thus, the program is a loop-free sequence of operations. Notably, this loop-free program could restrict us from undetermined number of iterations and then result in unrolled loops. However, we observe loop-free programs is successfully applied to prior works on data preparation [8, 10, 16], thus we also use loop-free programs as in the prior works.

Definition 2.2. (Tabular transformation program \mathcal{P}). \mathcal{P} is a tabular transformation program, which is a loop-free sequence of operations (o_0, o_1, \dots, o_k) . $o_i = (op_i, par_1, \dots)$

is an operation with operator op_i and its parameters (par_1, \dots) that transforms an input table t_i into an output table t_{i+1} , namely, $t_{i+1} = o_i(t_i)$.

In other expressions, a tabular transformation program \mathcal{P} is a sequence of operations (o_1, o_2, \dots, o_k) which obeys a chain rule where $t_1 = o_0(e_i)$, $t_2 = o_1(t_1)$, \dots , $e_o = o_k(t_k)$.

A domain-specific language determines a program space to be searched for programs consistent with an example. The domain-specific language for our tabular transformation is composed of a set of operators and parameters $\{op_1, op_2, \dots, par_1, par_2, \dots\}$ listed in Table 2.3.

2.2 Tabular Transformation

We formally describe *tabular transformation* in this section to make the scope of our problem clear. The tabular transformation is composed of syntactic transformations and layout transformations which are originally defined as a set of transforms for a data cleaning system studied in the work “Potter’s Wheel” [10].

We first describe the definitions of transforms for Potter’s Wheel, and then show the operations for our PBE system and how to define our operations drawing from the Potter’s Wheel transforms.

2.2.1 Transforms for Potter’s Wheel

Here, let us present the definitions of transforms for Potter’s Wheel. The operations for Foofah and our study are drawn from these transforms. We also discuss the expressive power of these transforms, and a scenario where data analysts often see in tabular transformation tasks.

Table 2.2 defines the transforms designed in Potter’s Wheel except for `Unfold` and `UnfoldSet` whose definition is complex and thus described in individual sections. We use notation defined in Table 2.1 here and throughout this section.

In the following section, we present all definitions of the transforms for Potter’s Wheel, which are categorized into three types of transformations: individual data values, one row to one row mappings (vertical transforms), and mappings of multiple rows (horizontal transforms).

TABLE 2.1: Notations for Table 2.2

Notation	Description
R	a relation with n columns
i, j	column indices and a_i denotes the value of a column in a row
x and glue	values
f	a function mapping a value into another value
$x \oplus y$	concatenates x and y
splitter	a string or a regular expression
$\text{left}(x, \text{splitter})$	the left part of x split by splitter
$\text{right}(x, \text{splitter})$	the right part of x split by splitter
predicate	a function returning a boolean

TABLE 2.2: Definitions of Potter’s Wheel’s transforms. This table is adapted from Potter’s Wheel [10]

Transform	Definition
Format	$\phi(R, i, f) = \{(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n, f(a_i)) \mid (a_1, \dots, a_n) \in R\}$
Drop	$\pi(R, i) = \{(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n) \mid (a_1, \dots, a_n) \in R\}$
Add	$\alpha(R, x) = \{(a_1, \dots, a_n, x) \mid (a_1, \dots, a_n) \in R\}$
Copy	$\kappa(R, i) = \{(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n) \mid (a_1, \dots, a_n) \in R\}$
Merge	$\mu(R, i, j, \text{glue}) = \{(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_{j-1}, a_{j+1}, \dots, a_n, a_i \oplus \text{glue} \oplus a_j) \mid (a_1, \dots, a_n) \in R\}$
Split	$\omega(R, i, \text{splitter}) = \{(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n, \text{left}(a_i, \text{splitter}), \text{right}(a_i, \text{splitter})) \mid (a_1, \dots, a_n) \in R\}$
Divide	$\delta(R, i, \text{predicate}) = \{(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n, a_i, \text{null}) \mid (a_1, \dots, a_n) \in R \wedge \text{predicate}(a_i)\} \cup \{(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n, \text{null}, a_i) \mid (a_1, \dots, a_n) \in R \wedge \neg \text{predicate}(a_i)\}$
Fold	$\lambda(R, i_1, i_2, \dots, i_k) = \{(a_1, \dots, a_{i_1-1}, a_{i_1+1}, \dots, a_{i_2-1}, a_{i_2+1}, \dots, a_{i_k-1}, a_{i_k+1}, \dots, a_n, a_{i_l}) \mid (a_1, \dots, a_n) \in R \wedge 1 \leq l \leq k\}$
Filter	$\sigma(R, \text{predicate}) = \{(a_1, \dots, a_n) \mid (a_1, \dots, a_n) \in R \wedge \text{predicate}((a_1, \dots, a_n))\}$

2.2.1.1 Transform of Individual Data Values

Format: Format applies a function to the value of a column in all rows. It includes functions of regular-expression-based substitutions and arithmetic operations. To support higher-order transformations, it allows for demoting names of columns and tables into column values. e.g. a value “George” in a *Name* column can be **Format**-ed into a self-describing representation “<\Name>George<Name>” using the substitution “.*” to “<\\C>\1<C>” where “\C” is a special character that denotes a column name. In addition, user defined functions (UDFs) can be defined to deal with situations involving complex types or specialized transformations (e.g. converting address, telephone number, date, etc. into another form).

Expressive Power: UDFs clearly allow **Format** to perform all transformations on individual data values.

2.2.1.2 Vertical Transforms: One Row to One Row Mappings

Vertical transforms are one-to-one mappings of tuples that typically perform column operations to unify data collected from multiple sources into a common format.

Drop, Copy, Add: Drop drops a column and Copy copies a column into next to the last column. Add adds a new column whose values can be set to a constant, a random number or a serial number. The last is often used for unique identifiers for data merged from different sources.

Merge with glue, Split by splitter: Merge concatenates values in two columns interposing a constant (the glue) in the middle to form a single new column. Split splits a column into two columns specifying either a position or a regular expression.

These operators are particularly useful for handling schematic differences. For example, suppose that names are specified in two columns `FirstName` and `LastName` in source A and `Name` column with a format of `LastName, FirstName` in source B. Assume that the `Name` column in source B has been mapped separately from the `FirstName` and `LastName` columns in source A, and the two sources are merged as seen in Figure 2.1. The user first `Format` names of the `LastName, FirstName` into `FirstName LastName`, then `Split` them into two columns, and then `Merge` the corresponding `LastName` columns and `FirstName` columns each other.

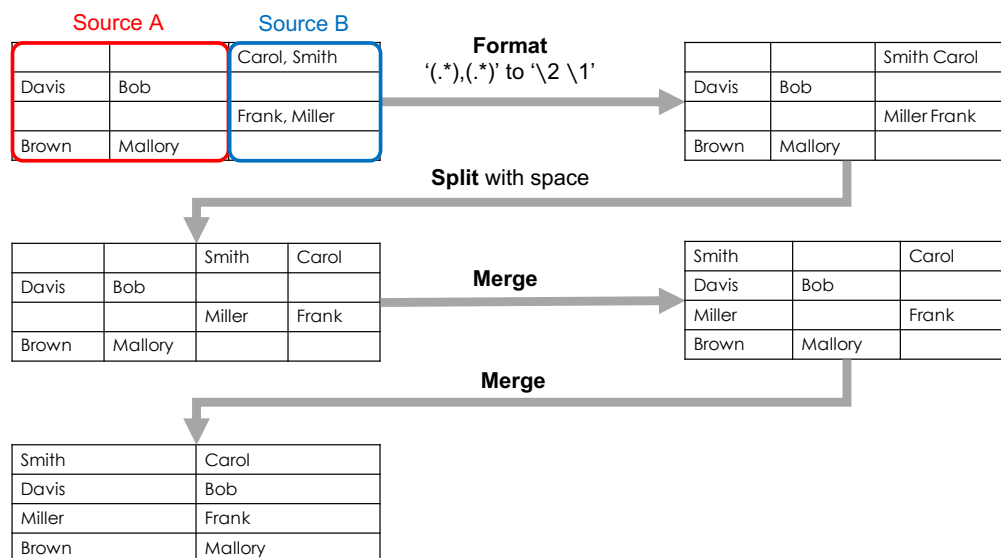


FIGURE 2.1: Merge and Split. This figure is based on Potter’s Wheel [10]

Divide: Divide divides a column according to the condition specified by predicate. Suppose that the `Name` column of source B has instead been mapped onto the `FirstName` column of source A as shown in the first column of the table in Figure 2.2. Divide is applied to vertically divide the first column into two columns using a predicate. As a result, the original values are going to the first or second column depending on whether they satisfy the predicate or not; the values consistent with the predicate moves to the first column and others to the second column.

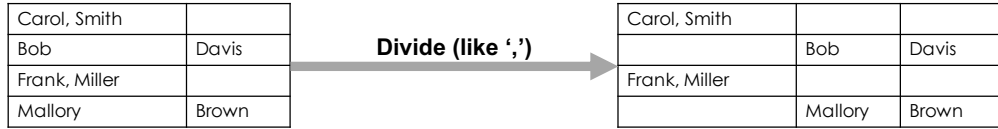


FIGURE 2.2: Divide. This figure is based on Potter’s Wheel [10]

The motivation for **Divide** is to support conditional transformation. It is needed to bunch values with logically different patterns (maybe from multiple sources) into separate columns. Predicates based on arithmetic and regular-expression-match are supported in Potter’s Wheel. Absence of the separate **Divide** transform complicates the GUI of Potter’s Wheel, because all other vertical transforms would have to accept predicates in their specification.

Expressive Power: The completeness of the vertical transforms for all one-to-one mappings of tuples is derived from the power of the **Format**; We can **Merge** all columns with a suitable glue, **Format** the result, and **Split** using a glue, to perform any one-to-one mapping of tuples. This is formally described as follows:

Theorem 2.3. *Vertical Transforms, along with **Format**, can be used to perform all one-to-one mappings of rows.*

Proof. Suppose we want to transform a row (a_1, a_2, \dots, a_n) into (b_1, b_2, \dots, b_m) . Let b_i be defined as $b_i = g_i(a_{i_1}, a_{i_2}, \dots, a_{i_l})$, where $\{a_{i_l} \mid 0 \leq l \leq n\}$ is a set of values picked up from (a_1, a_2, \dots, a_n) . Assuming that $|$ is a character that does not exist in the alphabets contained in the transforming or transformed rows, this transform is performed as follows: $\text{split}(\text{Format}(\text{merge}(a_1, a_2, \dots, a_n, |), \text{udf}), |)$ where **split** and **merge** are m -ary and n -ary variants of **Split** and **Merge** transforms respectively, and **udf** is a UDF that converts $a_1|a_2|\dots|a_n$ into $b_1|b_2|\dots|b_m$. □

Although **Format**, **Merge** and **Split** are functionally complete, **Divide**, **Add**, **Drop** and **Copy** are also provided as well for usability reasons, because many operations are more naturally specified through these transforms.

In addition, **Drop** and **Copy** allow us to perform transformations using only g_1, \dots, g_m which are often expressed in terms of regular-expression and arithmetic-expression based **Format**. It allows us to avoid user programming as possible. Since multiple functions g_i uses an argument a_j and a **Format** automatically drops old value, the value needs to be copied explicitly. We can obtain each b_i through first making a **Copy** of a_{i_1}, \dots, a_{i_l} , and then applying g_i over them to form b_i . After applying this process to form b_1, \dots, b_m , we **Drop** a_1, \dots, a_n .

2.2.1.3 Horizontal Transforms 1

One-to-Many Mappings: Horizontal transforms help to tackle higher-order schematic Heterogeneities [22] where information is stored partly in data values, and partly in the schema. Figure 2.3 shows an example of higher-order schematic heterogeneities where a student’s grades are listed as one row per course in the right schema of 2.3, and as multiple columns of the same row in the left schema. A transform from the left to right schemas in Figure 2.3 is an example of one-to-many transforms which are described in this section and any-to-many transforms are described in the next section.

Name	Math	English
Bob	70	75
Alice	87	42
Mallory	92	72

Name		
Bob	Math	70
Bob	English	75
Alice	Math	87
Alice	English	42
Mallory	Math	92
Mallory	English	72

FIGURE 2.3: Higher order schematic heterogeneities. This figure is based on Potter’s Wheel [10]

Fold: Fold converts one row into multiple rows; Fold folds a set of columns that follows from a specified column as defined in Table 2.2.

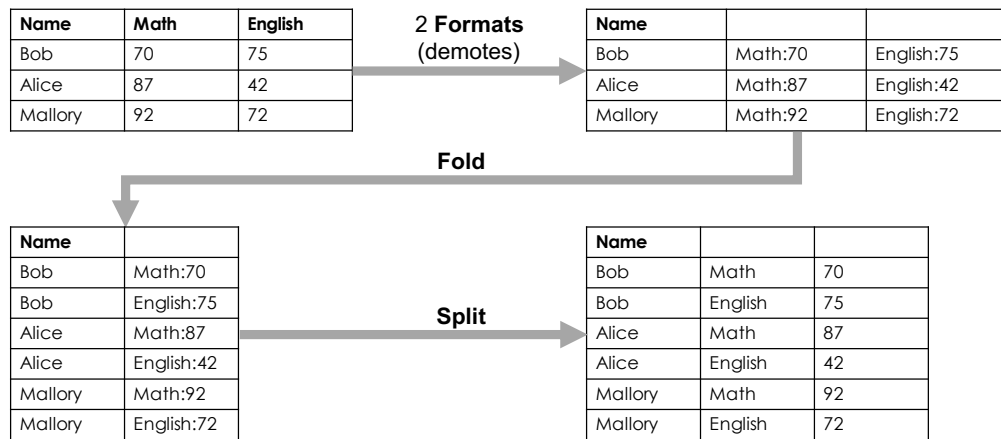


FIGURE 2.4: Demotes (Formats), Fold and Split. This figure is based on Potter’s Wheel [10]

Figure 2.4 shows an example of converting a table that contains some pieces of information about student and the corresponding grades. To resolve higher-order heterogeneities of this table, the subject names are demoted into the rows through Format, multiple grades are folded together through Fold and finally separated the subject from the grade through Split. Fold is drawn from the fold restructuring operator of SchemaSQL [23, 24] except that the fold operator of SchemaSQL both demotes column names and folds

columns together at the same time. Potter’s Wheel separates the demoting and folding functions into two operators, **Format** and **Fold**, because there are many situations where there are no meaningful column names to demote. Figure 2.5 shows an example of folding a set of values into a single column without demoting. It is notable that **Fold** needs the ability of performing arbitrary many columns at the same time because a series of two-column folds does not lead correct semantics as shown in Figure 2.6.

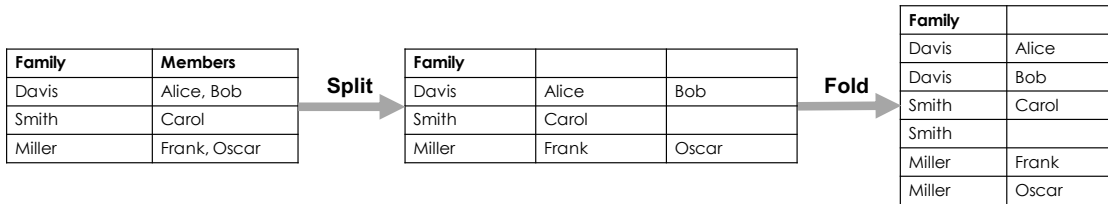


FIGURE 2.5: Folding without demoting. This figure is based on Potter’s Wheel [10]

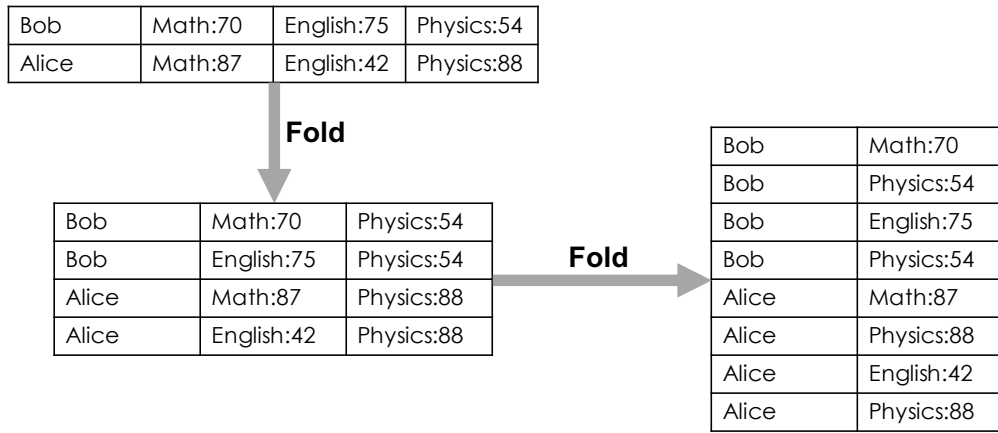


FIGURE 2.6: A series of Two-column Fold. This figure is based on Potter’s Wheel [10]

Filter: Filter is one of 1-to-1 or 1-to-0 mappings. Filter eliminates unnecessary rows using the predicate provided by users. Potter’s wheel supports arithmetic and regular expression-based predicates as well as UDFs.

Expressive Power: The combination of **Fold**, **Filter**, and vertical transforms and **Format** allows us to perform all one-to-many mapping of rows in a table. This theorem is proved as described below.

Theorem 2.4. *Horizontal Transforms when combined with Vertical transforms and Format can perform all one-to-many mappings of rows.*

Proof. Suppose that we want to map a row (a_1, a_2, \dots, a_n) to a set of rows $\{(b_{1,1}, \dots, b_{1,m}), (b_{2,1}, \dots, b_{2,m}), \dots, (b_{k,1}, \dots, b_{k,m})\}$. The number of output rows k itself can vary as a function of (a_1, a_2, \dots, a_n) . Let K be the maximum value of k for all rows in the domain of the desired mapping. Assume that $|$ is a character not present in the alphabet from which the values are chosen.

We first perform a one-to-one mapping on (a_1, a_2, \dots, a_n) to form $a(b_{1,1}|b_{1,2}|\dots|b_{1,m}, b_{2,1}|b_{2,2}|\dots|b_{2,m}, \dots, b_{k,1}|b_{k,2}|\dots|b_{k,m}, NULL, NULL, \dots)$ with $K - k$ NULLs at the end. We then perform a K way Fold and a Filter to remove all the resulting *NULL*s. Finally, we perform a m -way Split by $|$ to get the desired mapping. □

In addition to this, demoting through **Format** and folding/unfolding through **Fold/Unfold** allow us to move information between schema and data, which consequently flatten and unflatten tables.

Fold and **Unfold** transforms are basically similar to those of restructuring operators of SchemaSQL [23, 24]. For more detailed analysis of the expressive power of these transforms, see the literatures [23–25].

2.2.1.4 Horizontal Transforms 2: Many-to-Many Mappings

Transforms mapping multiple rows into one or more rows are the most generic ones. Such transforms are often quite useful for higher-order transformations.

Unfold: Unfold unflattens tables and moves information from data values to column names. Figure 2.7 shows an example of this unflattening process for a table using **Unfold** transform.

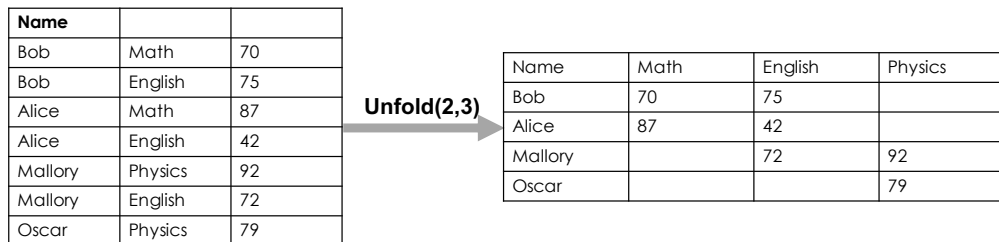


FIGURE 2.7: An **Unfold** transform. This figure is based on Potter’s Wheel [10]

Notably, **Unfold** is not the exact inverse of **Fold**. **Fold** takes a set of columns and folds them into one column, replicating other columns, whereas **Unfold** takes two columns (not one column) and collects rows that have the same values in all the other columns, and finally unfolds the two chosen columns. **Unfold** takes two columns because values in one column are used as column names to align the values in the other column.

Unfold transform is formally defined as follows. $\text{Unfold}(T, i, j)$ on the i -th and j -th columns of a table T with n columns named $(c_1 \dots c_n)$ produces a new table with $n+m-2$ columns named $(c_1 \dots c_{i-1}, c_{i+1} \dots c_{j-1}, c_{j+1} \dots c_n, u_1 \dots u_m)$ where $\{u_1 \dots u_m\}$ are the maximal set of distinct values of the i -th column in T .

Every set of rows produces exactly one row, where the set of rows have identical values in all columns except the i -th and j -th columns. Specifically, suppose a set \mathcal{S} is a set of k rows in $T (a_1 \cdots a_{i-1}, u_l, a_{i+1} \cdots a_{j-1}, v_l, a_{j+1} \cdots a_n)$ where l takes $k \in \{1, \dots, m\}$ distinct values, the set of rows \mathcal{S} produces a row $(a_1 \cdots a_{i-1}, a_{i+1} \cdots a_{j-1}, a_{j+1}, \dots, a_n, v_1, v_2, \dots, v_m)$. If the set of rows \mathcal{S} does not have a row with $u_p \in \{u_1 \cdots u_m\}$ in the j -th column, the v_p in a produced row is set to *NULL*.

This Unfold transform is exactly similar to the unfold restructuring operation of SchemaSQL [23, 24]. Moreover, there is also another variant of Unfold transform that restructures set valued attributes. This variant is used in a situation where there is no explicit column name as we can see an example in Figure 2.8.

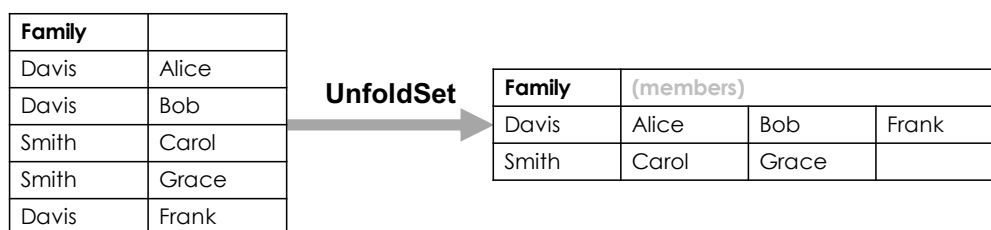


FIGURE 2.8: An Unfold transform without an explicit column name to align. This figure is based on Potter’s Wheel [10]

Unfold Variant: We often need to unfold sets of value without column names to align the unfolded values as seen in an example in Figure 2.8. The variant of Unfold is formally defined as UnfoldSet transform as below.

$\text{UnfoldSet}(T, i)$ on the i -th column of a table T with n columns named $(c_1 \cdots c_n)$ produces a new table with $n+m-1$ columns named $(c_1 \cdots c_{i-1}, c_{i+1} \cdots c_n, \text{NULL}, \text{NULL}, \dots, \text{NULL})$ with the m number of *NULL*s where m is the size of the largest set of rows in T with identical values in all columns except the i -th column. Specifically, suppose a set of k rows $\mathcal{S} = \{(a_1, a_2, \dots, a_{i-1}, v_l, a_{i+1}, \dots, a_n) \mid l = 1, 2, \dots, k\}$ with identical values in all columns except the i -th column, every set of rows \mathcal{S} in T produces one row $(a_1, a_2, \dots, a_{i-1}, a_{i+1}, \dots, a_n, v_1, v_2, \dots, v_k, \text{NULL}, \dots, \text{NULL})$ with the $(m - k)$ number of *NULL*s.

Note that the ordering of $u_1, u_2 \cdots u_m$ is not specified, because UnfoldSet does not make an alignment. The family members’ names could be permuted in the unfolded table in the example in Figure 2.8. A default ordering, such as lexical order, depends on implementations.

Expressive Power: Fold allows us to flatten tables into a unified schema where all schematic information is in columns, whereas UnFold allows us to revert the unified schema into a form where some information is in column names. Fold (with Format)

and `Unfold` transforms are able to move information between schema and data, thereby resolving schematic heterogeneities.

These transforms are basically similar to the operators of SchemaSQL. For more detailed analysis of their expressive power, refer to the literature [23–25].

2.2.1.5 Summary of Expressive Power of Potter’s Wheel

The transforms of Potter’s Wheel support all one-to-many (including one-to-one and one-to-zero) transformations of rows in table. In addition, they support moving information between schema and data, and flattening and unflattening tables, thus resolving schematic heterogeneities.

2.2.2 Operations for Our Study

The *tabular transformation* in our study is defined as a sequence of operations. Each operation is composed of an operator and corresponding parameters. Table 2.3 lists the operations which are inspired by Foofah [8]. They are very similar to those of Foofah with differences highlighted in the table. Foofah derives their operations from transforms of Potter’s Wheel described in Section 2.2.1, re-defining them in order to meet them to purposes of PBE tasks. Some operators are newly added for simplifying programs or increasing expressive power, whereas other operators are deleted because they do not match to the concept of PBE: program specifications are given by examples.

Our Operations are described in the following sections with the reasons why they are employed or not employed.

2.2.2.1 Operations Almost Similar to Those of Potter’s Wheel

Operations `Drop`, `Copy`, `Split`, `MergeToOne`, `Divide`, `Fold` and `Unfold` have almost similar functions to those with analogous names of Potter’s Wheel transforms.

Each of our `Fold` and `Unfold` corresponds to each variant of fold/unfold transforms without moving information between data and schema (namely, demoting). That is, our `Fold` corresponds to `Fold` for Potter’s Wheel and our `Unfold` to `UnfoldSet` for Potter’s Wheel. Another pair of variants of fold/unfold operations with demoting, `FoldHeader`/`UnFoldHeader`, are detailed in Section 2.2.2.3.

TABLE 2.3: Operations for tabular transformation in our study (**differences from the original Foofah operator is flagged in bold**)

Operator	Description
MoveToEnd(<i>pos</i>)	Moves a column to the last position (replaces Foofah's original Move operator)
MoveFromEnd(<i>pos</i>)	Moves the last column to another position (replaces Foofah's original Move operator)
Drop(<i>pos</i>)	Drops a column
Copy(<i>pos</i>)	Duplicates a column to the following position
CopyToDest(<i>pos</i> , <i>pos</i>)	Duplicates a column to another position (newly added)
Fill(<i>pos</i>)	Fills empty cells in a column with the one above
Split(<i>delim</i> <i>delimregex</i> , <i>pos</i>)	Splits a column into two columns at the occurrence of a delimiter or the match of a regex
MergeToOne(<i>delim</i> , <i>pos</i>)	Merges two or more columns after a position by a delimiter
Divide(<i>predicate</i> , <i>pos</i>)	Divides a column into two columns whereby one satisfies the predicate and the other does not
Extract(<i>regex</i> , <i>pos</i>)	Extracts the first match at a column with addition of the matched column (newly added)
Fold(<i>pos</i>)	Folds all the columns after the specified column into one column
Unfold()	Expands the folded columns of the table
FoldHeader(<i>pos</i>)	Folds the column names and the values of all the columns after the specified position
UnFoldHeader(<i>pos</i>)	Transform the folded columns into a relational form whereby the column names are placed in the table header
Delete(<i>pos</i>)	Deletes rows which have an empty cell at the specified column
DeleteEmptyCols()	Deletes columns where the all cells are empty
Transpose()	Transposes rows and columns of the table
Wrap(<i>pos</i>)	Concatenates rows which have the same values at the specified column
WrapToOne()	Concatenates all rows to one row of the table
WrapEveryKRows(<i>k</i>)	Concatenates every <i>k</i> row of the table into one row
Numeric parameters	Description
<i>pos</i>	Position of a column
<i>k</i>	Number of rows wrapped
Non-numeric parameters	Description
<i>delimregex</i> (newly added)	[<code>~!@#%\$%^&* ,./:;<>\(\)\[\]]</code>
<i>delim</i>	One of delimiter from: <code>- , @, #, \$, &, *, , ', ', ', ', ', ', /, :, ;, <, >, [,], (,)</code>
<i>regex</i> , <i>predicate</i>	One of regular expression from: <code>\w+, \d+, \W+, [^\d]+, \d+\.*\d+, [^\d\.]+, ^0+\$, \d{1,2}:\d{1,2}, (\d{1,4}/)?\d{1,2}/\d{1,2}</code> (some regular expressions are newly added)

2.2.2.2 Operations Deleted

Our operators do not include transforms `Format`, `Filter`, and `Add`. PBE synthesizes programs through a specification provided as examples that users demand. Accordingly, PBE, with this special programming paradigm, is not suited to such operations which are originally defined to use in tools with graphical user interface like Potter's Wheel.

Potter's wheel has `Format` that includes functions of regex-based substitutions, arithmetic operations, and UDFs. We provide `Extract` (see Section 2.2.2.3) instead of the regex-based substitution. `Extract`, combining with `MergeToOne`, theoretically performs similar transformation to the regex-based substitution of `Format`.

The arithmetic operations have less meaning in PBE. When users make arithmetic examples, they need to calculate output values using a calculator or functions in programming language except for pretty simple calculations. Therefore, synthesizing an arithmetic program conflicts the concept of PBE where users describe desired output example.

We do not prepare UDF related function for our PBE system. Users cannot define customized functions on demand in using PBE system, because user-defined operators need to be registered to PBE system previously.

We do not prepare `Filter` that eliminates unnecessary rows using a predicate. Predicates are expressed by arithmetic conditions or regex-based conditions. As to arithmetic conditions (for example, $a_i > 3$), they are difficult to be specified by example, because infinite number of examples are needed in the worst case to specify such an arithmetic condition. As to regex-based conditions, `Divide` (see in Section 2.2.2.4), `Drop` (see in Section 2.2.2.1), and `Delete` (see in Section 2.2.2.4) in our operations can instead theoretically perform a transformation similar to `Filter` with regex-based predicate.

Thus, our operations restrict transforms with the arithmetic calculation, arithmetic predicates, and UDFs, whereas it supports regex-based functions by combining other operations.

2.2.2.3 Operations Added or Modified for Simplifying Program

Operations `CopyToDest`, `MoveToEnd`, `MoveFromEnd`, `Extract`, `FoldHeader`, `UnFoldHeader` are additionally defined for simplifying generated programs and thus searching programs efficiently. Although these operations have already semantically equivalent combination of operations, we provide them to generate a simple and natural program.

Copy and Drop in Potter’s Wheel need a lot of transforms to perform coping or moving arbitrary i -th column to j -th column (coping required columns to next to the last column in desired order, and then, drooping original columns). We added `CopyToDest`, `MoveToEnd`, `MoveFromEnd` to prevent such too many transforms. We can copy i -th column to j -th column directly using `CopyToDest`, and move i -th column to j -th column using `MoveToEnd` and `MoveFromEnd` in order.

We newly added `Extract` that extracts a word or number from a column based on a pattern expressed by regular-expression based predicate. The operation partially performs the regex-based substitution in `Format` of Potter’s Wheel as described in Section 2.2.2.2.

We also provide `FoldHeader` and `UnFoldHeader`. These operations are variants of fold/unfold that involve operations moving column names between column headers and data. `FoldHeader`, for example, performs all operations (`2Formats`, `Fold`, and `Split`) in Figure 2.4 in one operation, consequently it transforms an unflatten table with column headers into a flatten table with column names embedded in data. In contrast, `UnFoldHeader` is equivalent to `Unfold` of Potter’s Wheel. It transforms a flatten table with column names embedded in data into an unflatten table with column headers.

2.2.2.4 Operations Added for Increasing Expressive Power

Operations `Fill`, `Delete`, `DeleteEmptyCols`, `Transpose`, `Wrap`, `WrapToOne` and `WrapEveryRows` are added in the previous work `Wrangler` [16] and `Foofah` [8]. The works found these operations are needed for resolving problems in real scenarios and added them to their work. Many tabular data practically include some blocks of information on a spreadsheet which are divided with empty lines, so that humans can visually recognize each disparate information. Furthermore, there are also many tabular data where schema information is embedded vertically on the left side of a spreadsheet for human’s readability or other reasons. We suppose `Fill`, `Delete`, and `DeleteEmptyCols` are useful for converting such data with empty lines, `Transpose`, `Wrap`, `WrapToOne` and `WrapEveryRows` for restructuring schema.

2.2.2.5 Summary of Expressive Power of Our Operations

Our operations support all one-to-many transformations of rows in table. They also support moving information between schema and data, and flattening and unflattening tables as well. Thus, they can resolve schematic heterogeneities, as Potter’s Wheel do, with a few restrictions on arithmetic calculations, arithmetic predicates, and UDFs.

In addition, they support additional operations that resolve problems in practical data preparation scenarios including conversions of tables with the empty rows and columns and restructure of the schema formats.

2.2.3 A Tabular Transformation Example

A tabular transformation program that converts from Table1.1 into Table1.2 can be constituted by an appropriate sequence of operations listed in Table 2.3. The following list is an example of sequence of operations converting the input Table 1.1 into the output Table 1.2

- Split(“:”, 2): split the column 2 with the delimiter “:”, which converts Table 1.1 to Table 2.4.
- Delete(2): delete rows including an empty cell at the column 2, which converts Table 2.4 to Table 2.5.
- Fill(1): fill empty cells by the value above at the column 1, which converts Table 2.5 to Table 2.6.
- UnFold(): unfold the table, which converts Table 2.6 to Table 1.2.

TABLE 2.4: The table converted by splitting the second column of Table 1.1 with ‘:’

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

TABLE 2.5: The table converted by deleting rows that are blank in the second column from Table 2.4

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

TABLE 2.6: The table converted by filling each blank cell in the first column of Table 2.5 by the cell above.

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

2.3 Overview of ML-based PBE system

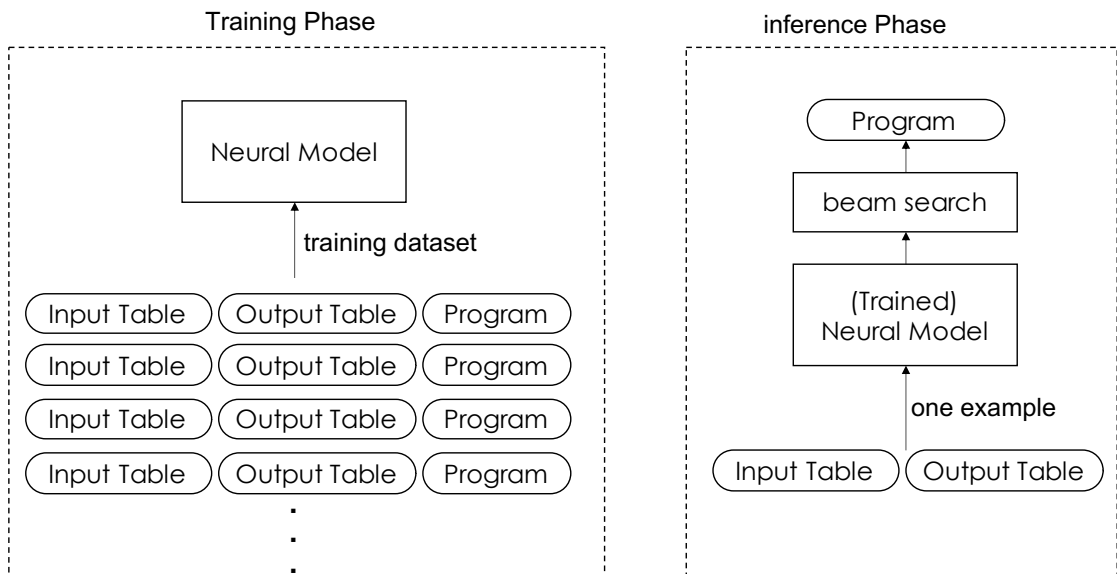


FIGURE 2.9: System overview of our approach

Fig. 2.9 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.

Chapter 3

Related Work

3.1 Data Preparation

3.1.1 SchemaSQL

Miller et al. [22] argues that SchemaSQL [23, 24] can be used to tackle schematic heterogeneity problems. Schematic heterogeneity means a situation where multiple databases and tables have disparate schemas. Data utilization requires integration of these databases and tables into one determined schema. Thus, our *tabular transformation* problem is quite similar to schematic heterogeneity problem which SchemaSQL solves.

SchemaSQL is a declarative language proposed in the works [23, 24]. It is a higher order extension of SQL. SQL makes use of tuple variables that range over the tuples of a relation declared in the *from* clause of a query. Extending SQL, SchemaSQL permits three additional types of variables: database, relation and attribute variables and creates views using these variables.

We show an example that explains how SchemaSQL creates views with extra variables. Figure 3.1 illustrates three relational schemas for stock information. All three schemas specify similar information about prices of company stocks. Suppose we want to obtain views with relational schema **s1** translating from schemas **s2** and **s3**. In that case, SQL can not create such views independent on data stored in tables, because SQL does not

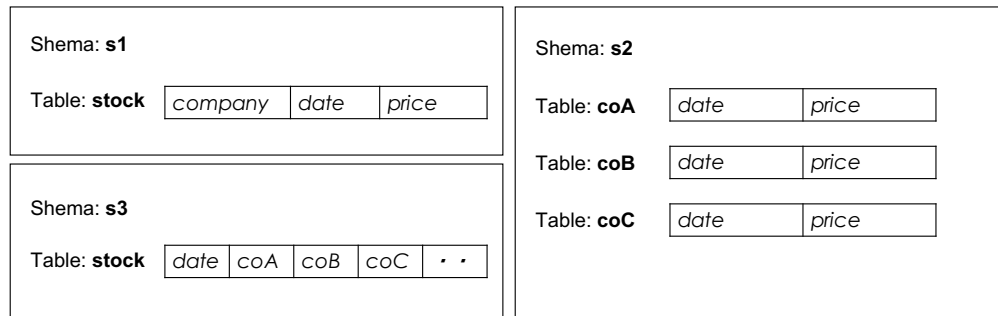


FIGURE 3.1: Three stock schemas. Attribute names are described in italic. Schema **s1** has a relational table whose attributes are a company name, a date, and the price of the company’s stock on that date. Schema **s2** has separate relational tables for all companies. Labels of the tables represent company names. Schema **s3** has a table that contains the price of a company’s stock of that date recorded in separate attributes. In Schema **s3**, the company names are labels for a set of attributes each containing price information. This example is adapted from Miller et al. [22]

permit quantification over relation or attribute names. If the set of company names changes, the view definition by SQL would need to be altered. On the other hand, SchemaSQL can create views by higher order queries independent on data described in Listing 3.1.

```

create view v2 (company, data, price) as
  select R, T.data, T.price
  from s2->R, R T

create view v3 (company, data, price) as
  select A, T.data, T.A
  from s3::stock->A, S3::stock T
  where A <> 'date'

```

LISTING 3.1: SchemaSQL views. This example is adapted from Miller et.al [22]

In Listing 3.1, `s2->R` declares `R` to be a relation variable ranging over all relations in `s2`. The `s3::stock->A` declares the variable `A` to be an attribute variable ranging over all attribute names in the relation `stock` in `s3` as well. And the variable `T` is a tuple variable declared ranging over the tuples of the relation `stock` in `s3`.

Thus, SchemaSQL introduces higher-order transformations used in Potter’s Wheel [10].

3.1.2 Potter’s Wheel

Potter’s Wheel [10] proposes an interactive framework for data preparation. It enables users to build transformations interactively by doing or undoing transforms in a intuitive, graphical manner through a spreadsheet-like interface. It has served as an exemplary

model for following studies and products on data preparation tools like Wrangler [16] which is the original study of Trifacta [12].

It supports a set of transformations composed of syntactic and layout transforms, which we call *tabular transformations*. These transforms are drawn from SchemaSQL which can be used for transformations within data records as well as higher-order transformations that resolve schematic heterogeneities. The transforms can also be used for conversions between some nested self-describing data and flat data formats. For further detail of the transforms of Potter’s Wheel, refer to Section 2.2.1.

3.1.3 Wrangler and Recent Works

Wrangler [16] is an interactive system for creating data transformations. It allows users to manipulate data visualized on its user-interface (UI) directly, with automatic inference and suggestion of relevant transforms. Users can iteratively explore the candidate transforms and preview their effects on the UI. Thus, Wrangler significantly reduces users’ time to specify transforms.

Futhermore, Wrangler has an underlying declarative transformation language. It also keeps histories of user’s interactive manipulations and support review, refinement, and annotation of the transformation histories written in the transformation language, thus promoting robust, auditable transforms instead of manual editing.

Wrangler extends the Potter’s Wheel with some key differences. It inherits **map** transforms like *delete*, *extract*, *cutting*, and *splitting*, and **reshape** transforms like *fold* and *unfold*, and in contrast extending their transforms with **lookups**, **joins**, **positional**, **sorting**, **aggregatoin**, and **schema** transforms.

Wrangler is an original work of Trifacta [12] and has been a pioneering work for the line of recent data preparation tools like Tableau Prep [13], Paxata [14] and Alteryx [15]. They equip many modern features needed for efficient data preparation processing within their sophisticated GUIs.

The GUIs equip interactive grids that allow users to perform simple text transformations, such as trimming a string, cleaning up data columns and extracting necessary rows through filters. Moreover, Trifacta have a feature for suggesting contextually the most relevant transform with a temporal window that shows the preview of result of the transform as in Wrangler (See Figure 3.2). Furthermore, their GUIs can give an overview of data type, data distribution and transform code to understand the characteristics of the overall data the user has and transforms the user is adapting (See Figure 3.3).

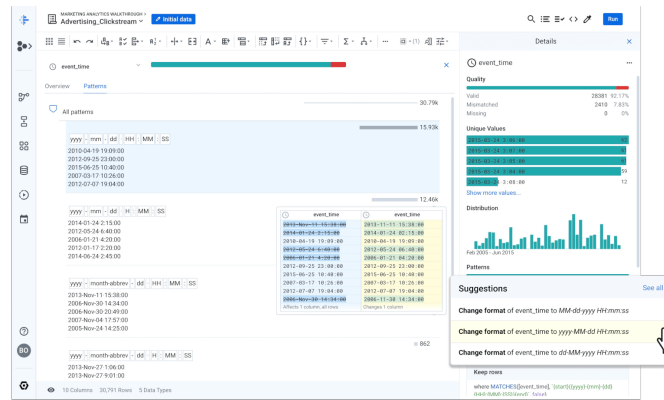


FIGURE 3.2: Trifacta Interface for Suggestion of Transform. This figure is a screenshot from the web cite [12]

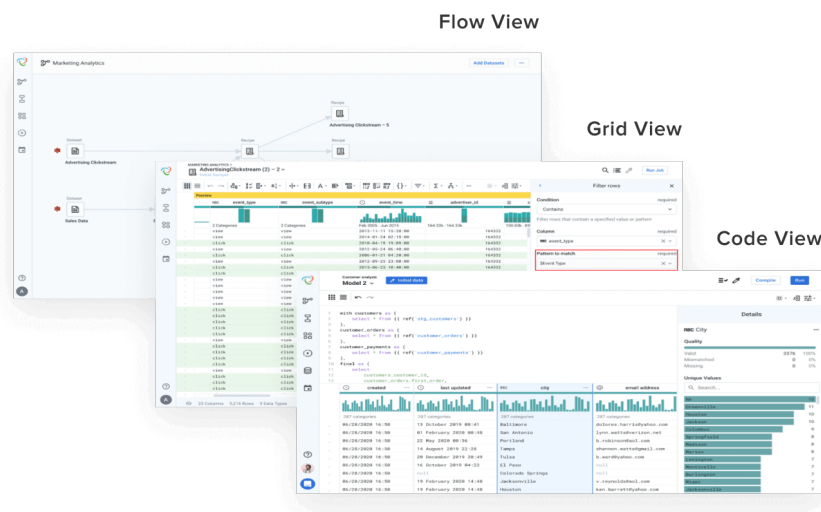


FIGURE 3.3: Trifacta Interface for Overview. This figure is a screenshot from the web cite [12]

Though these recent data preparation tools have many features that reduce efforts and costs of analysts, specifying transforms and their parameters is still difficult for novice users. To address this challenge, we pursue to apply PBE techniques to data preparation task, thus enabling novice users to automatically transform by just describing how they want to transform.

3.2 PBE

PBE is a subfield of program synthesis, where a program specification is given by input-output examples. Although writing the specification in program synthesis is difficult for those even with programming skills, writing the concrete input-output examples in PBE is much easier for users without programming expertise, thereby enables them to create

programs without tasks such as designing, coding and debugging. PBE is a promising approach that enables users like business users, domain specialists, and data analysts to generate programs for automating repetitive tasks.

Techniques that synthesize programs from given specifications are used to many applications such like graphics [26], code repair [27, 28], code suggestion [29, 30], concurrent programming [31]. Among these applications, as described in Section 1, we focus on the study about applying the PBE technique on “data preparation” (or data wrangling in some literatures) especially for tabular transformation.

PBE is realized on the top of search algorithms that search a program consistent with input-output examples from a huge hypothesis program space. In order to tackle solving exponentially large program space, many search techniques that reduce the program space by using pruning rules or semantic constraints have been developed. On the other hand, in recent years, stochastic search algorithms that leverage statistical or machine learning techniques have emerged. This approach is what we are interested in this dissertation.

We describe some PBE studies closely relevant to our study in the following sections. We call the PBE that uses stochastic search algorithms as “ML-based PBE” and the PBE that does not use stochastic search algorithms as “search-based PBE” in the following sections.

3.2.1 Search-Based PBE Studies

Many researchers have been studied PBE for long years. Here, we take a look around search-based PBE studies for data transformation. Data transformations, including tabular transformations we focus on in this dissertation, has been one of the most useful applications of PBE as mentioned in the book by Gulwani et al. [17].

FlashFill [9] is a representative work on PBE generating string transformations (syntactic transformations) once a user provides a set of examples. Refer to the further detail of this study in the next section Section 3.2.1.1.

Singh et al. [32] propose a transformation language for semantic transformations that combines table lookup operations and syntactic transformations. The semantic transformations involve string manipulations that need to be interpreted using a column entry from some relational table, or some standard data type such as date, time, or currency. They have developed the synthesizer that generate programs of the semantic transformations from examples users provide.

PBE can be applicable to number transformations and date transformations. Singh et al. present a PBE that generates programs for number transformations like formatting and rounding [32]. They also present a PBE system for cleaning data types [33], proposing a DSL with probabilistic semantics that is parameterized with declarative data type definitions. The PBE system allows for learning transformations on non-uniform, unstructured, and ambiguous data from a few input-output examples, for instance, normalizing data with complex date formats into a unified format.

Raza et al. [34] propose a predictive synthesis technique for splitting a long string into various sub-fields. Data extraction tasks are easy for a human observer to predict the desired extraction by just observing the input data itself. This technique enables splitting without any user specification but from just the input data: namely, splitting from given input-only examples.

FlashRelate [35] allows extracting relational data from semi-structured spreadsheets. Users often create a virtually high-dimensional data through spacial layouts involving headers, whitespace, and relative positioning on a two-dimensional spreadsheet. Although such semi-structured data suits for human understanding with the visual representation, whereas, complicates data manipulation. FlashRelate allows users to provide examples of tuples in an output table, and then generate a program that extract more tuples from an input semi-structured spreadsheet. FlashExtract [36] is a framework for data extraction by examples as well. It synthesizes programs that extract tabular data from text files, log files, or webpages. It allows users to give some positive/negative instances as examples, and then extracts all instances into the output table.

ProgFromEx [37] is a PBE system that generate programs for layout transformation that rearranges the layout of semi-structured tables into structured relational form, keeping the textual content of any data element unchanged.

These PBE algorithms described above allow generating programs for various data types and transformations. However, none of these supports tabular transformations that involves both syntactic transformations and layout transformations.

Foofah [8] is a PBE system that firstly tackle the tabular transformations. This system supports the tabular transformations that involve almost transforms proposed in Potter's Wheel [10] detailed in Section 2.2. In this dissertation, we aim to realize an ML-based PBE system that handles such transformations supported in Foofah. We take a look around Foofah in Section 3.2.1.2.

3.2.1.1 FlashFill

FlashFill [9] is a typical example of research that applies PBE to solve data preparation tasks, especially string transformation in Excel cells. Fig. 3.4 illustrates that once the user provides desired examples in (row 2, col. B) and (row 3, col.B), FlashFill learns a program that extracts the two words preceding “@” mark from col. A and converts them to lowercase and concatenates by a white space. Thus, it allows users to transform strings simply by giving examples of what they want, without using complicated Excel macros.

	A	B
1	E-mail	
2	Alice.Davis@someuniversity.ac.jp	alice davis
3	Bob.Brown@someorganization.org	bob brown
4	Carol.Smith@somecompany.com	carol smith
5	Grace.Smith@someuniversity.ac.jp	grace smith
6	Frank.Hill@somecompany.com	frank hill
7	Marolly.Scott@someuniversity.ac.jp	marolly scott

FIGURE 3.4: FlashFill: an extension of Microsoft Excel

Designing a search algorithm that finds programs consistent with the input-output examples from a huge search space is the most important challenge for realizing the PBE system. One concept for restricting search space is defining a domain specific language (DSL) for the focusing domain. FlashFill DSL focuses on the string transformation. The FlashFill DSL represents programs that transform an n-ary tuple of strings into another string. These programs involve computing substrings of the strings in the input tuple, and then concatenating them appropriately. There is also support for loops and conditional computation for rich string transformations.

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

3.2.1.2 Foofah

Foofah [8] is a PBE study for dealing with *tabular transformation* that includes syntactic and layout transformations among the data preparation tasks. Foofah targets the same problem that we focus on in our study (see Section 2.1).

Foofah uses the A* algorithm, a graph search algorithm, to make the search process more efficient. Figure 3.5 illustrates the A* algorithm processing that solves the problem

described in Section 2.1. A* algorithm searches the shortest path that reaches from the initial state (T_i) to the final state (T_o), and then the path represents the program (P_1, P_2, \dots, P_n) where each P_i is defined as operators in Table 2.3. To find the path, A* algorithm computes the cost $f(n) = g(n) + h(n)$ where $g(n)$ is the cost to reach state n from the initial state and $h(n)$ is the heuristic function that approximates cost of the cheapest path from state n to the goal state. A* algorithm finds the shortest path by visiting the state that has the cheapest $f(n)$ at each search step.

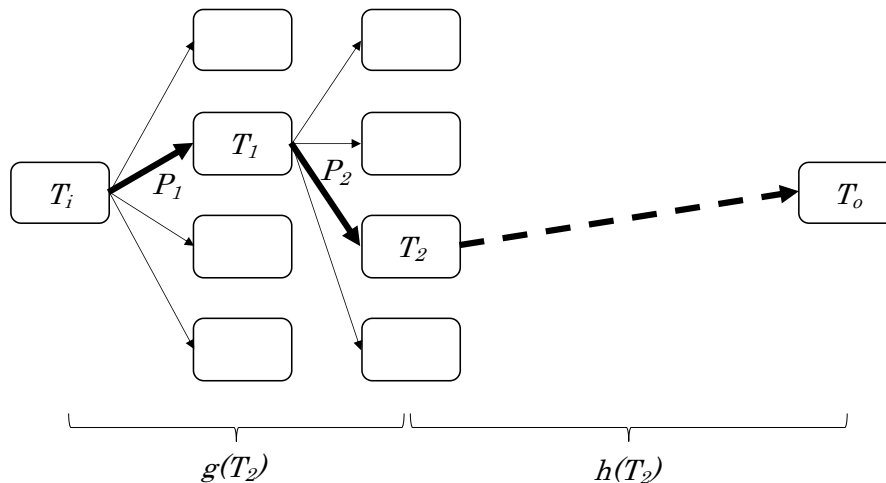


FIGURE 3.5: A* algorithm searches the path that is cheapest in the cost $f(n) = g(n) + h(n)$. $g(T_2)$ in this figure is the cost from T_i to T_2 which is the addition of the cost between T_i and T_1 and the cost between T_1 and T_2 . $h(T_2)$ is the approximated cost between T_2 and T_o which is computed by TED Batch.

The correct estimation of the heuristic function $h(n)$ is essential for the A* search performance. In order to estimate the heuristic functions, Foofah introduces the Table Edit Distance (TED) which estimates a distance between tables. TED is defined as the table dissimilarity as below.

$$TED(T_1, T_2) = \min_{(p_1, \dots, p_k) \in P(T_1, T_2)} \sum_{i=1}^k cost(p_i) \quad (3.1)$$

TED is the minimum total cost of *table edit operations* needed to transform T_1 to T_2 where $P(T_1, T_2)$ denotes the set of edit paths transforming T_1 to T_2 and $cost(p_i)$ is the cost of each table edit operation p_i which is defined in Table 3.1. Computing TED in real time is not practical, therefore Foofah uses an efficient greedy algorithm to approximate TED (see the literature [8] for the algorithm in detail).

Furthermore, *Table Edit Distance Batch (TED Batch)* is designed to modify TED to fit into their problem. The number of operations is overestimated in definition 3.1 where the unit of operations is cell level. In order to fit the TED into their problem, they

TABLE 3.1: Table edit operators. This list is adapted from Foofah [8]

Operator	Description
Add	Add a cell to table
Delete	Remove a cell from table
Move	Move a cell from location (x_1, y_1) to (x_2, y_2)
Transform	Syntactically transform a cell into a new cell

developed *TED Batch* that approximates the number of operations defined in Table 2.3 where the unit of operations is column or row level by grouping operations into batches and compacting the cost of each batch.

Foofah realizes a search-based PBE by leveraging the A* search algorithm, the TED Batch as the heuristic function to estimate the cost between tables. In addition, pruning rules are developed with their heuristic insight in order to reduce the program space and improve the runtime of the search.

Although Foofah is able to solve the same problem we are interested in, *tabular transformation*, it is necessary to be developed on the basis of developer’s heuristic insight and engineering efforts to enable the search algorithm to work efficiently.

Thus, our study aims to establish an ML-based PBE for tabular transformation which is based on a stochastic approach where the PBE system is able to be improved in a data-driven manner.

3.2.2 ML-based PBE Studies

The neural methods are expected to extend a model by simply increasing the number of training data without the need for laborious technical development. Thus, realizing PBE using neural methods has attracted much attention in recent years [18, 38–40].

RobustFill proposed by Devlin et al. [18] is a pioneering PBE system using neural networks. We introduce this work in the next section Section 3.2.2.1.

Kalyan et al. [38] propose a hybrid synthesis technique that combines both symbolic logic techniques and neural models for string transformation with DSLs similar to that of FlashFill. The model realizes a real-item program synthesis, leveraging powerful RNN encoders and a deductive symbolic search framework.

Singh [39] presents a PBE system called BlinkFill, whose DSL also consists of syntactic string transformations similar to that of FlashFill. BlinkFill uses semi-supervised learning technique to reduce ambiguity by using a data structure that represents a large set

of logical patterns that are shared across the input data and efficiently learn substring expressions.

Bavishi et al. [40] propose a neural program candidate generator that produces functions and arguments of APIs for Python pandas libraries for an input-output example provided in data frame form.

In these studies described above, neural encoders encode input-output examples into a representation, and neural decoders or search algorithms generate programs or APIs from the representation given from the encoder. These neural system can enhance their expressiveness by training new training data. However, no system generates programs for tabular transformations defined in Chapter 2.

3.2.2.1 RobustFill

RobustFill [18] is a fundamental study for PBE using neural methods and is used as a basis of many other studies on ML-based PBE. It achieves a string transformation PBE by leveraging an LSTM model.

RobustFill generates a string transformation which is composed of operations for string manipulations from user-given examples. The string transformation of RobustFill is constructed from a DSL which is defined based on substring extractions, string conversions and constant strings. The DSL of RobustFill is similar to those of FlashFill [9].

A neural model is trained fully supervised on a large corpus of synthetic I/O Example + Program pairs in training phase. It takes n pairs of input-output strings $(I_1, O_1), \dots, (I_n, O_n)$ and is trained so that it generates the corresponding program P as output.



FIGURE 3.6: Neural sub-network architecture for single example. A dotted line from x to y means that x attends to y . This figure is based on RobustFill [18]

The architecture of the neural model is illustrated in Figure 3.6 and Figure 3.7. Figure 3.6 shows single-example sub-network models which only take a single example (I, O) as input and produce a program P as output. Each single-example sub-network is constructed by long short term memory (LSTM) networks which are illustrated as rectangles and characterized by variations of attention mechanisms [41] and bi-directional LSTM

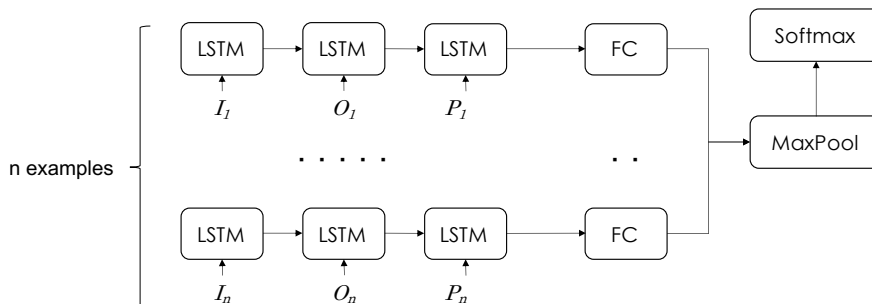


FIGURE 3.7: Neural network architecture for multi-example pooling. This figure is based on RobustFill [18]

networks. In order to solve the problem with multiple pairs of I/O examples, a multiple-example network model which is composed of multiple single-example sub-networks is proposed. Figure 3.7 shows the multi-example network model. The multi-example network model takes multiple pairs of I/O examples and generates a program corresponding to the pairs of I/O examples. It pools hidden states of P_1, \dots, P_n at max pool layer and feed into a single output softmax layer and eventually outputs probabilities of a program corresponding to the pairs of I/O examples.

Robustfill is an ML-based PBE but not for tabular transformation. To the best of our knowledge, no neural PBE for tabular transformation has been proposed and implemented.

3.3 Encoder-Decoder Neural Networks

As in Robustfill [18], some studies on ML-based PBE use encoder-decoder models [19, 38, 42]. The encoder-decoder model is typically applied to machine translation problems [41, 43, 44], For this reason, ML-based PBE studies have applied the encoder-decoder model to the PBE problem, regarding the PBE problem as a translation problem from input-output examples to a corresponding program.

Parisotto et al. [19] propose an encoder-decoder model that uses a cross correlation I/O network as the encoder and a recursive-reverse-recursive neural network (R3NN) as the decoder. The cross correlation I/O network produces a representation of I/O examples, and given the representation, R3NN synthesizes a program by incrementally expanding partial programs.

Kalyan et al. [38] propose an ML-based model using an LSTM-based encoder that encodes I/O examples and production rules, and predicts the score of a candidate production for the given examples. Consequently, their proposed search method generates programs based on the scores that the encoder outputs.

Although we also formulate our ML-based PBE model as an encoder-decoder model, we use more sophisticated model, Transformer, in our encoder-decoder model. In this Section, we overview the two kinds of models used in the encoder-decoder model. The first is the conventional LSTM neural network and the second is the Transformer.

3.3.1 Long Short Term Memory (LSTM)

An RNN is a network with loops. Applied to a time-series sequence or a word sequence, this loop allows information in previous loops persist in the network and enables RNN to deal with dependencies between different time steps (iterations of the loop). RNNs' limitation is difficulty of learning with a too long gap between two time steps.

LSTM is one type of recurrent neural networks (RNN) and capable of handling such a long-term dependencies. The computational graph of a LSTM network is illustrated in Figure 3.8. A LSTM network takes an input sequence (x_0, \dots, x_t, \dots) and output a hidden sequence (h_0, \dots, h_t, \dots) . x_t and h_t is computed by the values on the previous time step x_{t-1} and h_{t-1} .

$$\begin{aligned}
 f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
 i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
 \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
 C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t
 \end{aligned}
 \tag{3.2}$$

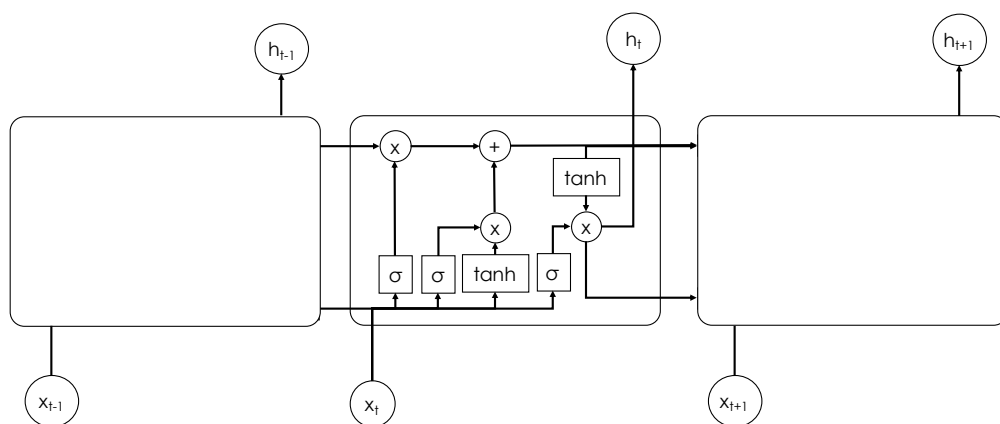


FIGURE 3.8: LSTM network. An unrolled representation of the repeated LSTM module from time step $t - 1$ to $t + 1$

Where $\sigma(\cdot)$ is a sigmoid function and W_f , W_i and W_C are the weights and b_f , b_i and b_C are the biases of the LSTM network. The sigmoid layer outputs numbers between zero and one. A value of zero means “forget the value” and a value of one means “retain

the value”. This layer is called “forget gate layer” and is essential for capability of memorizing a long-term temporal dependency in LSTM.

A sequence-to-sequence model can be constructed with an LSTM encoder and an LSTM decoder [45]. A sequence-to-sequence model is illustrated in Figure 3.9. The encoder inputs “A B C” and outputs the last hidden state and the decoder takes the last hidden state from the encoder and the end-of-sequence token and outputs “W X Y Z” decoding using a simple left-to-right beam-search. Thus, a sequence-to-sequence model realizes a machine translation system using LSTM encoder and decoder.

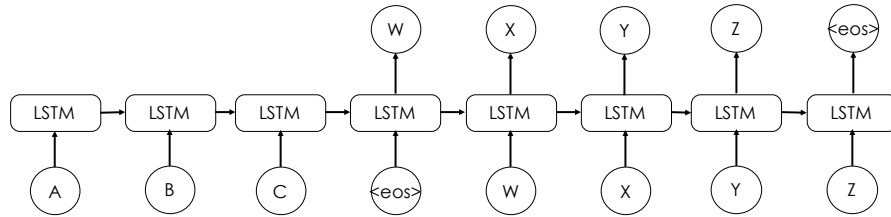


FIGURE 3.9: A illustration for sequence to sequence model. This is an example model that reads “A B C ” and produces “W X Y Z” as the output sentence. This figure is based on Sutskever et al. [45]

3.3.2 Transformer

The RNN like LSTM has been studied and applied as a basis of sequence-to-sequence models and has achieved a high performance in sequence modeling and transduction problems such as language modeling and machine translation.

However, recurrent models need a sequential computation where, aligning the positions of an input sequence to time steps, they generate a sequence of hidden states h_t at time t using h_{t-1} at the previous time $t - 1$ recursively. This sequential computation prevents parallelization, which becomes critical at longer sequence lengths.

The Transformer model achieves the simple model based on attention mechanism, dispensing with the recurrent nature and establishes the state-of-the-art performance in translation quality [46].

The model architecture of the Transformer is shown in Figure 3.10. It is composed of an encoder (left half of Figure 3.10) and a decoder (right half of Figure 3.10).

The encoder is composed of a stack of $N = 6$ layers. Each layer has a multi-head self-attention sub-layer, and a position-wise fully connected feed-forward sub-layer network. Furthermore, it has a residual connection around each sub-layer followed by layer normalization.

The decoder is also composed of a stack of $N = 6$ layers. Each layer has a sub-layer which computes multi-head attention over the output of the encoder stack in addition to the sub-layers similar to that of encoder. The self-attention sub-layer in the decoder is masked to prevent predictions for position i from unseen outputs at positions larger than i .

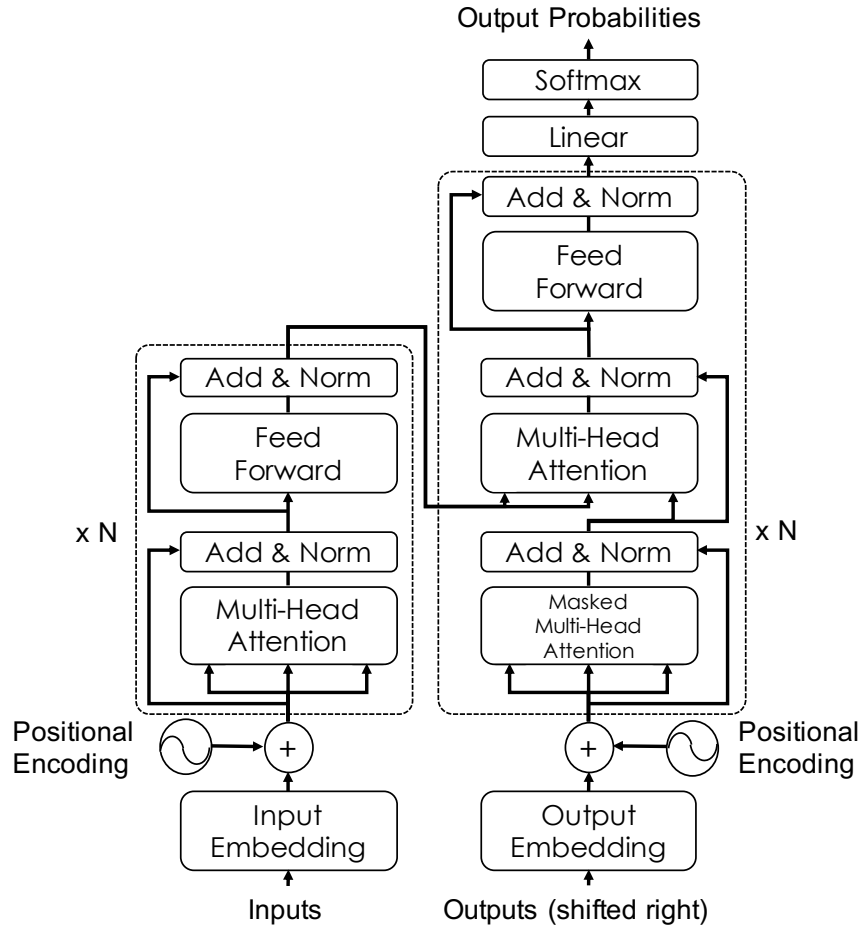


FIGURE 3.10: Transformer model architecture. This figure is based on Vaswani et al. [46]

The sub-layer that has the most essential role in the Transformer model is the attention sub-layer. The Transformer employs the “scaled dot-product attention” as the attention mechanism. $\sqrt{d_k}$, and apply a softmax function to obtain the weights on the values. The scaled dot-product attention is computed as follows.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.3}$$

Where Q is a set of query vectors packed together into a matrix, and keys and values are also packed into matrix K and V , and the dimension of queries and keys are d_k and values d_v .

The Transformer performs a multi-head attention instead of single scaled dot-product attention as seen in Figure 3.11. The multi-head attentions linearly project the queries, keys and values h times with different, learned linear projections to d_k , d_k and d_v dimensions, respectively. And then, these are concatenated and then projected again.

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O \tag{3.4}$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$

where the projections are learned parameters of $W_i^Q \in \mathbb{R}_i^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}_i^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}_i^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}_i^{d_v \times hd_{model}}$. The value of $h = 8$ in the Transformer original work. The individual h attention heads perform differently with each different projection. The Transformer uses multi-head attentions in “encoder-decoder attention layers”, “self attention layers in the encoder” and “self attention layers in the decoder”.

Although the Transformer is entirely based on only attentions and therefore has a simple architecture, the powerful performance and simple structure of Transformer has attracted many researchers with variety of applications rather than neural translation machines. The representative applications of Transformer model are the pre-trained large-scale language model such like Bidirectional Encoder Representations from Transformers(BERT) [47], Generative Pre-trained Transformer (GPT)-1/2/3 [48–50] which employ the Transformer encoder [47] or decoder [48–50] as a basic component of the architecture and achieve remarkable results in various tasks.

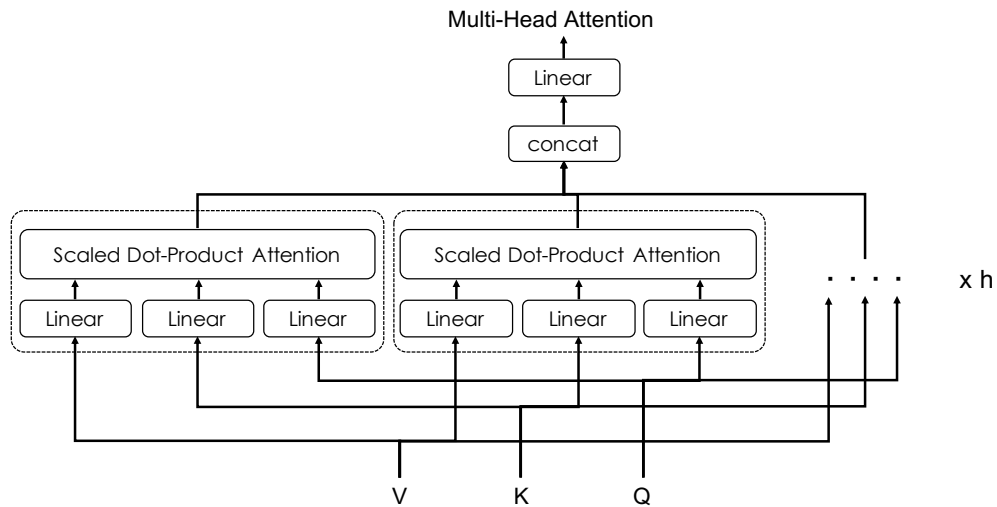


FIGURE 3.11: Multi head attention

3.3.3 Transformer-based model on Larger Data

As described in later chapters, we achieved better performance than the conventional methods. However, performance of training and inferring larger data remains a challenge. Simply increasing the table size for the training data does not improve performance, as described in 4.6.2.2. We are planning to improve our model to accept larger table sizes in the training data by revising the model network referring to the studies like Longformer [51] and related other works [52, 53]. Transformer-based models in these studies have a local windowed attention mechanism that enables the model to perform the attention processing in lower computational and memory costs and consequently scale to larger input documents. The attention mechanism is realized by sparsifying the self attention matrix according to a periodical attention pattern.

3.3.4 Synthesizing Realistic Training Data

Preparing a training dataset which has a distribution similar to the real data is an important issue for any neural network model. There are studies to synthesize a training dataset for neural networks for program synthesis [54–57].

Shin et al. [54] propose a methodology for controlling and evaluating the bias of synthetic data distributions by ensuring greater uniformity over the *salient* random variables. Their method shows good performance in a deep neural model for Karel domain that have more complex DSLs than that for string transformations.

Clymo et al. [55] propose a constraint based method using an SMT solver to synthesize inputs which cover a diverse set of behaviors for a given program. Their method synthesizes datasets for DeepCoder [58] which is a program synthesis system for manipulating list of integers.

Suh et al. [56] propose an adversarial method to generate a training set applicable to PCCoder [59]. Their adversarial approach builds a training set iteratively. In each iteration, it finds data distributions on which the PBE model performs poorly and then adds data drawn from those distributions to the training set, thus generating test sets that are less likely to overestimate the performance of a PBE model.

Although some works study on synthesizing training datasets for ML-based PBE as described above, no work studies on synthesizing tabular data and corresponding tabular transformation. Thus, we first formulate a method synthesizing tabular data in uniform distribution as described in Section 4.6.1.2.

3.3.5 Training Objectives

The encoder-decoder models, which is typically used in neural machine translation [43], are trained to generate sentences that are similar to the reference (target) sentences. The training is performed based on maximum likelihood optimization, specifically minimizing cross-entropy loss between the reference and the generated sentence. ML-based PBE approaches using an encoder-decoder model [18, 19] are also trained in the similar objective optimization.

However, the true objectives for these models are rather different from maximum likelihood. Indeed, many translation models pursue better BLEU scores [43, 60, 61]. Similarly, our PBE model aims to generate programs valid for input examples or consistent with input-output examples.

In order to directly optimize the model to the true objective, some studies [43, 60–63] has been considered to refine their models using policy gradient reinforcement learning as in REINFORCE algorithms proposed in the work by Williams [64]. Ranzato et al. [60], Wu et al. [43], Wiseman et al. [65] applied this approach to translation, Guu et al. [62] to “program to language”, Zaremba & Sutskever [61] to Neural Turing Machine (NTM), and Bunel et al. [63] to program synthesis. Studying using this approach will be proposed in future work.

3.4 Two-dimensional Tabular Data Embeddings

Researchers have paid their attentions on handling structured tabular data in neural network models in table understanding and document understanding studies such as cell-classification [66, 67], table classification [68], table-to-text [69], and document classification [70]. The neural network models used in these studies are the conventional ones like RNN [66, 67], LSTM and CNN [68], RNNs and CNN [69], and hierarchical attention mechanisms [70].

Whereas, some studies [71, 72] employing embedding layers for two-dimensional data to the Transformer model have emerged these days. The embedding layers enable the Transformer model, which originally learns one-dimensional positions on a sequential data, to learn the positions of two dimensional data. In the following section, we introduce the literatures embedding two-dimensional data into the Transformer-based model.

3.4.1 TAPAS

TAPAS (for **T**able **P**arser) [71] is a question answering model that reasons out an answer drawn from a table with respect to a question that users provide. TAPAS predicts a minimal program by selecting a subset of the table cells and a possible aggregation operation to be executed on top of them (See Figure 3.12). It can predict the aggregation operations from natural language without the need to specify them in some formalism.

Table			
Rank	Name	No. of reigns	Combined days
1	Lou Thesz	3	3749
2	Ric Flair	8	3103
3	Harley Race	7	1799
4	Dory Funk Jr.	1	1563
5	Dan Severn	2	1559
6	Gene Kiniski	1	1131

Q1: Which wrestler had the most number of reigns?
A1: Ric Flair

Q2: Average time as champions for top 2 wrestlers?
A2: AVG(3749, 3103) = 3426

Q3: How many world champions are there with only one reign?
A3: 7

Q4: What is the number of reigns for Harley Race?
A4: 7

Q5.1: Which of the following wrestlers were ranked in the bottom 3?
A5.1: { Dory Funk Jr, Dan Severn, Gene Kiniski }

Q5.2: Out of these, who had more than one reign?
A5.2: Dan Severn

FIGURE 3.12: A table (left) corresponding example questions (right). TAPAS predicts a minimal program that create an answer for a question using the contents of the **Table**. This figure is based on TAPAS [71]

TAPAS is implemented based on BERT architecture [47] with additional positional embeddings that capture tabular structure and with two classification layers for selecting cells and predicting a corresponding aggregation operator. TAPAS firstly flattens a table into a sequence of words by splitting the table into word tokens and concatenating it with the question tokens in front of the tokens of the table. And then, it appends several types of positional embeddings into the sequence of words. The types of positional embeddings are *Position ID*, *Segment ID*, *Column ID*, *Row ID*, and *Rank ID*. Introducing these positional embeddings in the model based on the BERT architecture, TAPAS succeeded to capture the two-dimensional table positions and achieved higher accuracy than other existing works.

3.4.2 Vision Transformer

The Vision Transformer [72] is a Transformer-based model targeting at solving computer vision tasks. Whereas Convolutional architectures has remained dominant in computer vision research in recent years, some studies applying the attention-based Transformer architectures to such tasks like image recognition has emerged. The Vision Transformer is the representative work among these studies.

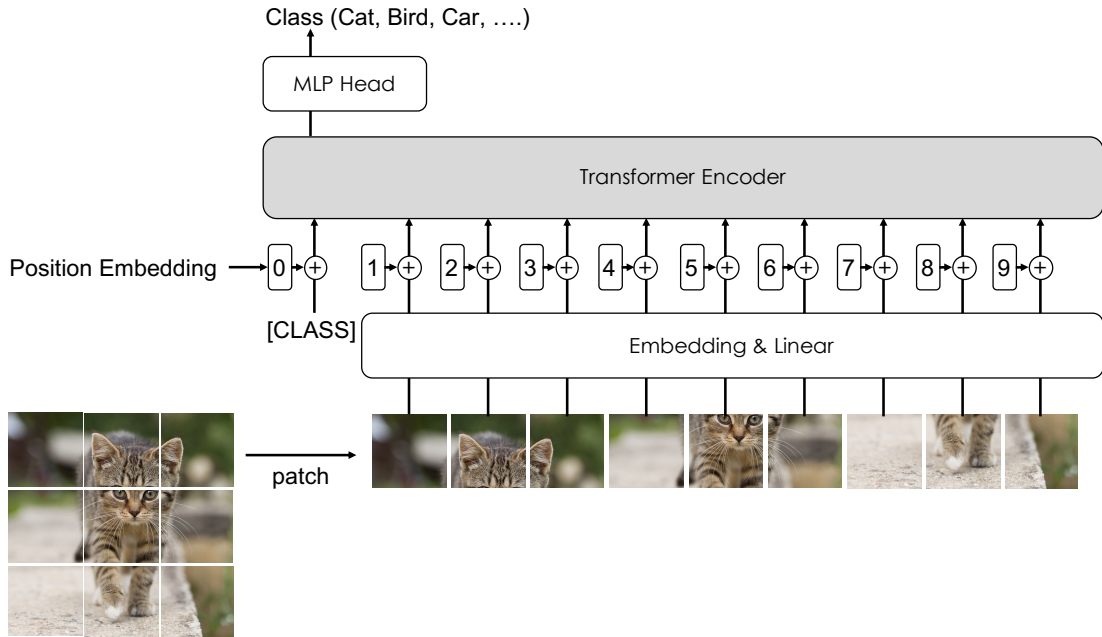


FIGURE 3.13: Vision Transformer. This figure is based on Dosovitskiy et al. [72]

An overview of the model is illustrated in Figure 3.13. To handle 2D images in the Transformer, image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ is reshaped into a sequence of flattened 2D patches $\mathbf{x}_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$, where (H, W) is the resolution of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and $N = HW/P^2$ is the resulting number of patches which is the input sequence length. The flattened patches ($P^2 \cdot C$ dimension) are mapped to the constant latent vector size of the model (D dimension) through a trainable linear projection.

Positional embeddings are added to the patch embeddings to retain positional information. One-dimensional position embeddings are used as the positional embeddings. Although 2D-aware positional embeddings are also used in this work, they do not show a significant performance gain in these settings. Two-dimensional positional embeddings are composed of X -embedding and Y -embedding, which are learned and have size $D/2$. Two embeddings are concatenated to get the final positional embedding for the patch.

Inspired from these studies like TAPAS (described in Section 3.4.1) and Vision Transformer (described in Section 3.4.2), we have designed our tabular positional encoding that embeds a pair of input-output tabular data and handle them in the Transformer-based model. We propose this tabular positional encoding in Chapter 4.

3.5 Beam Search

The beam search techniques have been studied for many years, particularly as a search method for generating from sequence-to-sequence model [45] [65] [73] and for natural-language-processing (NLP) applications [74] [75]. Our model, which is based on the sequence-to-sequence model, also generates a program by search algorithms like beam search. However, the beam search strategy of NLP may be different from that of our program generation. Thus, our beam search requires new method for optimizing the metric where generating more programs consistent with user-given examples is better.

Some studies aim to find metric-driven search technique. Yang et al. [76] propose a re-scoring method for candidate scores based on the candidate size, focusing on solving the sequence length ratio problem which they call as “beam search curse”. They propose several methods to address this problem and show their hyperparameter-free method achieves a performance improvement in the BLEU metric. Leblond et al. [77] have investigated many decoding algorithms and found which algorithm is best heavily depends on the characteristics of the goal metric. Additionally, they introduce a Monte-Carlo Tree Search based method and show its competitiveness. These studies show the importance of selecting search method depending on the goal metric, thus inspiring us to develop variants of beam search methods for program generation described in Chapter 5.

Other Studies aim to make the beam search more efficient, and resulting in more accurate sequence generation as below.

Freitag et al. [73] introduce several variants of beam search for the machine translation task. They propose a flexible beam search strategy whose candidate size varies at each time step depending on the candidate scores. The beam search prunes the search graph, thus, speeds up the decoding process without losing translation quality.

Diverse beam decoding [78] modifies the basic beam search by penalizing scores of candidates that are siblings, thus exploring hypotheses from diverse parents. Studies like *iterative beam search* [75] also diversely explore search space by performing multiple beam searches, and thus achieving good performance on generating diverse sequences for such tasks as dialog response generation, collaborative story generation, and image captioning. These studies can speed up the beam search and improves decoding performance.

In this dissertation, we pursue beam search techniques that fit to our problem, program generation, and consequently propose two types of novel techniques described in Chapter 5 and compare them with *iterative beam search* [75]. Refer to the next section for detail of iterative beam search.

3.5.1 Iterative Beam Search

Iterative beam search [75] focuses on generating diverse sequences for tasks such as dialog response generation. This study is an example that adopts beam search results suitable for applications. It searches wider search space by performing beam searches multiple times. Eliminating the explored candidates in previous iterations from search space in successive search iterations, it explores wide search space with less computational cost.

Suppose that beam search maintains a set of beam size K hypotheses \mathcal{H}_t at time step t :

$$\mathcal{H}_t = \{(y_1^1, \dots, y_t^1), \dots, (y_1^K, \dots, y_t^K)\} \quad (3.5)$$

Where (y_1^K, \dots, y_t^K) is the K -th candidate at time step t .

The search space over which beam search has explored can be characterized by the union of all partial hypothesis sets: $\mathcal{S}_0 = \bigcup_{t=1}^T \mathcal{H}_t$, where the subscript 0 means the first iteration of multiple beam searches. Re-running beam search with an increased beam width K results in overlapping the search space significantly with \mathcal{S}_0 . Instead, the interactive beam search keeps the beam size K constant and performs multiple iteration of beam search, ensuring that any previously explored space $\tilde{\mathcal{S}}_{<l} = \bigcup_{l'=0}^{l-1} \mathcal{S}_{l'}$ is not included in a subsequent iteration of beam search.

This can be done by setting the score of each candidate to negative infinity when the candidate is included in $\tilde{\mathcal{S}}_{<l}$. This procedure ensures that a new partial hypothesis set of beam search in the l -th iteration minimally overlaps with the search space explored in the previous iterations of beam search.

Thus, the iterative beam search produces more diverse set of candidates than traditional beam search by exploring larger hypothesis spaces with minimum overlaps of spaces in multiple iterations of beam search.

Transformer-based Neural Model for Tabular Data

4.1 Transformer-based Encoder-Decoder Model

Since tabular transformations need larger structured data and complex transforms on its data, the neural network model for tabular transformations must have higher expressive power than that for string transformations. Transformer is a state-of-the-art ML model for NLP tasks proposed by Vaswani et al.[46]. Its architecture is based on attention mechanisms and has the ability of training their large scale parameters by high-level parallelization and short training times, thus achieving higher performance in even complicated tasks than conventional recurrent neural network such as LSTM. Hence, our Transformer-based PBE model for tabular transformations achieves better performance than LSTM-based one. This model is realized by dispensing with recurrent and convolutional processing entirely.

Fig. 4.1 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 Sections 4.2 and 4.3, respectively.

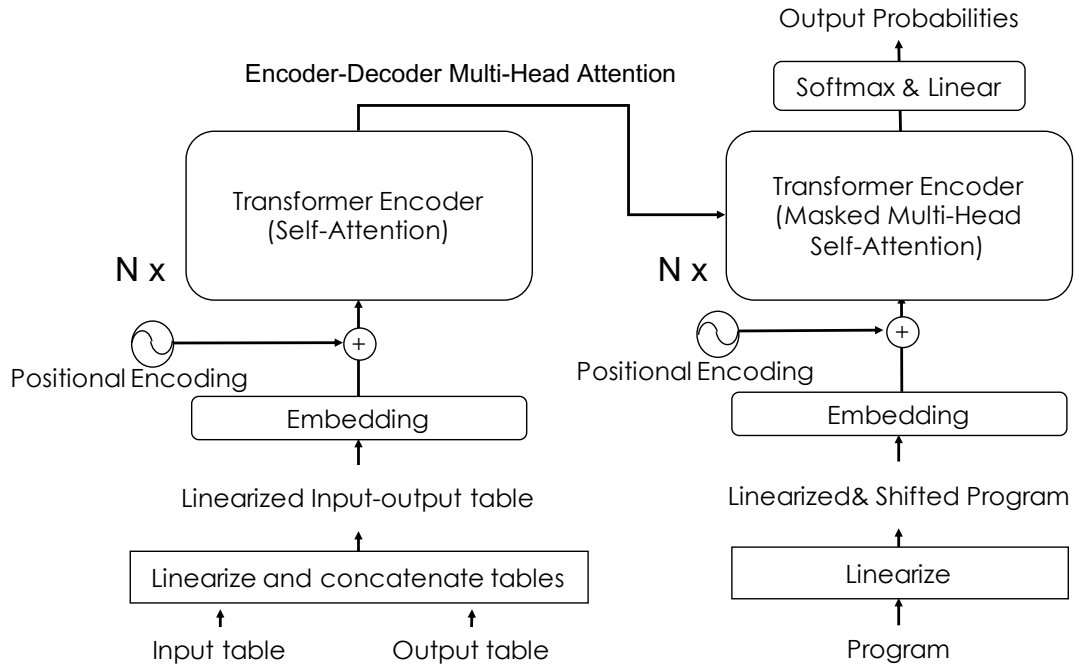


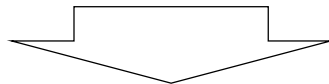
FIGURE 4.1: The Transformer-based model

4.2 Tabular Data Linearization

Tabular data is linearized into a serialized structure in order to feed the tabular data into the sequential encoder-decoder model. Fig. 4.2 shows the preliminary model that we developed in our study as this linearization 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.

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

Linearizes into a character-level sequence
inserting the separators between row and column boundaries



'N' 'a' 'm' 'e' <eoc> 'N' 'u' 'm' 'b' 'e' 'r' 's' <eol> 'A' 'l' 'l' 'c' 'e' <eoc> 'T' 'e' 'l' ':' ' ' '(' '0' '3' ' ...

FIGURE 4.2: Linearization processing for tabular data

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

4.2.1 Scanning Direction

There are two possible options for scanning directions in linearizing the tables, namely horizontal and vertical (see Fig. 4.3). Because an actual table generally stores information about the same record in the row direction and information about the same attribute in the column direction, we can expect to observe some differences in model performance depending on the scanning direction.

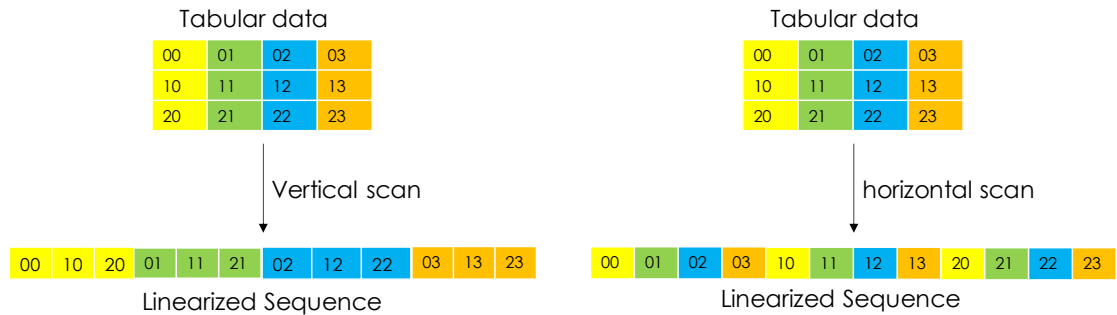


FIGURE 4.3: Two types of scanning directions: vertical scanning on the left and horizontal scanning on the right

4.3 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 2.3 according to the tokenization rules. We have the following two options for tokenization rules.

4.3.1 Separated Token Rule

In separated token rule, a combination of operator and non-numeric parameter corresponds to one token, and a numeric parameter to one token individually. 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 used in our preliminary experiments in Section 4.6.2, 4.6.3.

4.3.2 Combined Token Rule

In combined token rule, a combination of operator, non-numeric parameter and numeric parameter corresponds to one token. 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. Combined token rule is used in our experiment of Section 4.6.4 and Chapter 5.

4.4 Model Description

We explain the model architecture of the Transformer-based encoder-decoder model in this section. Let a token sequence of input-output tables be \mathbf{t} and a token sequence of program be \mathbf{o} . The Transformer embedding layers embed \mathbf{t} and \mathbf{o} and append the positional encoding PE .

$$\begin{aligned}\mathbf{x}_0 &= EncoderEmbedding(\mathbf{t}) + PE \\ \mathbf{y}_0 &= DecoderEmbedding(\mathbf{o}) + PE\end{aligned}$$

Let $\mathbf{x}_i = (x_0^i, x_1^i, \dots, x_n^i)$ 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 $\mathbf{y}_i = (y_0^i, y_1^i, \dots, y_m^i)$ 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 with the stacked layers as described in the following expressions.

$$\begin{aligned}\mathbf{x}_i &= TransformerEncoderLayer_{i^{th}}(\mathbf{x}_{i-1}) \\ \mathbf{y}_i &= TransformerDecoderLayer_{i^{th}}(\mathbf{y}_{i-1}, \mathbf{x}_i)\end{aligned}$$

When all computations in the encoder and decoder sublayers have terminated, the output from the last sublayer of decoder \mathbf{y}_{last} is provided. The linear and softmax layers

at the last part of the Transformer-based model (see Fig. 4.1) compute the program probabilities from \mathbf{y}_{last} .

Thus, the program probabilities are output by feeding a sequence of tokens of input-output tables and a sequence of tokens of a program into the sequential encoder-decoder model. For a set \mathcal{T}_{io} 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 $\mathbf{t} = (t_0, t_1, \dots, t_n)$ where $t_i \in \mathcal{T}_{io}$. For a set \mathcal{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 $\mathbf{o} = (o_0, o_1, \dots, o_m)$ where $o_i \in \mathcal{O}$. Now, program probabilities can be represented as a sequence $\mathbf{p} = (p_0, p_1, \dots, p_m)$, where $p_i \in \mathbb{R}^{|\mathcal{O}|}$. The variable $p_i = P(z | o_{<i}, \mathbf{t})$ represents the conditional probability distribution of the discrete random variable $z \in \mathcal{O}$ at position i when $(o_0, o_1, \dots, o_{i-1})$ and \mathbf{t} are given.

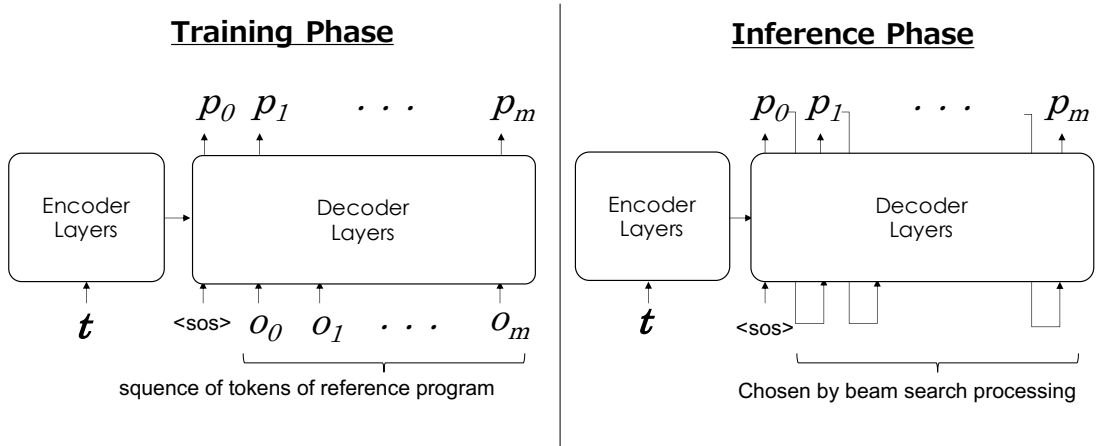


FIGURE 4.4: Decoding process in the encoder-decoder model

The left part of Fig. 4.4 shows the decoding process during the training phase. The program probabilities \mathbf{p} are output after inputting a sequence of tokens for input-output tables \mathbf{t} into the encoder layer and a sequence of tokens for the program (shifted by 1 token ($\langle \text{sos} \rangle$)) into the decoder layer. We can compute the cross-entropy loss from \mathbf{p} and (the one-hot vector of) \mathbf{o} , and then proceed to training with the aim of reducing the loss value.

The right part of Fig. 4.4 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 $\langle \text{sos} \rangle$ at the decoder layer and the token sequence \mathbf{t} at the encoder layer, the decoder outputs the conditional probability $p_0 = P(z | \langle \text{sos} \rangle, \mathbf{t})$ at position $t = 0$. Using beam search with the beam size of 1 gives the token $z_0 \in \mathcal{O}$ having the highest probability to give to the decoder at the next position. In the next step, given

the token sequence \mathbf{t} at the encoder layer and tokens $\langle \text{sos} \rangle$ and z_0 at the decoder layer, the decoder outputs the conditional probability $p_1 = P(z | z_0, \langle \text{sos} \rangle, \mathbf{t})$ at position $t = 1$. Again, beam search determines the token $z_1 \in \mathcal{O}$ having the highest score given by the equation 4.1 with respect to the tokens $\langle \text{sos} \rangle$, z_0 and z_1 to give to the decoder at the next position. In this way, the token z_t is found having the highest score at each position t .

$$\begin{aligned} \text{score} &= \log\left(\prod_{i=0}^t P(z | z_{<i}, \mathbf{t})\right) \\ &= \sum_{i=0}^t \log(P(z | z_{<i}, \mathbf{t})). \end{aligned} \tag{4.1}$$

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

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 $\mathcal{H}_t^{\text{cand}}$ 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 $\mathcal{H}_{t-1}^{\text{cand}}$ is concatenated by each token $z \in \mathcal{O}$ (line 8 in Algorithm 1), and the scores for concatenated hypotheses h are given by equation 4.1 (line 9 in Algorithm 1). The hypotheses are ranked by the score and the top- K hypotheses are kept in the candidates $\mathcal{H}_t^{\text{cand}}$ (line 10-22 in Algorithm 1). When $\langle \text{eos} \rangle$ token is selected as a candidate, the K value is reduced by one and the candidate is placed in the “final-candidate” list $\mathcal{H}_{\text{final}}$ (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 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.

Algorithm 1 basic beam search

Require: K ▷ Beam size
Ensure: \mathcal{H}_{final} ▷ Hypotheses obtained finally

- 1: $\mathcal{H}_0^{cand} \leftarrow \{h\}$ where $h = (\langle \text{sos} \rangle)$
- 2: $\mathcal{H}_{final} \leftarrow \emptyset$
- 3: **for** $t \in \mathbb{N}$ in ascending order **do**
- 4: $\mathcal{H}_t \leftarrow \emptyset$
- 5: $\mathcal{S}_t \leftarrow \emptyset$
- 6: **for** $h \in \mathcal{H}_{t-1}^{cand}$ **do**
- 7: **for** $z \in \mathcal{O}$ **do**
- 8: $h \leftarrow h||z$ ▷ $||$ concatenates tokens
- 9: $s \leftarrow \text{score}(h)$ ▷ score given by equation 4.1
- 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: $\mathcal{H}_t^{cand} \leftarrow \emptyset$
- 15: **for** $h \in \mathcal{H}_t$ in order **do**
- 16: **if** The last token of h is $\langle \text{eos} \rangle$ **then**
- 17: $\mathcal{H}_{final} \leftarrow \mathcal{H}_{final} \cup \{h\}$
- 18: $K \leftarrow K - 1$
- 19: **if** $K = 0$ **then**
- 20: **return** \mathcal{H}_{final}
- 21: **if** $|\mathcal{H}_t^{cand}| < K$ **then**
- 22: $\mathcal{H}_t^{cand} \leftarrow \mathcal{H}_t^{cand} \cup \{h\}$

4.5 Transformer-based Model with Tabular Positional Encodings

4.5.1 Positional Encoding

Figure 4.5 illustrates our Transformer-based model. The attention-based layers of the Transformer-based model are invariant with respect to the order of input sequences. Therefore, the network modules that encode positions of sequences are necessary in order to learn the positions of sequences on the Transformer-based model. The network modules are called as the “positional encoding” described in leftmost side of Figure 4.5. The positional encoding uses a sinusoidal function as expressed by the following expressions [46].

$$\begin{aligned}
 PE_{(pos,2i)} &= \sin(pos/10000^{2iD/d_{model}}) \\
 PE_{(pos,2i+1)} &= \cos(pos/10000^{2iD/d_{model}})
 \end{aligned} \tag{4.2}$$

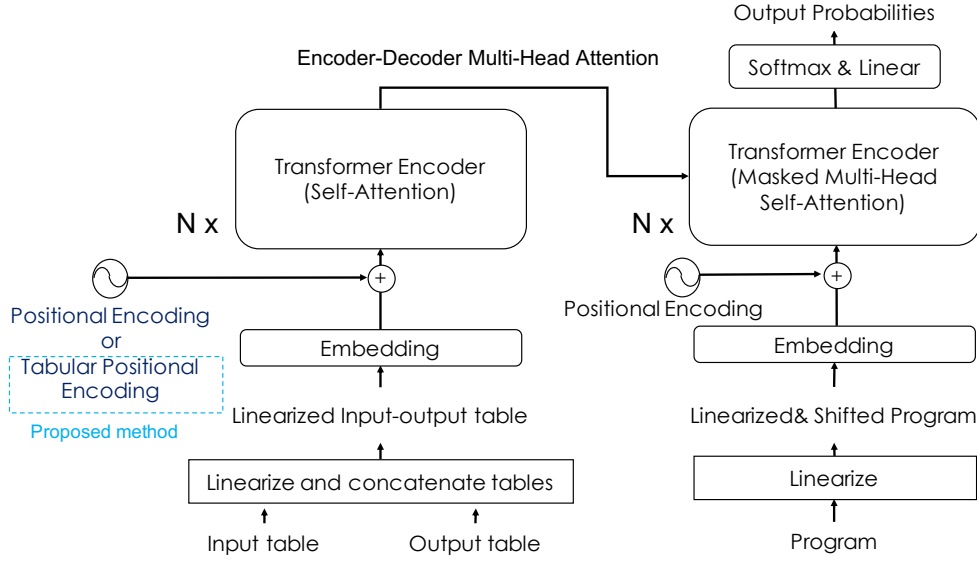


FIGURE 4.5: The Transformer-based model

Here, $2i$, $2i - 1$ are the dimension indexes of the model, pos is the position, d_{model} is the dimension size of the model and $D = 1$. Positions are embedded by adding this PEs into the linearized input-output table at each position pos .

Our Transformer-based model uses positional encoding to embed the one-dimensional positions in the linearized tabular data. Once a token sequence of the I/O tables are fed into the model, they are embedded in the embedding layer. After that, the positional encoding is added to the embedded sequence in order to embed the position of each embedded token.

4.5.2 Tabular Positional Encoding

We propose the Tabular Positional Encoding (TPE) that improve the positional encoding of Transformer-based model described in the previous sections. Replacing the positional encoding to proposed tabular positional encoding (as seen in Figure 4.5), we introduce the encoding method which embeds the positions of two-dimensional structured tabular data and thereby deals with a pair of I/O tables well in the Transformer-based model.

We propose two types of tabular positional encoding which represents the positions in two-dimensional tabular data.

- Additive Tabular Positional Encoding (ATPE)
- Concatenative Tabular Positional Encoding (CTPE)

The positional encoding is replaced by either ATPE or CTPE. It is added to the embedded linearized input-output table at the former step of Transformer encoder layers. Both types of tabular positional encodings are composed of four kinds of indexes in tabular data as in Figure 4.6.

- **Row Index** represents the indexes of rows in a table
- **Column Index** represents the indexes of columns in a table
- **Separator Index** denotes input table or output table
- **Local Position Index** represents the indexes of strings inside a table cell

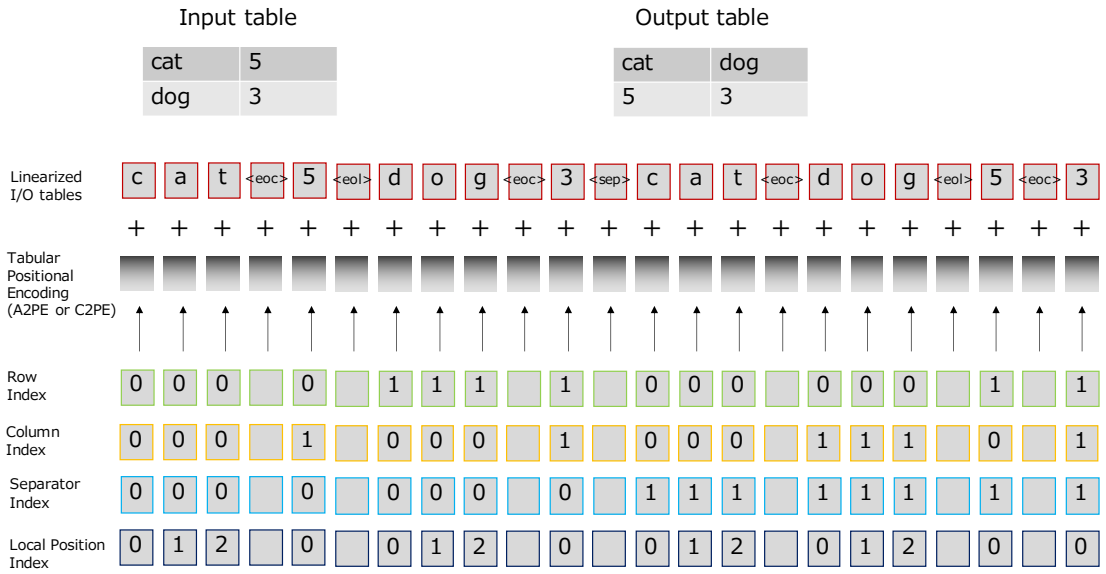


FIGURE 4.6: Tabular positional encodings

4.5.2.1 Additive Tabular Positional Encoding (ATPE)

We designed the additive tabular positional encoding inspired from the work TAPAS [71]. The vector of ATPE PE_{pos}^A is computed by addition of all positional encodings. Let be the vector of positional encoding of row, column, separator, local position as PE_{pos}^r , PE_{pos}^c , PE_{pos}^s , and PE_{pos}^l respectively, then

$$PE_{pos}^A = PE_{row}^r + PE_{col}^c + PE_{sep}^s + PE_{local}^l \quad (4.3)$$

where each row , col , sep , and $local$ represents row index, column index, index denoting input or output table, and local position of a table cell, respectively at position pos .

4.5.2.2 Concatenative Tabular Positional Encoding (CTPE)

We also designed the concatenative tabular positional encoding (CTPE) inspired from the work by Wang et al. [79]. The vector of CTPE \mathbf{PE}_{pos}^C is computed by concatenating all positional encodings, namely

$$\mathbf{PE}_{pos}^C = [\mathbf{PE}_{row}^r; \mathbf{PE}_{col}^c; \mathbf{PE}_{sep}^s; \mathbf{PE}_{local}^l] \quad (4.4)$$

4.5.3 Positional Encoding Implementations

There are two types of implementations for positional encodings, namely fixed encoding and learned encoding. We evaluated in our experiments each tabular positional encoding (ATPE, CTPE) with each encoding implementation (fixed, learned).

4.5.3.1 Fixed Encoding

In fixed encoding, each positional encoding, namely \mathbf{PE}_{pos}^r , \mathbf{PE}_{pos}^c , \mathbf{PE}_{pos}^s and \mathbf{PE}_{pos}^l is encoded into fixed value as expressed by the sinusoidal function in Equation 4.2.

$$\begin{aligned} \mathbf{PE}_{pos}^r &= \mathbf{PE}_{pos}^c = \mathbf{PE}_{pos}^s = \mathbf{PE}_{pos}^l \\ &= (PE_{(pos,0)}, PE_{(pos,1)}, \dots, PE_{(pos,d_{model}/D-1)}) \end{aligned} \quad (4.5)$$

The parameter $D = 1$ for ATPE, whereas $D = 4$ for CTPE.

4.5.3.2 Learned Encoding

In learned encoding, each positional encoding is encoded by a feed forward network $\mathbb{R}^{maxlen_{pos} \times d_{model}}$ with learnable parameters, where $maxlen_{pos}$ is the maximum length of position pos . Thus, each vector \mathbf{PE}_{pos}^r , \mathbf{PE}_{pos}^c , \mathbf{PE}_{pos}^s , and \mathbf{PE}_{pos}^l at a position pos are learned during supervised training at the same time with other neural network parameters of the Transformer-based model.

4.6 Experiments

4.6.1 Experimental Settings

4.6.1.1 Metrics

There are two metrics for evaluating the performance of PBE systems [18], namely *consistency* and *generalization*. We can formally describe the definitions of consistency and generalization with notation used in Section 2.1 as below.

Definition 4.1. (*Consistency*) Given an example $\mathcal{E} = (e_i, e_o)$, consistency measures whether the PBE system generates a program \mathcal{P} such that $e_o = \mathcal{P}(e_i)$, where e_i is an example tabular data extracted from the raw tabular data \mathcal{R}_i , and e_o is the tabular data the user wants to be transformed from e_i .

Definition 4.2. (*Generalization*) Given an example $\mathcal{E} = (e_i, e_o)$, generalization measures whether the PBE system generates a program \mathcal{P} such that $e_o = \mathcal{P}(e_i) \wedge \mathcal{R}_o = \mathcal{P}(\mathcal{R}_i)$, where \mathcal{R}_o is the tabular data the user wants to be transformed from \mathcal{R}_i .

Consistency is the important metric and is used mainly to evaluate our proposed models in this dissertation. We define *accuracy* as the proportion of benchmark tests for which the PBE system is successful in terms of consistency.

Generalization is also another important metric, with several past works [80–82] particularly focusing on it. The evaluations in terms of generalization are described in 6.2.2.

4.6.1.2 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 [18, 54, 55].

In order to construct the training data, we employed the random sampling and generation methodology shown in Fig. 4.7. An input table and a program are randomly synthesized with tokens, namely characters and operations respectively, which are randomly sampled in a uniform distribution.

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

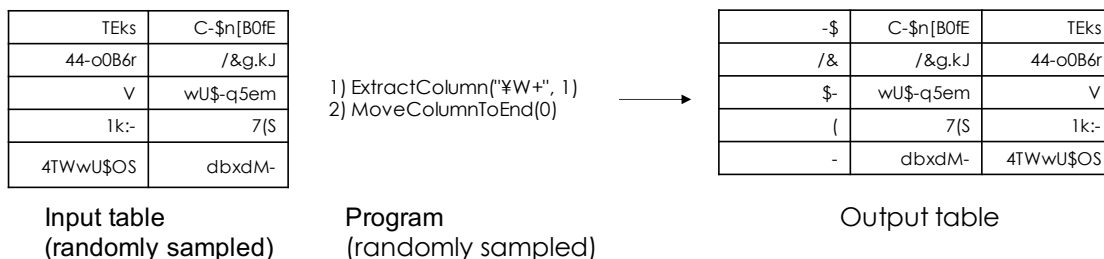


FIGURE 4.7: An example of synthesizing training data

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 2.3. 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 in transforming the input table, program synthesis is repeated until the synthesized program no longer fails in transforming 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.

4.6.1.3 Benchmark Datasets

We evaluated our models using the evaluation benchmark proposed and used in Foofah [8]¹. This benchmark is composed of 250 tests gathered from previous works and constructed to evaluate tabular-data transformation tasks.

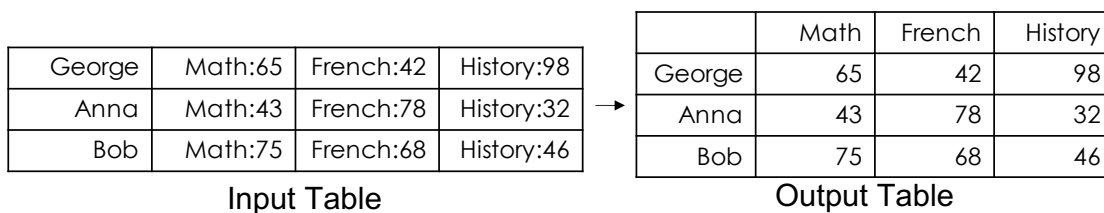


FIGURE 4.8: An example of “small” benchmark: exp0_potters_wheel_fold_2.3. This is picked from Foofah [8]

¹https://github.com/markjin1990/foofah_benchmarks

Bureau of I.A.	
Regional Director	Numbers
Niles C.	Tel:(800)645-8397
	Fax:(907)586-7252
Jean H.	Tel:(918)781-4600
	Fax:(918)781-4604
Frank K.	Tel:(615)564-6500
	Fax:(615)564-6701
Eddard S.	Tel:(404)555-0121
	Fax:(404)555-0139

→

	Tel	Fax
Niles C.	(800)645-8397	(907)586-7252
Jean H.	(918)781-4600	(918)781-4604
Frank K.	(615)564-6500	(615)564-6701
Eddard S.	(404)555-0121	(404)555-0139

Output Table

Input Table

FIGURE 4.9: An example of “large” benchmark: exp0_proactive_wrangling_complex_4. This is picked from Foofah [8]

We divided the benchmark into two parts according to the size of the table. The “small” benchmark involves tables with rows smaller than 5 and columns smaller than 5, whereas the remaining ones are referred as “large”. Figure 4.8 shows an example of “small” benchmark, and Figure 4.9 “large” benchmark. The “small” benchmark was composed of 73 tests, as opposed to 177 for the “large” benchmark. The “all” benchmark includes all 250 tests.

4.6.1.4 Hyperparameters

We describe hyperparameters for the structures of neural networks and those for training.

TABLE 4.1: Hyperparameters common for all experiments

Hyperparameters	LSTM-based	Transformer-based
Learning rate	0.005	0.0001
Optimizer	SGD w/ gradient clip	Adam w/ gradient clip
Loss function	Cross-entropy loss	Cross-entropy loss
Emb. dimension	256	256
FFN dimension	-	2048
Multihead atten.	-	8
Dropout rate	0.2	0.2
Network structure	Bidirectional LSTM	Transformer layers = 2 or 6
inference timeout	30 (seconds)	30 (seconds)

Common Parameters Table 4.1 lists the hyperparameters commonly used in all experiments in this study. This lists up hyperparameters for LSTM-based model and those for our Transformer-based model.

Note that the inference timeout is set to 30 seconds. The inference process in neural model (LSTM-based and Transformer-based model), where the beam search are exploring an answer program, is terminated when elapsed time exceeds 30 seconds. In other words, we measured the performance of our system through examining whether the system generates a program consistent with examples in 30 seconds or not. The searching process of search-based method is equally terminated at 30 seconds as well.

Parameters different depending on Experiments Table 4.2 lists the parameters that varies depending on experiments.

TABLE 4.2: Parameters varying on experiments.
 “default” in “special tokens” means ‘<sep> + <eoc> + <eol>’.
 “*” means that various parameter values are examined in the Section.

Section	token rule	special tokens	scan direction	table size	# samples	program length	# layers	beam size	TPE
4.6.2.1	sep	*	*	5×5	1M	6	2 or 6	100	-
4.6.2.2	sep	default	horizontal	*	1M	6	2 or 6	100	-
4.6.2.3	sep	default	horizontal	5×5	*	6	2 or 6	100	-
4.6.2.4	sep	default	horizontal	5×5	1M	*	2 or 6	100	-
4.6.3	sep	default	horizontal	5×5	1M	6	2 or 6	100	-
4.6.4	comb	default	horizontal	5×5	10M	6	6	100	*
5	comb	default	horizontal	5×5	10M	6	6	*	-
6.1	comb	default	horizontal	5×5	10M	6	6	100, 500	best

The “Section” column denotes the section where each experiment is conducted. The “token rule” is the tokenization rule for program linearization (see Section 4.3.1 and Section 4.3.2); “sep” corresponds to “separated token rule” and “comb” to “combined token rule”. The “special tokens” denotes special tokens used for tabular data linearization (see Section 4.2), and similarly the “scan direction” is variations of scan directions (see Section 4.2.1).

The columns “table size”, “# samples”, and “program length” are parameters relevant to training data. Section 4.6.1.2 describes how to synthesize the set of training data using these parameters. The column “# layers” denotes the number of sub-layers that constitute the encoder and decoder of Transformer-based model. It is set to 2 or 6.

The beam size was set to 100 as default in the experiments in Chapter 4. Experiments for variety of beam sizes are conducted in Chapter 5 in order to evaluate the impact of beam size to the performance of program generation.

The “TPE” column means *Tabular Positional Encodings*. The experiments with ‘-’ for the “TPE” column evaluates the Transformer-based model without Tabular Positional Encoding, namely the model with conventional Positional Encoding. The experiment

'best' for the "TPE" column selected the best performed model among the models with various types of TPEs conducted in the experiments in Section 4.6.4.

4.6.1.5 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 [83]² and our Transformer-based model using the fairseq toolkit [84]³.

4.6.1.6 Baselines

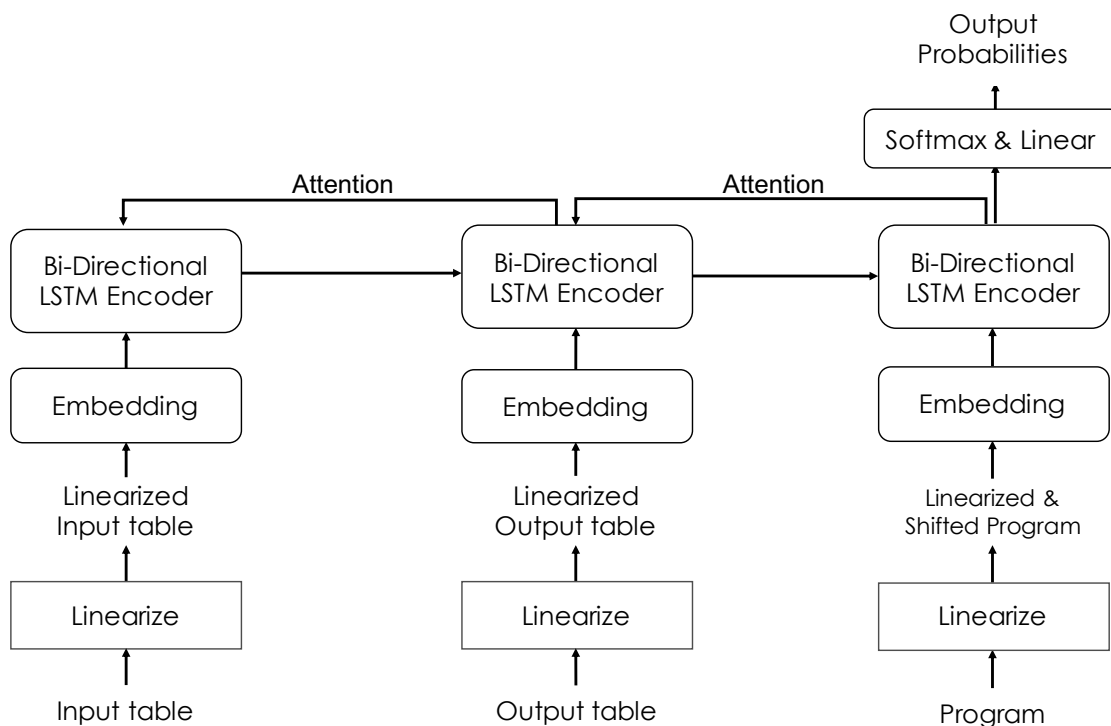


FIGURE 4.10: LSTM-based model

The LSTM-based Baseline System We developed a baseline system using an LSTM model, which is a well-known sequential encoder-decoder model, with reference to the Attention-A model in the work RobustFill [18]. Note that RobustFill is designed for the string PBE. We adopted the model as tabular transform PBE to compare with our

²<https://pytorch.org/>

³<https://github.com/pytorch/fairseq>

neural model using *tabular data linearization* (see Section 4.2) and *program linearization* (see Section 4.3). See Figure 4.10 for an overview of the model.

As in RobustFill, we composed our model with three bi-directional LSTM [85] modules which can memorize information from both directions between past and future. Thus, we expect that bi-directional LSTM modules can learn well vertical and horizontal interrelationships of components of tabular data in both directions.

An LSTM module is an encoder for an input table and another encoder for an output table, and the last one is a decoder for the program. We use the attention mechanism introduced in the work by Luong et al. [41] for our model to memorize the interrelationships between an input table, an output table and the program in training phase. The attention structure of ours is similar to those of **Attention-A** structure in Robustfill. That is, the output encoder makes attention to the input encoder and the program decoder makes attention to the output encoder. Although more complicated attention structures are proposed in Robustfill literature, they concluded **Attention-A** was most simple and sufficiently effective, so we employed the similar structure.

A pair of input-output tables is linearized to a sequence of tokens as in Section 4.2. After that, the sequence of tokens is embedded at the embedding layer and fed into the bi-directional LSTM encoder. Each encoder encodes the embedded sequence to a fixed length hidden vector and provides the vector to the next LSTM network as the initial vector. The encoder for input table provides the hidden vector to the one for output table, and the encoder for output table to the decoder for program.

In the training phase, the program is linearized to a sequence of tokens in a similar manner as in Section 4.3. The program tokens are composed of the operators and parameters listed in Table 2.3. Additionally, the start of sequence token, denoted by $\langle \text{sos} \rangle$, is inserted at the foremost left side of the sequence of program tokens, shifting the sequence to right, and the end of sequence token, denoted by $\langle \text{eos} \rangle$, is appended to the last of the sequence. The shifted sequence of program tokens is embedded at the embedding layer and fed into the bi-directional LSTM decoder. Provided the hidden vector from the encoder and embedded shifted sequence of program tokens, the decoder predicts each program token at the next time stamp in auto-regressive manner using teacher forcing [86], eventually decodes entire sequence of program tokens probabilistically.

In the evaluation phase, once training is complete, the decoder decodes multiple candidate sequences of program tokens using beam search [45]. And then, a program which satisfies the input-output examples is selected from the candidates as the answer.

The Search-based Baseline System We implemented Foofah as a baseline search-based system to compare our neural model using A*graph search as the core search algorithm with the table-edit-distance (TED) batch as the heuristic function and the same pruning rules as with Foofah. Foofah searches a program which consistent with the given input-output example tables by combining the operators and parameters listed in Table 2.3 with this search algorithm.

4.6.2 Preliminary Experiments using a Variety of Training Data

We first conducted some experiments to decide several default values for parameters in synthesizing training data. The parameters have some options in linearization methods (scanning direction and special tokens), program length, number of samples and table size. The experiments for the parameters are described in Section 4.6.2.1, 4.6.2.4, 4.6.2.3 and 4.6.2.2, respectively.

4.6.2.1 Comparison of Linearization Methods

We show how much impact our linearization methods proposed in Section 4.2 and 4.2.1 make on the performance of the PBE task. In order to investigate the impact of special symbols, we ablated each special symbols in the linearization processing and experimented over the ablated models. We conducted this ablation study on the Transformer-based model with layer 2 and 6. The corresponding experimental results for the special symbols are listed in Table 4.3.

TABLE 4.3: Experimental results of various linearization methods. The bold values denote best performance over the results in each Transformer layer.

Transformer layers	special symbols	scan direction	accuracy on small data	accuracy on large data
2	no special (A)	horizontal	54.7	4.5
2	+sep (B)	horizontal	64.3	20.3
2	+sep +eoc (C)	horizontal	80.8	29.3
2	+sep +eol (D)	horizontal	76.7	20.9
2	+sep + eoc + eol	horizontal (E)	87.6	33.8
2	+sep + eoc + eol	vertical (F)	87.6	40.1
6	no special (A)	horizontal	50.6	3.3
6	+sep (B)	horizontal	64.3	25.4
6	+sep +eoc (C)	horizontal	78.0	23.7
6	+sep +eol (D)	horizontal	72.6	20.9
6	+sep + eoc + eol	horizontal (E)	86.3	36.1
6	+sep + eoc + eol	vertical (F)	86.3	31.0

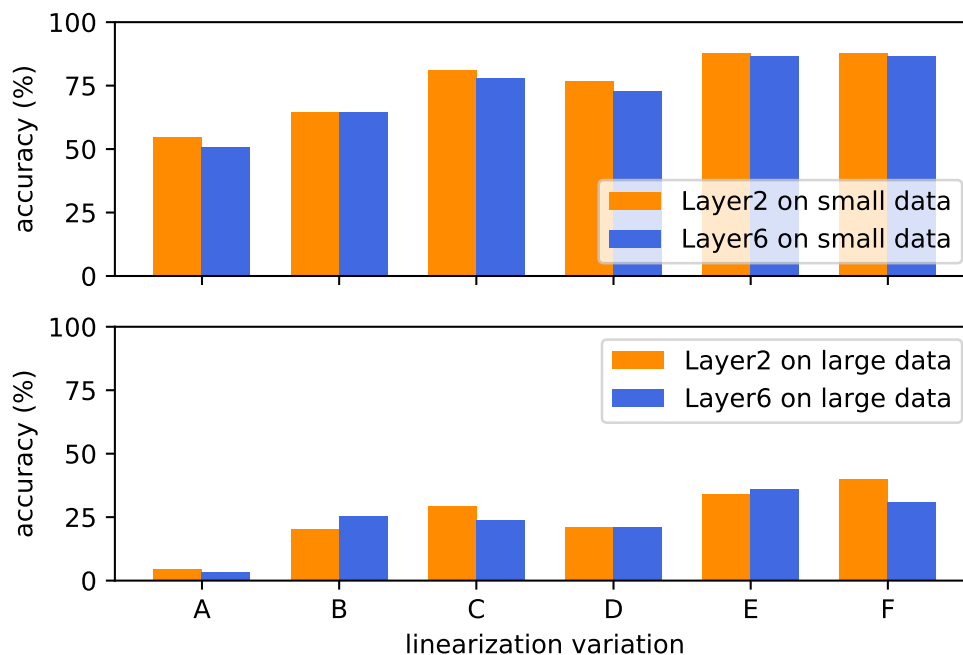


FIGURE 4.11: Accuracy with respect to linearization variations

These results are also shown as a bar graph in Fig. 4.11. The vertical axis denotes accuracy on the small data (top) and on the large data (bottom). The horizontal axis denotes linearization variations and each variation is denoted by the alphabetical symbols below.

A no special symbol is used

B only special symbol `<sep>` is used

C special symbols `<sep>` and `<eoc>` are used

D special symbols `<sep>` and `<eol>` are used

E special symbols `<sep>` and `<eoc>` and `<eol>` are used and horizontal scanning direction.

F special symbols `<sep>` and `<eoc>` and `<eol>` are used and scanning direction is vertical

In variations A-E, scanning direction is horizontal, and in F, scanning direction is vertical. Fig. 4.11 shows accuracy performance improves as the special symbols are introduced. The symbol `<sep>` which is the boundary of input-output tables (B) and `<eoc>` which is the boundary of columns of tables (C) improve accuracy performance. On the

other hand, $\langle \text{eol} \rangle$ which is the boundary of rows of tables (D) has less impact on performance improvement than $\langle \text{eoc} \rangle$. The model with all symbols has the most impact on the performance improvement. In this study, we have confirmed that the linearization method helps the model to capture the table structure of a two-dimensional input-output table pair to some extent with the special symbols that represent two-dimensional table structures.

We also compare the difference of the performance between horizontal scan (E) and vertical scan (F). We did not observe notable difference of accuracy performance in scanning direction in our experiments.

We use the variation E as the default value in the following experiments.

4.6.2.2 Comparison of Models Trained on Large Size Tables

We trained our model with a training dataset containing only large-table-size items and reevaluated the models. The results are given in Table 4.4 and shown as a bar plot in Fig. 4.12. (The notation $m \times n$ in the results indicates that the row size of the table is m and the column size is n , respectively.) This result shows that using large-table-size training datasets does not improve the performance. Training the model with table sizes of 20×20 and 30×30 in the training datasets did not converge the learning process and gave even worse results. We consider this problem to be a limitation of our current Transformer-based model. Since the Transformer is known that it might not train and perform well with much longer input sequence with more than thousands tokens, increasing the size of tables in training data does not have effect on improving the performance of the Transformer model.

TABLE 4.4: Experimental results for models trained on large size of tables. The bold values denote best performance over the results in each Transformer layer.

Transformer layers	table size	accuracy on small data	accuracy on large data
2	5×5	87.6	33.8
2	6×6	82.1	36.1
2	7×7	79.4	39.5
2	8×8	86.3	35.5
2	9×9	79.4	34.4
6	5×5	86.3	36.1
6	6×6	83.5	35.5
6	7×7	87.6	33.8
6	8×8	87.6	40.1
6	9×9	82.1	32.2

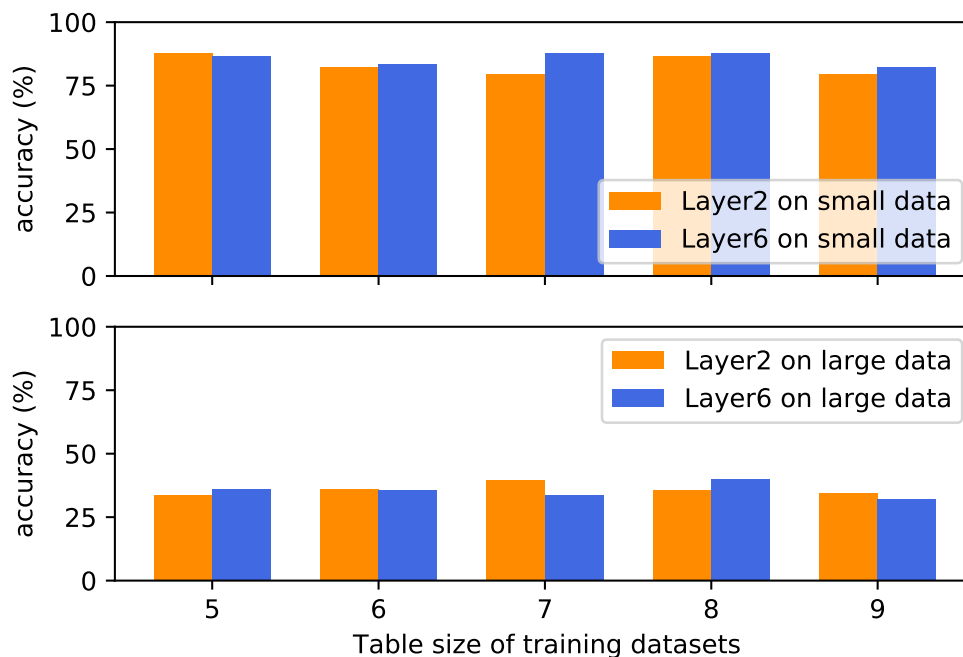


FIGURE 4.12: Accuracy with respect to table size for the training data

Some studies like Longformer [51] handle such a long input sequence in Transformer-based models. A pair of input-output tables with table-size of more than 10×10 could be converted to a sequence with more than thousands of tokens. Applying Longformer’s technique is possibly effective to address the problem on such large-size tables. We would work on this challenge in future work.

4.6.2.3 Comparison of Models Trained on Various Numbers of Samples

We examined the performance on training datasets of various sizes. The corresponding experimental results are listed in Table 4.5. Fig. 4.13 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.

TABLE 4.5: Accuracy with respect to # of samples of training data. The bold values denote the best performance over the results in each Transformer layer.

Transformer layers	# of samples	accuracy on small data	accuracy on large data
2	10K	52.0	6.2
2	100K	67.1	18.0
2	1M	87.6	33.8
2	2M	80.8	31.7
2	4M	90.4	34.4
2	10M	82.1	36.7
6	10K	52.0	2.25
6	100K	65.7	12.4
6	1M	86.3	36.1
6	2M	90.4	34.4
6	4M	93.1	41.8
6	10M	95.8	42.3

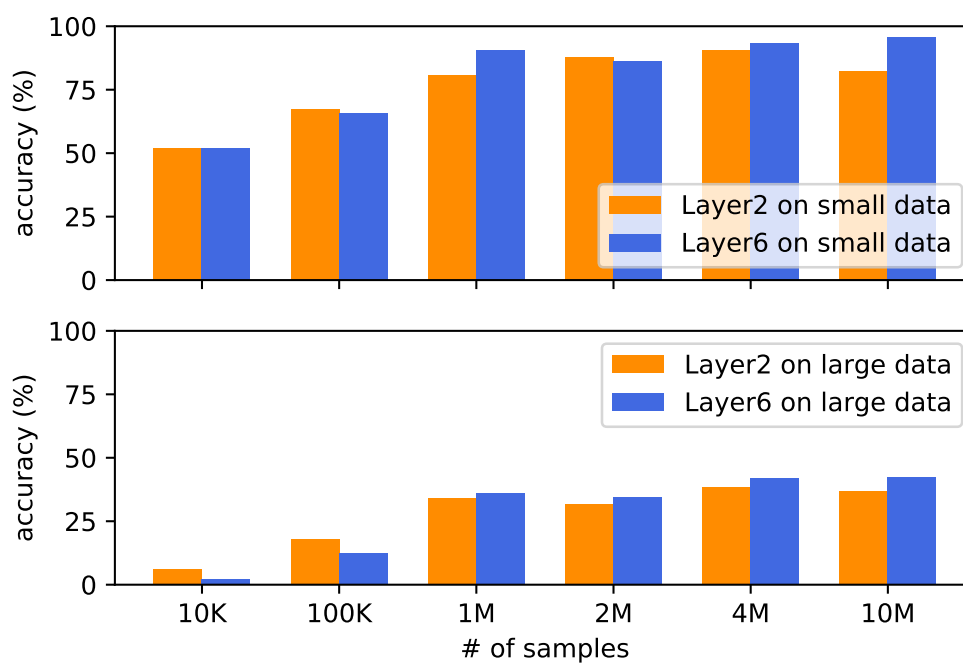


FIGURE 4.13: Accuracy with respect to number of samples in training data

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 hypothesize that the model with the larger capacity (the 6-layer model) can learn expressive representations more effectively.

4.6.2.4 Comparison of Models for a Variety of Program Lengths

We evaluated the accuracy with respect to program length. The corresponding experimental results are listed in Table 4.6. Fig. 4.14 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 over 6. This indicates that the 6-layer model has sufficient capacity to train via datasets containing long programs (as was also found in Section 4.6.2.3).

In the following Section 4.6.4, Chapter 5, and Section 6.1, 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 Sections 4.6.2.3 and 4.6.2.4.

TABLE 4.6: Accuracy with respect to program length in training data. The bold values denote the best performance over the results in each Transformer layer.

Transformer layers	program length	accuracy on small data	accuracy on large data
2	3	87.6	33.8
2	4	87.6	37.8
2	5	94.5	36.7
2	6	82.1	36.7
2	7	89.0	37.8
2	8	82.1	37.8
6	3	86.3	36.1
6	4	91.7	35.5
6	5	91.7	40.1
6	6	95.8	42.3
6	7	90.4	41.2
6	8	93.1	41.8

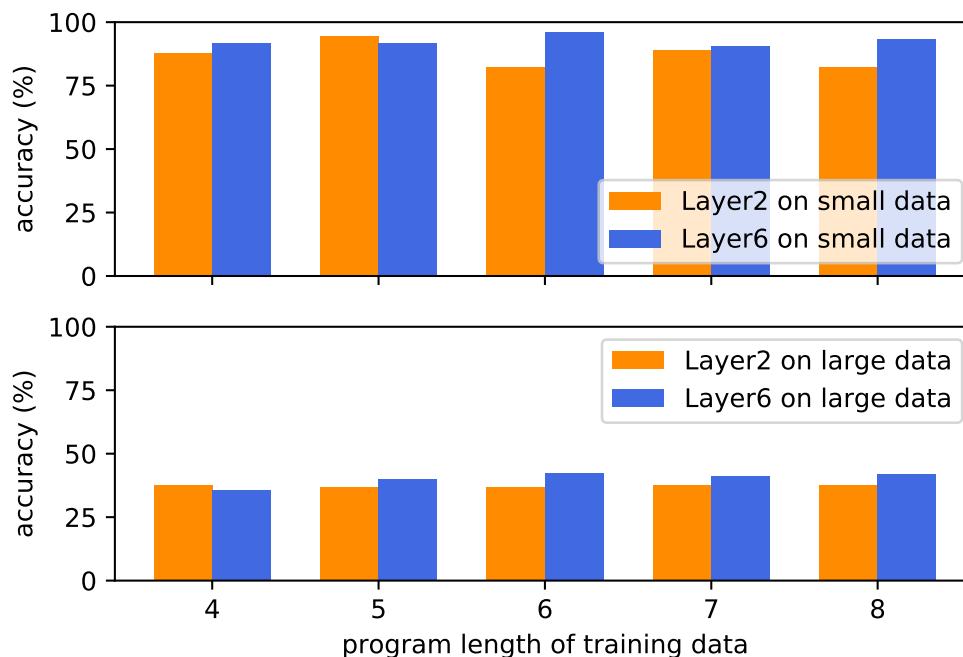


FIGURE 4.14: Accuracy with respect to program length in training data

4.6.3 Comparison of the LSTM-based Model and the Transformer-based Model

We compared a baseline (LSTM-based) model (see Section 4.6.1.6) and our Transformer-based models with 2 and 6 layers in terms of the learning time required to train them. Table 4.7 gives the training times for each model.

TABLE 4.7: Average time required to train each model in one epoch and the number of parameters for each model

Model	No. of parameters	Time / epoch (seconds)
LSTM-based	3,538,432	6729
Transformer (2 layers)	14,861,824	480
Transformer (6 layers)	44,287,488	1125

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. 4.15 shows the learning curves of these 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.

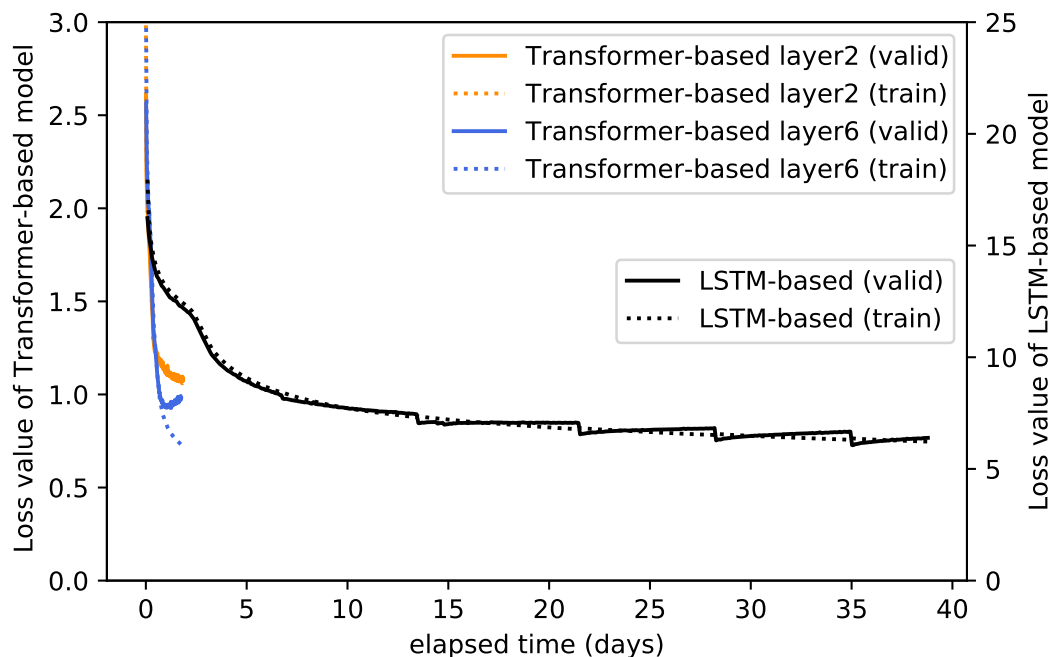


FIGURE 4.15: Learning curves

Experimental results in Table 4.8 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.

TABLE 4.8: Accuracy of LSTM-based model and Transformer-based models. The bold values denote the best performance over all results.

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

4.6.4 Comparison of Variations of Tabular Positional Embeddings and Baselines

In this section, we evaluate the accuracy performance of the tabular positional encoding (TPE). The experimental results are listed in Table 4.9. We set the parameters as defined in Section 4.6.1.4.

“PE” as the TPE type in the list denotes the result of the Transformer-based model with the original positional encoding and is the baseline in this evaluation. “ATPE” and “CTPE” denotes the results of the Transformer-based model with the tabular positional encoding proposed in this study in Section 4.5.2.1, and 4.5.2.2 respectively.

We found that the Transformer-based models with tabular positional encoding especially outperform the baseline model significantly regarding accuracy on large-size data. Considering accuracy on small-size data, they are almost comparable to the baseline model. This means that the proposed tabular positional encoding can deal with two-dimensional structural data and provide an ability of encoding two-dimensional positions with Transformer-based model.

Next, we compared the performance of ATPE and CTPE, and found that the performance of CTPE is better than that of ATPE in both cases of sinusoidal and learned.

Furthermore, We also compared the performance of sinusoidal and learned, and found that the performance of sinusoidal is better than that of learned in both cases of ATPE and CTPE. This tendency of ATPE and CTPE and that of sinusoidal and learned are a little different in the result with proposed decoding methods. The result is shown in the experiments described in Section 6.1.

TABLE 4.9: Accuracy of the Transformer-based model with tabular positional encoding and baselines. The bold values denotes the best performance over the benchmarks.

TPE type	sinusoidal or learned	accuracy on small data	accuracy on large data
PE	sinusoidal	93.1	46.8
ATPE	sinusoidal	91.7	55.3
CTPE	sinusoidal	91.7	55.9
ATPE	learned	91.7	49.1
CTPE	learned	91.7	54.2

4.7 Visualization of Attentions

4.7.1 Attention Weights

In this section, we investigate how attentions of our proposed model learn their weights. Transformer has attentions as their core mechanism. Our Transformer-based model consists of encoder and decoder, and they have three types of attentions in their neural networks: self-attention in encoder, cross-attention across encoder and decoder, and self-attention in decoder. Equation 3.3 in Section 4.1 shows that attention is computed with query Q , key K , and value V . The weight of attention

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

denotes how strongly the query Q relates the key K . By inspecting these weights of attentions, we can interpret how the Transformer-based model learns their neural parameters.

In this section, we inspect the average weights of multi-head attention in our proposed model. They are computed by the weights of multi-head attention expressed as below.

$$\text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right)$$

Where $Q_i = W_i^Q Q$, $K_i = W_i^K K$. W_i^Q and W_i^K are the learned weight parameters for the head i . Thus, we obtain an average weight of multi-head attentions as in Equation 4.6.

In the next section, we inspect the attention weights in our model using this definition.

$$\text{avg}_{i \in \text{multiheads}} \left(\text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) \right) \quad (4.6)$$

4.7.2 An Example of Attention

We select a dataset “exp0_33_3” from our evaluation benchmarks as an example to take a deeper look regarding attention weights. Figure 4.16 shows the input-output example of exp0_33_3. Listing 4.1 shows the program “WrapOneRow” that transforms multiple rows to one row, which transforms the input table into the output table of exp0_33_3.

```
[ 'WrapOneRow', {} ]
```

LISTING 4.1: Program for exp0_33_3

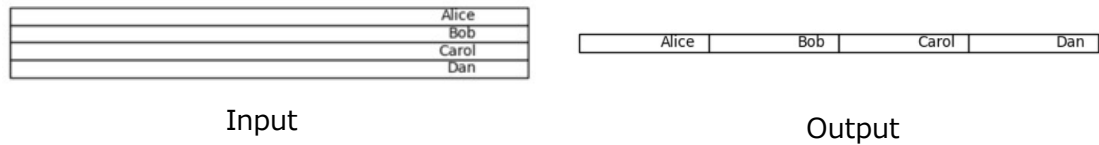


FIGURE 4.16: Input-output example for exp0_33_3

Figure 4.17 shows heatmaps of the attention weights of self-attention in the encoder of our model. They are the attention weights computed with the given input-output examples and corresponding reference program of exp0_33_3.

Each of six heatmaps in Figure 4.17 shows the attention weights of each sub-layer in the Transformer encoder. The top-left heatmap is that of 0th sub-layer (the bottom sub-layer), and the right-lower heatmaps are those of upper sub-layers, and then the bottom-right heatmap is that of 5th sub-layer (the top sub-layer). The queries are aligned on the horizontal axis of the heatmaps, and the keys on the vertical axis. In the case of the self-attention in encoder, the queries and keys are both input-output example linearized into a sequence.

We observed from these heatmaps that a query attends to the same key strongly particularly in lower sub-layers. In other words, the attentions strongly appears at the diagonal elements of the heatmap. That is, the queries locally attend to the keys around the diagonal elements of the heatmap in lower sub-layers, and contrarily they globally attend to the keys through the whole heatmap in higher sub-layers. The rectangle patterns in the higher sub-layers mean word-to-word attentions in cells of tables, which captures the tabular cell structure of the table, whereas the line patterns in the lower sub-layers mean character-to-character attentions.

Additionally, queries and keys of boundary special characters, `<eoc>`, `<eol>`, `<sep>` attend strongly. These characters are crucial to represent the tabular structure of the table and its transformation, thus, these attention weights may reflect this transformation.

Figure 4.18 shows heatmaps of the attention weights of cross-attention across encoder and decoder for exp0_33_3. The queries in the horizontal axis represents the input-output example and the keys in the vertical axis the program.

We observed that heatmaps of 0th, 1th, and 5th sub-layer show that the query `<eoc>` attends strongly to the key “WrapOneRow” operation. This attention weights reflect the function of this operation which converts `<eol>` into `<eoc>`.

Figure 4.19 shows heatmaps of the attention weights of self-attention in decoder for exp0_33_3. The queries in the horizontal axis and the keys in the vertical axis the program represents both the program. Apparently, the attention weights in the right

upper region of heatmaps are zero. This means that the self-attention in decoder is masked from the attentions from the future queries in order to prevent the decoder process from cribbing the next operation of the reference program in training data.

We cannot find other features to mention here regarding the self-attentions in decoder, except for the strong attention weight at < sos >, which is the start of the sequence of a program and thus is always anchored at the start position.

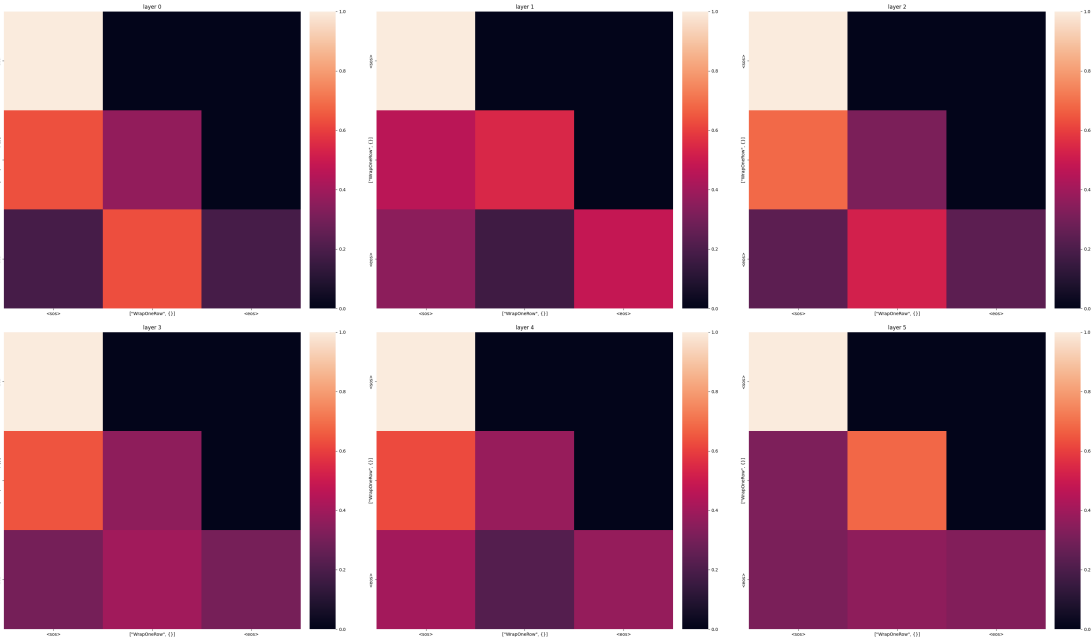


FIGURE 4.19: Self attention in decoder for exp0_33_3

Decoding Methods for Program Space Exploration

As written in 4.4, in our encoder-decoder model, beam search generates a sequence which corresponds to a program for an input-output example from the Transformer decoder. The Transformer decoder firstly outputs how likely each program component occurs for the input-output examples provided, and then beam search generates the most likely programs from the likelihoods.

Although beam search is a widely-used technique to generate sequences from the decoder of the encoder-decoder model and is especially used in NLP tasks, it is not a technique designed for program generation, thereby causes an inefficient exploration of the program search space. 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 chapter.

5.1 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 the input table. Therefore, the beam search that originates from the intermediate table is more successful than the one that stems from the input table.

On this assumption, we designed a multistep beam search which repeats beam search for 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 2 multistep beam search

Require: n_{input} ▷ Node that represents the input table
Require: n_{output} ▷ Node that represents the output table
Require: K ▷ Beam size
Ensure: P ▷ Program consistent with input-output tables

- 1: $\mathcal{T} \leftarrow (\{n_{input}\}, \emptyset)$
- 2: **while** timeout doesn't occur **do**
- 3: $n_{pick} \leftarrow \text{pickup_node}(\mathcal{T})$
- 4: $\mathcal{H} \leftarrow \text{beam_search}(n_{pick}, n_{output}, K)$ ▷ basic beam search
- 5: **for** $i \in \{0 \cdots K - 1\}$ **do**
- 6: $\mathcal{S} \leftarrow \text{generate_sequence}(n_{pick}, \mathcal{H}_i)$ ▷ $\mathcal{S} = (\mathcal{N}', \mathcal{E}')$
- 7: **if** $\exists n \in \mathcal{N}'$ such that n is identical to n_{output} **then**
- 8: **return** P ▷ operation sequence from n_{input} to n_{output}
- 9: $\mathcal{T} \leftarrow \text{graft}(\mathcal{T}, \mathcal{S})$
- 10: $\mathcal{T} \leftarrow \text{prune}(\mathcal{T})$

In order to keep the candidates of the intermediate tables and efficiently handle them, we maintain a directed tree structure $\mathcal{T} = (\mathcal{N}, \mathcal{E})$ that comprises a set of nodes $\mathcal{N} = \{n_i \mid i \in \mathbb{Z}\}$ and a set of directed edges $\mathcal{E} = \{e_i \mid i \in \mathbb{Z}\}$. (see Fig. 5.1). 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.

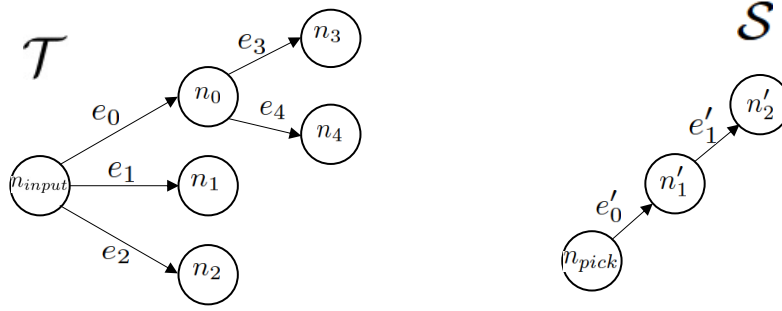
\mathcal{T} is a tree structure that has the following features.

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

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

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

For each hypothesis \mathcal{H}_i , a sequence $\mathcal{S} = (\mathcal{N}', \mathcal{E}')$ is generated, which contains the operation sequence $\mathcal{E}' = (e'_0, e'_1, \dots, e'_t)$, where $e'_t = \langle \text{eos} \rangle$. Applying the operation sequence \mathcal{E}' to each n_{pick} in order, we obtain a sequence $\mathcal{S} = (\mathcal{N}', \mathcal{E}')$, such that $\mathcal{N}' = \{n_{pick}, n'_1, n'_2, \dots, null, null\}$ and $\mathcal{E}' = \{e'_0, e'_1, \dots, null, null\}$, where $n'_1 = e_0(n_{pick})$, $n'_2 = e'_1(n'_1)$, etc. (see Line 6 in Algorithm 2 and Fig. 5.1). 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}' and \mathcal{E}' .


 FIGURE 5.1: Tree structure \mathcal{T} and sequence \mathcal{S}

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

To merge the sequence \mathcal{S} into the directed tree \mathcal{T} , the first node n_{pick} of \mathcal{S} is grafted onto the n_{pick} of \mathcal{T} (see Line 9 in Algorithm 2 and Fig. 5.2). It then prunes the edge and node that reach the identical nodes among \mathcal{T} and \mathcal{S} that have longer paths from the root node n_{input} of \mathcal{T} (see Line 10 in Algorithm 2 and Fig. 5.3). Using this pruning process, we can avoid identical nodes in \mathcal{T} , keep \mathcal{T} simple, and make it efficient to run the operations over all \mathcal{T} .

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

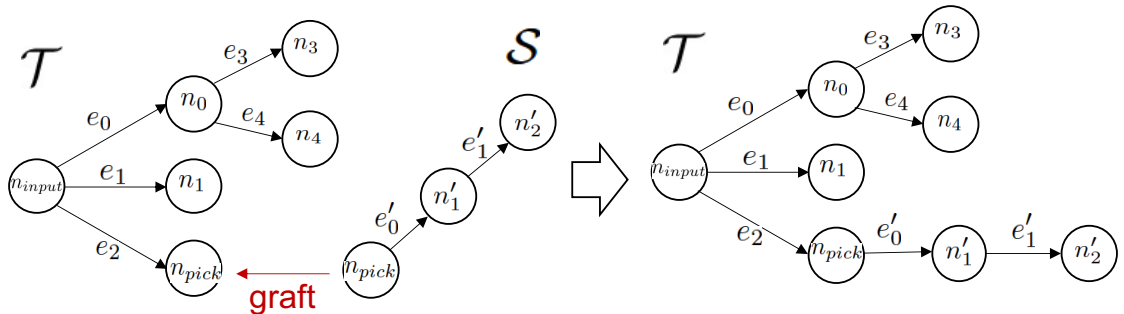


FIGURE 5.2: Graft processing

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

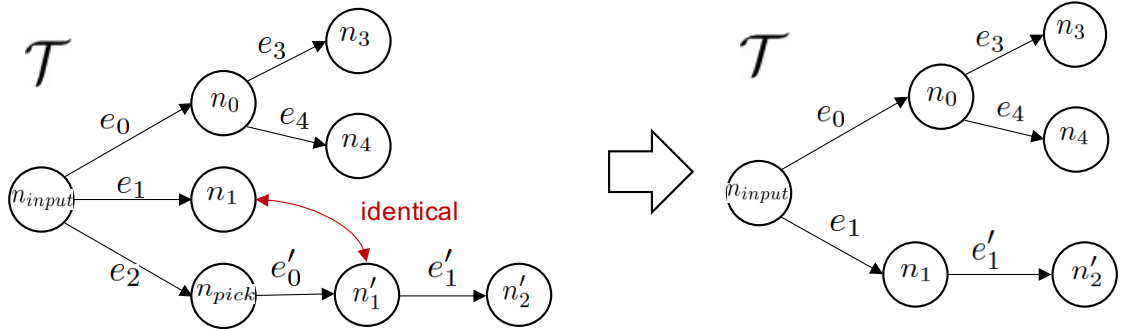


FIGURE 5.3: Prune processing

shortest path from n_{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.

5.2 PV-Beam Search

Here we propose a beam search method called as PV-beam search. Basic beam search keeps candidates including even programs that cannot create any output table and never become any correct programs. In order to improve the efficiency of the basic beam search procedure, we propose the PV-beam search that removes invalid hypotheses in the inner loop of beam search procedure and keeps only valid hypotheses.

The PV-beam search is described in Algorithm 3. It is different from the basic beam search at line 16-19 in Algorithm 3.

It synthesizes a program P from hypothesis h , and checks whether the program can transform the input table into any table, and if not, removes the corresponding hypothesis from candidates \mathcal{H}_t^{cand} . Therefore it searches only the valid hypothesis space and finds a correct program efficiently.

5.3 Experiments

5.3.1 Experimental Results

We conducted some experiments in order to evaluate our proposed beam search variants *multistep beam search* and *PV-beam search* comparing conventional basic beam search

Algorithm 3 PV-beam search

Require: *input_table* ▷ input table
Require: K ▷ Beam size
Ensure: \mathcal{H}_{final} ▷ Hypotheses obtained finally

- 1: $\mathcal{H}_0^{cand} \leftarrow \{h\}$ where $h = (\langle \text{sos} \rangle)$
- 2: $\mathcal{H}_{final} \leftarrow \emptyset$
- 3: **for** $t \in \mathbb{N}$ in ascending order **do**
- 4: $\mathcal{H}_t \leftarrow \emptyset$
- 5: $\mathcal{S}_t \leftarrow \emptyset$
- 6: **for** $h \in \mathcal{H}_{t-1}^{cand}$ **do**
- 7: **for** $z \in \mathcal{O}$ **do**
- 8: $h \leftarrow h||z$ ▷ $||$ concatenates tokens
- 9: $s \leftarrow \text{score}(h)$ ▷ *score* given by equation 4.1
- 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: $\mathcal{H}_t^{cand} \leftarrow \emptyset$
- 15: **for** $h \in \mathcal{H}_t$ in order **do**
- 16: synthesize a program P from hypothesis h
- 17: $\text{output_table} = P(\text{input_table})$
- 18: **if** output_table is not a valid table **then**
- 19: **continue**
- 20: **if** the last token of h is $\langle \text{eos} \rangle$ **then**
- 21: $\mathcal{H}_{final} \leftarrow \mathcal{H}_{final} \cup \{h\}$
- 22: $K \leftarrow K - 1$
- 23: **if** $K = 0$ **then**
- 24: **return** \mathcal{H}_{final}
- 25: **if** $|\mathcal{H}_t^{cand}| < K$ **then**
- 26: $\mathcal{H}_t^{cand} \leftarrow \mathcal{H}_t^{cand} \cup \{h\}$

and iterative beam search in previous literatures.

We used the experimental settings similar to those of Chapter 4.6.4 except for the parameter of beam size as described in Section 4.6.1.4. In order to examine how each beam search variants react for various beam size, we conducted experiments varying the beam size beginning from 1, gradually increasing, finally reaching at 1000. Table 5.1 shows these experimental results.

5.3.2 Comparison of Various Beam Search Methods

In this section, 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 [75] to compare with our proposed decoding methods. The iterative beam search repeats the basic beam

TABLE 5.1: Accuracy with respect to beam size for various decoding methods. The bold values are the best performance over the results.

Model	Decoding method	Beam width	accuracy on small data	accuracy on large data
Transformer-based	basic	1	76.7	22.5
		5	89.0	32.2
		10	89.0	35.5
		50	91.7	43.5
		100	93.1	46.8
		500	93.1	51.4
	iterative	1000	94.5	50.2
		1	86.3	40.6
		5	90.4	37.2
		10	90.4	40.6
		50	91.7	43.5
		100	93.1	46.8
	MS shortest	500	93.1	51.4
		1000	94.5	50.2
		1	86.8	29.9
		5	94.5	41.8
		10	94.5	45.1
		50	94.5	49.1
	PV-beam	100	94.5	50.8
		500	95.8	53.1
		1000	97.2	51.4
		1	80.8	24.2
		5	89.0	38.4
		10	89.0	44.6
50		91.7	57.6	
100		93.1	62.7	
	500	94.5	64.4	
	1000	97.2	60.4	
lstm-based	-	100	69.8	10.7
search-based [8]	-	-	73.9	63.2

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 4.3 is employed as a program linearization method.

The experimental results here are given in Table 5.1 and Fig. 5.4 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 is 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 execute 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. We selected the shortest path strategy because we observed the performance of it 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 all data and achieves the best accuracy among other beam search methods.

5.3.3 Comparison of the Proposed Beam Search Methods and Baselines

We show the experimental results of two baseline systems (LSTM-based system and search-based system) and the Transformer model with/without our proposed beam search methods (basic beam search, multistep beam search and PV-beam search) and compare their performance. The results are listed in Table 5.1 and summarized in Table 5.2.

Note that experimental results in this Section are not those of the Transformer-based model with tabular positional encoding in Chapter 4, but of the Transformer-based model only with decoding methods in this Chapter 5.

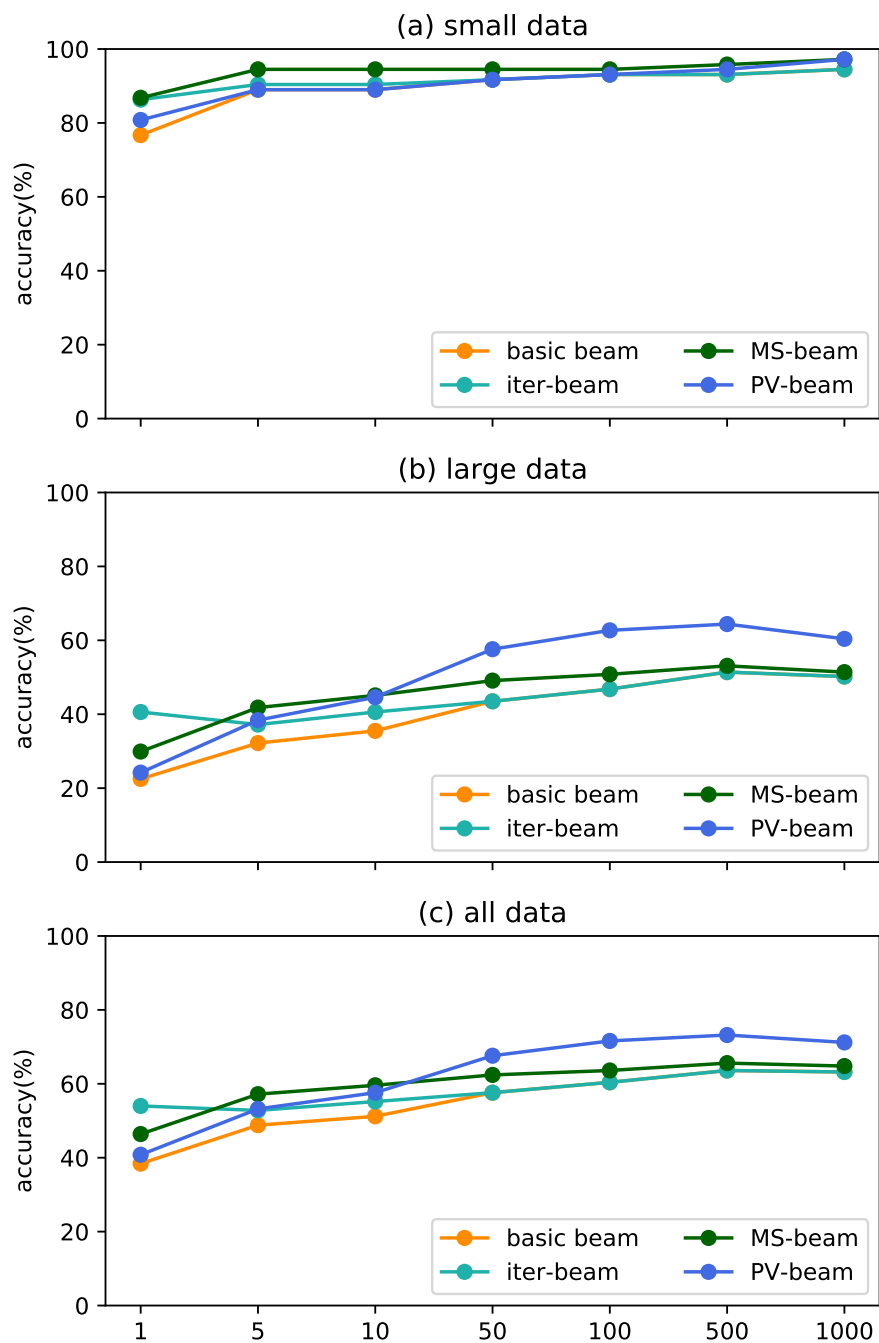


FIGURE 5.4: Accuracy of various types of beam search with respect to beam width. The top panel (a) is for benchmarks of small data, the center for those of large data, and the bottom for those of all data.

TABLE 5.2: Accuracy of baselines and proposed methods. The bold values denotes the best performance over the benchmarks.

model	decoding method	accuracy (small data)	accuracy (large data)
search-based [8]	-	73.9	63.2
LSTM	-	69.8	10.7
Transformer	basic beam (beam=100)	93.1	46.8
Transformer	basic beam (beam=500)	93.1	51.4
Transformer	multistep (beam=100)	94.5	50.8
Transformer	multistep (beam=500)	95.8	53.1
Transformer	PV-baem (beam=100)	93.1	62.7
Transformer	PV-beam (beam=500)	94.5	64.4

The LSTM-based system shows the worst performance on both small and large data. As described in Section 4.6.3, 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 on large data. However, the Transformer-based system with our proposed PV-beam search (beam=500) achieved 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. 5.5 shows the comparisons of the response time of experiments from Table 5.2, namely two baseline systems and the Transformer models with/without 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 outperforms the LSTM-based system on every benchmark data from start to the timeout (30 seconds).

The top panel (a) 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 (b) 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.

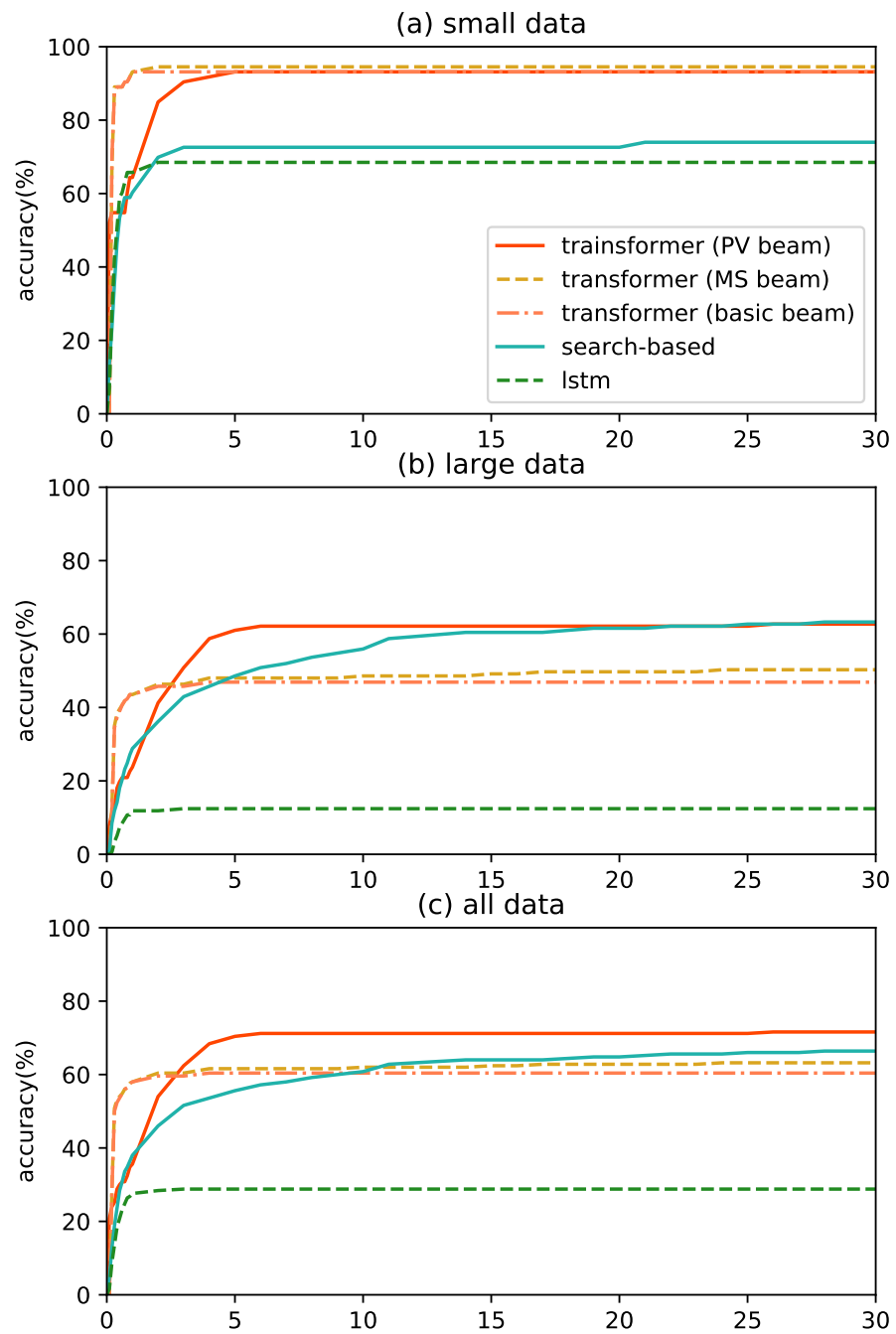


FIGURE 5.5: Accuracy with respect to elapsed time (beam size = 100). The top panel (a) shows the response performance for small benchmark data, center panel (b) for large benchmark data, and bottom panel (c) for all benchmark data.

The bottom panel (c) 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 observe in our experiments the Transformer model with PV-beam search with beam size of 500 (see Fig. 5.6) is inferior to the search-based method for a few seconds from the start time due to the high computational cost of the program validation process (line 16-18 in Algorithm 3). Thus, 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 performance.

We also observe 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 where a quick response is more desirable.

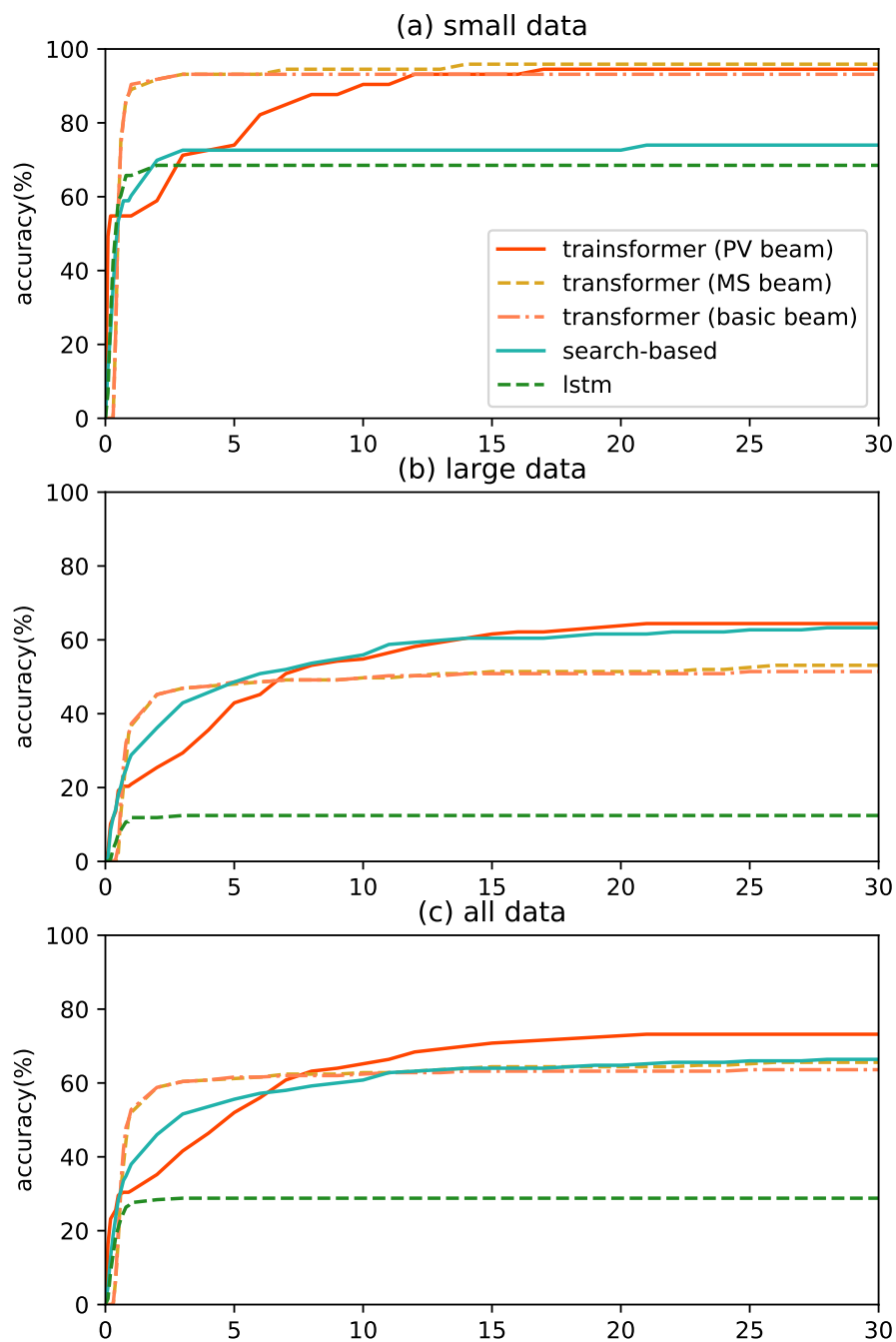


FIGURE 5.6: Accuracy with respect to elapsed time (beam size = 500). The top panel (a) shows the response performance for small benchmark data, center panel (b) for large benchmark data, and bottom panel (c) for all benchmark data.

Chapter 6

Transformer-based Model with TPE and PV-Beam Search

In this chapter, we focus on the neural model that we finally obtained through this study: the Transformer-based model with both tabular positional encoding (TPE) in Chapter 4 and PV-beam search, which performs best in Chapter 5. We first experimentally evaluate the model in Section 6.1, and then discuss how to construct a practical PBE system using this model in Section 6.2.

6.1 Experimental Evaluation

In this Section, we show the experimental results of the Transformer-based model with both tabular positional encoding (from Chapter 4) and PV-beam search, which performs best in Chapter 5. We used the experimental settings written in Section 4.6.1. We set a timeout to 30 seconds. Each experiment aborts when it exceeds the timeout. Table 6.1) shows the experimental results of our proposed models and the baseline systems in terms of accuracy.

We compare our proposed model with two types of baselines in our experiments. The first baseline is a search-based (non-neural) method that simulates the work Foofah [8] described in Section 4.6.1.6. It is referred as "Search-based" in the experimental results. The second one is the Transformer-based model that was the best in the accuracy

TABLE 6.1: Accuracy of baselines and proposed models. The bold number represents the best performance over a benchmark set.

Model	beam size	accuracy (small data)	accuracy (large data)	accuracy (all data)
Search-based [8]	-	73.9	63.2	66.4
Transformer (best in Section 5.3.3)	100	93.1	62.7	71.6
Transformer w/ ATPE (sinusoidal)	100	91.7	68.3	75.2
Transformer w/ ATPE (learned)	100	91.7	64.4	72.4
Transformer w/ CTPE (sinusoidal)	100	91.7	69.4	76.0
Transformer w/ CTPE (learned)	100	91.7	70.6	76.8
Transformer (best in Section 5.3.3)	500	94.5	64.4	73.1
Transformer w/ ATPE (sinusoidal)	500	94.5	72.3	78.8
Transformer w/ ATPE (learned)	500	93.1	70.0	76.8
Transformer w/ CTPE (sinusoidal)	500	91.7	69.4	76.0
Transformer w/ CTPE (learned)	500	91.7	72.3	78.0

performance in Section 5.3.3 with the PV-beam method and beam size 500. It is referred as "Transformer" in the experimental results. We no longer compare with the baseline of LSTM-based neural method described in Section 4.6.1.6 because we already know our proposed Transformer-based model outperforms it significantly on the experiments in previous Sections.

Our models proposed in Section 4.5.2 are referred as "ATPE" and "CTPE". Each model is implemented both by the fixed encoding with sinusoidal function referred as "sinusoidal" and by the learned encoding denoted as "learned".

First, these experimental results show the ML-based models, namely Transformer, ATPE and CTPE outperform the search-based method in both the large and the small datasets. Especially, the Transformer has a large advantage in small datasets. This shows the ML-based models can deal well with small-size tabular data resulting in better performance in the small datasets compared to the conventional search-based method.

Second, our proposed models, ATPE and CTPE, are comparable to the Transformer in the small datasets, and are significantly better than the Transformer in the large datasets. This result shows that our proposed models represent the vertical and horizontal dependencies between tabular cells by embedding the positions of two-dimensional structure directly, and are promising models for encoding two-dimensional tabular data.

We do not find remarkable difference in the accuracy performance between ATPE and CTPE and between sinusoidal and learned. Their performance is comparable to each other. The model that achieves the best performance among all models is the ATPE with sinusoidal function.

Finally, we compare our proposed models and the baselines in terms of their response time performance. We choose the ATPE with sinusoidal function which achieved the

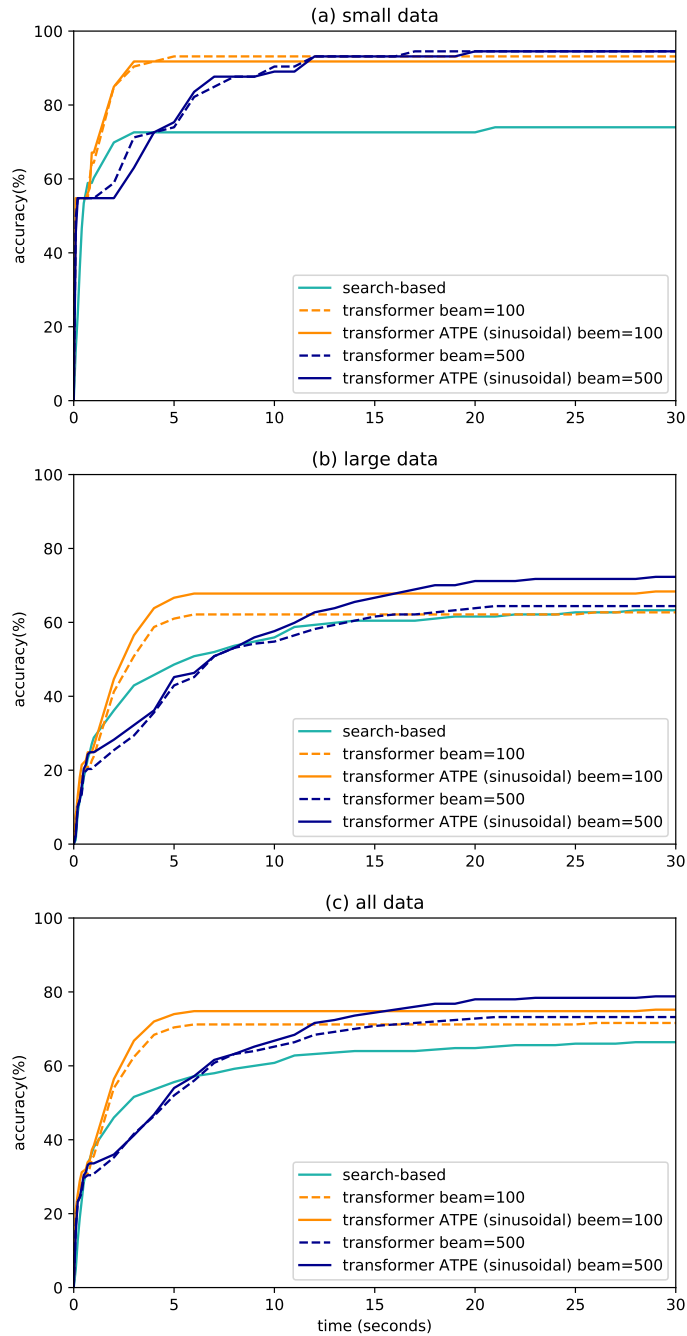


FIGURE 6.1: Accuracy with respect to the elapsed time. The top panel (a) shows the performance in terms of the response with the small benchmark datasets, the center panel (b) shows the results with the large benchmark datasets, and the bottom panel (c) shows the results with all benchmark datasets.

best performance in Table 6.1. The bottom panel (c) of Figure 6.1 compares the response performance of the search-based method and ML-based models with beam size 100 and 500 with all datasets. Our proposed model, ATPE with beam size 100, achieved the best performance in the time range from start to about 15 seconds. And ATPE with beam size 500 outperforms that with beam size 100 when the time elapses longer than 15 seconds. This means a larger beam size expands the solution space of beam search and takes a longer time to solve the problem.

Thus, we suggest that our proposed model with beam size 500 is better if the final accuracy is the preferred metric, whereas our proposed model with beam size 100 would be better if a rapid response is more desirable.

6.2 Towards Practical PBE Systems

6.2.1 Iterative PBE System

We now explain the practical goal of our PBE system by demonstrating examples.

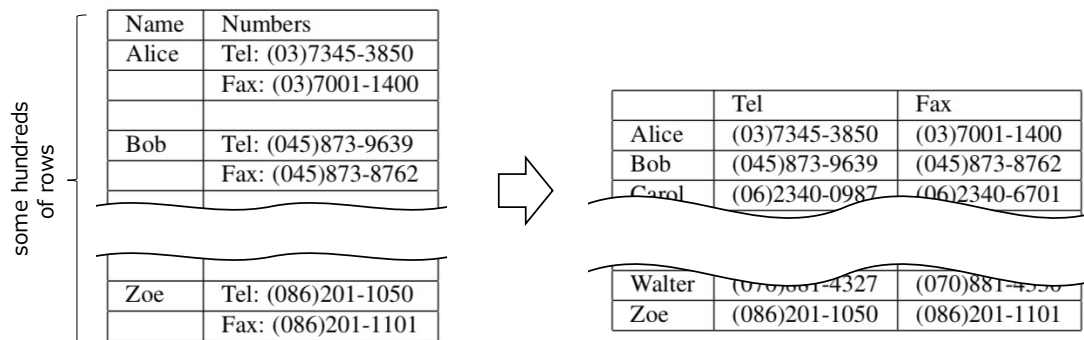



FIGURE 6.2: The left-hand table is the original table to be transformed, and the right-hand table is the table desired

Suppose a data analyst wants to transform a large table with some hundreds of rows as seen in Fig. 6.2. Since transforming such a large table is a very cumbersome and time-consuming task, she instead tries creating a program that transforms the whole original table into the desired table using a PBE system. Creating a transformation program using the PBE system requires iteration steps across the PBE system and the human as described below.

In the initial step, she creates a small input-output example table through extracting an input example from the original table and creating an output example from scratch as shown in Fig. 6.3. Then, when she commands the PBE system, it synthesizes a program (in Listing 6.1) based on her example.


Alice	Tel: (03)7345-3850
	Fax: (03)7001-1400



	Tel	Fax
Alice	(03)7345-3850	(03)7001-1400

FIGURE 6.3: Input-output example tables used in the first step: the left-hand table is the input example and the right-hand table is the desired output

Name	Numbers
Alice	Tel: (03)7345-3850
	Fax: (03)7001-1400



	Tel	Fax
Alice	(03)7345-3850	(03)7001-1400

FIGURE 6.4: Input-output example tables used in the second step: the left-hand table is the input example and the right-hand table is the desired output

```
Split(1, ':')
Fill(0)
UnFoldHeader(1)
```

LISTING 6.1: Transformation program for the initial step

After that, she runs the synthesized program on the whole original table to validate that the program outputs the table as she intends. In this case, the program runs incompletely because the program did not learn the transformations for the table headers and the blank lines; the examples does not include these items like headers and blank lines.

In the second step, she modifies the first example including the table headers and the blank lines as shown in Fig. 6.4. This time, she ensures that the synthesized program outputs the table she intends and consequently finds the correct program (in Listing 6.2).

```
Split(1, ':')
Delete(2)
Fill(0)
UnFoldHeader(1)
```

LISTING 6.2: Transformation program for the second step

She can eventually obtain a user-intended program through repeating the iteration of modifying examples, synthesizing a program, and validating the program.

The goal of the PBE system is not only synthesizing a correct program quickly. There are five requirements for the PBE system as below.

- Reducing the cost of one iteration
 - Easiness of creating and giving examples

- Consistent and fast program synthesis
- Easiness of validation
- Reducing the number of iterations
 - Generalization of program
 - Resolving Failure cases

Easiness of Creating and Giving Examples Creating examples and giving them into the PBE system is repeated many times iteratively until the intended program is completely synthesized. Creating examples for tabular transformations is relatively more complicated than string transformations. Therefore, creating input-output examples easily and quickly is critical for the usability of PBE for tabular transformations. Furthermore, giving the examples into the PBE system must be also an easy and quick operation for users like one-click action. Although it is an interesting issue, we do not expand our discussion on user-interface related questions as they are out of the scope of this study.

Consistent and Fast Program Synthesis The PBE system needs to be able to rapidly synthesize the program consistent with user-given examples. We have focused on the consistency metric through this dissertation and achieved good performance using our proposed model.

Easiness of Validation The validation process is also repeated many times iteratively. In the validation process, users transform the original data using the synthesized program and then check if the output data is what they want or not. Large original and transformed data, which possibly ranges to some thousands of rows or columns, prevent users to view the entire data in one screen at a time. Thus, finding erroneous region from such large data efficiently is difficult for users. An ideal PBE system involves functions of highlighting or visualizing the regions that should be focused from the whole data. Although it is also an interesting issue, we do not expand our discussion on user-interface related questions as they are out of the scope of this study.

Generalization of Program The PBE system that successively synthesizes the user-intended program from smaller and clueless examples can reduce the iteration number. The metric that measures this ability is called *generalization*. We discuss the performance of generalization for our proposed model in Section 6.2.2.

Resolving Failure Cases The PBE system often fails generating programs for several reasons. Providing some good hints is crucial for amending the failure and consequently reduce the iteration number. We discuss the challenges in several failure cases in Section 6.2.3.

6.2.2 Generalization

Generalization (see Section 4.6.1.1) is a key metric for a practical PBE system. The goal of PBE users is synthesizing a program that is able to transform not only the user-provided example data but also the original data successfully. The generalization performance is the ability of synthesizing a program that can transform the original data and is regarded as more important than consistency in a practical PBE system. A PBE system with good generalization performance suppresses the number of user’s iteration seen in the Section 6.2.1 and allows users to obtain the solution they want quickly.

In this section, we evaluate the generalization of our proposed model and existing search-based method. The Foofah datasets are composed of 250 datasets with 50 independent benchmarks. Each benchmark has five datasets that represent an identical tabular-transformation but different in table size.

Let \mathcal{D}_i be a dataset (input-output tables) of a benchmark for size $i \in 1, 2, 3, 4, 5$. Inclusion relation holds as follows.

$$\mathcal{D}_1 \subset \mathcal{D}_2 \subset \mathcal{D}_3 \subset \mathcal{D}_4 \subset \mathcal{D}_5$$

Where the relation $\mathcal{D}_i \subset \mathcal{D}_j$ means dataset \mathcal{D}_i is a part of dataset \mathcal{D}_j . Notably here, we define that the inclusion symbol “ \subset ” denotes whether a table includes the cells of another table or not, apart from the strict definition in set theory. Figure 6.5 shows an example dataset of \mathcal{D}_1 (left) and \mathcal{D}_2 (right) from a benchmark named “exp0_10”. It shows that the input–output tables of \mathcal{D}_1 are part of those of \mathcal{D}_2 . Additionally, Figure 6.6 shows the example for \mathcal{D}_3 and \mathcal{D}_4 , and Figure 6.7 shows the example for \mathcal{D}_5 . These examples show datasets \mathcal{D}_i satisfy the relation $\mathcal{D}_i \subset \mathcal{D}_j$ for $i < j$.

We evaluated our model by ensuring whether each program synthesized from datasets \mathcal{D}_i for $i \in 1, 2, 3, 4$ successfully transforms an input table of \mathcal{D}_5 into an output table of \mathcal{D}_5 or not. Table 6.2 shows the generalization performance of our proposed model that performs best over experiments in Section 6.1. “Consistent programs” denotes the number of benchmarks that can synthesize programs that transform the input table of \mathcal{D}_i into the output table of \mathcal{D}_i . “Generalized programs” are those that can transform

Clothing	Color1	Color2	Color3	Color4	Color5
Shirt	Red	Yellow	Blue	Green	Orange

Input

Shirt	Red
Shirt	Yellow
Shirt	Blue
Shirt	Green
Shirt	Orange

Output

Clothing	Color1	Color2	Color3	Color4	Color5
Shirt	Red	Yellow	Blue	Green	Orange
Pants	Yellow	Blue	Green	Orange	Red

Input

Shirt	Red
Shirt	Yellow
Shirt	Blue
Shirt	Green
Shirt	Orange
Pants	Yellow
Pants	Blue
Pants	Green
Pants	Orange
Pants	Red

Output

FIGURE 6.5: Input-output tables of \mathcal{D}_1 (left) and \mathcal{D}_2 (right) in exp0.10 benchmark

Clothing	Color1	Color2	Color3	Color4	Color5
Shirt	Red	Yellow	Blue	Green	Orange
Pants	Yellow	Blue	Green	Orange	Red
Hat	Blue	Green	Orange	Red	Yellow

Input

Shirt	Red
Shirt	Yellow
Shirt	Blue
Shirt	Green
Shirt	Orange
Pants	Yellow
Pants	Blue
Pants	Green
Pants	Orange
Pants	Red
Hat	Blue
Hat	Green
Hat	Orange
Hat	Red
Hat	Yellow

Output

Clothing	Color1	Color2	Color3	Color4	Color5
Shirt	Red	Yellow	Blue	Green	Orange
Pants	Yellow	Blue	Green	Orange	Red
Hat	Blue	Green	Orange	Red	Yellow
Scarf	Green	Orange	Red	Yellow	Blue

Input

Shirt	Red
Shirt	Yellow
Shirt	Blue
Shirt	Green
Shirt	Orange
Pants	Yellow
Pants	Blue
Pants	Green
Pants	Orange
Pants	Red
Hat	Blue
Hat	Green
Hat	Orange
Hat	Red
Hat	Yellow
Scarf	Green
Scarf	Orange
Scarf	Red
Scarf	Yellow
Scarf	Blue

Output

FIGURE 6.6: Input-output tables of \mathcal{D}_3 (left) and \mathcal{D}_4 (right) in exp0.10 benchmark

Clothing	Color1	Color2	Color3	Color4	Color5
Shirt	Red	Yellow	Blue	Green	Orange
Pants	Yellow	Blue	Green	Orange	Red
Hat	Blue	Green	Orange	Red	Yellow
Scarf	Green	Orange	Red	Yellow	Blue
Glove	Blue	Purple	Red	Yellow	Green

Input

Shirt	Red
Shirt	Yellow
Shirt	Blue
Shirt	Green
Shirt	Orange
Pants	Yellow
Pants	Blue
Pants	Green
Pants	Orange
Pants	Red
Hat	Blue
Hat	Green
Hat	Orange
Hat	Red
Hat	Yellow
Scarf	Green
Scarf	Orange
Scarf	Red
Scarf	Yellow
Scarf	Blue
Glove	Blue
Glove	Purple
Glove	Red
Glove	Yellow
Glove	Green

Output

FIGURE 6.7: Input-output tables of \mathcal{D}_5 in exp0.10 benchmark

the input table of \mathcal{D}_5 into output table of \mathcal{D}_5 . “Consistency (%)” is the ratio of the benchmarks where a consistent program is synthesized, whereas “generalization (%)” is the ratio of the benchmark where a generalized program is synthesized.

TABLE 6.2: Generalization (our proposed model)

dataset size	consistent programs	generalized programs	consistency (%)	generalization (%)
\mathcal{D}_1	46	22	86.0	44.0
\mathcal{D}_2	41	38	82.0	76.0
\mathcal{D}_3	38	38	76.0	76.0
\mathcal{D}_4	38	38	76.0	76.0
\mathcal{D}_5	34	34	68.0	68.0

Table 6.2 shows that the consistency decreases as the dataset size increases because synthesizing problem is more difficult for larger dataset size, whereas the generalization increases as the dataset size increases from \mathcal{D}_1 to \mathcal{D}_2 because expressiveness of \mathcal{D}_2 is higher than \mathcal{D}_1 . The generalization of \mathcal{D}_1 is extremely low, because a considerable number of datasets of \mathcal{D}_1 are too small to express the transformation of \mathcal{D}_5 . This means PBE systems have no way that synthesizes generalized programs from such small examples without enough clues for generalization. The generalization of \mathcal{D}_2 significantly better than that of \mathcal{D}_1 and achieves 76%. The generalization and consistency are equal from \mathcal{D}_3 to \mathcal{D}_5 . This means dataset size larger than \mathcal{D}_3 have sufficient expressiveness in all tests for generating programs consistent with the dataset of \mathcal{D}_5 . However, the generalization of \mathcal{D}_3 to \mathcal{D}_5 does not better than that of \mathcal{D}_2 because consistency of those dataset size never better than 76%. Thus, the generalization achieves the best at \mathcal{D}_2 in our proposed model.

TABLE 6.3: Generalization (search-based)

dataset size	consistent programs	generalized programs	consistency (%)	generalization (%)
\mathcal{D}_1	37	19	74.0	38.0
\mathcal{D}_2	36	34	72.0	68.0
\mathcal{D}_3	34	33	68.0	66.0
\mathcal{D}_4	30	30	60.0	60.0
\mathcal{D}_5	30	30	60.0	60.0

Table 6.3 shows the generalization of search-based method. Our model outperforms the existing search-based method in terms of both consistency and generalization in all

datasets \mathcal{D}_i for $i \in \{1, 2, 3, 4, 5\}$. Notably, better generalization of our model comes from the better consistency. Accordingly, the performance feature in sole generalization of our proposed model is almost similar to that of search-based method. For example, the generalization of search-based method is the best (68%) at \mathcal{D}_2 as well as that of our proposed model.

6.2.3 Resolving Failure Cases

PBE is a black-box and obscure system for PBE users. When the PBE system fails to generate a program, all users can do is providing additional or renewed examples. However, PBE systems merely offer hints that help users to reason out what causes the failure, and merely provides a synthesis process and generated programs easy to understand.

Failures in program synthesis are categorized into three failure cases:

- **Ambiguous Specification:** the PBE system successes generating programs, but it's wrong solution.
- **Solvable Failure:** the PBE system fails generating programs despite with solvable program space.
- **Unsolvable Failure:** the PBE system fails generating programs due to unsolvable program space”.

These failure cases can be formally described as follows. Let P (Program space) to be the set of all programs that the PBE system can express, E (Explored space) $\subset P$ to be the set of all programs the PBE system has explored, C (Consistent programs) to be the set of all consistent programs that user-given examples can express, and G (Generalized programs) $\subset C$ to be the set of all programs that the user wants to obtain. Specifically, a success situation can be expressed using a set relation $E \cap G \neq \emptyset$ and this relation illustrated in Figure 6.8.

6.2.3.1 Ambiguous Specification

PBE systems possibly generate programs consistent with user-given examples but inconsistent with unseen data. This case is formally expressed with $E \cap G = \emptyset \wedge E \cap C \neq \emptyset$ and is illustrated in Figure 6.9.

This failure case results from ambiguity of the user-given examples. However, providing additional examples that disambiguate their intent is often hard for users. Mayer et

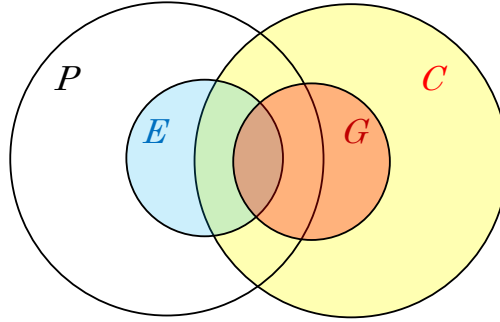


FIGURE 6.8: Venn Diagram for a success case

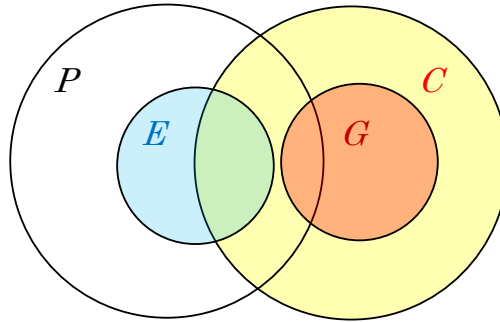


FIGURE 6.9: Venn Diagram for an ambiguous specification case

al. [87] and some studies [88, 89] tackle this problem in an active learning manner by automatically suggesting input examples that distinguish multiple programs. Peleg et al. [90] allows users to directly specify which parts of a generated program must be included or excluded in the next iteration. These approaches can be applied to our proposed system in principle, and consequently improve our interactive PBE system discussed in Section 6.2.1.

6.2.3.2 Solvable Failure

PBE systems possibly fail generating programs consistent with user-given examples. The case means the PBE system fails finding a solution despite the program space explored includes solutions (i.e., programs consistent with examples). This case is formally expressed with $E \cap C = \emptyset \wedge P \cap C \neq \emptyset$ and is illustrated in Figure 6.10.

This failure mainly results from high complexity of user-given example, thus is possibly resolved by providing examples decomposing a complex example to simpler example. Little work has been done against this failure. Zhang et al. [91] proposes an interpretable synthesizer for regular expressions that shows the underlying synthesis process to users and thus providing hints to them in the next iteration. However, users hardly understand the underlying synthesis process (for example, program space explored in the current step). This challenge remains difficult for now and have to be solved in future work

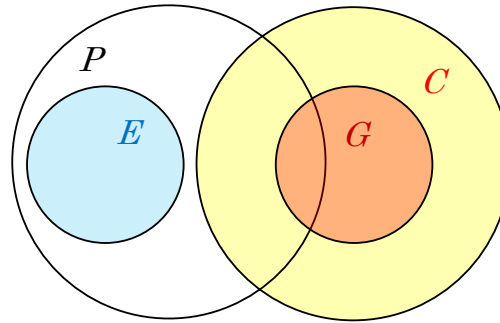


FIGURE 6.10: Venn Diagram for a solvable failure case

6.2.3.3 Unsolvable Failure

This case means the PBE system fails finding a solution because the program space explored never includes solutions. This case is formally expressed with $P \cap C = \emptyset$ and is illustrated in Figure 6.11.

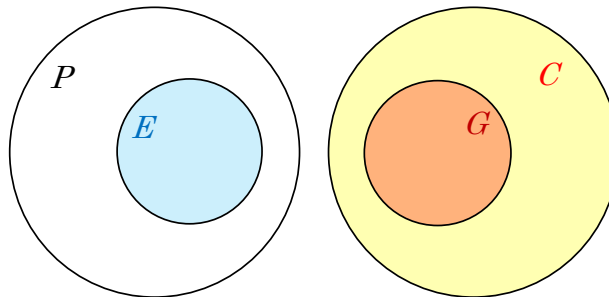


FIGURE 6.11: Venn Diagram for an unsolvable failure case

This case often occurs because many users are novice for the PBE system, and naturally unfamiliar with how expressive programs the PBE system can generate. In this case, users have to give up using the PBE system, and attempt to select other techniques, including those equipped in data preparation tools to solve their problem.

However, users hardly know which is better providing simpler examples to the PBE system or giving up using the PBE system. A technique of successfully detecting if the user-given example is solvable or not is strongly expected, but is difficult to realize for now.

6.2.4 Practical Data Preparation System

Data preparation includes a lot of tasks: not only data transformation, but also importing data, cleaning & cleansing data, enriching data, understanding data.

Importing data Data preparation leverages various formats of data from various sources: structured data collected from relational database management systems (RDBMS) and DWH; semi-structured data from NoSQL, CSV, XML, and the Web; and unstructured data from text documents, and log files. Hence, data preparation tools require functions able to read various data formats and convert them into an inner format viewable in a spreadsheet of the tools.

Cleaning & Cleansing data Data preparation cleans the data and makes its quality high enough to inject into post-process systems like business intelligence (BI) tools and statistical tools. Pre-processing includes removing outliers, filling missing values, filtering, and entity resolution as well as tabular transformation discussed in this dissertation.

Enriching Data Data preparation enriches the data with external data, collected external data from various sources like Web sites, open data from governments, and free or commercial data published by various organizations. Enrichment techniques include schema matching, lookup, and joining of external tables.

Understanding data Data preparation requires the data understanding process for knowing the nature of the unfamiliar data and validating if the data is transformed well. Data understanding includes detecting erroneous/missing values, datatype inference, profiling, aggregation, and visualizing these results. Data preparation includes many tasks as described above. Users tackle data preparation by conducting these tasks back and forth each other. Thus, data preparation requires tools that integrate features for these tasks seamlessly and allow users to complete these tasks in one package. Many data preparation tools equip this feature in a user interface and integrate these tasks.

Our neural model should be integrated in such a data preparation tools. PBE can solve only tabular transformation tasks, which is just one part of data preparation, thus it is difficult to be used as a tool isolated from other data preparation functions.

FlashFill [9] is a representative product integrated into Microsoft Excel. However, it supports only syntactic (string) transformations. Wrex [92] proposes a PBE system integrated as a Jupyter notebook extension. This system allows synthesizing readable codes in Python language for dataframe transformation by providing an output dataframe in the code block on the Jupyter notebook. It integrates the PBE task and other tasks on the Jupyter notebook. However, Jupyter notebooks are suitable for Python programmers but not for novice users without coding skills.

For those considerations, we conclude that our model best be embedded into data preparation tools like OpenRefine [11], and Trifacta [12]. Our model embedded in a data preparation tools will help users make tabular transformations only by providing examples in iterative manner without specifying operators and parameter.

Chapter 7

Conclusion

7.1 Summary

In this dissertation, we have proposed a new ML approach to realize PBE for tabular transformations. As far as we know, our Transformer-based neural model is the first of its kind, even for non-tabular-transformation PBE.

In order to handle tabular transformations, we have proposed the Transformer-based model that enables learning large scale parameters of neural network and therefore acquires the ability to express programs of tabular data. The Transformer-based model achieves faster and more expressive performance than the LSTM model (baseline) used in conventional ML-based PBE research.

We have also proposed the tabular positional encoding that enables the Transformer-based model to learn the two-dimensional positions of each element and thus learn the tabular structure of tabular data. We have designed two types of tabular positional encoding called ATPE and CTPE and also evaluated the model with learned parameters and fixed ones. We have experimentally ensured the Transformer-based model with this tabular positional encoding outperforms the models without the tabular positional encoding.

In addition, we have proposed multistep beam search and the PV-Beam search, which optimize the conventional beam search for program generation. We had experiments that compare the accuracy performance of the model with our proposed beam search

and that with basic beam search. As a result, we found that the PV-beam search surpasses the accuracy and response of the baselines and the multistep beam search realizes much shorter response time than the search-based system.

Finally, we compared our proposed Transformer-based models that includes tabular positional encoding and multistep beam search and PV-beam search as the decoding methodology to the baselines (LSTM-based model and search-based model) in experiments. We found that our proposed models achieve superior performance to baselines and provide a high-performance and scalable ML model for tabular transformation PBE. Thus, the model enables users to perform table transformation easily and effortlessly.

We can conclude that our model allows users to generate a correct program quickly from user-supplied small examples and obtain a final program using the PBE system interactively.

7.2 Other Applications

Our achievements are applicable not only to PBE for tabular transformation but also other applications. Especially, two-dimensional tabular positional encoding can be used in many table understanding tasks using the Transformer encoder.

For example, we can consider applying it to cell-classification studies [66, 67, 93] that classifies cells in a spreadsheet into semantic types like: “data”, “attributes”, “meta-data”, “header” or “derived”, to discover tables and then infer the layout of the table automatically. The table-to-text tasks can also benefit from it [69, 94]. In a table-to-text task, given a region of a table as input, a natural language sentence is generated from the selected table region. Additionally, semantic parsing (query answering) on tabular data [95] is also a promising application as seen in TAPAS [71] which uses a close technique to ours. Given tabular data and query (in natural language in many cases) related to the tabular data, a semantic parsing task reads the given tabular data and then answers to the query the most salient data in the tabular data.

Our method helps the Transformer encoder to learn the structure of tabular data adequately, thus improves many table understanding tasks described above.

7.3 Future Work

In this section, we give some future directions for the remaining work in this dissertation.

7.3.1 Directions for Study of Improving our model

Large-Scale Tabular Data Although our proposed model has accomplished a significant improvement from the preliminary LSTM-based model for large-size benchmark datasets, the model is less efficient for large-size benchmarks than small-size ones. Applying variants of Transformer model for a large document or sequence including Longformer [51] possibly solves this challenge. Refer to Section 3.3.3 for more details.

Synthesis of Training Data Our synthesis method of training data proposed in Section 4.6.1.2 relies on uniformly sampling values from all possible parameter ranges. More sophisticated methods including [54], which as described in Section 3.3.4 makes uniform sampling for only given salient variables, can synthesize training data with more real data distribution.

Reinforcement Learning Our ML-based models are trained to maximize the likelihood of reference programs in training data. However, if possible, we want to train the model to maximize the possibility of generating a program that transforms an input example into a valid table or the output example ideally. Reinforcement learning may be a possible solution as described in Section 3.3.5.

Benchmark Datasets We used the benchmark datasets from Foofah [8] to evaluate the performance of our model. The benchmark datasets of Foofah are constructed by broadly collecting from the prior works like Wrangler [16] and Potter’s Wheel [10]. However, we think more heterogeneous benchmark data are needed from various sources including open data from governments for example, to study a PBE system applicable to the real-world scenarios. Collecting and organizing such benchmark datasets is left as a work to be done.

7.3.2 Directions for Study of Managing the PBE system

Extensibility Neural network models can extend their expressiveness by retraining the model with the data including the already trained one and newly added one. However, the cost (time) of retraining a huge training data is considerably high. We expect techniques of fine-tuning or transfer-learning with small additional training data could possibly solve the challenge. Studies on extending our model using these techniques are proposed as future work.

Practical PBE System As we discussed in Section 6.2, many challenges remain to be solved in realizing a practical PBE system. Such system requires not only studies on the program synthesizer, but also, more crucially, those on how to integrate the synthesizer into the system with helpful user interactions and adequate expressive power to eliminate the user's high hurdles to data transformation.

Bibliography

- [1] PowerCenter. <https://www.informatica.com/products/data-integration/powercenter.html>. Last accessed: 2022/08/15.
- [2] IBM DataStage. <https://www.ibm.com/products/datastage>. Last accessed: 2022/08/15.
- [3] ETL vs. Data Preparation: What Are the Differences? <https://www.precisely.com/blog/big-data/etl-data-preparation-differences>. Last accessed: 2022/08/15.
- [4] Enterprise resource planning. https://en.wikipedia.org/wiki/Enterprise_resource_planning. Last accessed: 2022/08/15.
- [5] Customer relationship management. https://en.wikipedia.org/wiki/Customer_relationship_management. Last accessed: 2022/08/15.
- [6] Colin Shearer. The CRISP-DM model: the new blueprint for data mining. *Journal of data warehousing*, 5(4):13–22, 2000.
- [7] New York Times, For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights. <https://www.nytimes.com/2014/08/18/technology/for-big-data-scientists-hurdle-to-insights-is-janitor-work.html>. Last accessed: 2022/08/15.
- [8] Zhongjun 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.
- [9] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 46(1):317–330, 2011.
- [10] Vijayshankar Raman and Joseph M Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, volume 1, pages 381–390, 2001.
- [11] OpenRefine. <https://openrefine.org>. Last accessed: 2022/08/15.

- [12] Trifacta. <https://www.trifacta.com/>. Last accessed: 2022/08/15.
- [13] Tableau prep. <https://www.tableau.com/products/prep>. Last accessed: 2022/08/15.
- [14] Paxata. <https://www.datarobot.com/platform/dataprep/>. Last accessed: 2022/08/15.
- [15] Alteryx. <https://www.alteryx.com/ja>. Last accessed: 2022/08/15.
- [16] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the sigchi conference on human factors in computing systems*, pages 3363–3372, 2011.
- [17] Sumit Gulwani, Alex Polozov, and Rishabh Singh. *Program Synthesis*, volume 4. NOW, August 2017. URL <https://www.microsoft.com/en-us/research/publication/program-synthesis/>.
- [18] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. In *International conference on machine learning*, pages 990–998. PMLR, 2017.
- [19] Emilio Parisotto, Abdel-rahman Mohamed, Rishabh Singh, Lihong Li, Dengyong Zhou, and Pushmeet Kohli. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*, 2016.
- [20] Yoshifumi Ujibashi and Atsuhiko Takasu. Neural Network Approach to Program Synthesis for Tabular Transformation by Example. *IEEE Access*, 10:24864–24876, 2022. doi: 10.1109/ACCESS.2022.3155468.
- [21] Yoshifumi Ujibashi and Atsuhiko Takasu. Two-Dimensional Encoding Method for Neural Synthesis of Tabular Transformation by Example. In *International Conference on Artificial Neural Networks*. Lecture Notes in Computer Science, 2022. (in press).
- [22] Reée J Miller. Using schematically heterogeneous structures. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of data*, pages 189–200, 1998.
- [23] Laks VS Lakshmanan, Fereidoon Sadri, and Iyer N Subramanian. SchemaSQL—a language for interoperability in relational multi-database systems. In *VLDB*, volume 96, pages 239–250. Citeseer, 1996.
- [24] Laks VS Lakshmanan, Fereidoon Sadri, and Subbu N Subramanian. On Efficiently Implementing SchemaSQL on an SQL Database System. In *VLDB*, volume 99, pages 471–182, 1999.

- [25] Weidong Chen, Michael Kifer, and David S Warren. HiLog: A foundation for higher-order logic programming. *The Journal of Logic Programming*, 15(3):187–230, 1993.
- [26] Brian Hempel and Ravi Chugh. Semi-automated svg programming via direct manipulation. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 379–390, 2016.
- [27] Hoang Duong Thien Nguyen, Dawei Qi, Abhik Roychoudhury, and Satish Chandra. Semfix: Program repair via semantic analysis. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 772–781. IEEE, 2013.
- [28] Sumit Gulwani, Ivan Radiček, and Florian Zuleger. Feedback generation for performance problems in introductory programming assignments. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 41–51, 2014.
- [29] Veselin Raychev, Martin Vechev, and Eran Yahav. Code completion with statistical language models. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 419–428, 2014.
- [30] Hongyu Zhang, Anuj Jain, Gaurav Khandelwal, Chandrashekhar Kaushik, Scott Ge, and Wenxiang Hu. Bing developer assistant: improving developer productivity by recommending sample code. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 956–961, 2016.
- [31] Martin Vechev, Eran Yahav, and Greta Yorsh. Abstraction-guided synthesis of synchronization. In *Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 327–338, 2010.
- [32] Rishabh Singh and Sumit Gulwani. Learning semantic string transformations from examples. *arXiv preprint arXiv:1204.6079*, 2012.
- [33] Rishabh Singh and Sumit Gulwani. Transforming spreadsheet data types using examples. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 343–356, 2016.
- [34] Mohammad Raza and Sumit Gulwani. Automated Data Extraction using Predictive Program Synthesis. In *AAAI 2017*. Association for the Advancement of Artificial Intelligence, February 2017. URL <https://www.microsoft.com/en-us/research/publication/automated-data-extraction-using-predictive-program-synthesis/>.

- [35] 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.
- [36] Vu Le and Sumit Gulwani. Flashextract: A framework for data extraction by examples. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 542–553, 2014.
- [37] William R Harris and Sumit Gulwani. Spreadsheet table transformations from examples. *ACM SIGPLAN Notices*, 46(6):317–328, 2011.
- [38] 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.
- [39] Rishabh Singh. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *Proceedings of the VLDB Endowment*, 9(10):816–827, 2016.
- [40] 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.
- [41] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.
- [42] Matej Balog, Rishabh Singh, Petros Maniatis, and Charles Sutton. Neural program synthesis with a differentiable fixer. *arXiv preprint arXiv:2006.10924*, 2020.
- [43] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [44] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [45] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

- [46] 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.
- [47] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [48] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.
- [49] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [50] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [51] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- [52] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.
- [53] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
- [54] 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.
- [55] Judith Clymo, Haik Manukian, Nathanaël Fijalkow, Adrià Gascón, and Brooks Paige. Data generation for neural programming by example. In *International Conference on Artificial Intelligence and Statistics*, pages 3450–3459. PMLR, 2020.
- [56] Alexander Suh and Yuval Timen. Creating synthetic datasets via evolution for neural program synthesis. *arXiv preprint arXiv:2003.10485*, 2020.
- [57] Shivam Handa and Martin Rinard. Program Synthesis Over Noisy Data with Guarantees. *arXiv preprint arXiv:2103.05030*, 2021.
- [58] M Balog, AL Gaunt, M Brockschmidt, S Nowozin, and D Tarlow. DeepCoder: Learning to write programs. In *5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings*, 2019.

- [59] Amit Zohar and Lior Wolf. Automatic Program Synthesis of Long Programs with a Learned Garbage Collector. In *NeurIPS*, 2018.
- [60] Marc’Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.
- [61] Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines-revised. *arXiv preprint arXiv:1505.00521*, 2015.
- [62] Kelvin Guu, Panupong Pasupat, Evan Zheran Liu, and Percy Liang. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. *arXiv preprint arXiv:1704.07926*, 2017.
- [63] Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. *arXiv preprint arXiv:1805.04276*, 2018.
- [64] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.
- [65] Sam Wiseman and Alexander M Rush. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*, 2016.
- [66] Majid Ghasemi Gol, Jay Pujara, and Pedro Szekely. Tabular Cell Classification Using Pre-Trained Cell Embeddings. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 230–239. IEEE, 2019.
- [67] Majid Ghasemi-Gol, Jay Pujara, and Pedro Szekely. Learning cell embeddings for understanding table layouts. *Knowl. Inf. Syst.*, 63(1):39–64, jan 2021.
- [68] Kyosuke Nishida, Kugatsu Sadamitsu, Ryuichiro Higashinaka, and Yoshihiro Matsuo. Understanding the semantic structures of tables with a hybrid deep neural network architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [69] Junwei Bao, Duyu Tang, Nan Duan, Zhao Yan, Yuanhua Lv, Ming Zhou, and Tiejun Zhao. Table-to-Text: Describing Table Region With Natural Language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. doi: 10.1609/aaai.v32i1.11944.
- [70] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

- [71] 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.
- [72] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xi-aohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [73] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*, 2017.
- [74] Daphne Ippolito, Reno Kriz, Maria Kustikova, João Sedoc, and Chris Callison-Burch. Comparison of diverse decoding methods from conditional language models. *arXiv preprint arXiv:1906.06362*, 2019.
- [75] Iliia 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.
- [76] Yilin Yang, Liang Huang, and Mingbo Ma. Breaking the beam search curse: A study of (re-) scoring methods and stopping criteria for neural machine translation. *arXiv preprint arXiv:1808.09582*, 2018.
- [77] Rémi Leblond, Jean-Baptiste Alayrac, Laurent Sifre, Miruna Pislari, Jean-Baptiste Lespiau, Ioannis Antonoglou, Karen Simonyan, and Oriol Vinyals. Machine translation decoding beyond beam search. *arXiv preprint arXiv:2104.05336*, 2021.
- [78] Jiwei Li, Will Monroe, and Dan Jurafsky. A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*, 2016.
- [79] Zelun Wang and Jyh-Charn Liu. Translating math formula images to LaTeX sequences using deep neural networks with sequence-level training. *International Journal on Document Analysis and Recognition (IJDAR)*, 24(1):63–75, 2021.
- [80] 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.
- [81] 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*.

- [82] Raza Mohammad and Gulwani Sumit. Disjunctive Program Synthesis: a Robust Approach to Programming by Example. *2018, Association for the Advancement of Artificial Intelligence*.
- [83] 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. URL <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [84] 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.
- [85] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional LSTM. In *2013 IEEE workshop on automatic speech recognition and understanding*, pages 273–278. IEEE, 2013.
- [86] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [87] Mikaël Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Oleksandr Polozov, Rishabh Singh, Benjamin Zorn, and Sumit Gulwani. User interaction models for disambiguation in programming by example. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 291–301, 2015.
- [88] Susmit Jha, Sumit Gulwani, Sanjit A Seshia, and Ashish Tiwari. Oracle-guided component-based program synthesis. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 215–224. IEEE, 2010.
- [89] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. Interactive query synthesis from input-output examples. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1631–1634, 2017.
- [90] Hila Peleg, Sharon Shoham, and Eran Yahav. Programming Not Only by Example. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 1114–1124, 2018. doi: 10.1145/3180155.3180189.

- [91] Tianyi Zhang, Zhiyang Chen, Yuanli Zhu, Priyan Vaithilingam, Xinyu Wang, and Elena L. Glassman. Interpretable program synthesis. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445646. URL <https://doi.org/10.1145/3411764.3445646>.
- [92] Ian Drosos, Titus Barik, Philip J Guo, Robert DeLine, and Sumit Gulwani. Wrex: A Unified Programming-by-Example Interaction for Synthesizing Readable Code for Data Scientists. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pages 1–12, 2020.
- [93] Elvis Koci, Maik Thiele, Wolfgang Lehner, and Oscar Romero. Table Recognition in Spreadsheets via a Graph Representation. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, pages 139–144, 2018. doi: 10.1109/DAS.2018.48.
- [94] Ankur P Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. ToTTo: A controlled table-to-text generation dataset. *arXiv preprint arXiv:2004.14373*, 2020.
- [95] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. TaBERT: Pretraining for joint understanding of textual and tabular data. *arXiv preprint arXiv:2005.08314*, 2020.