PLOG: Table-to-Logic Pretraining for Logical Table-to-Text Generation

Ao Liu[1]*, Haoyu Dong[2]†, Naoaki Okazaki[1], Shi Han[2], Dongmei Zhang[2]
[1] Tokyo Institute of Technology, [2] Microsoft Research
`liu.ao@nlp.c.titech.ac.jp, okazaki@c.titech.ac.jp`
`{hadong,shihan,dongmeiz}@microsoft.com`

Abstract

Logical table-to-text generation is a task that involves generating logically faithful sentences from tables, which requires models to derive logical-level facts from table records via logical inference. It raises a new challenge on the logical-level content planning of table-to-text models. However, directly learning the logical inference knowledge from table-text pairs is very difficult for neural models because of the ambiguity of natural language and the scarcity of parallel data. Hence even large-scale pretrained language models present low logical fidelity on logical table-to-text. In this work, we propose a Pretrained Logical Form Generator (PLOG) framework to improve generation fidelity. Specifically, PLOG is first pretrained on a table-to-logical-form generation (*table-to-logic*) task, then finetuned on downstream table-to-text tasks. The logical forms are formally defined with unambiguous semantics. Hence we can collect a large amount of accurate logical forms from tables without human annotation. In addition, PLOG can learn logical inference from table-logic pairs much more reliably than from table-text pairs. To evaluate our model, we further collect a controlled logical table-to-text dataset CONTLOG based on an existing dataset. On two benchmarks, LOGICNLG and CONTLOG, PLOG outperforms strong baselines by a large margin on logical fidelity, demonstrating the effectiveness of table-to-logic pretraining.

1 Introduction

Table-to-text generation is a sub-task of data-to-text generation, aiming to generate natural language descriptions from structured tables. There are two main steps to performing table-to-text generation: content planning (selecting table contents and determining the plan to describe them) and surface realization (verbalizing the plan into fluent natural language). Traditional table-to-text systems adopt a pipeline architecture to complete the two procedures with separate modules (Kukich, 1983; McKeown, 1985). Recent work has shown the advantage of using a neural encoder-decoder model to directly generate sentences from the tables, which presents the strong capability to produce fluent and natural text (Wiseman et al., 2017; Nie et al., 2018; Puduppully et al., 2019b). Researchers have also attempted to finetune pretrained language models such as BART (Lewis et al., 2020) and T5 (Raffel et al., 2020) on downstream table-to-text tasks and achieved remarkable success on a broad range of benchmarks (Xie et al., 2022; Kale and Rastogi, 2020).

Previous studies have mainly focused on surface-level realization, i.e., simply restating surface-level facts in natural language (Wiseman et al., 2017; Liu et al., 2018; Puduppully et al., 2019a,b). Recently, logical table-to-text generation (Chen et al., 2020a), i.e., generating textual descriptions that require logical reasoning over surface-level facts in the table, has attracted increasing attention. Logical table-to-text generation poses a new challenge of *logical-level* content planning, requiring models to perform logical inference to derive facts from surface-level table records. End-to-end neural models often suffer from low logical fidelity on this task, i.e., the generated sentences are not logically entailed by the tables despite their reasonable fluency (Chen et al., 2020a, 2021). We attribute this to the fact that the ambiguity of natural language target sentences hinders neural models from learning accurate logical inference from table-text pairs. In addition, the amount of such table-text pairs is limited because of the labor-intensive human annotation for logic-focused descriptions, which also limits the performance of neural models.

To achieve high fidelity of logical-level generation, Chen et al. (2020b) have attempted to annotate

---

Title: dwbl				
branding	callsign	frequency	power	location
dwbl 1242	dwbl	1242khz	20kw	metro manila
mellow 947	dwll	94.7mhz	20kw	metro manila
mellow 887	dyku	88.7mhz	10kw	Iloilo city
mellow 957	dxbl	95.7mhz	10kw	cagayan de oro
mellow 947 davao	dxll	94.7mhz	20kw	davao city

(a) Table-to-Logical-Form Generation

Input: Table + (Highlighted Cells)
Target: greater { hop { filter_eq { all_rows ; callsign ; dwll } ; power } ;
hop { filter_eq { all_rows ; callsign ; dyku} ; power } }

(b) (Controlled) Logical Table-to-Text Generation

Input: Table + (Highlighted Cells)
Target: For dwbl, the power for dwll is higher than the power for dyku.

Pretrained LM → Intermediate Pretraining on Task (a) → PLOG → Finetuning on Task (b) → PLOG(Finetuned)
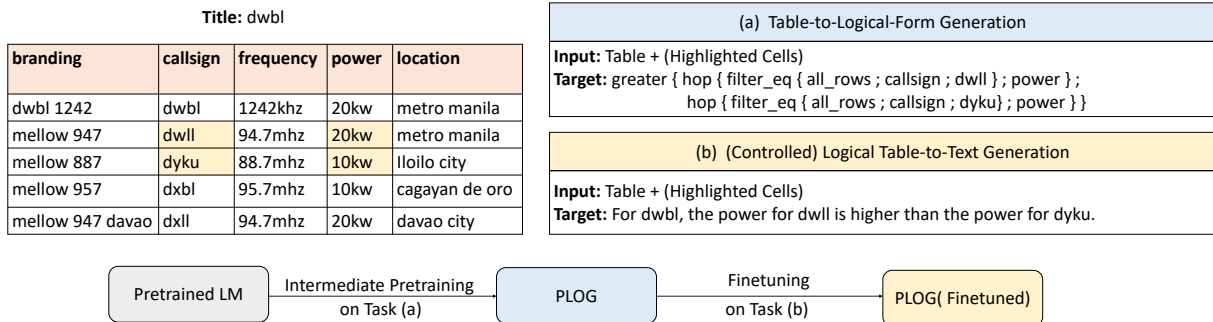
Figure 1: Examples of the tasks and the training procedure of our proposed PLoG model. Task (a) is the table-to-logic pretraining task we propose; task (b) is the downstream logical table-to-text task we target. The yellow-colored table cells are annotated as control features for the CONTLOG task, while for LOGICNLG, such highlighted cells are not available. We collect different table-to-logic datasets for CONTLOG and LOGICNLG separately and perform intermediate pretraining for pretrained language models on the collected data, then finetune the model on the downstream tasks.

logical forms to guide the text generation and proposed a LOGIC2TEXT dataset. With logical forms as mediators conveying accurate logical-level facts, models can focus on surface realization from associated logical forms and achieve high fidelity. However, annotating accurate logical forms for textual descriptions requires intensive human efforts. Moreover, generating from a self-contained logical form is actually a different task from table-to-text generation. Prior studies on this dataset (Liu et al., 2021a; Shu et al., 2021; Xie et al., 2022) mostly focus on converting the logical forms into texts rather than tables into texts.

In this study, we propose a **P**re-trained **LO**gical Form **G**enerator (**PLOG**) model to achieve more faithful logical table-to-text. Specifically, PLoG is first pretrained on a large-scale synthetic corpus of table-to-logical-form generation (*table-to-logic*) to learn how to generate accurate logical forms from tables, then finetuned on downstream table-to-text tasks to transfer the logical inference knowledge learned from pretraining to text generation. Our insights are three-fold. (i) Unlike natural language sentences, logical forms are formally defined with unambiguous semantics; hence it is much easier and more reliable for models to acquire logical inference knowledge via learning from logical form generation. (ii) It is viable to collect large-scale logical form corpora via rule-based sampling over tables without the efforts of human annotators. (iii) Via pretraining on large amounts of table-to-logic data, the proposed model can better understand the table and organize the logical-level content planning, leading to faithful table-to-text generation. Here, we treat logical forms as intermediate mean-

ing representations of logical-level texts, while we do not need them when performing the downstream task. To collect the pretraining data, we propose an execution-guided sampling approach to sample accurate logical forms from tables automatically.

We formulate the pretraining task in the same sequence-to-sequence (seq2seq) generation to achieve smooth transfer learning to the downstream table-to-text task. We adopt several strong pretrained language models, BART and T5, as the backbone models. Because the previous benchmark for logical table-to-text, LOGICNLG, lacks control features, leading to uncontrollable content selection and poor logical fidelity, we collect a a new **CONT**rolled **LOG**ical Natural Language Generation (CONTLOG) dataset as a complementary testbed towards controlled logical table-to-text generation. Specifically, we re-organize the LOGIC2TEXT dataset by detecting highlighted cells based on their annotated logical forms. Figure 1 presents examples of the table-to-logic pretraining task and the (controlled) logical table-to-text task.

On the two benchmarks, LOGICNLG and CONTLOG, PLoG outperforms the strong baselines such as T5 by a large margin on the logical fidelity, demonstrating the effectiveness of table-to-logic pretraining. Human evaluation and analysis experiments further demonstrate that our approach can considerably promote the fidelity of logical table-to-text generation.[1]

---

[1]Code and data will be released at https://github.com/microsoft/PLOG.

2 Related Work

Table-to-Text Generation Early table-to-text generation tasks are limited to surface-level generation with little focus on logical inference (Lebret et al., 2016). LOGICNLG is the first dataset to focus on logical table-to-text generation, with Wikipedia tables and human-annotated logical descriptions. Chen et al. (2021) proposed a deconfounded variational encoder-decoder model to encourage the model to generate non-surface-level predictions; however, the logical reasoning process is not explicitly considered, leading to low fidelity scores on human evaluation. Chen et al. (2020b) proposed to annotate logical forms to guide the generation and released a LOGIC2TEXT dataset. In this work, we focus on direct logical table-to-text generation without any explicit logical forms. Another related line of datasets are ToTTo (Parikh et al., 2020) and HiTab (Cheng et al., 2021), which incorporate highlighted cells to promote controllable generation. The CONTLOG dataset we propose is similar to the task settings of these datasets but differs in that we focus on logical-level generation. At the same time, only a small portion of examples in ToTTo and HiTab involve logical reasoning. ROTOWIRE (Wiseman et al., 2017) and Numeric-NLG (Suadaa et al., 2021) also involve numerical reasoning over table records, while they focus on document-level table summarization instead of sentence generation.

Table Pretraining Table pretraining (Eisenschlos et al., 2020; Liu et al., 2021b; Dong et al., 2022; Iida et al., 2021) has been popular for table understanding tasks such as Table Question Answering (TableQA) (Zhong et al., 2017; Pasupat and Liang, 2015) and Table Fact Verification (TableFV) (Chen et al., 2019). With large-scale pretraining corpora, the table pretraining models can learn a better joint understanding of tabular and textual data through well-defined pretraining objectives. Most table pretraining works are based on table-text corpora, while TAPEX (Liu et al., 2021b) learns from synthetic SQL programs, which is the closest to our work. Specifically, TAPEX is first pretrained on a table-based SQL execution task, where the input is a table and a SQL program, and the output is the answer to the SQL query. Then, the pretrained model can be finetuned on TableQA and TableFV tasks where the input is a table associated with a textual query/statement, and the output is the answer. However, our work differs from TAPEX in that we focus on table-to-text generation, where the input is a structured table and the output is a textual statement of the table contents. Our task requires deriving a complete logical-level fact from the table without the guidance of any query. In addition, our pretraining task also requires generating a self-contained logical form from the table, while TAPEX aims to learn the neural execution of an existing SQL program. Similarly, FLAP (Anonymous, 2021) proposes to enhance the numerical reasoning ability of table-to-text models with an artificial pretraining task. This task is a synthetic QA task similar to TAPEX pretraining.

Another line of related works adopts pretraining techniques to solve the text-to-SQL parsing (Yu et al., 2021; Shi et al., 2021) task, also involving collecting synthetic SQL data and pretraining models on SQL generation tasks. However, text-to-SQL still requires an explicit NL query as the input, which is different from our task. Although table pretraining is popular in table understanding tasks, it has not been well-explored in table-to-text. Previous works on table-to-text tend to directly utilize pretrained language models by flattening structured tables into sequences (Gong et al., 2020; Kale and Rastogi, 2020; Xie et al., 2022). A recent work (Andrejczuk et al., 2022) incorporates structural positional embeddings of tables into T5 (Raffel et al., 2020) and performs intermediate pretraining in a similar way to TAPAS (Eisenschlos et al., 2020). Similarly, PLOG can also be seen as intermediate pretraining of language models for table-to-text generation.

3 Downstream Tasks

In this work, we focus on logical table-to-text. The previous benchmark LOGICNLG aims at generating sentences from a full table without control features, which causes uncontrollable content selection and hinders faithful generation (Chen et al., 2020b). Therefore, we propose a new controlled logical table-to-text dataset CONTLOG as a complementary testbed to LOGICNLG. Inspired by previous studies on controlled table-to-text (Parikh et al., 2020; Cheng et al., 2021), we incorporate highlighted cells as additional supervision signals in CONTLOG (Figure 1) to narrow down the scope of content selection, such that models can focus more on planning and generation.

3.1 CONTLOG Dataset Construction

We reuse the LOGIC2TEXT dataset to build CON-TLOG. In LOGIC2TEXT, there is an annotated logical form for each target sentence. The logical form can convey the accurate logical semantics of the sentence. Hence, we execute the logical forms on the context tables and extract the table cells relevant to the execution process. These cells are also the ones most relevant to the target sentence. Although built upon LOGIC2TEXT, CONTLOG does not contain logical forms because we focus on the direct table-to-text generation. Figure 1 shows an example of CONTLOG.

3.2 Task Formulation

The input of LOGICNLG is a table T with an NL title W. $T = \{T_{ij}|1 \leq i \leq R_T, 1 \leq j \leq R_C\}$, where R_T and R_C are the numbers of rows and columns, respectively, and T_{ij} is the table cell value at row i and column j. Each column also has a column header Col_j. The output is a sentence y. The task objective is to find a model $P(y|T)$ to generate a sentence $\hat{y}$ that is both fluent and logically entailed by the table. In CONTLOG, an additional set of highlighted cells $H = \{(i, j)\}$ are included as a part of the input, where i and j denote the row index and column index of a highlighted cell. The objective thus becomes $P(y|T; H)$.

4 Table-to-Logic Pretraining

Logical table-to-text is difficult mainly because of the ambiguity of natural language sentences. For example, a sentence `Alice was the first player that achieved champion in 2010` has two possible meanings: (1) *Alice got the first champion of 2010*; (2) *Alice became the first champion in history, and this achievement happened in 2010.* This prevents end-to-end neural models from inferring unambiguous logical facts from the table, especially when the parallel data are scarce.

To achieve faithful logical table-to-text generation, we propose a table-to-logic pretraining task that involves generating a logical form from an input table. In this task, the model needs to mine logical-level facts from tables and organize the facts into formally defined meaning representations, i.e., logical forms. Each logical form can be regarded as an abstract content plan of a logical-level description. Therefore, we expect a model to learn logical-level content planning from the pretraining task. We then finetune the model on the downstream table-to-text tasks to generalize the content planning to natural language generation. We formulate our pretraining and downstream tasks as the same seq2seq generation paradigm to realize successful transfer learning.

4.1 Pretraining Task Formulation

The input of the pretraining task is the same (sub-) table as we introduced in Section 3.2, while the target is a logical form instead of a sentence. We follow the same schema in LOGIC2TEXT to define the logical forms used in our task. Each logical form z is the composition of several logical functions. Each function $f_i(arg_1, ...)$ accepts several arguments relevant to the table T. z can be parsed into a tree and executed from bottom to up by a logical form executor. In this process, the execution result of f_i may be fed to its parent function as an argument. The root function always outputs a Boolean value (*true* or *false*) which indicates the factual correctness of z. We select this schema because of its several merits. (1) It is originally designed to represent logical-level textual statements in LOGIC2TEXT, and its definition is close to our downstream tasks. A similar schema has also been used for TableFV tasks (Ou and Liu, 2022; Chen et al., 2019). (2) It covers seven of the most commonly used logic types: `count`, `unique`, `comparative`, `superlative`, `ordinal`, `aggregation` and `majority`. (3) The logical forms can be executed on the tables to evaluate their exact correctness, allowing accurate evaluation of the pretraining task. A detailed description of the logic schema is provided in Appendix B.

4.2 Evaluation Metric of Table-to-Logic

We adopt the execution accuracy of generated logical forms as the evaluation metric for our pretraining task, similar to the setting in text-to-SQL tasks (Zhong et al., 2017). Specifically, a logical form is counted as correct if it can be successfully executed on the input table and returns a Boolean value *True* that indicates the table entails it.

4.3 Pretraining Data Collection

To perform table-to-logic pretraining, we must collect enough paired data of tables and associated logical forms. The formal definition of logical forms allows us to automatically collect a large amount of logical forms from tables via rule-based sampling.
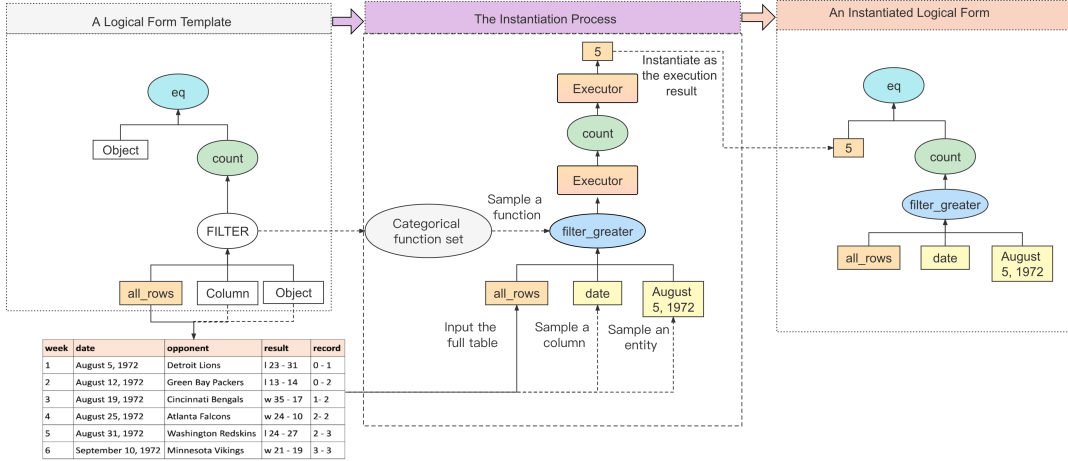
Figure 2: An example of instantiating a logical form template. The colored nodes in the template indicate nodes that do not need instantiation, while the white-background nodes are typed placeholders in the template. The dotted arrows indicate instantiation. We employ instantiation of these white nodes with a bottom-up execution-guided sampling approach. Finally, a logical form instance is obtained. `Column` and `Object` indicate a column header and an object (entity/number), respectively. `FILTER` indicates the category of row-filtering functions. `all_rows` is a special entity to represent the entire table.

Here, we propose instantiating existing logical form templates to sample logical forms similarly to how prior studies collect SQL programs (Zhong et al., 2020; Liu et al., 2021b). Specifically, we extract abstract templates from the logic schema we use. Then we adopt an execution-guided sampling method to instantiate the templates based on the context tables. Our approach has two merits: (1) By utilizing the pre-defined templates, we can control the distribution and diversity of collected logical forms. (2) With the execution-guided sampling, the correctness of the collected logical forms is guaranteed.

Templatization We first extract the templates based on our logic schema. We define them as trees with typed placeholder nodes that need to be instantiated into specific functions or entities. The placeholders include two entity types: `Column` represents a column header and `Object` means either a textual entity or a numerical value. In addition, we categorize some similar functions into smaller groups to obtain several function placeholders, which can reduce the number of templates and simplify the instantiation work. For example, `FILTER` represents a set of row-filtering functions. Table 7 shows the complete list of these function placeholders. The other functions that cannot be categorized need not instantiation. Finally, we obtain 35 templates, an average of 5 for each logic type. More examples of the templates are provided

in Appendix C.

Instantiation We propose an execution-guided bottom-up sampling strategy to instantiate the template trees. An example of template instantiation is depicted in Figure 2. We design rules to instantiate different placeholder nodes via sampling. For example, we uniformly sample a column from the table to instantiate a `Column` placeholder (e.g. `date` in Figure 2). For a function placeholder such as `FILTER`, we sample a specific function from the corresponding category it represents (e.g. `filter_greater` in Figure 2). For each instantiated function node, we execute it, obtain the execution result and feed the result to the parent function as an argument. Hence, the arguments of higher-level functions are guaranteed to be valid. The process lasts from bottom to up until finishing executing the root function node. We provide the detailed sampling rules in Appendix C. For each table, we conduct multiple trials of sampling. At each trial, we randomly select a template based on its distribution in LOGIC2TEXT, and perform the instantiation. Because of the randomness in selecting functions and entities, we can obtain different results from multiple trials. A trial may sometimes fail because of execution errors, but each successful trial will result in a correct logical form. We can perform the sampling as many trials as we want to obtain a large-scale table-to-logic corpus.

Dataset	#tables	#examples (train/val/test)
LOGICNLG	7,392	28,450/4,260/4,305
CONTLOG	5,554	8,566/1,095/1,092
LOGICNLG (pretrain)	5,682	426.6k/3,000/2,997
CONTLOG (pretrain)	4,554	800k/1,500/1,500

Table 1: Statistics of the downstream tasks and their corresponding table-to-logic pretraining data.

Table Source and Data Collection We collect pretraining data separately for the two datasets, LOGICNLG and CONTLOG. For each dataset, we use the tables in its training data as the source tables to avoid potential data leakage. In addition, we remove the sampled logical forms that have appeared in LOGIC2TEXT since they are semantically equal to some of the target sentences in CONTLOG. To evaluate the performance of table-to-logic models and enable the selection of pretrained models, we also split the collected logical forms into train/val/split sets. The statistics of the pretraining data and their corresponding downstream datasets are shown in Table 1. Although we can sample more logical forms with more trials, we find the current pretraining data enough to obtain ideal experimental results.

5 The PLOG Model

In this section, we introduce our proposed model PLOG and how we conduct the seq2seq generation for the pretraining and downstream tasks.

Backbone Model We utilize the same backbone model to address both tasks to achieve the knowledge transfer from the table-to-logic pretraining task to the table-to-text downstream task. Theoretically, any text generation model applies to our task, such as GPT-2 (Radford et al., 2019), BART, and T5. We test different backbone models, including BART-large, T5-base, and T5-large.

Model Input Similarly to prior work on table-to-text generation (Kale and Rastogi, 2020; Parikh et al., 2020), we employ a template-based method to serialize the input table. For the LOGICNLG task, we follow (Chen et al., 2020a) to encode the relevant table columns by concatenating the table cells in row-wise order. For CONTLOG, we only concatenate the highlighted table cells as the input, as suggested by prior works on controlled table-to-text generation (Parikh et al., 2020). This is to avoid the over-length issue with pretrained models

and the negative impacts caused by irrelevant table information.

Numerical Pre-Computation Numerical reasoning is difficult for neural language models, especially aggregation operations (e.g., the average of numerical values) and numerical ranking (e.g., the nth-maximum values of a column). Therefore, we conduct a pre-processing step by pre-computing some potentially useful numerical values. Similar approaches have also been proposed to improve the fidelity in table summarization (Suadaa et al., 2021) and text-to-SQL tasks (Zhao et al., 2022). First, we evaluate each numerical cell's rank in its column (or the scope of highlighted cells) and append this rank to the linearized cell representation. Hence, each table cell T_{ij} can be serialized into a sequence c_{ij} = <cell> T_{ij} <col_header> Col_j </col_header> <row_idx> i </row_idx> <max_rank> r_{ij}^+ </max_rank> <min_rank> r_{ij}^- </min_rank> </cell>, where r_{ij}^+ indicates the rank of T_{ij} in column j in the decreasing order and r_{ij}^- is the rank in the increasing order. The special tokens with angle brackets are used to indicate the structure of the input. In addition, we compute the average and sum of each numerical column in the input (sub-) table, and append two aggregation cell strings c_{sum} and c_{avg} to the flattened table sequence. c_j^{sum}/c_j^{avg} = <sum_cell>/<avg_cell> sum_value/avg_value <col_header> Col_j </col_header> </sum_cell>/</avg_cell>.

Finally, the input (sub-) table is serialized as S = <table> <caption> W </caption> c_1^{sum} c_1^{avg} ... c_{11} c_{12} ... </table>.

Model Output We linearize each logical form z into a string via a pre-order traversal of the logic tree following (Chen et al., 2020b). Special punctuations such as semicolons and braces are used to indicate the structural relationships between functions. For example, the logical form instance in Figure 2 can be linearized into eq { 5 ; count { filter_greater { all_rows ; date ; August 5, 1972 } } }. As for the downstream task, the output becomes a sentence. After pretraining a PLOG model, we directly finetune it on the downstream table-to-text tasks by changing the target from logical forms to sentences.

6 Experiments

6.1 Experimental Settings

Evaluation Metrics Following prior works (Chen et al., 2020a, 2021) on LOG-ICNLG, we evaluate our models on both surface-level matching metrics and logical fidelity scores. Surface-level metrics include BLEU-1/2/3, which are based on n-gram matching between the model generations and gold references. In terms of fidelity scores, prior works adopt SP-Acc and NLI-Acc. For SP-Acc, a sentence is first parsed into a logical program and evaluated as the execution accuracy of the program. NLI-Acc is based on TableBERT, a table-entailment model pretrained on the TabFact dataset (Chen et al., 2019). The model can predict whether a table supports a sentence.

However, these two fidelity metrics are not enough to verify the fidelity: we empirically find that the parsing algorithm for SP-Acc often generates irrelevant logical programs for the sentences, which renders the evaluation inaccurate. In addition, the TableBERT model used for NLI-Acc only achieves 65.1% accuracy on the TabFact dataset, and we find it overly positive about the predictions. To this end, we add two state-of-the-art table-entailment models for evaluation: TAPEX-large (Liu et al., 2021b) and TAPAS-large (Eisenschlos et al., 2020), which achieve 84.2% and 81.0% test accuracy on TabFact, respectively. We name the two metrics as TAPEX-Acc and TAPAS-Acc, respectively. We still evaluate SP-Acc and NLI-Acc to compare our method with previous studies. For CONTLOG, we adopt the evaluation metrics of LOGIC2TEXT: BLEU-4 and ROUGE-1/2/4/L to evaluate surface-level matching, and use TAPEX-Acc and TAPAS-Acc to evaluate the fidelity.

Models for Comparison For LOGICNLG, we compare our method with the following models: **GPT-TabGen (sm)** and **GPT-Coarse-to-Fine (sm)** (Chen et al., 2020a) are two baselines based on pretrained GPT-2; **DCVED+GPT-TabGen** (Chen et al., 2021) is a de-confounded variational model with GPT-TabGen (sm) as the backbone. We also include pretrained **BART-large**, **T5-base** and **T5-large** as the baselines models for both LOGICNLG and CONTLOG, for which we adopt our data pre-processing method introduced in Section 5. Our models are named **PLoG (BART-**

large), **PLoG (T5-base)** and **PLoG (T5-large)** when using different backbones. We adopt the same input serialization strategy with numerical pre-computation for BART, T5, and PLoG models.

Training Details We conduct our main experiments based on Transformers (Wolf et al., 2020) and PyTorch (Paszke et al., 2019). During training, the parameters of embedding layers of models are frozen. During inference, we adopt beam search with beam size 4 for all the experiments. We set the maximum length as 500 and 200 for source and target sequences, respectively. Each experiment was run only once because of the time cost. On LOGICNLG, model selection is based on the BLEU-3 score on the validation set, and on CONT-LOG, it is based on validation BLEU-4 scores. The selection of pretraining checkpoints is based on the Execution Accuracy of generated logical forms on the validation set of pretraining tasks. We provide detailed hyperparameters in Appendix A.

6.2 Automatic Evaluation

LOGICNLG Table 2 presents the results on LOGICNLG. We can observe that the BART and T5 models with our preprocessing strategies outperform all the previous models based on GPT-2 in terms of both surface-level metrics and logical fidelity scores. We also observe that the PLoG models mostly outperform their base models on BLEU scores while they can significantly improve the logical fidelity scores on all the metrics. For example, PLoG (T5-large) improves the TAPEX-Acc and TAPAS-Acc over T5-large by an average of 10% accuracy. However, PLoG (T5-base) achieves lower results on BLEU scores, possibly because of the uncontrollable task setting of LOGICNLG. LOGICNLG does not provide highlighted cells, so the potential space for content selection is usually very large. This makes models very likely to generate faithful sentences that describe different facts/contents from the gold references, causing low BLEU scores. Moreover, BLEU is based on local N-Gram matching which cannot capture the global faithfulness of generated sentences. Therefore, such surface-level metrics may not correlate well with fidelity metrics.

CONTLOG The results on CONTLOG are shown in Table 3. As observed, PLoG models outperform their base counterparts consistently on both

Model	Surface-level Evaluation			Logical Fidelity			
	BLEU-1	BLEU-2	BLEU-3	SP-Acc	NLI-Acc	TAPEX-Acc	TAPAS-Acc
GPT-TabGen (sm)	48.8	27.1	12.6	42.1	68.7	46.0	45.5
GPT-Coarse-to-Fine (sm)	46.6	26.8	13.3	42.7	72.2	44.6	45.6
DCVED + GPT-TabGen	49.5	28.6	15.3	43.9	76.9	–	–
T5-base	**52.6**	**32.6**	**19.3**	48.2	80.4	52.4	56.2
PLoG (T5-base)	51.7	32.3	18.9	**48.9**	**85.5**	**61.7**	**62.3**
T5-large	53.4	**34.1**	20.4	48.4	85.9	65.5	66.2
PLoG (T5-large)	**53.7**	**34.1**	20.4	**54.1**	**89.0**	**75.9**	**76.0**
BART-large	54.5	34.6	20.6	49.6	85.4	63.3	67.1
PLoG (BART-large)	**54.9**	**35.0**	**21.0**	**50.5**	**88.9**	**73.7**	**74.4**

Table 2: The experimental results of different models on the test split of LOGICNLG. For the previous models, we compute the TAPEX-Acc and TAPAS-Acc of the only two that have a released official output. We compare each pair of base and PLoG models and mark the better scores as bold.

Model	Surface-level Evaluation					Logical Fidelity	
	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-4	ROUGE-L	TAPEX-Acc	TAPAS-Acc
T5-base	29.7	60.2	36.4	16.4	50.2	67.4	64.8
PLoG (T5-base)	**30.4**	**61.4**	**37.3**	**16.8**	**51.4**	**78.3**	**74.0**
T5-large	31.2	62.1	37.9	**17.6**	51.4	73.8	71.3
PLoG (T5-large)	**31.7**	**62.3**	**38.3**	**17.6**	**52.0**	**81.9**	**76.8**
BART-large	29.3	59.6	36.0	16.3	48.9	70.3	64.8
PLoG (BART-large)	**32.1**	**63.2**	**39.2**	**18.1**	**53.0**	**85.9**	**82.0**

Table 3: The experimental results of different models on the test split of CONTLOG. We compare each pair of base and PLoG models and mark the better scores as bold.

Model	LOGICNLG		CONTLOG	
	AVG	ACC	AVG	ACC
T5-base	**1.87**	**40.5%**	2.15	58.0%
PLoG (T5-base)	1.84	40.0%	**2.42**	**71.5%**
T5-large	2.21	55.0%	2.42	70.5%
PLoG (T5-large)	**2.41**	**66.0%**	**2.58**	**79.0%**
BART-large	2.05	49.5%	2.12	56.5%
PLoG (BART-large)	**2.39**	**67.5%**	**2.50**	**74.5%**

Table 4: The human evaluation results of different models. AVG is the average score while ACC means the accuracy of logical fidelity. The average inter-annotator agreement is 0.82 when measured by Fleiss' Kappa.

surface-level and logical-level metrics. This suggests that adding highlighted cells to narrow down the scope of content selection is beneficial to more reliable evaluation. In addition, the consistent improvements with different backbone models demonstrates the general effectiveness of our approach.

6.3 Human Evaluation

To further investigate whether the models can generate faithful sentences, we perform a human eval-uation on the outputs of BART, T5, and PLoG models. Specifically, we randomly sample 200 examples from the test set of each dataset. We hire three human annotators to rate each sentence a score in the discrete range between 0 and 3, according to the criteria adopted in (Chen et al., 2020a). Non-sense (0): the sentence does not make sense, and people cannot understand its meaning. Wrong (1): the sentence is overall fluent, but the logic it describes is false. Partially correct (2): the sentence describes multiple facts. At least one of them is correct, but it still contains factual errors. Correct (3): the sentence is of high quality in both fluency and logical correctness. The model names are hidden to the annotators, and we collect their individual results to summarize two scores for each model: (1) the average of their scores on each sampled set; (2) the fidelity accuracy, i.e., the proportion of sentences scored as correct[2]. The evaluation is only based on the context table without considering gold references, because the generated sentences may

[2]We take a vote on the three evaluators' scores, i.e., a sentence is judged as correct if at least two of them give a score of 3.

5538

Model	LogicNLG		ContLog	
	Val	Test	Val	Test
PLoG (BART-large)	49.47	49.85	59.67	61.73
PLoG (T5-base)	90.93	88.86	91.87	92.20
PLoG (T5-large)	93.77	92.23	93.33	93.13

Table 5: Experimental results of different PLoG models on the validation and test sets of table-to-logic generation. The scores are reported as Execution Accuracy.
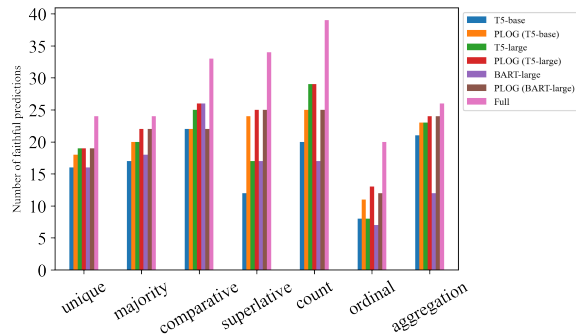

Figure 3: The human evaluation results of different models on the different logic types of ContLog. The y-axis indicates the number of samples scored as correct. *Full* indicates the number of samples of each logic type in the 200 human evaluation samples.

not describe the same fact as the references do but still present high quality in terms of fidelity and fluency.

As shown in Table 4, PLoG (T5-base) outperforms T5-base by a large margin on ContLog while it does not achieve superior results on LogicNLG, which is inconsistent with automatic scores. However, PLoG (T5-large) and PLoG (BART-large) achieve significant improvements over base models on both datasets, showing an improvement consistent with the automatic metrics.

6.4 Table-to-Logic Results

We report the Execution Accuracy of our pretrained models on the table-to-logic pretraining task in Table 5. As shown, PLoG (T5-base) and PLoG (T5-large) present over 90% accuracy in generating correct logical forms, demonstrating that table-to-logic pretraining indeed improves the model's ability to derive accurate logical facts. However, PLoG (BART-large) achieves much lower accuracy. We analyzed the error cases of BART-large and found that over 90% of the errors are caused by logical form parsing errors, i.e., the generated logic string cannot be successfully parsed into a structurally correct logical form tree because of misspelled function names and mismatched brack-

ets. It seems BART-large performs much worse than T5-base and T5-large at learning the structure of logic strings. We suppose that incorporating grammar-guided decoding methods (Wang et al., 2018) may alleviate this problem, which we leave to future work. Surprisingly, this does not affect the performance of PLoG (BART-large) on downstream tasks, showing that the model still acquired beneficial knowledge through the pretraining.

6.5 Analysis on Different Logic Types

In ContLog, each target sentence belongs to a predefined logic type inherited from Logic2Text, allowing us to analyze the performance of models on different logical reasoning types. In Figure 3, we can observe that our PLoG models generally improves the performance of their base models on most logic types, especially on superlative and ordinal. However, we still observe a considerable amount of incorrect generations of all the models, suggesting the potential room for improvement in the future.

7 Conclusion

We proposed a table-to-logic pretraining task to enhance the fidelity of logical table-to-text generation. In addition, we constructed a controlled logical table-to-text generation task by re-purposing an existing dataset. To realize pretraining on large-scale corpora, we proposed an execution-guided sampling scheme to extract accurate logical forms from tables automatically. With table-to-logic pretraining, our table-to-text model could significantly improve logical fidelity. Our work shows a novel way to utilize formal language to promote table-to-text generation, and may be extended to other related areas such as table representation learning.

Limitations

The first limitation of our approach is that it is initialized from pretrained language models such as T5 to inherit the language generation knowledge learned from large-scale text corpora. This requires the input of PLoG to be a text sequence, which may limit the structural encoding of table inputs and logical form outputs. Although it is possible for us to design and pretrain a new model from scratch, the computational cost will be too large. The second limitation is also caused by this. Because we adopt pretrained language models to perform table-to-logic and table-to-text generation, we have to

serialize the input (sub-) tables to fit them in the language model encoder. Therefore, the maximum sequence length of the encoder model limited the size of the input table. To address this, we only input relevant columns or highlighted cells instead of the full table to reduce the input sequence length. However, some potentially useful contextual information in the full table is omitted and may limit the model performance. The third limitation lies in the logical form schema we adopt, which is restricted to the domain of current logical table-to-text datasets. When applying our method to new downstream datasets with unseen logic types, e.g., median, proportion, the current schema should be extended to support the new logic. However, the schema is easy to extend by defining new logical operations as executable functions on tables.

Ethics Statement

This work presents PLOG, a pretrained language model for the research community to study logical table-to-text generation. In addition, we also propose a new dataset CONTLOG for the research of controlled logical table-to-text generation. Our dataset contains Wikipedia tables, annotations (target sentences, meta information such as logic types) and highlighted table cell information. We reuse the tables and annotations of LOGIC2TEXT. LOGIC2TEXT is a public dataset under MIT license. And to obtain the highlighted cell information, we use an automatic method without human annotation. We also use LOGICNLG, another public dataset for experiments, which is also under MIT license. All datasets are in English. In human evaluation, we hire human annotators to evaluate the performance of our models. We recruit 3 graduate students in electrical engineering, computer science, and English majors (1 female and 2 males). Each student is paid $7.8 per hour (above the average local payment of similar jobs).

Acknowledgements

References

Ewa Andrejczuk, Julian Martin Eisenschlos, Francesco Piccinno, Syrine Krichene, and Yasemin Altun. 2022. Table-to-text generation and pre-training with tabt5. *arXiv preprint arXiv:2210.09162*.

Authors Anonymous. 2021. Flap: Table-to-text generation with feature indication and numerical reasoning pretraining. *ACL Rolling Review Nov.*

Wenhu Chen, Jianshu Chen, Yu Su, Zhiyu Chen, and William Yang Wang. 2020a. Logical natural language generation from open-domain tables. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7929–7942.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. Tabfact: A large-scale dataset for table-based fact verification. *arXiv preprint:1909.02164*.

Wenqing Chen, Jidong Tian, Yitian Li, Hao He, and Yaohui Jin. 2021. De-confounded variational encoder-decoder for logical table-to-text generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5532–5542.

Zhiyu Chen, Wenhu Chen, Hanwen Zha, Xiyou Zhou, Yunkai Zhang, Sairam Sundaresan, and William Yang Wang. 2020b. Logic2text: High-fidelity natural language generation from logical forms. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2096–2111.

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2021. Hitab: A hierarchical table dataset for question answering and natural language generation. *arXiv preprint arXiv:2108.06712*.

Haoyu Dong, Zhoujun Cheng, Xinyi He, Mengyu Zhou, Anda Zhou, Fan Zhou, Ao Liu, Shi Han, and Dongmei Zhang. 2022. Table pretraining: A survey on model architectures, pretraining objectives, and downstream tasks. *arXiv preprint arXiv:2201.09745*.

Julian Eisenschlos, Syrine Krichene, and Thomas Mueller. 2020. Understanding tables with intermediate pre-training. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 281–296.

Heng Gong, Yawei Sun, Xiaocheng Feng, Bing Qin, Wei Bi, Xiaojiang Liu, and Ting Liu. 2020. Tablegpt: Few-shot table-to-text generation with table structure reconstruction and content matching. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1978–1988.

Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. Tabbie: Pretrained representations of tabular data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3446–3456.

Mihir Kale and Abhinav Rastogi. 2020. Text-to-text pre-training for data-to-text tasks. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 97–102.

Karen Kukich. 1983. Design of a knowledge-based report generator. In *21st Annual Meeting of the Association for Computational Linguistics*, pages 145–150.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1213.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Ao Liu, Congjian Luo, and Naoaki Okazaki. 2021a. Improving logical-level natural language generation with topic-conditioned data augmentation and logical form generation. *arXiv preprint arXiv:2112.06240*.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian guang Lou. 2021b. Tapex: Table pre-training via learning a neural sql executor.

Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. 2018. Table-to-text generation by structure-aware seq2seq learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Kathleen R McKeown. 1985. Discourse strategies for generating natural-language text. *Artificial intelligence*, 27(1):1–41.

Feng Nie, Jinpeng Wang, Jin-ge Yao, Rong Pan, and Chin-Yew Lin. 2018. Operation-guided neural networks for high fidelity data-to-text generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3879–3889.

Suixin Ou and Yongmei Liu. 2022. Learning to generate programs for table fact verification via structure-aware semantic parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7624–7638, Dublin, Ireland. Association for Computational Linguistics.

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. Totto: A controlled table-to-text generation dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1173–1186.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480.

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. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019a. Data-to-text generation with content selection and planning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6908–6915.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019b. Data-to-text generation with entity modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2023–2035.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(140):1–67.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR.

Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. 2021. Learning contextual representations for semantic parsing with generation-augmented pre-training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13806–13814.

Chang Shu, Yusen Zhang, Xiangyu Dong, Peng Shi, Tao Yu, and Rui Zhang. 2021. Logic-consistency text generation from semantic parses. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4414–4426.

Lya Hulliyyatus Suadaa, Hidetaka Kamigaito, Kotaro Funakoshi, Manabu Okumura, and Hiroya Takamura. 2021. Towards table-to-text generation with numerical reasoning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1451–1465.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*.

Sam Wiseman, Stuart M Shieber, and Alexander M Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I Wang, et al. 2022. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966*.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R Radev, Richard Socher, and Caiming Xiong. 2021. Grappa: Grammar-augmented pre-training for table semantic parsing. In *ICLR*.

Chen Zhao, Yu Su, Adam Pauls, and Emmanouil Antonios Platanios. 2022. Bridging the generalization gap in text-to-sql parsing with schema expansion. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5568–5578.

Victor Zhong, Mike Lewis, Sida I Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

A Experimental Setting Details

The following are the hyperparameters for different model configurations. During finetuning, each pair of base model and the corresponding PLoG model share the same hyperparameters for a fair comparison, while these hyperparameters are tuned only with the base model.

T5-base and **PLoG (T5-base)** : Hyperparamters are the same for both datasets.

- Optimizer: AdamW (Loshchilov and Hutter, 2017).

- Learning rate: 2×10^{-4} for pretraining and 1×10^{-5} for finetuning.

- Batch size: 5 for both pretraining and finetuning.

T5-large and **PLoG (T5-large)** : Hyperparamters are the same for both datasets.

- Optimizer: AdaFactor (Shazeer and Stern, 2018).

- Learning rate: 2×10^{-4} for both pretraining and finetuning.

- Batch size: 10 (2×5 gradient accumulation steps) for both pretraining and finetuning.

BART-large and **PLoG (BART-large)**:

- Optimizer: AdaFactor for both datasets.

- Learning rate: 5×10^{-4} for pretraining on LOGICNLG and 2×10^{-4} on CONTLOG; 2×10^{-4} for fine-tuning on both datasets.

- Batch size: 256 (4×64) for pretraining on LOGICNLG and 32 (4×8) on CONTLOG; 32 (4×8) for fine-tuning on both datasets.

The following is the additional information of each pretrained model.

- T5-base: 220M parameters with 12-layer, 768-hidden-state, 3072 feed-forward hidden-state, 12-heads.

- T5-large: 770M parameters with 24-layer, 1024-hidden-state, 4096 feed-forward hidden-state, 16-heads.

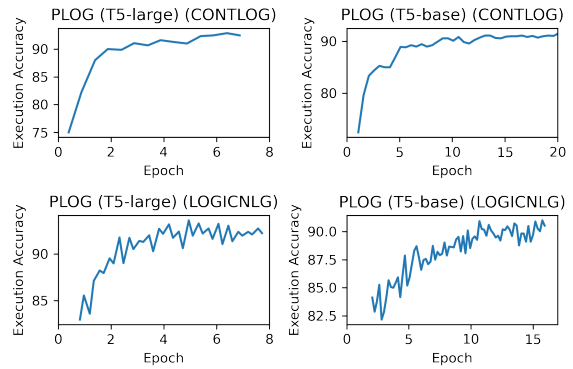- BART-large: 406M parameters with 24-layer, 1024-hidden-state, 16-heads,

Figure 4: Validation results of table-to-logic pretraining with T5-base and T5-large as the backbones. The results of LOGICNLG pretraining and CONTLOG pretraining are shown at different intervals for better illustration. The results within the first 160k steps are not computed.

Pretraining Details We pretrain our models on the collected table-to-logic data and evaluate their Execution Accuracy on the validation set (pretraining corpora) at an interval of a certain number of steps. We take the best pretraining checkpoints to finetune them on downstream tasks. Figure 4 presents the validation results of pretraining during the training process. We can observe that the models achieve higher accuracy when trained for more epochs. The pretraining is very time-consuming because of the large-scale pretraining data and models. For example, it takes approximately 17 hours to train PLoG (T5-base) for one epoch on the CONTLOG pretraining data, while it takes 5 days to train one epoch of PLoG (T5-large). Each experiment was done on a single NVIDIA V100 GPU. We suppose the time cost can be reduced by using more GPU resources.

B Logical Form Schema

LOGIC2TEXT (Chen et al., 2020b) defines 7 logic types, including `count`, `unique`, `comparative`, `superlative`, `ordinal`, `aggregation` and `majority`. For the definitions and examples of these logic types, please refer to the Appendix of (Chen et al., 2020b). In this section, we provide a complete list of the logical functions in Table 6, which we use to define our logical form schema.

C Details of Pretraining Data Collection

Here, we provide more details of the pretraining data collection procedure, including examples of abstract templates and the complete rules for logi-

Name	Arguments	Output	Description
count	view	number	returns the number of rows in the view
only	view	bool	returns whether there is exactly one row in the view
hop	row, header string	object	returns the value under the header column of the row
and	bool, bool	bool	returns the boolean operation result of two arguments
max/min/avg/sum	view, header string	number	returns the max/min/average/sum of the values under the header column
nth_max/nth_min	view, header string	number	returns the n-th max/n-th min of the values under the header column
argmax/argmin	view, header string	row	returns the row with the max/min value in header column
nth_argmax/nth_argmin	view, header string	row	returns the row with the n-th max/min value in header column
eq/not_eq	object, object	bool	returns if the two arguments are equal
round_eq	object, object	bool	returns if the two arguments are roughly equal under certain tolerance
greater/less	object, object	bool	returns if argument 1 is greater/less than argument 2
diff	object, object	object	returns the difference between two arguments
filter_eq/not_eq	view, header string, object	view	returns the subview whose values under the header column is equal/not equal to argument 3
filter_greater/less	view, header string, object	view	returns the subview whose values under the header column is greater/less than argument 3
filter_greater_eq /less_eq	view, header string, object	view	returns the subview whose values under the header column is greater/less or equal than argument 3
filter_all	view, header string	view	returns the view itself for the case of describing the whole table
all_eq/not_eq	view, header string, object	bool	returns whether all the values under the header column are equal/not equal to argument 3
all_greater/less	view, header string, object	bool	returns whether all the values under the header column are greater/less than argument 3
all_greater_eq/less_eq	view, header string, object	bool	returns whether all the values under the header column are greater/less or equal to argument 3
most_eq/not_eq	view, header string, object	bool	returns whether most of the values under the header column are equal/not equal to argument 3
most_greater/less	view, header string, object	bool	returns whether most of the values under the header column are greater/less than argument 3
most_greater_eq/less_eq	view, header string, object	bool	returns whether most of the values under the header column are greater/less or equal to argument 3

Table 6: Function definitions of the logic schema borrowed from (Chen et al., 2020b).

Category	Function
FILTER	filter_eq, filter_not_eq, filter_greater, ...
SUPERLATIVE	max, min
ORDINAL	nth_max, nth_min
SUPERARG	argmax, argmin
ORDARG	nth_argmax, nth_argmin
COMPARE	greater, less, eq, not_eq
MAJORITY	all_eq, all_not_eq, most_eq, all_greater, ...
AGGREGATE	avg, sum

Table 7: The categorized functions for template abstraction. Functions in the same category have the same argument definitions.

cal form sampling. Table 7 lists the function-type placeholders.

Templates We provide in Table 8 some examples of our logical form templates. These examples are all based on the example table in Figure 1.

Instantiation Here we provide the main rules we design for instantiating a logical form template by sampling from a table.

1. For placeholder type `Column`, we randomly sample a column header from the current input (sub-) table.

2. For placeholder type `Object`, the instantiation depends on the parent function node of this placeholder. If the function node is `only` or belongs to the category FILTER or MAJORITY, the placeholder is instantiated as a sampled value from a certain column of the

current input (sub-) table. Otherwise, if the function node is `eq`, this placeholder is instantiated as the execution result of its brother node. This is to guarantee the correctness of equality judgements.

3. The instantiation of a function-type placeholder depends on its function category, as listed in Table 7. If the placeholder belongs to the function category COMPARE or MAJORITY, we choose the specific function name based on the real relationships among its arguments. For example, the arguments of COMPARE functions are two objects whose relationship (equal, greater, less, etc.) can be pre-computed. Hence we can determine the actual function based on this relationship. If the placeholder belongs to another category, the function can be uniformly sampled from the function set.

D Case Study

We further conduct a case study by showing some qualitative examples of model generations. As presented in Figure 5, PLOG models can generate logically correct sentences with complex reasoning while the base models often fail to describe correct facts for the table.

Logic Type		Examples
Count	Template	eq { count { [FILTER] { all_rows ; [Column 1] ; [Object 1] } } ; [Object 2] }
	Instance	eq { count { filter_eq { all_rows ; power ; 20kw } } ; 3 }
	Explanation	In dwbl, there are 3 brandings with power 20kw.
Comparative	Template	[COMPARE] { hop { [FILTER] { all_rows ; [Column 1] ; [Object 1] } ; [Column 2] } ; hop { [FILTER] { all_rows ; [Column 1] ; [Object 2] } ; [Column 2] } }
	Instance	greater { hop { filter_eq { all_rows ; callsign ; dwbl } ; power } ; hop { filter_eq { all_rows ; callsign ; dyku } ; power } }
	Explanation	The callsign dwbl has a greater power than dyku.
Unique	Template	only { [FILTER] { all_rows ; [Column 1] ; [Object 1] } }
	Instance	only { filter_eq { all_rows ; location ; iloilo city } }
	Explanation	Only one brand is located in iloilo city.
Superlative	Template	eq { hop { [SUPERARG] { all_rows ; [Column 1] } ; [Column 2] } ; [Object 1] }
	Instance	eq { hop { argmin { all_rows ; frequency } ; callsign } ; dyku }
	Explanation	The callsign dyku has the lowest frequency.
Ordinal	Template	eq { hop { [ORDARG] { all_rows ; [Column 1] ; [Object 1] } ; [Column 2] } ; [Object 2] }
	Instance	eq { hop { nth_argmax { all_rows ; frequency ; 2 } ; branding } ; mellow 957 }
	Explanation	Mellow 957 is the brand that has the second highest frequency.
Majority	Template	[MAJORITY] { all_rows ; [Column 1] ; [Object 1] }
	Instance	most_less { all_rows ; frequency ; 1242khz }
	Explanation	Most of the brands have a frequency lower than 1242khz.
Aggregation	Template	round_eq { [AGGREGATE] { all_rows ; [Column 1] } ; [Object 1] }
	Instance	round_eq { avg { all_rows ; power } ; 16kw }
	Explanation	The average power of all the brands is 16kw.

Table 8: Examples of logical form sampling. For each logic type, we show an example of the abstract template, an instance sampled from the table in Figure 1, and a textual explanation of the instance.

Title: coast province

code	county	former province	area (km 2)	population census 2009	capital
1	mombasa	coast	212.5	939370	mombasa (city)
2	kwale	coast	8270.3	649931	kwale
3	kilifi	coast	12245.9	1109735	kilifi
4	tana river	coast	35375.8	240075	hola
5	lamu	coast	6497.7	101539	lamu
6	taita - taveta	coast	17083.9	284657	voi

Gold: Tana river is the county in coast province with the highest area in square kilometers.

T5-base: Kwale county has the highest area of any county in the coast province.
PLOG (T5-base): Tana river has the most square kilometers of area in the coast province.

T5-large: Mombasa county has the highest area in the coast province.
PLOG (T5-large): Tana river is the county with the highest area (km 2) in the coast province.

BART-base: Kilifi has the highest area (km 2) in the coast province .
PLOG (BART-base) : Kilifi county has the third largest area of any of the counties in coast province .

(a) An example of CONTLOG. The yellow-colored cells are highlighted cells.

Title: 2010 – 11 new jersey nets season

game	team	location attendance
75	philadelphia	wells fargo center 16695
76	miami	prudential center 18711
77	minnesota	prudential center 13461
78	detroit	the palace of auburn hills 14554
79	new york	prudential center 18023
80	toronto	air canada centre 17755
81	charlotte	prudential center 13853

Gold: The most highly attended Attendance game was against Miami at the Prudential Center.

T5-base: The New Jersey Net played against New York at Prudential Center 18023 and Air Canada Centre 17755.
PLOG (T5-base): The lowest Attendance for a game was 13461, against Minnesota.

T5-large: The game with the highest Attendance was against Minnesota at Prudential Center.
PLOG (T5-large): The game with the highest Attendance was a home game versus Miami.

BART-large: the New Jersey Net played against Miami at the Prudential Center and won by 18711 point.
PLOG (BART-large): Miami had the highest Attendance at the Prudential Center.

(b) An example of LOGICNLG. Some irrelevant columns are removed for illustration.

Figure 5: Qualitative examples of two datasets. The red color indicates incorrect facts while the blue color indicates correct facts.