論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Towards Faithful Logical Natural Language Generation from Structured Data
著者(和文)	LIU AO
Author(English)	Ao Liu
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第12583号, 授与年月日:2023年9月22日, 学位の種別:課程博士, 審査員:岡崎 直観,篠田 浩一,徳永 健伸,林 晋平,宮崎 純
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第12583号, Conferred date:2023/9/22, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Doctoral Dissertation

Towards Faithful Logical Natural Language Generation from Structured Data

Ao Liu

August 13, 2023

Artificial Intelligence Course
Department of Computer Science
School of Computing
Tokyo Institute of Technology

A Doctoral Dissertation
submitted to School of Computing,
Tokyo Institute of Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Ao Liu

Thesis Committee:

Professor Naoaki Okazaki	(Supervisor)
Professor Takenobu Tokunaga	(Co-supervisor)
Professor Koichi Shinoda	(Co-supervisor)
Professor Jun Miyazaki	(Co-supervisor)
Associate Professor Shinpei Hayashi	(Co-supervisor)

Towards Faithful Logical Natural Language Generation from Structured Data *

Ao Liu

Abstract

Natural Language Generation (NLG) is the process of generating natural language outputs from certain inputs, which is a vital component of human-machine interfaces, such as modern Artificial Intelligence (AI) assistants. A subfield of NLG is data-to-text, which aims to enhance the accessibility of complex data for humans by generating understandable natural language descriptions from structured tables. Traditionally, most methodologies have focused on the generation of surface-level descriptions of superficial facts explicitly present in structured data. However, a recent trend, known as logical natural language generation (logical NLG), is aiming to describe inferred facts in the data, which requires logical reasoning over structure data.

While deep learning models, particularly large-scale pretrained language models, have shown great promise in terms of fluency due to the wealth of contextual knowledge learned from large-scale corpora, they fail to address a critical aspect of logical NLG: faithfulness. Faithfulness, in this context, refers to the factual consistency of the generated content with the given data. Logical NLG poses challenges for maintaining faithfulness as it requires logical reasoning during content planning and non-trivial surface realization to transform an inferred fact into natural language. The joint learning of reasoning and generation is challenging for end-to-end neural models. The scarcity of logical NLG data exacerbates this difficulty.

This dissertation proposes two novel approaches to enhance the faithfulness of logical NLG from structured tables by leveraging logical forms (LFs) as additional

*Doctoral Dissertation, School of Computing
Tokyo Institute of Technology, August 13, 2023.

resources to address the primary challenges of logical NLG. The first methodology treats LFs as explicit control features for generating the target textual description. The LF control allows the model to primarily concentrate on surface realization, as the LF provides a complete logical fact for description, thereby reducing the need for logical reasoning over the table. However, training a model with explicit LF control requires large-scale parallel data of LFs and texts, which is limited due to the labor-intensive human annotation required. To address the issue of data scarcity, this proposed approach aims to augment parallel LF-to-text data based on the abundant information in tables. The method consists of two stages for data augmentation. The first part generates additional LFs and texts from existing tables using pretrained table-to-text and table-to-LF models, incorporating a predefined logic type as a prompt to foster topic-diversified augmented data. Because these LFs and texts are independently augmented and not aligned, the second part employs a semi-supervised dual learning framework for creating pseudo-parallel data from the new LFs and texts, thereby improving the training of the NLG model. Experimental results have proven the effectiveness of this approach.

The second approach, in contrast, addresses table-to-text generation without explicit LF control, thus requiring the model to engage in logical reasoning. This approach only augments LFs as pretraining resources for task transfer learning, proposing a pretraining task for generating correct LFs from a table. The model is first pretrained on this task and then finetuned on the logical NLG task. LFs represent unambiguous semantics, enabling models to learn more reliable logical reasoning from LF generation and transfer the knowledge to downstream NLG tasks. Furthermore, a rule-based sampling method is proposed to collect a large amount of LFs for pretraining to alleviate the data scarcity problem. To better assess logical NLG, we introduce a controlled logical NLG benchmark with annotated highlighted table cells serving as control features. Several pretrained language models were selected as the basis to evaluate our method, and experimental results have demonstrated that the pretraining improves the models' logical reasoning abilities over tables and enhances their faithfulness on two logical NLG benchmarks.

We also conducted a comparative analysis of the difficulties associated with using different model interfaces and control features of logical NLG. Experimental results showed that using explicit LFs eased the generation and led to the best performance of the same model. The no-control interface with only a table as the

input is the most challenging, where the model suffers from uncontrollability and low fidelity. The interface with highlighted cells offers moderate controllability and difficulty. The two proposed approaches offer different perspectives on the utilization of LFs, thus adopting different interfaces. The first approach adopts the interface with explicit LFs, focusing on the surface realization part of logical NLG. The second approach, on the other hand, utilizes LFs implicitly and adopts the interface with highlighted cells or only tables, tackling a more challenging task setting of logical NLG. Unlike the first approach, it jointly solves logical reasoning and surface realization, with a focus on the former step.

Keywords:

Logical NLG, Faithfulness, Data Augmentation, Pretrained Language Models, Data-to-Text, Semi-Supervised Learning, Table Pretraining

Acknowledgements

Foremost, I would like to thank my advisor, Prof. Naoaki Okazaki. His invaluable suggestions and guidance have been instrumental in shaping my research and this dissertation. Our regular research meetings have made me think of novel ideas, make progress on my research and obtain useful advice. I am also grateful for the opportunity he offered me to work as a research assistant (RA) in the lab, which provided crucial financial support for me. Moreover, I truly appreciate his support for my personal matters, such as fellowship applications, internships, and job hunting. My Ph.D. study has been a pleasant journey because of him.

I would also like to convey my deepest gratitude to my dissertation committee members: Prof. Takenobu Tokunaga, Prof. Koichi Shinoda, Prof. Jun Miyazaki, and Prof. Shinpei Hayashi. Their careful examination of my dissertation, insightful comments, and constant encouragement have been invaluable and their questions have encouraged me to widen my research from varied perspectives. Furthermore, I want to especially thank Prof. Shinpei Hayashi, who, as an academic advisor, generously devoted his time to regular meetings with me to offer valuable advice on my academic life.

My sincere thanks also go to the secretaries in our lab, Ms. Yukiko Konishi, Ms. Yuko Unzai and Ms. Naoko Furuya, for their support on lab matters, including the submission of RA reports every month, the reimbursement of travel expenses and publication fees, etc. Ms. Yukiko Konishi's cheerful encouragement during my dissertation writing phase has been especially inspiring.

Throughout my graduate school, I am very thankful to all my friends, collaborators and colleagues for the insightful discussions about my research, life and career. Here I want to thank An Wang, Zhishen Yang, Yasheng Sun, Sakae Mizuki, Youmi Ma, Marco Cognetta, Yu Pan, Yidong Wang and Yong Dai.

I am also grateful to Haoyu Dong at Microsoft Research Asia and Yang Zhao at IBM Research Tokyo for their wholehearted support during my research internships.

Finally, I want to thank my family for their unflagging love and support throughout my life; this dissertation is simply impossible without them.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Natural Language Generation from Structured Data

Human communication and expression of information predominantly occur through the conduit of natural language. The ongoing research in the realm of Artificial Intelligence (AI), with a specific emphasis on Natural Language Processing (NLP), is primarily directed towards understanding and emulating human language, thereby contributing to the development of AI applications that are eased by the human-machine interface. Current AI assistants, for instance, engage in dialogues with human users via natural language, comprehend user queries, and generate contextual responses in a natural language form that is readily accessible to the users. Central to these human-machine interfaces is the concept of Natural Language Generation (NLG), a crucial facet of NLP. In a widely-referenced survey pertaining to NLG [99], it is delineated as the generation of human language from some **non-linguistic** representation of information. Although there is consensus that the end product of NLG should be text, the requirement of the input being non-linguistic is yet to be universally accepted [107]. In a broader context, NLG is also associated with generation tasks such as Text Summarization [1], Dialogue Generation [124], and Machine Translation [5], where the input is a text. In this dissertation, our focus is on NLG from non-linguistic inputs, specifically, structured data, also known as data-to-text generation [99, 93], in contrast to text-to-text generation. For clarity, we exclusively use the term NLG to denote data-to-text generation in this dissertation.

The main objective of data-to-text generation is to formulate natural language descriptions for the given input data. The input can be composed of table records [87], spreadsheets [22], key-value lists [60], expert knowledge bases [83], and numerical data, like time series data from stock markets [106]. However,

the other data formats require minimal modifications to be transformed into structured tables, which is why the terms "table-to-text" and "data-to-text" are sometimes used interchangeably [54]. Data-to-text generation serves two primary purposes. Firstly, it enhances the accessibility of structured data to human users. The variations in domains, formats, and sizes of structured data restrict its availability to human experts. Converting or summarizing information in structured data to human language can broaden its accessibility to the general public. The second application is automatic content creation, such as writing weather reports, sports reports, or financial reports which usually entail tremendous human effort and are sometimes beyond the scope of limited human labor. Automating textual report generation for structured data can alleviate the burden on human writers. This is also connected to the recent trend of Artificial Intelligence Generated Content (AIGC), referring to content generation by AIs in accordance with user requirements.

The earlier research on NLG [58, 76, 99] utilized a pipeline approach, consisting of a series of submodules, each addressing a different aspect of NLG. These modules were constructed based on rules and templates, embodying the linguistic expertise of human professionals. They generally include processes such as deciding the information to incorporate in the text, organizing the order and structure of the information or facts, creating abstract templates as a plan, applying syntax rules to generate surface forms, and so forth. Nowadays, data-driven approaches, particularly those that are neural network-based, have taken center stage in this domain [93, 94, 52, 21, 82]. The advent of graphical processing units (GPUs) and parallel computing has enabled the training of large-scale deep learning models on extensive datasets, thereby directly learning the end-to-end mapping from input data to output text. More recently, the development of large-scale pretrained language models (PLMs) [96, 98, 62] has occurred, which are capable of being utilized in various downstream tasks and applications of NLP. The current cutting-edge approaches finetune PLMs on downstream NLG datasets by transforming structured inputs into serialized textual sequences, easily attaining high generation fluency and informativeness [52, 43]. In recent times, significant attention has been drawn towards large language models such as GPT-3/GPT-4 [12, 84], along with their application in dialogue systems like ChatGPT[1]. These models have

[1]https://openai.com/blog/chatgpt

garnered recognition due to their remarkable text generation capabilities and interactive conversational abilities. However, these models are often hidden from the public or too large for finetuning. Therefore, an intriguing approach [139] that has emerged involves directly prompting these models with minimal or even no additional training, relying on the inherent contextual knowledge within the language models.

1.2 Logical Natural Language Generation

The majority of existing NLG research primarily focuses on generating surface-level descriptions of table records. Essentially, these descriptions simply rephrase the superficial facts presented in the table without adding any additional insights or summaries. For instance, Lebret et al. [60] focused on creating biographical text from Wikipedia infoboxes. This required the model to select and rearrange key-value pair attributes into coherent sentences. Similarly, some researchers [36] have developed NLG models that translate structured meaning representations into sentences. Some studies [126, 22, 87] have explored the use of large multi-row tables as the generation context, which demands selecting and describing salient and interesting information from these tables. Nonetheless, these methods largely focus on producing superficial descriptions that do not incorporate logical inference. However, in real-world scenarios, there is often a need for higher-level summarization or conclusions drawn from the table information, as opposed to merely repeating the facts present in the table records [112]. Such a need implies the necessity of building advanced NLG systems that can produce more human-like natural language descriptions through logical inference.

Recent research trends are moving towards this direction to inject logical inference in text generation. Korn et al. [57], for example, have explored the automatic generation of interesting trivia for superlative relational tables to enhance user engagement in search apps like Google. While the aim is to generate fun facts involving superlative information, the facts are explicitly provided in the superlative tables, so no inference is required during the generation. Another line of work [126, 39, 64] has developed data-to-text systems for sports summary generation in the domain of basketball games, which sometimes requires inferring non-trivial facts via arithmetic reasoning such as addition or comparison. Parikh et al. [87], Su et al. [115] have explored the data-to-text generation

Figure 1.1: Example of table-to-text generation showing a comparison between surface-level generation and logical NLG, borrowed from [17]. Logical NLG requires logical inference over superficial facts in the table such that the generated sentence is logically entailed by the table. The logical NLG sentences are also examples of the LogicNLG dataset [17].

in open-domain tables, which sometimes also involves reasoning for the generation. However, these studies do not specifically solve the requirement of logical inference in NLG.

Chen et al. [17] have emphasized the importance of logical inference in NLG, and formally defined the logical natural language generation (*logical NLG*) task, the definition of which is generating textual descriptions that can be *logically entailed* by the given table. The table is viewed as a premise, and the target text is a conclusion inferred from that premise. The term "logical" in logical NLG refers to the logical reasoning process required to derive the fact included in the conclusion. The definition of logical NLG implies the following two steps:

1. *Logical reasoning*, which involves inferring a correct fact from the superficial information in the table. For example, given the superficial facts "Gold Medal of Canada is 3", "Gold Medal of Mexico is 2" and "Gold Medal of Colombia is 1" present in the table in Figure 1.1, one can infer a fact:

"Gold Medal of Canada is the most". The scope of logic in logical NLG is specifically focused on inferences involving symbolic operations over the tables [88], such as superlative operations like max/min, aggregation functions like sum/mean, comparison operations like same/different, and counting operations like total/only. This differentiates logical NLG from other reasoning or inference-related NLP tasks like natural language inference [11] and commonsense reasoning [117].

2. *Surface realization*, to describe the inferred fact in natural language. The inferred fact is, in essence, an abstract concept about what to generate and can be represented in different means other than natural language. Conveying the inferred fact with accurate, human-like language realizations is the focus of this step.

From the perspective of traditional NLG [99], the logical reasoning step is similar to the content planning process, i.e., deciding an abstract plan about what information to include in the final generation. Logical NLG differs from surface-level generation by placing a higher-level demand to perform logical reasoning over structured data during content planning. Figure 1.1 illustrates a comparison between surface-level generation and logical NLG.

Logical NLG is valuable for real-world applications for two reasons. Firstly, the ability to generate high-level descriptions is a key characteristic of human-machine interfaces, such as dialogue systems. Instead of generating mundane statements, humans expect an NLG system to provide more engaging and human-like responses. Secondly, the generated content can be more beneficial for human needs. For example, a data analyst might expect an NLG system to provide data reports with rich data analysis, which necessitates logical inference over data records. Therefore, logical NLG has attracted more and more attention in the NLG community. Some recent studies have explored generating analytical descriptions for scientific tables [116, 79] in academic papers or generating stock market comments from time series data [81, 51], which involves rich logical reasoning.

1.3 Problems of Logical Natural Language Generation

Early rule-based NLG methods have explored adding inference in the generation, incorporating deep reasoning over the user's intentions, the communication goal, and context data [112]. In this process, domain knowledge was necessary for designing the rules governing content determination. Contemporary NLG research, however, has discarded these rule-based approaches due to their limited flexibility and scalability. Instead, a strong emphasis is now placed on deep learning approaches, particularly pretrained language models, which demonstrate an adept ability to produce fluent, coherent sentences. Nonetheless, an essential property of NLG that poses a considerable challenge is **faithfulness**, or **fidelity**, denoting the factual consistency of the generated content with the input data.

Several studies have endeavored to address *surface-level fidelity* issue in NLG by incorporating verbatim input elements such as entities and relationships in tables. Employing techniques like the copy mechanism [37, 93, 94, 21], content selection, and planning [93, 94], these works have proven effective in generating factually accurate and coherent descriptions. Nonetheless, the advent of logical NLG presents a *logical fidelity* challenge, as the generated descriptions encompass inferred facts not explicitly provided in the input. Merely reusing the input content does not ensure the factual accuracy of logical descriptions, as these logical facts necessitate accurate inference from the tables. Thus, to achieve logically faithful generation, a model must perform logical reasoning and surface realization inherent to logical NLG.

Previous research [17] managed to accomplish satisfactory generation fluency by finetuning PLMs on the training data. Nonetheless, even state-of-the-art PLMs have achieved low faithfulness in the logical NLG task. These models can formulate fluent and varied statements that depict seemingly rational facts, but these facts may contradict the given table's content with minimal linguistic variation. For instance, two statements, *The Canada team scored more points than the Brazil team* and *The Canada team scored fewer points than the Brazil team* are only different in one word, but their correctness can be opposite. Correctly understanding symbolic operations such as comparison is the key to faithful generation.

The primary hurdle lies in training end-to-end neural models from the struc-

tured data and the target natural language statement. The intermediate procedure of logical NLG, encompassing content planning and surface realization, cannot be readily learned from mere table-text pairs. This is due to the ambiguity and diversity of natural language that hinder models from acquiring reliable logical inference knowledge, i.e., the composition of symbolic operations. For instance, a sentence such as *Alice was the first player that achieved champion in 2010* harbors two potential interpretations:

1. *Alice secured the first championship of 2010*;
2. *Alice became the first champion in history, an accomplishment that occurred in 2010.*

Another example is a sentence *Canada performed the best on the leaderboard.* The word *best* can be interpreted as the highest rank (inferred by an argmin operation over a *rank* column) or the most scores (obtained by an argmax operation over a *score* column). The same fact can be conveyed by different language representations, such as

1. *Canada ranked the first in this game.*
2. *Canada scored the most goals.*

Such linguistic ambiguity and diversity prevent end-to-end neural models from learning unambiguous logical reasoning, especially when the available training data is scarce. Regrettably, annotating logical NLG data is labor-intensive. Despite the availability of copious tabular data resources, logical descriptions necessitate human annotation. The data scarcity issue is particularly acute under real-world conditions, where the acquisition of domain-specific data is crucial.

Several efforts have been made to mitigate the low faithfulness issue in logical NLG. The strategy involves introducing mediators to bridge the gap between the input data and the text. Chen et al. [17] proposed a coarse-to-fine approach to generate a masked textual template of the target sentence, followed by generating the complete sentence by filling in the blanks of the template. Chen et al. [19] proposed a latent variable model which takes selected entities as latent mediators. However, the performances of these methods remain constrained due to the ambiguity and diversity of natural language references.

Jhamtani and Berg-Kirkpatrick [51] introduced logical forms (programs) as latent variables to improve the faithfulness of the generating captions from time series data. A program composed of logical operations, such as begin and increase, is sampled from the latent space, and the generation is conditioned on the programs with truth values. However, there is no program annotated for supervised training; instead, an exhaustive search of all candidate programs is conducted for inference. This is acceptable for the simple time series data with a small space of programs, but the expansion to other domains is difficult due to the potential feasibility issue in searching programs. Chen et al. [20] put forward a similar idea to use logical forms (LFs) as mediators, addressing the NLG from open-domain multi-row tables. They have defined the LFs to cover a large scope of logic types, including the most common operations over database-like tables. These LFs are annotated as meaning representations (MRs) of the target text, clearly defining compound logical facts and reasoning procedures. Figure 1.2 shows examples of the logical forms. Instead of treating LFs as latent variables, the authors utilized LFs as explicit inputs alongside the tables to control the generation process. Given such ground-truth LFs, the generation model can avoid error-prone logical reasoning and shift its focus towards surface realization. Nevertheless, the training of such models depends on the availability of large-scale parallel data of <LF, text> pairs, and obtaining such aligned LFs necessitates additional human annotation effort. Several subsequent studies [110, 68] also follow the explicit usage of LFs and proposed data augmentation techniques to augment such parallel data. However, these data augmentation methods are based on the perturbation of existing LFs, neglecting the faithfulness of the augmented data to the table. These studies hence focus more on the surface realization from LFs to texts rather than logical reasoning.

The previous studies have shown the difficulty of achieving faithfulness when building NLG systems involving logical inference. While introducing explicit mediators such as LFs can alleviate the difficulty of learning logical reasoning for end-to-end models, it exacerbates the data scarcity problem due to the annotation of ground-truth LFs. Therefore, building reliable logical NLG systems is still an unsolved challenge. In this study, we aim to bridge this gap by addressing the problems of (1) end-to-end learning of both logical reasoning and surface realization and (2) data scarcity. To this end, we treat LFs as auxiliary mediators and propose effective data augmentation approaches to improve the faithfulness

of neural-based logical NLG systems.

1.4 Solutions

This study proposes two approaches to augment and utilize LFs. Previous studies [110, 68] has developed data augmentation of LFs via perturbating existing LFs; however, such techniques can only produce semantically similar LFs and often renders the augmented LFs factually incorrect. We take a different view to augment data from tables. The large search space of logical relationships between entities in tables allows for inferring various additional logical facts as LFs. While being rooted in the same principle to augment LFs from tables, the two approaches investigate distinct usage of the auxiliary LFs: the first one adopts LFs as explicit control for generation, showing an **explicit** usage of LFs; the second one explores an **implicit** usage of LFs for transfer learning.

Notably, our approach primarily follows the previous work on logical NLG from open-domain Wikipedia tables [17, 20, 110, 125, 68], which specifies the type and domain of the tables addressed by the proposed methods. Here, we introduce the scope of our research by discussing these specifications.

- Firstly, such tables typically resemble a database–multi-row structures with each row representing an item and each column denoting an attribute. Examples of these attributes could be an athlete's place of birth, a country's area, or a tournament's date. These tables allow for logical reasoning operations similar to the scope of SQL functions over databases, which we will define in Section 2.2.1. The homogeneous data types (mostly numerical and categorical values) and the simple structure of these tables simplify the challenges associated with logical reasoning and text generation. In contrast, realistic tables are more diverse and may contain heterogeneous data types or hierarchical structures. We do not consider the more complex data resources in this study; however, our approaches can serve as the basis for future work to deal with such table types.

- Secondly, the target tables in this study are open-domain Wikipedia tables, a popular resource for developing data-to-text datasets [87, 17, 20]. Some other studies [116, 79] have addressed the tables from scientific articles; however, these scientific tables focus on a specific scenario of comparing,

ranking, or summarizing numerical values of experimental results from scientific papers. We adopt Wikipedia tables to cover a wider range of domains and reasoning types.

- In addition, our research does not involve the generation based on multiple tables or multi-modal context, which requires further adaptations of our methods.

- Furthermore, we address the problem of generating a single sentence rather than a full document. This allows us to separate the problem of logical inference from other linguistic complexities inherent in documents, providing a focused study. However, sentence-level generation can be viewed as the basis of document-level generation, with an added need for document planning. The investigation of these more complex cases is reserved for future work.

This study starts with a data augmentation approach for parallel <LF, text> data, adhering to existing work [20, 110] that employs LFs as explicit supervisory signals directing the generation process. It is observed that multi-row tables frequently encompass an abundance of logical facts, owing to the flexible combination of diverse logical operations in various sequences to construct distinctive logical facts. Each LF or textual description essentially constitutes a semantic representation of such a logical fact. Hence, we propose a method to extract additional logical facts as LFs or texts from tables, which can be converted into parallel instances to facilitate the training process. Our strategy comprises two stages: (1) augmenting unpaired LFs and texts, and (2) their transformation into pseudo-parallel data via semi-supervised learning. Specifically, the first stage incorporates a topic-conditioned data augmentation (TopicDA) strategy where we train two auxiliary data augmentation (DA) models, table-to-logic and table-to-text, which independently generate an LF or a text for a specified table. We utilize an available parallel corpus LOGIC2TEXT [20] (consisting of <table, LF, text> instances) to establish the supervisory data essential for training these DA models. Multiple logic topics are employed to control and prompt generation within the DA models, culminating in logic-diversified augmented data.

Subsequently, during the second stage, the augmented LFs and texts are converted into pseudo-parallel data to enhance training via a semi-supervised dual

learning framework. Specifically, we introduce the dual task of LOGIC2TEXT, namely, logical form generation (LG), aiming to generate an LF given a text and a context table. Dual models are jointly trained for these dual tasks. A LOGIC2TEXT model is trained to generate pseudo-text data for the augmented LFs, thereby creating pseudo-parallel data of <LF, text> that can be employed to boost the training of the LG model. Conversely, the LG model can benefit the LOGIC2TEXT model by generating pseudo-parallel data for the augmented texts. Concurrently, the models apply self-training to augment themselves. During the first stage, we use the existing tables in the training data for data augmentation, thereby avoiding the use of external resources. Experimental results on the LOGIC2TEXT dataset have demonstrated the superiority of the proposed approach compared to strong supervised baselines. Further improvements can be anticipated when using external tabular data.

While proven effective, the first approach addresses logical NLG under the assumption that such parallel data of LFs and texts are readily available. Additionally, this explicit usage of LFs separates the two stages of logical NLG, namely, logical reasoning and language realization, shifting the emphasis towards the latter stage. In this scenario, the NLG model may lack the capability for logical reasoning as the fact is explicitly provided in the LF input.

By contrast, the second proposed approach augments LFs for implicit task transfer learning and circumvents the explicit alignment of LFs and texts. This approach follows existing work addressing the logical NLG task as a direct table-to-text task [17, 125]. Unlike the first approach, we aim to use LFs as implicit mediators to help models acquire better logical reasoning knowledge. Specifically, we propose an auxiliary pretraining task, table-to-logical-form (*table-to-logic*) generation, which aims to generate a correct LF from a provided table. We introduce the PLOG framework, wherein a model is initially pretrained on the table-to-logic task, then finetuned on downstream logical table-to-text tasks. Table-to-logic is akin to table-to-text in that the input is solely a table, with the only variation being the generation target, i.e., LFs vs texts. Compared to texts, LFs are formal languages defined with unambiguous semantics and low language diversity; hence, we anticipate the model to gain more reliable logical reasoning by learning to generate LFs and transferring this knowledge to downstream tasks. In addition, we can acquire a large-scale LF corpus with automatic data augmentation methods rather than human efforts, for which we propose

an execution-guided sampling method to extract correct LFs from tables. We adopt several different PLMs as the base models for continual pretraining on the collected table-to-logic corpus. We consider the LogicNLG [17] dataset the benchmark for evaluation; however, it lacks control features and results in uncontrollable content selection, which may impede evaluation and undermine faithfulness. We further construct a **Cont**rolled **Log**ical Natural Language Generation (ContLog) dataset based on the existing Logic2Text dataset, which employs highlighted table cells to control the generation. The experiments on three PLMs, BART-large [62], T5-base and T5-large [98], and two benchmarks, LogicNLG and ContLog, demonstrate the effectiveness of our approach in enhancing faithfulness. Both quantitative and qualitative experiments demonstrate that the models can learn superior logical reasoning via transfer learning from the augmented LFs.

While sharing the same idea of treating LFs as additional resources, the two approaches differ in the focused task settings and model interfaces. The first approach addressed the interface with explicit LF control, which relies on the readily available parallel corpus of LFs and texts; however, this may not be the case in real-world scenarios. Moreover, this interface implies non-trivial specification of a complete LF which may not be feasible for end users. The second approach, on the other hand, utilizes LFs implicitly and follows the interfaces with only tables or with highlighted cells as the control features. Unlike the first approach, it enables joint learning of logical reasoning and surface realization, with a focus on the former. We also conduct a comparative experiment on the performance of the same model and dataset but with different interfaces, including using explicit LFs, highlighted cells, and a no-control interface with only a table as the input. Experimental results show that the explicit usage of LFs leads to the lowest difficulty and best performance on faithfulness. The highlighted cells pose moderate difficulty and the no-control interface is the most difficult due to the uncontrollable content selection. This suggests that the second approach addresses a more difficult task setting compared to the first approach.

The two methodologies also differ in their approaches to tackling data scarcity with augmented data. The first is based on semi-supervised learning with complex training strategies and has high time complexity for each round of training. The second adopts transfer learning and keeps the finetuning on NLG data as usual supervised training. The method is hence simple yet effective and sidesteps the

	Approach 1	Approach 2
Task settings / Model interfaces	table+LF→text	table→text table+highlighted cells→text
Evaluation benchmarks	LOGIC2TEXT	LOGICNLG, CONTLOG
Types of augmented data	LFs & texts	only LFs
Usage of augmented data	Semi-supervised learning	Task transfer learning
Requirement of <LF, text> parallel data	Yes	No

Table 1.1: The comparison between the two approaches.

extra training time during finetuning. Table 1.1 offers a comparison between the two methods.

To summarize, this study proposes two approaches to enhance logical NLG by augmenting LFs, demonstrating their effectiveness in distinct task settings.

1.5 Contributions

The contributions of this study are as follows.

1. This study introduces an innovative, automatic model-based data augmentation method that leverages pretrained language models to augment parallel logical form (LF) and text data from tables. This methodology effectively addresses the data scarcity issue encountered when using explicit LFs as a means of controlling logical NLG.

2. This study has developed an automatic rule-based LF augmentation method for sampling LFs from tables, which facilitates the collection of large-scale, high-quality LFs. The devised sampling algorithm and the resulting corpus could potentially benefit a broader domain of table-related tasks.

3. It proposes the first table pretraining framework to improve logical NLG. Using augmented LFs as pretraining data, the logical reasoning ability of pretrained language models is improved in data-to-text generation, as demonstrated by quantitative and qualitative experiments.

4. It is the first study that showed the effectiveness of pretraining on formal language to improve natural language generation from structured data.

5. This work provides a comprehensive comparative analysis on different task settings of logical NLG, delivering valuable insights on the design of user interfaces for real-world applications.

Through these contributions, this study paves the way for more robust and reliable logical NLG models and promotes further studies of NLG systems in practical applications that include inference in the generation.

1.6 Thesis Outline

In the rest of this thesis, Chapter 2 describes the related work for logical NLG. This includes the prior works on data-to-text generation and the recent works on logical NLG. The related work is described in two veins, datasets, and approaches. The datasets part, in particular, provides the details of the two datasets used in this study. In addition, Chapter 2 introduces the preliminary knowledge relevant to this study, including the definition of logical forms, the evaluation metrics of logical NLG, and the pretrained language models which serve as the base models in this study.

Chapter 3 introduces the first approach to improve the faithfulness of logical NLG. This approach is focused on the task setting of logical NLG that uses LFs as explicit control, and aims to augment parallel <LF, text> data via semi-supervised learning.

Chapter 4 introduces the second proposed approach, which augments only LFs and utilizes them implicitly via task transfer learning. This chapter also discusses the difficulties of different model interfaces using different control features, that is, strong controls like LFs, softer controls like highlighted cells, and no control.

Chapter 5 concludes this thesis by summarizing the proposed methods. It also discusses the limitations of this work and the future directions.

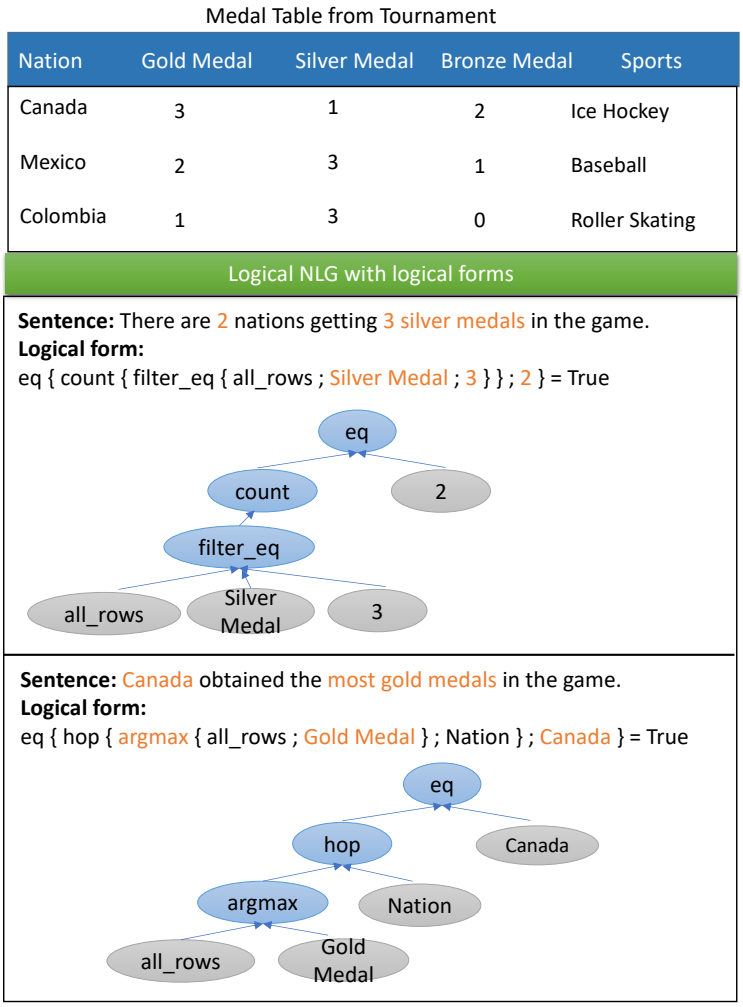


Figure 1.2: Examples of logical forms aligned with target texts introduced in [20]. The two logical forms belong to logic types *count* and *superlative*. Each logical form can be represented as a tree-structured program with function nodes like `filter_eq` and argument nodes like `all_rows`. A string form of the LF can be obtained by pre-order traversal of the program tree, with punctuations like '{' and ';' to indicate the structure. We provide a complete definition of such logical forms in Section 2.2.1.

2 Related Work and Preliminary

In this chapter, we introduce the related work for logical natural language generation, including a comprehensive review of the general data-to-text generation and the vein of NLG systems from traditional rule-based systems to the current mainstream neural systems. The other related work about data augmentation, semi-supervised learning, and table pretraining are given in the corresponding chapters of the proposed approaches. This chapter also introduces the necessary preliminary knowledge for understanding this research.

2.1 Related Work on Logical NLG

2.1.1 Datasets

Data-to-text tasks [58, 76] aim to generate a textual target text y from a source input x, which usually involves structured data, such as tables. This practice of automated transformation of data into user-friendly descriptions has numerous business applications, with robo-journalism [61, 27] being one of the most publicly known examples. It involves generating news articles automatically based on the incoming data from news events. Real-world commercial frameworks that leverage such systems include ArriaNLG[1], Narrative Science[2], and Automated Insights[3], etc. Depending on the type of input data, data-to-text systems can generate various outputs, including:

1. sports reports such as summaries for soccer games [102, 118, 7];
2. weather forecasts [101, 10, 66];

[1] `https://www.arria.com/`
[2] `https://narrativescience.com/`
[3] `https://automatedinsights.com/`

3. financial reports such as stock market summaries [4, 81];

4. clinical reports such as summaries of patient information [91, 92].

Many datasets have been introduced to support data-to-text research. These datasets vary in terms of the output length. Some focus on generating longer text, usually a document-level summary of the data, such as the ROTOWIRE dataset [126], which generates a summary for NBA basketball game records, with an average length of 337 words. On the other hand, most datasets concentrate on generating sentences. For example, TOTTO [87] requires generating a sentence-level description for certain salient regions in a table.

Early data-to-text datasets tackled MR-to-text generation within a limited domain. Datasets like BAGEL [75] and SF [123] align an MR with a single sentence in the restaurant domain. WeatherGov [66] pairs weather statistics with short weather forecasting texts. E2E [83] is a large-scale MR-to-text dataset with 50K MR-sentence pairs within the domain of restaurants. Unlike earlier datasets, E2E necessitates content selection from the provided attributes in the MR. WikiBio [60] pairs Wikipedia infoboxes with a single-sentence biography text taken from the corresponding Wikipedia article's first sentence. The data in E2E and WikiBio are attribute-value pairs and can be viewed as single-row tables. Some datasets, like TOTTO, include larger tables. TOTTO is a controlled table-to-text generation dataset with open-domain multi-row Wikipedia tables. However, these datasets consist of mostly verbalized summaries of the data records, providing surface-level descriptions of the data. While ROTOWIRE involves logical inference for sports statistics, this inference is infrequent and confined to the NBA basketball domain (e.g., double-double, smash, triple-double).

Chen et al. [17] created a LogicNLG dataset as a benchmark for logical NLG based on an existing table-based fact verification dataset TabFact [16]. Chen et al. [20] have proposed another logical NLG dataset, Logic2Text, consisting of triplets of <table, logical form, text>. Both datasets contain only logical descriptions without simple surface-level descriptions, making them focused benchmarks for logical NLG. Therefore, we choose these benchmarks as the basis for all the experiments in this dissertation. Unlike LogicNLG, Logic2Text is annotated with an aligned logical form for each textual statement. In addition, Logic2Text can be formulated as a table-aware logical form to text generation

task since the logical forms strictly control the generation. Next, we provide more details about the two datasets.

LogicNLG is a table-to-text dataset consisting of 37K descriptions paired with 7.3K open-domain tables from Wikipedia. The domains of tables include Sports, Politics, Entertainment, Celebrity and Science. An example of LogicNLG is shown in Figure 1.1.

Each table is paired with 5 different descriptions encompassing diverse types of logical inference. Since LogicNLG is automatically built from TabFact, its annotation process mirrors that of TabFact. TabFact is collected by asking human annotators to write statements about Wikipedia tables, each statement labeled either `entailed` or `refuted`, denoting whether the table factually supports it or not. The annotations are done through two separate channels: the *simple* channel and the *complex* channel. The *simple* statements correspond to a single row/record of tables and can be verified without complex logical inference. The *complex* ones involve multiple rows of tables and require compound higher-order logical semantics like summary, average, argmax, argmin, difference, etc. The annotators are encouraged to produce statements with their own wisdom. Hence further quality control is necessary. The collected data are filtered by human works based on the following criteria: a statement is accepted only if (1) it does not contain grammatical errors or typos, (2) it does not contain vague expressions (many, few, etc.), and (3) it should be explicitly supported or refuted by the table without additional knowledge. The negative (`refuted`) statements are collected via manual rewriting of entailed statements. To focus on the complex logical inference, LogicNLG is built by selecting only the *complex* and `entailed` statements in TabFact.

Logic2Text is another logical NLG dataset collected through human annotation. Its tables are collected from WikiTables [88], the same resource as LogicNLG. Similarly to the collection process of TabFact, the tables are filtered to include only those of less than 20 rows and 10 columns. Therefore, tables in Logic2Text and LogicNLG are, to some extent overlapped. Unlike LogicNLG, Logic2Text includes aligned logical forms (LFs) for each textual statement. The task setting of Logic2Text assumes the existence of these LFs, such that the generation of the target text is conditioned on both the context table and the LF. Therefore, it is a highly-controllable NLG dataset. Nonetheless, it is sometimes used as an uncontrolled NLG dataset by discarding the LFs [125].

An example of LOGIC2TEXT is shown in Figure 1.2.

The LFs and texts in LOGIC2TEXT are annotated based on seven strictly-defined logic types, including *count*, *superlative*, *comparative*, *aggregation*, *majority*, *unique*, and *ordinal*. These logic types pre-define the possible logical semantics involved. For example, the logic type superlative is defined as "describing the maximum or minimum value in a column, with the scope of all table rows or a subset of rows." The human workers are first asked to compose statements of a certain logic type, describing interesting facts in the table. The collected statements must pass a quality control stage similar to the settings of collecting TabFact. An additional requirement is that the statements must be consistent with the assigned logic types. After collecting the statements, the core step is constructing the LF for each statement. This is realized via conversations with human workers. For each logic type, an LF structure prototype is designed by the authors. Then, human workers answer a series of template questions to derive the complete LF based on the prototype. After obtaining the LFs, multiple verification steps are performed to guarantee factual correctness (i.e., whether the table supports it) and semantic correctness (i.e., whether it is semantically equivalent to the corresponding statement). The scale of LOGIC2TEXT is relatively small, including 10K examples and 5.5K tables. The LFs used in LOGIC2TEXT is a Python-like program that can be easily converted to other types of formal language. It is an extension of the logical form schema used in [16] to represent the semantics of statements in TabFact. Therefore, it is now known that LOGIC2TEXT and LOGICNLG share a similar scope of logical reasoning types. We will describe in Section 2.2.1 the definition and grammar of the LFs since they form the basis of the methods proposed in this research.

2.1.2 Approaches

Data-to-text systems are usually designed to address the two challenges: *what to say*, the selection and organization of the partial information in the data that should be included in the under-constructed text, and *how to say* it, the surface realization of the generation, involving how to represent the selected information in natural language. Traditional approaches usually handle the two challenges with sophisticatedly designed separate modules. Reiter and Dale [99] proposed a pipeline architecture involving six stages: content determination, text struc-

turing, sentence aggregation, lexicalization, referring expression generation, and linguistic realization. In the case of document-level generation, the pipeline can also be summarized into text planning (or document planning, or macro planning), sentence planning (or micro planning), and surface realization [100]. For sentence-level generation, the pipeline can be roughly divided into content planning and surface realization [80].

The early conventional systems design rule-based hand-built modules to realize the pipeline [58, 76, 99]. Some researchers have resorted to theories of discourse reference [47, 104], generic planners [24], or schemas [30] to build the content planner. These planners are manually constructed via rules. Mellish et al. [78], Karamanis [53] adopt a generate-then-rank approach to select from a set of generated content plans based on a ranking function. There are also many rule-based systems designed for surface realization [8, 34]. Another branch is data-driven approaches that adopt statistical methods to learn the system from input and output data. In these studies, some researchers still resort to a pipeline with individual components [56, 77, 6, 31], while another line of work addresses the generation problem in an end-to-end manner [9, 66].

With the development of neural network models and deep learning, neural network models trained on large-scale parallel data have shown superiority in generating fluent and diverse natural language. Most neural approaches follow the sequence-to-sequence (Seq2Seq) approach with LSTM [46] or Transformer [119] as the backbone models. A line of studies still follows the traditional pipeline methods to modularize each step of NLG, i.e., content planning and surface realization. Ma et al. [74] first adopt a classifier to select the content to describe, then resorts to a neural model to perform the sentence planning and language realization in one pass. Castro Ferreira et al. [13] propose to include the multiple steps proposed in [99], including discourse ordering, lexicalization, and regular expression generation. Su et al. [115] propose a plan-then-generate model which consists of a content planner and a sequence generator. The content planner first predicts the favorable content plan in the form of an ordered list of tokens; then, the sequence generator predicts the final output conditioned on both the initial input and the content plan.

Most neural approaches, however, are built as end-to-end models to tackle the entire NLG procedure directly. They do not explicitly separate the stages but rely on representation learning. Due to the structural gap between the input

and output, such models are prone to generate unfaithful hallucinations. Various strategies have been proposed to improve the faithfulness of such models, including using soft templates [127, 133] or retrieved prototypes [65, 114] to control the generation, adopting copy mechanism to reuse the verbatim in the structured data [37, 93, 94, 21], and enhancing the structural awareness [70, 23] of models. The majority of these approaches still rely on specific encoders to encode the structured input. In recent years, with the rapid development of pretrained language models (PLMs) [25, 26, 96, 98, 62], researchers have tried to adapt PLMs to data-to-text tasks and achieved remarkable success in the generation quality. To fit the structured input into PLMs, a common technique is to linearize the structured data into sequences based on structure-aware templates such that the data-to-text generation can be solved as a text-to-text problem [52, 43]. PLMs prevail in the current studies because of the strong capability to understand structured data and generate fluent natural language. They can achieve high fidelity on some easier tasks that do not require much inference, like E2E and TOTTO [52, 43]; however, they are still prone to unfaithful hallucinations on logical NLG benchmarks.

Chen et al. [17] propose a BERT-TabGen and a GPT-TabGen model based on BERT [26] and GPT-2 [96], respectively, to solve logical NLG by linearizing the input table into a paragraph. While achieving good fluency, these models fail to generate logically faithful sentences in most cases, leading to a fidelity score of slightly over 20% reported by human annotation. Chen et al. [19] claim that the pretrained models may not correctly learn the logical inference because the surface-level spurious correlations are easier to capture than the causal relationships between the table and the text. They propose a de-confounded variational encoder-decoder (DCVAE) model to de-confound the negative effects of such spurious correlations, which is realized by introducing selected entities as latent mediators between the table and text. Despite improvements over the GPT-TabGen baseline, DCVAE still achieves a low logical fidelity score on the logical NLG datasets. Chen et al. [19] claim the performance might be improved by enhancing the mediators, such as using logical forms, which are limited due to the costly manual annotation.

Chen et al. [20] address logical NLG by using LFs. They show that neural models can achieve much higher fidelity scores on the LOGIC2TEXT dataset with manually annotated LFs as explicit controls, compared with directly modeling

the table-to-text generation. Some later studies follow this vein on using LFs to control the generation. Since LFs explicitly define the logical facts to be generated, the difficulty of learning logical relationships between the table and text is reduced. Therefore, this line of work focuses more on the logical consistency between the LF and the text. Xie et al. [130] propose the UNIFIEDSKG framework that unifies 21 structural knowledge grounding tasks into a text-to-text format, including LOGIC2TEXT. They have shown the advantage of finetuning T5 [98] on LOGIC2TEXT and the benefits of multi-task prefix-tuning that enables task knowledge transfer from other related tasks. Shu et al. [111] propose an iterative training framework SNOWBALL, which augments the training set recursively with a generator model and includes an evaluator model to control the quality of the augmented data. The data augmentation is based on a rule-based perturbation of the correct LFs in LOGIC2TEXT, and a pretrained generator generates corresponding silver target sentences for such LFs. Liu et al. [68] question that models may learn spurious correlations between the table headers and logical operators in LFs, which may damage the robustness of models. They propose that training the models with an additional set of counterfactual samples can improve the robustness of models on unseen data. Similarly to [111], the counterfactual data is also constructed via automatic LF perturbation, which replaces a specific table header in the LFs with another one. The target sentences are not generated by neural models; instead, they are automatically constructed by replacing the same table headers as perturbed in the LFs. Although these methods [111, 68] adopt data augmentation to improve the performance of models, their methods only perturb existing LFs without considering the factual correctness of the augmented LFs, i.e., whether they are entailed by specific tables. The augmented data do not convey factual information of real tables; hence these studies are less relevant to the logical NLG task, but more focused on translating logical forms into texts. The approaches proposed in this thesis also seek to augment LFs, but they are based on a different view to augment data from real tables.

2.2 Preliminary

This section describes the necessary preliminary knowledge for this study. First, we describe the formal definition and grammar of the logical form used in this study, which also defines the scope of logical reasoning in logical NLG. Then, we

Stat ::= and *Stat Stat* | only *View* | eq *Obj Obj* | not_eq *Obj Obj* | greater *Obj Obj* | less *Obj Obj*
| round_eq *Obj Obj* | all_eq *View Col Obj* | all_not_eq *View Col Obj*
| all_greater *View Col Obj* | all_less *View Col Obj* | all_greater_eq *View Col Obj*
| all_less_eq *View Col Obj* | most_eq *View Col Obj* | most_not_eq *View Col Obj*
| most_greater *View Col Obj* | most_less *View Col Obj*
| most_greater_eq *View Col Obj* | most_less_eq *View Col Obj*

View ::= *all_rows* | filter_eq *View Col Obj* | filter_not_eq *View Col Obj*
| filter_greater *View Col Obj* | filter_less *View Col Obj*
| filter_greater_eq *View Col Obj* | filter_less_eq *View Col Obj* | filter_all *View Col*

Num ::= count *View* | sum *View Col* | avg *View Col* | max *View Col* | min *View Col*
| nth_max *View Col Ord* | nth_min *View Col Ord*

Obj ::= hop *Row Col* | diff *Obj Obj* | *Num* | *Val*

Row ::= argmax *View Col* | argmin *View Col*
| nth_argmax *View Col Ord* | nth_argmin *View Ord*

Col ::= *column*
Ord ::= *order*
Val ::= *value*

Figure 2.1: The logical form grammar.

introduce the evaluation metrics used for logical NLG. Finally, we describe the basis of the pretrained language models used in this study, which serve as the base models in the proprosed approaches.

2.2.1 The Logical Form Definition

The logical forms used in this study are formally defined meaning representations to represent a statement about a database-like table, originally defined by [16]. An execution engine is also implemented, which supports the execution of logical forms as programs. Chen et al. [20] have extended the function set of [16], which we adopt in this study. As depicted in Figure 1.2, a logical form is a tree structure whose nodes represent either a logical function/operation or an argument/operand of an operation. The root of the tree is a Boolean operation, as the logical forms represent descriptive statements. The output of the Boolean operation, *true* or *false*, can indicate the factual correctness of the logical form.

The Grammar

The logical form is a meaning representation language. We follow [2] to define the context-free grammar in Figure 2.1 with slight modifications. The non-terminals in Figure 2.1 are defined as follows.

- *Stat* indicates the Boolean results of statements, including results returned by comparative operations such as equal (`eq`), greater than (`greater`), less than (`less`), not equal (`not_eq`), most equal (`most_eq`) and all equal (`all_eq`). It can also produce a joint operation (`and`).
- *Col* means a specific column header in the context table.
- *Val* indicates a specific value, either an entity in the table or an arbitrary value used in comparative operations or filters (e.g., `filter_eq`).
- *View* represents a set of rows of the table. It is either the full table (denoted by *all_rows*) or a partial table selected by filters.
- *Num* refers to numerical values returned by certain operations such as maximum (`max`), minimum (`min`), counting (`count`), summary (`sum`) and average (`avg`).
- *Row* means a single row selected by superlative operations such as `argmax` and `argmin`.
- *Obj* represents objects returned by the `hop` operation that locates a value in a certain Col and a Row. An Obj is either a Num or a Val and can be applied to the difference (`diff`) operation.
- *Ord* is a numerical order specified for ordinal operations such as `nth_max`, `nth_min`, `nth_argmax` and `nth_argmin`, which refer to the logic of selecting the n-th maximum or minimum values from a View and a column.

The terminals include the following categories.

- Operation names, including `and`, `sum`, `avg`, `count`, `filter_eq`, etc. These names correspond to executable logical operations that can be verified in the execution engine. Each operation accepts several operands that follow the operation name, abstracted by *View*, *Obj*, and *Stat*.

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

Table 2.1: Definitions of the logical operations, borrowed from [20].

- Column names, denoted by *column* in Figure 2.1. This represents specific column headers in tables.
- Values, denoted by *value* in Figure 2.1. This represents specific entities in tables or arbitrary values, which could be any words based on the context.
- Order numbers, denoted by *order* in Figure 2.1. This represents specific order numbers used for ordinal operations.
- A special terminal *all_rows* is defined to denote the full table.

The non-terminals, as can be observed, are mostly argument or output types of logical operations, which do not explicitly appear in a produced logical form via grammar. The terminals, such as logical operations and columns, form the final logical form tree, as shown in Figure 1.2. Therefore, the logical forms explicitly include all the involved operations in a compound logical fact. It is noteworthy that a logical form generated by the grammar in Figure 2.1 does not have separators between the operation names and arguments. However, we follow the previous practice in [20] to add punctuation characters '{', '}' and ';' to separate the operations and arguments in the string-form of LFs, as shown in Figure 1.2. This is to represent the inherent structure in the logical form tree.

The Scope of Logic

The definition of logical forms covers a variety of logic types, including count, comparative, superlative, aggregation, unique, ordinal, and majority, as categorized in [20]. Each logic type corresponds to a specific category of logical operations, while some operations are shared by all the logic types, such as filter operations and the `hop` operation. Table 2.1 provides the full list of all the logical operations used in this study as well as their detailed explanations.

2.2.2 Evaluation Metrics for Logical NLG

The evaluation of logical NLG is different from traditional data-to-text tasks, since faithfulness is particularly important and hard-to-measure in logical NLG. Specifically, we adopt the following automatic evaluation metrics, which can be divided into *surface-level* metrics and *logical fidelity* metrics.

Surface-level Metrics

Here, *surface-level* metrics refer to the commonly used n-gram matching metrics, such as BLEU [86] and ROUGE [67]. These metrics evaluate the comparison between the surface forms of the generated output and the gold reference text. As the comparison is purely based on local word-level overlaps, such metrics cannot accurately represent the faithfulness of the generated outputs.

BLEU (Bilingual Evaluation Understudy) compares the candidate output and the reference text by measuring the precision of n-grams, i.e., how often the sequence of words (n-grams) in the candidate output also appears in the reference. It also includes a brevity penalty (BP) to penalize shorter outputs. BLEU measures a modified precision score from unigram to 4-grams and computes a weighted average of these scores for the final score. Formally, the modified precision is computed by

$$p_n = \frac{\sum_{\text{n-gram} \in y} \text{Count}_{\text{clip}}(\text{n-gram})}{\sum_{\text{n-gram} \in y} \text{Count}(\text{n-gram})}, \tag{2.1}$$

where $\text{Count}_{\text{clip}}(\text{n-gram})$ is the maximum number of times an n-gram of the candidate output y occurs in any reference sentence, and $\text{Count}(\text{n-gram})$ is the total

number of n-grams in the candidate output. The final BLEU score is computed as follows.

$$\text{BLEU} = \text{BP} \cdot \exp(\frac{1}{N}\sum_{n=1}^{N}\log p_n), \tag{2.2}$$

$$\tag{2.3}$$

where BP is the brevity penalty. Usually, $N = 4$, which means considering up to 4-grams.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a recall-oriented metric. We adopt two variants of ROUGE, ROUGE-N and ROUGE-L. ROUGE-N is similar to BLEU by computing the n-gram matching score, but focuses more on the recall.

$$\text{ROUGE-N} = \frac{\sum_{S \in \textit{References}} \sum_{\text{n-gram} \in S} \text{Count}_{\text{match}}(\text{n-gram})}{\sum_{S \in \textit{References}} \sum_{\text{n-gram} \in S} \text{Count}(\text{n-gram})}, \tag{2.4}$$

where $\text{Count}_{\text{match}}$(n-gram) is the maximum number of an n-gram co-occurring in a candidate output and a set of references, and Count(n-gram) is the total number of times an n-gram appears in the references. In this study, we adopt $N = 1, 2$, and 4 following [20].

ROUGE-L is a variant of ROUGE based on longest common subsequences (LCS) between the candidate and references. Given a candidate X and a reference Y, ROUGE-L is computed as follows.

$$\text{ROUGE-L} = \frac{(1+\beta^2)R_{lcs}P_{lcs}}{R_{lcs} + \beta^2 P_{lcs}}, \tag{2.5}$$

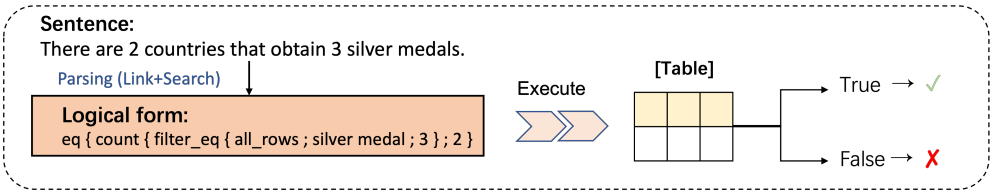
$$R_{lcs} = \frac{\text{LCS}(X,Y)}{m}, \tag{2.6}$$

$$P_{lcs} = \frac{\text{LCS}(X,Y)}{n}. \tag{2.7}$$

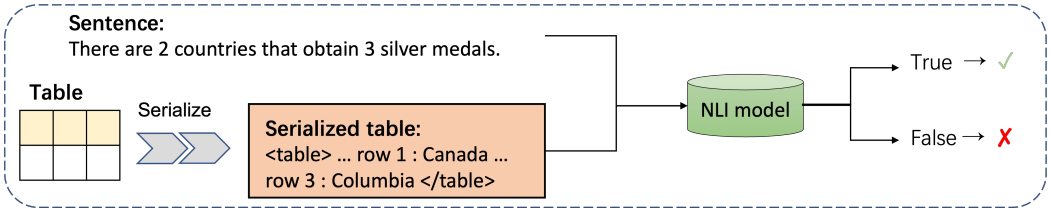
where m is the length of the reference Y, n is the length of the candidate X, and $\text{LCS}(X,Y)$ is the length of the longest common subsequence between X and Y.

Logical Fidelity Metrics

We follow the previous works on logical NLG to define the logical fidelity metrics. Chen et al. [17] proposed two metrics, SP-Acc and NLI-Acc, to measure whether



(a) SP-Acc.



(b) NLI-Acc, TaPEx-Acc and TaPas-Acc.

Figure 2.2: The evaluation method used for SP-Acc (2.2a) and NLI-Acc/TaPEx-Acc/TaPas-Acc (2.2b). For the latter three metrics, the method is the same but with different NLI models.

a generated text is faithful to the given table. We further introduce TaPEx-Acc and TaPas-Acc to complement these metrics. Another line of work [110, 130, 68] on the Logic2Text dataset treats LFs as explicit supervision signals and directly measures the logical consistency between the generated text and the input LF. Shu et al. [110] proposes the Blec metric to evaluate such consistency automatically. While this metric only considers the semantical faithfulness of the generated text to the given LF, it can, to some extent, reflect the faithfulness to the table since the LFs are annotated as exactly supported by the table. Next, we introduce these metrics in detail.

SP-Acc is a parsing-based method aiming to transform the generated sentence into a meaning representation, i.e., a logical form, and execute it against the table to verify its correctness. The conversion to logical forms is based on a weakly-supervised semantic parsing method, which first links the entities and predicates in the sentence with the table and then performs a breadth-first search to construct potential logical forms. The logical forms adopted in this method are similar to those defined in Section 2.2.1 and can be executed to return a boolean value, *true* or *false*, indicating its exact correctness. An illustration of SP-Acc

is provided in Figure 2.2a. However, we empirically find the parsing method often generates irrelevant logical forms to the sentence, rendering the evaluation inaccurate. Therefore, we adopt this metric only for comparison with the previous work.

NLI-Acc is based on the idea of measuring the entailment score between the table and the sentence (Figure 2.2b). A TableBERT [16] model serves as the evaluator, which linearizes the table into a sequence and performs natural language inference between the table sequence and the sentence. It is pretrained on the TabFact dataset, and for inference, it can predict a likelihood score $P_{NLI}(Y|T)$ to indicate the probability that table T entails sentence Y. The final score is computed as the ratio of model outputs that obtain $P_{NLI}(Y|T) > 0.5$.

TaPEx-Acc and TaPas-Acc However, the TableBERT model used for NLI-Acc only achieved 65.1% accuracy on the test set of TabFact dataset, and we find it overly positive about the predictions. Therefore, we add two similar scores to NLI-Acc by adopting two stronger table-NLI models: TaPEx-large [69] and TaPas-large [33], which achieved 84.2% and 81.0% test accuracy on TabFact. We name the two metrics as **TaPEx-Acc** and **TaPas-Acc**, respectively.

Blec is a rule-based method based on a string match between the LF and the generated sentence. Given an LF $L = (l_1, l_2, \ldots, l_n)$ consisting of n tokens, the algorithm first detects whether each token l_i is a key token, i.e., an operation name or a number. If yes, the algorithm will search for the token in the sentence. If all the key tokens in the LF can be found exactly in the sentence, this generation is counted as correct. The overall score is computed as

$$\textsc{Blec} = \frac{\sum_{s \in \mathcal{D}} \text{match}(s)}{|\mathcal{D}|}, \tag{2.8}$$

where s is an example of the dataset $\mathcal{D}$, match(s) is an indicator function that returns 1 or 0 to represent whether the generation matches the logical form of s or not, $|\mathcal{D}|$ is the total size of the dataset.

All these automatic metrics can not represent the exact generation quality; hence human evaluation is also necessary. We introduce the details of human evaluation in the experiment parts of the corresponding chapters of proposed methods.

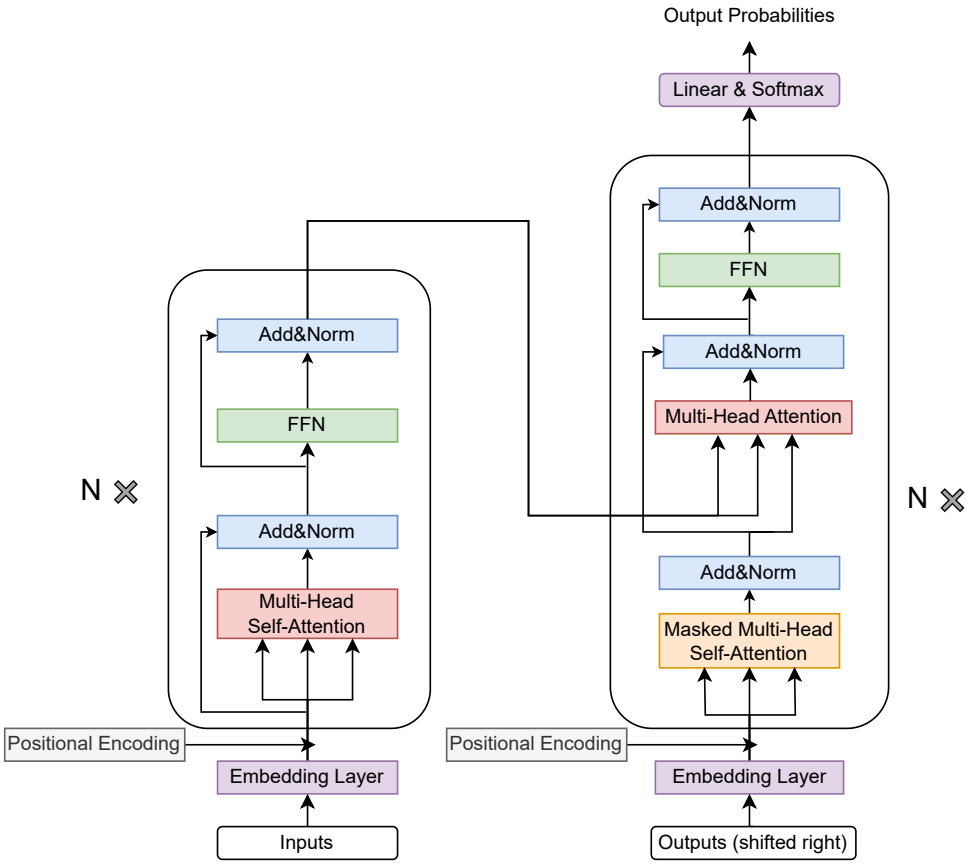


Figure 2.3: The Transformer architecture. On the left side is the encoder, and on the right is the decoder. They are connected via a source-target attention mechanism, illustrated by the Multi-Head Attention block in the decoder.

2.2.3 Pretrained Language Models

Next, we introduce the basic knowledge about the pretrained language models we use in this study. To begin with, we first describe the Transformer [119] architecture, which is the base model of the PLMs.

Transformer

The Transformer architecture is well-known for its general effectiveness in NLP and is adopted as the base model by most PLMs. Transformer adopts multi-head

self-attention mechanism to handle sequential data. It enhances the efficiency of parallel computing and learns better long-term dependency between tokens in a sequence than LSTM. The original Transformer network adopts an encoder-decoder framework. Given an input sequence $\boldsymbol{X} = (x_1, x_2, \cdots, x_m)$ and a target sequence $\boldsymbol{Y} = (y_1, y_2, \cdots, y_n)$, the learning objective is the probability $P(\boldsymbol{Y}|\boldsymbol{X})$ to transform $\boldsymbol{X}$ into $\boldsymbol{Y}$. The encoder transforms $\boldsymbol{X}$ into an intermediate feature representation $\boldsymbol{Z}$. Then the decoder generates the target $\boldsymbol{Y}$ based on $\boldsymbol{Z}$. The encoder can be seen as a feature extractor to obtain the context-aware representation of the input based on the self-attention mechanism. The decoder is a sequence generator that produces tokens step-by-step autoregressively. The connection between the encoder and the decoder lies in a source-target attention mechanism.

As shown in Figure 2.3, the encoder comprises N stacked identical encoder layers. Each layer consists of a multi-head self-attention sub-layer and a position-wise feed-forward network (FFN) sub-layer. After each sub-layer, a residual connection is employed, followed by a layer normalization (illustrated by Add&Norm blocks). The decoder also comprises stacked decoder layers, while the architecture is different from the encoder. A masked multi-head self-attention sub-layer is first applied to the target sequence embeddings, which adopts masked attention to make each token only visible to its subsequent tokens since the generation of a token should not rely on its future generation in the autoregressive generation. Then, a multi-head source-target attention is applied to model the attention relationship between the encoder's output and the decoder's input after the masked multi-head attention. Finally, the vector representation obtained by source-target attention passes a positional feed-forward layer. Similar to the encoder, each sub-layer in the decoder is also followed by a residual connection and a layer normalization. The final representation obtained by the decoder is fed to a linear classifier to predict the output probability of generating the target tokens. Next, we introduce the details of each layer.

Attention Mechanism The attention mechanism exploits the relationship between the elements in the input (in this case, the tokens in a sequence) to compute better representations. Transformers adopt a Query-Key-Value (QKV) attention

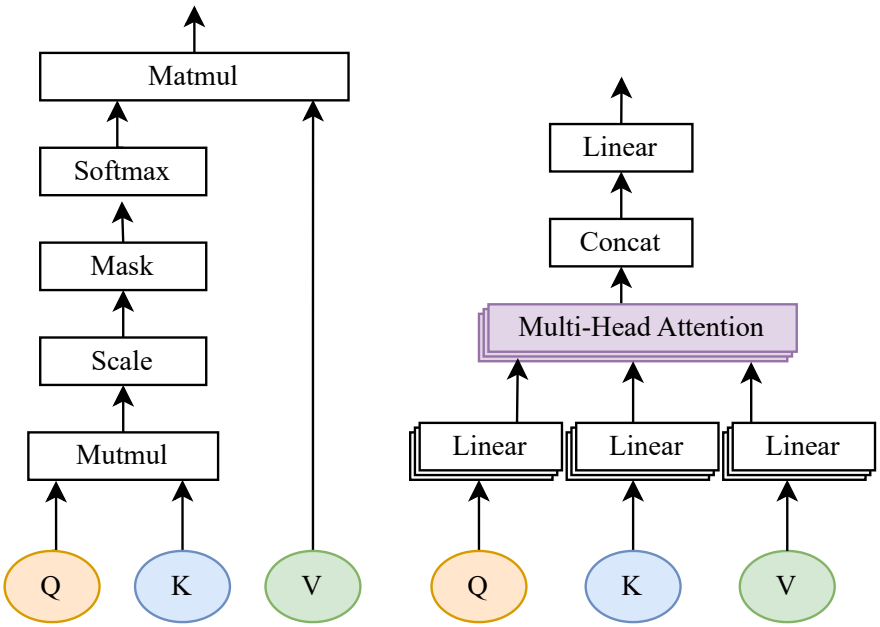


Figure 2.4: The Multi-head attention mechanism.

computed via the following formula (illustrated in the left side of Figure 2.4).

$$\text{att}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V, \tag{2.9}$$

where $Q \in \mathbb{R}^{N_Q \times d_k}$, $K \in \mathbb{R}^{N_K \times d_k}$, $V \in \mathbb{R}^{N_V \times d_k}$ are vector representations of certain sequences, N_Q, N_K, N_V ($N_K = N_V$) are the numbers of tokens in the sequences they represent and d_k is the feature dimension of the vectors. The inner product QK^T computes the pair-wise correspondence between tokens of Q and K. The softmax operation transforms the correspondence scores into normalized attention weights. The multiplication of the attention weights and V leads to a weighted sum of the token representations in the sequence V represents. Q, K and V are set differently in self-attention and source-target attention.

In self-attention layers, Q, K and V are obtained from the same sequence to compute the attention weights between tokens in the same sequence. In the source-target attention in the decoder, Q is the representation obtained from the decoder's self-attention layer. In contrast, K and V are obtained from the output of the encoder. This is to model the dependency between the input and target sequences. To enhance the expressiveness of attention layers, Transformer

uses multi-head attention (illustrated in the right side of Figure 2.4) to linearly project Q, K, and V for h times into different same-dimension partial representations. The self-attention or source-target attention is employed for each partial representation, denoted as a "head" in the following formula.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{2.10}$$

$$\text{head}_i = \text{att}(QW_i^Q, KW_i^K, VW_i^V). \tag{2.11}$$

The projection matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_k}$ project the full hidden dimension d_{model} to the partial dimension $d_k = d_{model}/h$, and $W_O \in \mathbb{R}^{hd_k \times d_{model}}$ projects the multi-head representations back to the original dimension.

Position-wise Feed-forward Network Each layer in the encoder and decoder contains a fully-connected feed-forward network (FFN) for linear transformation of the representations.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2.12}$$

Embeddings The initial embeddings of the input and target sequences comprise a token embedding and a positional embedding. The token embedding is obtained by learned embeddings with dimension d_{model}. The positional embedding aims to include the order information of tokens in the sequence. In [119], a sinusoidal encoding method is used to compute the positional embeddings:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{2.13}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \tag{2.14}$$

Here, *pos* indicates the absolute position of each token; i refers to the i-th element in the embedding vectors. These embeddings are fixed upon deciding the sequence order, while the later PLMs usually adopt a learned embedding that can be changed during the training.

GPT-2

PLMs modify the architecture of the original Transformer and adopt different components of the original encoder-decoder framework. Thus, PLMs can be

roughly classified into encoder-only, decoder-only, and encoder-decoder models. The most representative encoder-only model is BERT [26], followed by its variants. Such models mainly focus on extracting the representations of sequences with an encoder and use the representations for natural language understanding tasks. To apply them to generation tasks, a separate decoder model is necessary. Therefore, decoder-only models such as GPT-2 and encoder-decoder models such as BART and T5 better fit NLG tasks. GPT-2 serves as the base model used in Chapter 3, while BART and T5 are base models used in Chapter 4.

Generative Pre-trained Transformer 2 (GPT-2) is a direct scale-up of OpenAI's GPT model ("GPT-1") with the same architecture but slight changes in the normalization.

GPT-2 was pretrained on WebText, a large corpus of around 40GB of text collected from Reddit. The unsupervised pretraining on raw unlabeled corpora is realized by a language modeling task, i.e., predicting the next token given the previous context. Given an unlabeled text corpus $\boldsymbol{U} = \{u_1, u_2, \ldots, u_N\}$, the optimization objective is

$$\mathcal{L}(\boldsymbol{U}) = \sum_{i=1}^{N} \log(P(u_i|u_{i-k}, \ldots, u_{i-1}, \Theta)) \tag{2.15}$$

where Θ is the model parameters, k is the sliding window size that represents the number of visible previous tokens for each prediction. Similar to the decoder part of Transformer, GPT-2 adopts masked attention to realize left-to-right generation, where the attention weight computed for each token only depends on the previous tokens.

The finetuning of GPT-2 on specific downstream tasks can be easily adapted from the pretraining objective. Given a supervised corpus ($\boldsymbol{X} = (x_1, x_2, \ldots, x_M)$, $\boldsymbol{Y} = (y_1, y_2, \ldots, y_N)$ of an NLG task, the finetuning objective is as follows.

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}) = \sum_{i=1}^{N} \log(P(y_i|\boldsymbol{X}, y_1, \ldots, y_{i-1}, \Theta)) \tag{2.16}$$

As GPT-2 is decoder-only, both the input and output sequences are concatenated together to feed into the decoder, but we do not include the output probabilities on the input sequence in the training objective.

BART

Bidirectional and Auto-Regressive Transformers (BART) is an encoder-decoder PLM. Unlike GPT-2, BART has the same architecture as the original Transformer, such that the encoder can learn bidirectional contextual representations and the decoder generates auto-regressively. BART is a denoising auto-encoder, pretrained to denoise corrupted text. The pretraining corpus is constructed by corrupting a text with arbitrary noising functions, and the original text serves as the ground-truth of the denosing task. Thus, the model learns to reconstruct the original text from the corrupted text. Specifically, Lewis et al. [62] tested the following noising transformations and adopt a composition of Text Infilling and Sentence Permutation for pretraining the final models.

1. Token Masking: randomly replacing a token with a special placeholder `[MASK]`. The same method was used in BERT [26].
2. Token Deletion: randomly deleting a token.
3. Text Infilling: randomly replacing a span of consecutive tokens with a `[MASK]`.
4. Sentence Permutation: randomly shuffling the sentences in a document.
5. Document Rotation: randomly rotating a document to let it start from a uniformly sampled position.

The finetuning of BART on sequence generation tasks is straightforward because of the encoder-decoder architecture.

T5

Text-to-Text Transfer Transformer (T5) [98] is also an encoder-decoder model. Similarly to BART, T5 is pretrained on a denoising autoencoding task to recover corrupted text. T5 adopts span-level masking as in BART, but the generation target is not the entire original text but just the masked tokens. The pretrainig data is Colossal Clean Crawled Corpus (C4), a large corpus of 750GB, crawled and cleaned from Common Crawl. BART and T5 are similar PLMs and both can be directly adapted to NLG tasks.

All these PLMs have multiple pretrained checkpoints of different model sizes, allowing the public to adapt the models to different downstream tasks. The following is the information about the pretrained checkpoints adopted in this study.

- GPT-2-small: 117M parameters with 12-layer, 768 feed-forward hidden-state, 12-heads.
- 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.

3 Data Augmentation for Logical NLG with Explicit Logical Form Control

This chapter describes the first approach to augment logical forms (LFs) for logical NLG. As described in Chapter 1, the original form of logical NLG [17], directly generating a logical description from a table, is difficult because of implicit logical reasoning and the lack of training data. To address the difficulty in learning logical reasoning, Chen et al. [20] first proposed to use LFs as explicit mediators to guide the generation. They collected the LOGIC2TEXT dataset, which contains multiple parallel <LF, Text> examples within each table; hence the NLG task can be formulated as an LF-to-text task, with the table as the context. Such aligned LFs allow training logical NLG systems that can achieve more controllable and faithful generations than directly generating a random text from a table, which resulted in a significant boost in terms of the human-evaluated fidelity score from 20.2% to 80.4% [20].

Nevertheless, the labor-intensive human work of pairing LFs and textual descriptions limited the scale of the LOGIC2TEXT dataset (ca. 10.8k instances), which is much smaller than those of the common benchmarks on the surface-level NLG [83, 60] and the other logical NLG dataset LOGICNLG. Pretrained models such as GPT-2 [97] could work on this small amount of supervision data, using the rich contextual knowledge learned from large-scale corpora; however, there is a lot of room for improvement in terms of the generation quality [20]. Therefore, previous work [110] has explored augmenting additional LFs and generating paired texts via iterative self-training to improve the performance. However, their approach is based on perturbing the original LFs in LOGIC2TEXT based on static noise, which leads to mostly unfaithful LFs. The paired texts are generated from these augmented LFs, thus their faithfulness cannot be guaranteed either. While

the augmented LFs and texts boost the performance, this approach is dedicated to improving the conversion from LFs to texts, not considering their consistency to the context table.

In contrast, we first propose to augment reliable parallel <LF, text> pairs from tables. We observe that each table in the LOGIC2TEXT dataset is only associated with at most three examples. However, a table can contain abundant logical-level facts derived from the composition of various logical operations, and each fact can be abstracted into a logical form and associated text translations. The space of such compound facts is quite large. Suppose a table with m rows and n columns is given; the theoretical search space of logical facts will be up to $O(n*2^m+m*2^n)$ for an aggregation operation (e.g., `avg`, `count`), and $O(n * m^2 + m * n^2)$ for a 2-operand operation (e.g., `diff`, `greater`).[1]

Inspired by this, we propose to augment new examples from existing tables to take advantage of the rich information in multi-row tables. The proposed approach consists of two stages:

1. *Topic-conditioned data augmentation* (TopicDA) In this stage, we aim to obtain logical facts represented in either LFs or texts from tables. Specifically, we train two auxiliary topic-conditioned *table-to-logic* and *table-to-text* models on the re-organized supervision data of LOGIC2TEXT. Provided pre-defined logic types as *topics*, the two data augmentation (DA) models can generate LFs and texts with diverse logic types from tables in the original training data. The augmented examples are expected to be consistent with the given tables and logic types. As a result, we can mine more logical facts from existing tables for data augmentation without resorting to any additional resources. The augmented examples, however, are unpaired LFs and texts, which we use in the next stage to obtain parallel examples.

2. *Iterative joint training* To obtain parallel examples from the augmented LFs and texts, this stage aims to generate their corresponding paired counterparts. To this end, we introduce *logical form generation (LG)*, a dual task of LOGIC2TEXT that requires generating a valid logical form based on a text description and a corresponding table. Inspired by previous works on the joint learning of dual NLP tasks [15, 14, 95, 42, 103], we propose

[1]This is a rough computation and does not consider the composition of different operations.

the simultaneous solution of the LOGIC2TEXT task and LG task by generating pseudo-parallel data from the augmented data through iterative joint training, a semi-supervised training strategy. With iterative back-translation and pseudo-labeling, the augmented LFs and texts (obtained in stage 1) can be further used to construct paired data. The dual models are iteratively refined while benefiting from each other. We also propose a *round-trip data weighting* strategy to balance the effects of different pseudo-parallel examples during semi-supervised training.

We evaluate the proposed methods on the LOGIC2TEXT dataset and its dual task LG. Experimental results on both automatic and human evaluation demonstrate the effectiveness of the proposed framework in leveraging augmented data. In particular, our framework advances the state-of-the-art on LOGIC2TEXT with an average improvement of over 2 points on multiple evaluation metrics, including both surface-level metrics like BLEU and faithfulness scores like BLEC. Furthermore, analytical experiments on data augmentation demonstrate that the proposed TopicDA method can generate topic-diversified data with reasonable quality. We finally illustrate a few qualitative examples to show the performance of the TopicDA models and the proposed models.

3.1 Related Work

Our approach is highly related to the research on semi-supervised NLG. Iterative back-translation (IBT) is an extension of back-translation, in which forward and backward models are trained together to generate pseudo parallel instances for each other. Such a dual-reconstruction technique is popular in many text generation tasks such as machine translation [45], text style transfer [137, 73] and data-to-text generation [15, 14, 95, 42, 103]. Our approach is the first trial to apply dual learning on logical NLG. In prior work on IBT, the source-to-target model is only trained on pseudo-labeled data generated by the target-to-source model (together with labeled data), and vice versa. In this work, pseudo-labeled data generated by the two teacher models are combined and used for training both student models. This is related to the research on pseudo-labeling or self-training [44] for natural language generation. Our method combines both IBT and pseudo-labeling to fully utilize the unpaired augmented data.

Different from previous works [95, 113, 42] that directly exploited off-the-shelf unpaired data, Chang et al. [15] posed the problem that it is unrealistic to have so many unpaired texts for data-to-text tasks, and adopt language models (LM) like GPT-2 [97] to augment additional texts. The requirement of parallel LFs and texts in this research further toughens the problem because we must augment both supported by the tables, suggesting the difficulty of applying dual model learning. In this work, we realize the data augmentation from existing tables without the availability of additional resources. Previous work [110] on LOGIC2TEXT also performs data augmentation by perturbing the existing logical forms with rule-based noise and employs self-training. In contrast, our work generates both LFs and texts directly from tables with controllable topics and employs dual reconstruction to benefit the LOGIC2TEXT model with the dual model, showing superior results in logic-consistency between logical forms and texts.

3.2 Task Formulation

This section defines the task setting of logical NLG adopted in this chapter, i.e., LOGIC2TEXT.

In LOGIC2TEXT, an input consists of a table d and a logical form l that can be executed on d, and an output is a sentence description $t = [w_1, w_2, \ldots, w_n]$. Each d is a database-like table consisting of T_R rows and T_C columns, a table title/caption and a series of column headers $\{h_i\}_{i=1}^{T_C}$. We aim to train a model $P_\theta(t|l, d)$ to generate t^* that can be supported by both the table d and logical form l.

In this approach, we propose the dual task of LOGIC2TEXT, named **Logical Form Generation (LG)**, for which we train a model $P_\phi(l|t, d)$ to estimate l^* from the input description t, also supported by the table d. LG is a "conditionally reverse" task of LOGIC2TEXT, as the table acts as a common condition of generation in both tasks.

Each d can have multiple associated $\langle l, t \rangle$ pairs and each pair has a pre-defined logic type $c \in \mathcal{C}$ indicating its main logical operation, where $\mathcal{C} = \{$*count, comparative, superlative, unique, ordinal, aggregation, majority*$\}$, as described in Section 2.2.1. These logic types have different preferences on the patterns of LFs and texts. As shown in Figure 1.2, *count*-type LFs tend to contain *eq* and *count*

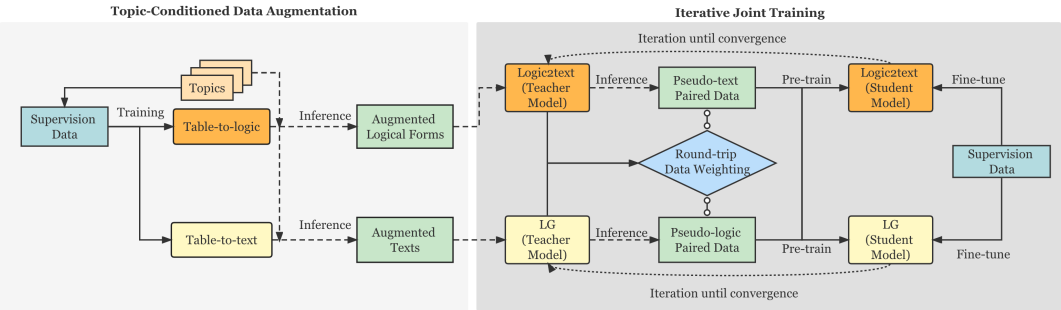


Figure 3.1: An overview of the proposed framework. We first train two topic-conditioned data augmentation models, *table-to-logic* and *table-to-text*, with the supervision data and pre-assigned topics. These models generate additional logical forms and texts from tables with different topics. The augmented unpaired data are utilized in iterative joint training, where the LOGIC2TEXT model and LG model are jointly improved via a teacher-student semi-supervised training procedure.

functions; *superlative*-type texts usually have superlative words like *most, least, highest*, etc. Both tasks share the same supervision data from the LOGIC2TEXT dataset, $\mathcal{S} = \{(c_i, d_i, l_i, t_i)\}_{i=1}^{k}$, where k is the number of instances.

3.3 Proposed Approach

Our framework is composed of two stages: (1) topic-conditioned data augmentation (TopicDA) for augmenting logical forms and texts. (2) iterative joint training of LOGIC2TEXT and LG, to utilize the augmented data and improve both models in an iterative procedure. Figure 3.1 depicts an overview of the framework.

3.3.1 Base Model

Following [20], we use a pretrained language model GPT-2 as the supervised base model and adopt data serialization strategies to linearize structured tables into sequences. Specifically, the sequence of each table starts with the concatenation of its caption and table headers. Then it includes table cells in a row-wise order from the top-left corner to the bottom-right corner. Each table cell is serialized into

T_{ij} = "⟨*cell*⟩ d_{ij} ⟨*row_idx*⟩ i ⟨/*row_idx*⟩ ⟨*col_header*⟩ h_j ⟨/*col_header*⟩ ⟨/*cell*⟩", where d_{ij} denotes the content of the cell at row i and column j of table d, h_j is the header of column j. The special tokens in brackets are defined to indicate the table structure.

As an example, the table in Figure 1.2 is linearized as a sequence

T = "⟨*table*⟩ ⟨*caption*⟩ *Medal Table from Tournament* ⟨/*caption*⟩ ⟨*col_header*⟩ *Nation* ⟨/*col_header*⟩ ⋯ ⟨*cell*⟩ *Canada* ⟨*row_idx*⟩ *1* ⟨/*row_idx*⟩ ⟨*col_header*⟩ *Nation* ⟨/*col_header*⟩ ⟨/*cell*⟩ ⟨*cell*⟩ *3* ⟨*row_idx*⟩ *1* ⟨/*row_idx*⟩ ⟨*col_header*⟩ *Gold Medal* ⟨/*col_header*⟩ ⟨/*cell*⟩ ⋯ ⟨/*cell*⟩ ⟨/*table*⟩".

As shown in Figure 1.2, the LFs can be linearized as a pre-order traversal of the logic form tree, with special tokens such as curly braces and semi-colons to separate functions and arguments. Thus, each instance has a serialized table T and LF l. We additionally consider the logic type (topic) c as explicit prior knowledge to guide the generation. We hence prepend a topic sequence "⟨*topic*⟩ *[logic type]* ⟨/*topic*⟩" to the table sequence T.

The objective of a LOGIC2TEXT model is to generate a sequence $t^* = [u_1, u_2, \ldots, u_N]$:

$$t^* = \arg\max \prod_{i=1}^{N} P(u_i | u_{<i}, [c; T; l]; \theta), \tag{3.1}$$

where $[;]$ denotes the concatenation of multiple sequences. Similarly, for the LG task, the goal is to generate a serialized LF $l^* = [v_1, v_2, \ldots, v_M]$:

$$l^* = \arg\max \prod_{j=1}^{M} P(v_j | v_{<j}, [c; T; t]; \phi). \tag{3.2}$$

3.3.2 Topic-Conditioned Data Augmentation

Researchers have used pretrained language models such as GPT-2 to augment text samples for text classification [85, 59] and generation tasks [15] by finetuning it on in-domain texts. The prior knowledge integrated in such pretrained models leads to a considerable quality of the generated text. Different from previous work [14] that bootstraps new text instances from the original texts, we seek to generate LFs and texts from tables.

To this end, we propose TopicDA, to construct two *conditional table-to-logic* ($P_{d2l}(l|d, c)$) and *table-to-text* ($P_{d2t}(t|d, c)$) models, which require the generation

of an LF l or text t directly from a table d following a certain logic type c. A logic type serves as a topic to control the generation pattern. Our insights are two-folds: (1) tables contain rich logical information that can be described by additional LFs or texts; (2) topics can ensure the diversity of augmented data.

TopicDA contains a training pass to pretrain DA models and an inference pass for them to generate new data. The training objectives are

$$\mathcal{L}_{d2l} = \mathbb{E}_{(c,d,l)\sim\mathcal{S}}[-\log P_{d2l}(l|d,c)], \tag{3.3}$$

$$\mathcal{L}_{d2t} = \mathbb{E}_{(c,d,t)\sim\mathcal{S}}[-\log P_{d2t}(t|d,c)]. \tag{3.4}$$

TopicDA theoretically applies to any pretrained language models that can generate sequences. Here, we implement the backbone model based on GPT-2, a widely-used decoder-only pretrained model. After training, the models can be utilized for data augmentation through inference. Specifically, we perform inference with the trained *table-to-logic* and *table-to-text* models to generate extra LFs and texts based on a table d_u, where d_u can be tables in the original training data or from other resources. In this study, we only adopt the tables in the training data of LOGIC2TEXT. However, further improvements can be expected if we use additional tables. We assign each of the seven logic types to d_u to generate outputs with diverse logical patterns.

A generated text or LF is filtered out if: (1) its length exceeds 200 tokens in Byte-Pair Encoding (BPE) [105]; (2) it is identical to an existing instance in the training data. In this study, we only consider the seven pre-defined logic types as topics. However, our method can be easily adapted to other domains and topics if it is provided with topic-dependent supervision data. For instance, it is possible to use more fine-grained logic patterns such as specific logical functions and words.

3.3.3 Iterative Joint Training

After data augmentation, we obtain $\mathcal{U}_l = \{(c_u, d_u, l_u)\}$ and $\mathcal{U}_t = \{(c_u, d_u, t_u)\}$, where l_u and t_u are unpaired logical forms and text descriptions, d_u is a context table, and c_u indicates the assigned logic type. They are not directly usable as supervision data for LOGIC2TEXT and LG because l_u and t_u are generated independently and unaligned. Therefore, we propose a semi-supervised learning

approach to jointly improve LOGIC2TEXT and LG models with the augmented data.

Let $P_\theta(t|l,d)$ denote a LOGIC2TEXT (L2T) model and $P_\phi(l|t,d)$ an LG model. Logic type c is also prepended to d as an extra control variable. Both models are first pretrained on supervision data $\mathcal{S}$ for several epochs. We then employ semi-supervised training based on the duality between the two tasks.

Iterative Back-translation

Specifically, we consider iterative back-translation (IBT), a popular method in machine translation [32] for augmenting pseudo-parallel data. Its core idea is to translate a target-side monolingual sentence y into a source-language pseudo-sentence $\hat{x}$, with a target-to-source translation model M_{yx}, which forms a pseudo-parallel sentence pair $(\hat{x}, y)$ that can be used to train the source-to-target model M_{xy}, while M_{xy} produces pseudo-parallel data $(x, \hat{y})$ to train M_{yx}. In our case, $P_\theta(t|l,d)$ and $P_\phi(l|t,d)$ are conditionally inverse to each other, with the table d acting as a condition of the conversion between the logical form l and text t.

Pseudo-labeling

A drawback of back-translation is that it can only improve a model with its target-side unpaired data. In our case, the L2T model is trained only on the pseudo instances constructed from U_t and the LG model only leverages pseudo data from U_l. To fully utilize the augmented data, we incorporate a pseudo-labeling scheme, where each model predicts pseudo targets for its source-side unpaired data, constructing pseudo-parallel instances to re-train itself.

Round-trip Data Weighting

The augmented logical forms and texts can be noisy because they are generated from imperfect neural models. Moreover, the pseudo-parallel data generated via IBT and pseudo-labeling are also noisy generations. Therefore, we propose a round-trip data weighting (RTW) strategy to assign per-sample weights to the pseudo-paralell data, which can balance their effects during unsupervised training. In round-trip translation, we first translate a logical form l into a text using an L2T model and then back-translates it to a reconstructed $\tilde{l}$ with an LG model.

We then compute the similarity between the l and $\tilde{l}$ with BERTScore [135] as the weight for this instance. We also apply the same method to unlabeled text instances. Finally, we obtain a weight vector $W_l = [w_l^{(1)}, w_l^{(2)}, \ldots, w_l^{(k_l)}]$ for $\mathcal{U}_l$ and $W_t = [w_t^{(1)}, w_t^{(2)}, \ldots, w_t^{(k_t)}]$ for $\mathcal{U}_t$, where k_l and k_t are the sizes of $\mathcal{U}_l$ and $\mathcal{U}_t$, respectively.

Our method is inspired by a line of works [50, 55] on unsupervised machine translation, which adopt round-trip translation to measure the quality of pseudo instances.

We adopt the BERTScore as the weighting score, a popular evaluation metric for text generation, which leverages the pretrained contextual embeddings from BERT [25] and compute cosine similarity between words in candidate and reference sentences/logical forms. We adopt the F1-measure of BERTScore in practice.

Training Strategy

We adopt a teacher-student training strategy to iteratively optimize the two models. At each iteration, we have both a teacher copy and a student copy of each model. The initial teacher models are pretrained on the supervision data at first. Then, they are frozen to generate pseudo-parallel data for the student models. Round-trip data weighting is then performed on these data. At the end of each iteration, the teacher models are updated from the best checkpoints of corresponding student models.

Within each iteration, we perform a pretrain-finetuning (PT-FT) scheme: the student models are first pretrained on the pseudo-parallel data generated from both IBT and pseudo-labeling, then finetuned on the clean supervision data $\mathcal{S}$ as teacher forcing. This scheme can also reduce the negative impacts caused by noisy pseudo-parallel data [44].

During the PT stage in semi-supervised training, we optimize the following objectives to train the models on the combination of pseudo-parallel data from both IBT and pseudo-labeling.

$$
\begin{aligned}
\mathcal{L}(\theta) =& \mathbb{E}_{(d,t)\sim\mathcal{U}_t}[-w_t \log P_\theta(t|\hat{l},d)] + \\
& \mathbb{E}_{d,l)\sim\mathcal{U}_l}[-w_l \log P_\theta(\hat{t}|l,d)], \\
\mathcal{L}(\phi) =& \mathbb{E}_{(d,l)\sim\mathcal{U}_l}[-w_l \log P_\phi(l|\hat{t},d)] + \\
& \mathbb{E}_{(d,t)\sim\mathcal{U}_t}[-w_t \log P_\phi(\hat{l}|t,d)],
\end{aligned} \tag{3.5}
$$

where $\hat{t} \sim \text{argmax}_t P_\theta(t|l,d)$ and $\hat{l} \sim \text{argmax}_l P_\phi(l|t,d)$ are the pseudo target sequences generated by the dual models; w_l and w_t are the corresponding weights for the pseudo-examples. A formal description of iterative joint training is illustrated in Algorithm 1.

Algorithm 1 Training procedure

1: **Input:** Augmented logical form dataset $\mathcal{U}_l$; augmented text dataset $\mathcal{U}_t$; Supervised dataset $\mathcal{S}$;
2: **Output:** A LOGIC2TEXT model P_θ; an LG model P_ϕ.
▷ Pretraining
3: Initialize student models P_θ^S and P_ϕ^S, and teacher models P_θ^T and P_ϕ^T with pretrained LMs.
4: Train P_θ^T and P_ϕ^T on $\mathcal{S}$.
▷ Iterative Joint Training
5: **repeat**
6: With P_θ^T and P_ϕ^T as the generation models, generate pseudo-paired data for $\mathcal{U}_l$ and $\mathcal{U}_t$.
7: Compute data weights W_l for $\mathcal{U}_l$ and W_t for $\mathcal{U}_t$ via round-trip weighting.
8: Train P_θ^S and P_ϕ^S on combined pseudo-paired data, according to Equation (3.5).
9: Train P_θ^S and P_ϕ^S on $\mathcal{S}$.
10: Update teacher models with the best parameters of student models: $P_\theta^T = P_\theta^S$; $P_\phi^T = P_\phi^S$.
11: Re-initialize P_θ^S and P_ϕ^S as in step 3.
12: **until** convergence or max iteration reached

# of instances for training DA models	8566
# of tables for training DA models	4554
# of tables for DA inference	4554
# of augmented logical forms	29042
# of augmented texts	31247

Table 3.1: Statistics of data augmentation (DA).

3.4 Experiments

3.4.1 Datasets

We conduct experiments on the LOGIC2TEXT dataset. Each table in LOGIC2TEXT is associated with 1 to 3 instances with different logic types. We reuse the L2T dataset to construct a logical form generation (LG) semantic parsing dataset, in which a logical form is parsed from a table and a text description. LG follows the same train/validation/test split as LOGIC2TEXT. Table 3.1 lists the statistics of the augmented data.

3.4.2 Evaluation Metrics

We use BLEU-4[2], ROUGE-1,2,4,L[3] to evaluate the models on the LOGIC2TEXT task, following [20]. In addition, we adopt BERTScore [135] to measure the semantic similarity between generated sentences and gold references. We also adopt BLEC[4] [110], an automatic metric that measures the logic consistency between logical forms and texts through rule-based string match and presents reasonable correlation with human evaluation. For the LG task, we adopt Logical Form Accuracy (LFA) as the main evaluation metric. LFA is the accuracy of the generated logical forms that have the exact string match with gold references. In addition, we use two fuzzy metrics to complement LFA: Pattern Accuracy (PTA) and Execution Accuracy (EXA). PTA only measures whether a generated LF has the same compositional pattern of logical functions with the ground truth, allowing mismatch of specific entities. EXA is the proportion of generated LFs

[2]multi-bleu.pl
[3]rouge-1.5.5
[4]https://github.com/chatc/BLEC

that can be executed on the table and returns a true value.

3.4.3 Experimental Settings

We implement our model on the Huggingface Transformers library [129] and PyTorch [90]. Our framework is theoretically applicable to any sequence generation model. In this work, we use GPT-2-small as our base model for both TopicDA and iterative training, because of its strong performance and reasonable model complexity. We adopt the AdamW [72] optimizer. We employ beam search with the size of 4 for the decoding in both data augmentation and semi-supervised learning. The best checkpoints of the LOGIC2TEXT models are chosen based on the BLEU score on the validation set, and for LG, they are chosen based on the LFA on the validation set. The word embeddings and positional embeddings of the model are frozen during training. The source sequence, i.e., concatenation of a logic type, caption, headers, table content and logical form or sentence, has a maximum of 800 tokens. The maximum length of target sequence is set to 200.

We experimented on a set of combinations of batch size (bs) and learning rate (lr) by manually tuning on the validation set results. With trial and error on bs=$\{1, 2, 4, 8, 20\}$ and lr=$[1 \times 10^{-5} : 3 \times 10^{-4}]$ with step size 1×10^{-5}. we found (bs=2, lr=2×10^{-5}), (bs=4, lr=3×10^{-5}), (bs=8, lr=5×10^{-5}) consistently good. For base models without semi-supervised learning, we set the batch size to 8 and the learning rate to 5×10^{-5}. We adopt (bs=2, lr=2×10^{-5}) for the full model because of the extra memory cost of pseudo data generation and dual model training. We use a gradient clipping with L2-norm and threshold 5.0. We fix the random seed for all experiments.

Here, we provide hyper-parameters of the semi-supervised training strategy. The initial teacher models of L2T and LG are pretrained on the supervision data for 6 epochs. During each iteration, the student models are pretrained on the pseudo-parallel data for 4 epochs, then finetuned on the supervision data for 7 epochs. The number of maximum iteration is set to 3. It takes approximately 24 hours to finish one iteration on an NVIDIA A6000 GPU. Empirically, the best results are obtained from the second or the third iteration.

3.4.4 Models for Comparison

We compare the proposed method with the following strong baselines proposed by previous work.

- **Graph2seq+copy** [20]: Graph2seq [131] builds a graph-based encoder to encode the logical forms with the copy mechanism.
- **Transformer+copy** [20]: This approach is based on the vanilla Transformer [119] network with extra copy mechanism.
- **GPT-2** [20]: A pretrained GPT-2 model is finetuned on the LOGIC2TEXT dataset, where the input tables and logical forms are represented as text sequences.
- **Snowball** [110]: This framework adopts iterative training of both a generator and an evaluator, while perturbing the existing logical forms for data augmentation and reports the best results on BLEC with pretrained BART [63] as the base model. This work does not report the results on BLEU and ROUGE.

Moreover, we re-implement the GPT-2 model with our data serialization methods. We adapt the GPT-2 model to both LOGIC2TEXT and LG as described in Section 3.3.1.

Besides supervised baselines, we employ a semi-supervised baseline: **GPT-2 + self-training (GPT-2-ST)**. Specifically, we adopt a similar iterative training strategy to our approach, where each model generates pseudo-labeled data from the augmented data of TopicDA and refine itself through self-training. The same PT-FT strategy is adopted, following the wisdom of [44]. It can also be seen as ablating our framework by removing the entire dual-task interaction.

3.4.5 Main Results

In Table 3.2 and Table 3.3, we can observe that the models based on pretrained GPT-2 completely outperform previous neural models without pretraining. GPT-2+ST improves supervised GPT-2 by utilizing our augmented data in a self-training manner. Although the LOGIC2TEXT and LG models are trained separately without interaction in GPT-2+ST, we still observe consistent improvements in all the evaluation metrics. This demonstrates the advantage of using our

Models	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-4	ROUGE-L	BERTScore	BLEC
Graph2seq+copy	25.38	58.15	32.79	12.25	49.47	–	–
Transformer+copy	26.42	58.77	33.05	12.83	49.01	–	–
GPT-2 [20]	31.44	64.16	39.48	17.46	53.99	–	–
SNOWBALL	–	–	–	–	–	–	88.60
Our Implementations							
GPT-2	31.95	64.71	40.28	18.47	54.19	0.581	86.72
GPT-2+ST	32.58†	64.93†	40.73†	18.63†	55.18	0.589	88.74†
Ours	**33.78***	**65.98***	**41.77***	**19.70**	**56.05***	**0.593**	**89.38***
– RTW	32.72†	65.53	40.96†	18.91	55.52	0.592	89.01
– RTW & PL	32.65†	65.27†	40.91†	19.15	55.20†	0.591	87.45†

Table 3.2: Main and ablation results on the test sets of LOGIC2TEXT. * indicates that the scores are significantly higher than the baseline model (GPT-2) with paired bootstrap resampling [29] ($p < 0.05$). † indicates that the scores are significantly lower than the full framework (Ours) with paired bootstrap resampling ($p < 0.05$).

augmented data. Moreover, our full model further outperforms these baselines and achieves state-of-the-art results on all the automatic metrics. By comparison with GPT-2+ST, our method shows significant improvements, demonstrating the effectiveness of the joint learning of LOGIC2TEXT and LG. In particular, our model improves our supervised baseline by about 1.8 points in BLEU-4 and ROUGE-L and 2.6 points in BLEC.

3.4.6 Ablation Study

To validate the effectiveness of our joint training method, we conduct ablation studies on two ablated variants of the full model with full supervised data: (i) – RTW: remove round-trip data weighting; (ii) – RTW & PL: remove both RTW and pseudo-labeling. Model (ii) can be seen as a standard IBT method with epoch-level iteration [136, 132]. As shown in Table 3.2 and Table 3.3, both round-trip data weighting and pseudo-labeling contribute to the best performance of our model.

3.4.7 Human Evaluation

We conduct a human evaluation to evaluate the fidelity of LOGIC2TEXT generation. We randomly sampled 200 instances from the test set and compared

Models	LFA	PTA	EXA
GPT-2	68.13	85.90	84.25
GPT-2+ST	68.59	86.36	84.71
Ours	**70.05**	**86.90**	**86.26**
– RTW	69.60	86.17	85.44
– RTW & PL	69.41	85.90	85.44

Table 3.3: Main and ablation results on the test sets of LG.

Models	Factual Acc.	Semantic Acc.
Gold	98.5	96.5
GPT-2	72.0	75.0
Ours	76.0	79.0

Table 3.4: Human evaluation results on 200 sampled instances from the test set of LOGIC2TEXT.

the generations of two models: (1) GPT-2 and (2) Ours, along with (3) the gold references. We follow [20] to evaluate on two metrics: (1) *factual correctness*, i.e., whether the generated description is factually supported by the table; (2) *semantic correctness*, i.e., whether the generated description is consistent with the meaning of the logical form. We hired 2 human experts with a master's degree in Computer Science to evaluate each example. The annotators were instructed to fully understand the logical form schema before the annotation. They first performed evaluation separately, with an inter-annotator agreement of 0.61 averaged over the results of Ours and GPT-2, measured by Cohen's Kappa [120]. The decisions of inconsistent judgements are finally made on discussion. The outputs were shuffled and the model names were hidden to the annotators during the annotation.

As can be observed in Table 3.4, our method outperforms the base model on both metrics by 4.0%, proving the effects of our framework to the logical fidelity. Although there is still a large gap between Ours and gold references, our improvement is reasonable because we did not use outside data nor special model designs compared to GPT-2.

	Topic Acc.	Factual Acc.	Structural Validity
Logical forms	98.52%	13.47%	96.66%
Texts	94.30%	27.85%	-

Table 3.5: Results of the quality of the augmented data.

3.4.8 Analysis of Data Augmentation

Here, we analyze whether TopicDA can generate high-quality data from the following aspects.

Topic Consistency

The main motivation of our TopicDA method is to use logic types as topics to encourage the DA models to generate topic-diversified data. Therefore, we analyze whether the augmented data are consistent with the pre-assigned topic. To realize this evaluation, we train two auxiliary topic classifiers LF-CLR and Text-CLR for classifying the logic type of a logical form and a textual description, respectively. The two classifiers are pretrained BERT-base [25] models finetuned on the training set of Logic2Text and then validated on the test set. The validation accuracy of LF-CLR reaches 100% and the Text-CLR achieves 97.7%. These classifier models are then used to evaluate whether the augmented data are consistent with their assigned logic types during augmentation. The evaluation results listed in Table 3.5 suggest that the augmented data are generally topic-consistent with an accuracy of 98.52% and 94.30% for LFs and texts, respectively. This demonstrates that the generation of TopicDA is controlled well by the topic prompt.

Topic Diversity

We then analyze the topic diversity of the augmented data, i.e. how well they cover the logic types in Logic2Text. Figure 3.2 shows the distribution of the augmented LFs/texts on different logic types. As can be observed, the augmented data cover all the seven logic types almost uniformly.

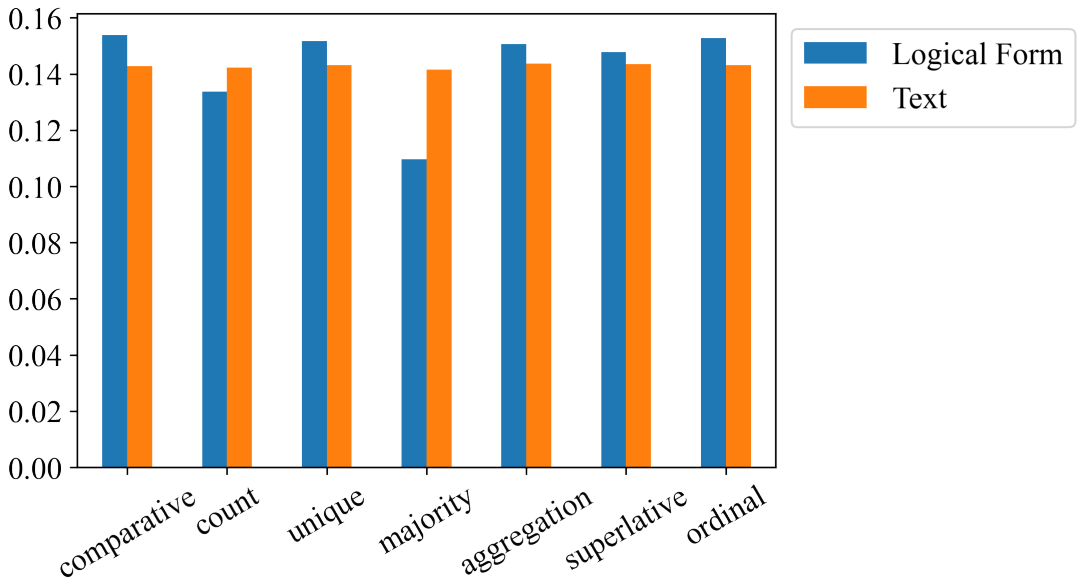


Figure 3.2: Distribution of the augmented logical forms and texts on different logic types.

Factual Correctness

Similarly to the factual correctness defined in Section 3.4.7, we validate whether the augmented data can be exactly supported by their assigned tables. We consider a logical form as factually correct if it can be executed on the table and returns a *true* value. For texts, we back-translate them to logical forms with an LG model and then execute the logical forms on the tables. The accuracy of factual correctness is 13.47% and 27.85% for logical forms and texts, respectively. This result is reasonable because the factual correctness is evaluated with the exact match of the logical form execution, which allows no ambiguity in the natural language arguments. However, the noisy data can still benefit the training with the RTW module.

Structural Validity

We then evaluate the structural validity of the augmented LFs w.r.t. three rules: (1) functions are valid according to the logic schema defined in [20]; (2) the types and numbers of arguments are consistent with the pre-defined function schema;

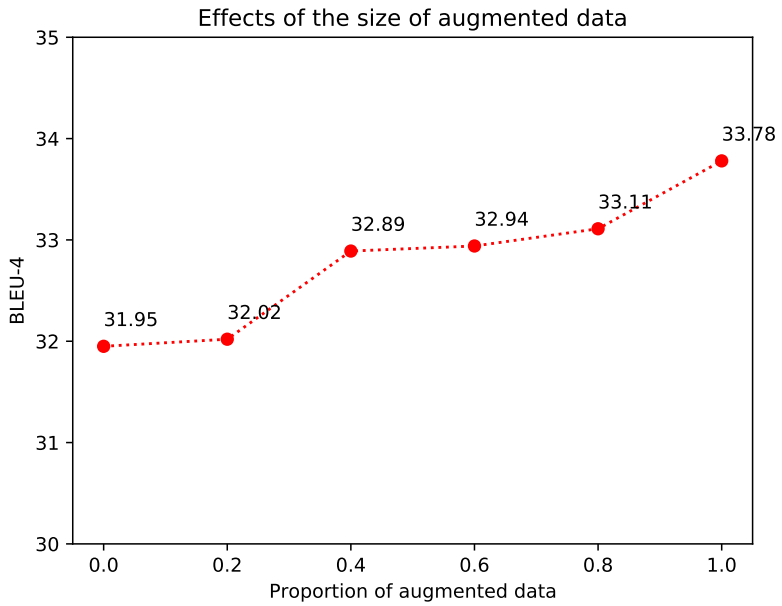


Figure 3.3: The proportion of augmented data vs. the BLEU-4 score on the test set of LOGIC2TEXT.

(3) the LF sequence can be successfully parsed into a tree-form LF. As shown in Table 3.5, 96.66% of the LFs are valid, proving the effectiveness of the proposed DA model in learning the logical form structure.

3.4.9 Effects of the Size of Augmented Data

We analyze the effects of the amount of augmented data by varying the proportion of the data. Specifically, we randomly sample from the full augmented data with proportions 0.2, 0.4, 0.6, and 0.8, and evaluate the performance on the LOGIC2TEXT task. As illustrated in Figure 3.3, the size of augmented data matters. Significant gains over the supervised baseline can be observed with only 40% of the augmented data. While the difference within 0.4, 0.6, and 0.8 seems relatively marginal, we can expect more gains if the augmented dataset is scaled up.

fahed attal

date	venue	score	result	competition
16 february 2006	bahrain national stadium , manama	2 - 0	2 - 0	friendly
18 february 2006	bahrain national stadium , manama	1 - 0	1 - 0	friendly
18 february 2006	bahrain national stadium , manama	2 - 0	2 - 0	friendly
1 march 2006	king abdullah stadium , amman	1 - 0	1 - 0	2007 afc asian cup qualifier
1 april 2006	bangabandhu stadium , dhaka	2 - 0	11 - 0	2006 afc challenge cup

Logic type : *unique*
Logical Form: and { only { filter_eq { all_rows ; date ; march 2006 } } ; eq { hop { filter_eq { all_rows ; date ; march 2006 } ; score } ; 1 - 0 } } = true

Gold: the only score in march 2006 was at the 2007 afc asian cup qualifier .
GPT-2: the only time fahed attal scored a goal in a friendly competition was on march 6th .
GPT-2+ST: the only time fahed attal scored a goal was on march 26 , 2006
Ours: the only game that fahed attal scored in the month of march was on march 1 , 2006 .
Ours – RTW: fahed attal scored only one international goal in the month of march .
Ours – RTW&PL: the only time fahed attal scored a goal in a friendly competition was on march 6th .

Figure 3.4: A qualitative example sampled from the test set. Some irrelevant parts of the table are hidden. Incorrect information in the model outputs is marked as red. The correct entity generation is marked as green.

3.4.10 Qualitative Examples

Here, we show a qualitative example of model generations in Figure 3.4. We can infer that the difficulty of this example mainly lies in the correct generation of the date entity *1 march 2006*. There is a slight mismatch between the table and the logical form as the logical form only provides the information of month (*march*) and year (*2006*), without the day (*1*). This leads to hallucinations of GPT-2, GPT-2+ST and Ours - RTW&PL in generating the incorrect day. Ours - RTW can generate a factually correct sentence, but with only the month information. By contrast, only our full model (Ours) can correctly generate the full entity *march 1, 2006* in the table, although the meaning is slightly different from the gold reference. However, the model outputs are still limited to the information in the logical form, while a human-written sentence (Gold) often contains additional

	All	Correct	Incorrect
Texts	0.65±0.20	0.70±0.19	0.64±0.20
Logical forms	0.86±0.16	0.92±0.12	0.85±0.16

Table 3.6: Mean and standard deviations of the data weights for different subsets of the augmented data.

supporting information such as *2007 afc asian cup qualifier* which comes from the table. This suggests potential future work on better information fusion of the table and logical form.

3.4.11 Effectiveness of Round-trip Data Weighting

In Section 3.4.8, we measured the factual accuracy of the augmented data, which showed 13.47% of the logical forms and 27.85% of the texts are logically supported by the tables. However, we adopted a soft data weighting approach instead of hard data filtering based on factual correctness. This is because factual correctness is evaluated based on exact match based execution of the augmented logical forms and the back-translated logical forms from the augmented texts. Therefore, it may strictly filter out some logically correct examples. In comparison, our round-trip data weighting method tends to assign lower scores to low-quality augmented data while still considering their potential effects. To analyze whether round-trip data weighting can measure the quality of augmented data, we split the augmented dataset ("All") into "Correct" and "Incorrect" based on their exact factual correctness. In Table 3.6, we show that "correct" data tend to have higher round-trip weights.

We show some qualitative examples in Section 3.4.12 to showcase the potential effects of imperfect augmented data.

3.4.12 Qualitative Examples of TopicDA

We list two examples of our TopicDA method, which show perfect quality in terms of topic consistency and logical fidelity. In Figure 3.5, we show the augmented logical form generated by assigning logic type *superlative* to the given table, which describes correct information in the table and matches the given topic. We also list its forward and round-trip translations here, which were used to compute

utah jazz all - time roster

player	nationality	position	years for jazz	school / club team
rick adelman	united states	guard	1974 - 75	loyola (ca)
john amaechi	england	center / forward	2001 - 03	penn state
louis amundson	united states	forward	2007	unlv
j j anderson	united states	forward	1982 - 85	bradley
shandon anderson	united states	guard / forward	1996 - 99	georgia
rafael araãjo	brazil	center	2006 - 2007	byu
carlos arroyo	puerto rico	guard	2002 - 05	florida international
isaac austin	united states	center	1991 - 93	arizona state
anthony avent	united states	forward	1998 - 99	seton hall

Logic type : *superlative*
Augmented Logical Form: eq { hop { argmin { all_rows ; years for jazz } ; player } ; rick adelman } = true
Forward translation: rick adelman was the earliest player to join the utah jazz all - time roster .
Round-trip translation: eq { hop { argmin { all_rows ; years for jazz } ; player } ; rick adelman } = true

Figure 3.5: An example of an augmented logical form generated via TopicDA.

duneland athletic conference

school	location	mascot	county	enrollment 08 - 09	ihsaa class	ihsaa class football	year joined	previous conference
chesterton	chesterton	trojans	64 porter	1921	4a	6a	1970	calumet
crown point	crown point	bulldogs	45 lake	2426	4a	6a	1993	lake suburban
lake central	saint john	indians	45 lake	3141	4a	6a	2003	independents
laporte	laporte	slicers	46 laporte	1956	4a	5a	1976	northern indiana
merrillville	merrillville	pirates	45 lake	2241	4a	6a	1975	lake suburban
michigan city	michigan city	wolves	46 laporte	1919	4a	5a	1995	none (new school)
portage	portage	indians	64 porter	2618	4a	6a	1970	calumet
valparaiso	valparaiso	vikings	64 porter	2072	4a	6a	1970	independents

Logic type : *ordinal*
Augmented Text: in the duneland athletic conference , the school with the 2nd highest enrollment is portage .
Forward translation: eq { hop { nth_argmax { all_rows ; enrollment 08 - 09 ; 2 } ; school } ; portage } = true
Round-trip translation: portage has the 2nd highest enrollment in the duneland athletic conference .

Figure 3.6: An example of an augmented text generated via TopicDA.

1972 miami dolphins season

week	date	opponent	result	record
1	august 5 , 1972	detroit lions	l 23 - 31	0 - 1
2	august 12 , 1972	green bay packers	l 13 - 14	0 - 2
3	august 19 , 1972	cincinnati bengals	w 35 - 17	1 - 2
4	august 25 , 1972	atlanta falcons	w 24 - 10	2 - 2
5	august 31 , 1972	washington redskins	l 24 - 27	2 - 3
6	september 10 , 1972	minnesota vikings	w 21 - 19	3 - 3

Logic type : *unique*
Augmented Text: in the 1972 miami dolphins season , the only time the opponent was green bay packers was on august 19th .
Forward translation: and { only { filter_eq { all_rows ; opponent ; green bay packers } } ; eq { hop { filter_eq { all_rows ; opponent ; green bay packers } ; date } ; august 19 , 1972 } } = true
Round-trip translation: in the 1972 miami dolphins season , the only time the opponent was the green bay packers was on august 19 , 1972 .

Figure 3.7: An incorrect example of an augmented text generated via TopicDA.

its round-trip data weight. These translations are of considerable quality, indicating a high data weight during semi-supervised training. Similarly, Figure 3.6 demonstrates a good example of text augmentation and its translations. However, sometimes the DA models can generate factually incorrect outputs. As shown in Figure 3.7, although the augmented text is consistent with the assigned topic "unique", the correct date should be "august 12" instead of "august 19". Nonetheless, it is still close to a perfect generation. In addition, its forward translation logical form is grammatically correct and semantically consistent with the text, which suggests that this augmented text may still benefit the models on the translation between logical forms and texts. Our data weighting method will also assign a high score to this example because the round-trip translation perfectly matches the original text.

3.5 Time Complexity of the Proposed Method

Here, we provide an analysis of the time complexity of the models in our experiments. Let N_s denote the number of supervision data, N_{ut} the number of augmented texts and N_{ul} the number of augmented logical forms. Let $T(n)$ denote the time complexity of training a model on n examples for one epoch, and

$I(n)$ the time complexity of sequence generation on n examples for one epoch. We assume that $T(a+b) = T(a) + T(b)$ and $I(a+b) = I(a) + I(b)$. Within **each iteration** of joint training, the time complexity of our method is as follows:

1. The generation of pseudo-parallel data, which costs $I(N_{ut} + N_{ul})$.

2. The round-trip generation for computing round-trip weights, which again costs $I(N_{ut} + N_{ul})$. The computation of round-trip weights is omitted because it costs relatively trivial time.

3. The pretraining on pseudo-parallel data, which costs $2 \times M_p \times T(N_{ut} + N_{ul})$, where M_p is the number of pretraining epochs. The 2 here indicates that we have two models to train, both a LOGIC2TEXT and an LG model.

4. The finetuning on supervision data, which costs $2 \times M_f \times T(N_s)$, where M_f is the number of finetuning epochs. Again, 2 indicates the training of two models.

In total, the time complexity of one iteration is $2 \times I(N_{ut} + N_{ul}) + 2 \times M_p \times T(N_{ut} + N_{ul}) + 2 \times M_f \times T(N_s)$. Similarly, we also list the overall time complexity of all the models we experiment with in Table 3.7. Compared with the base model GPT-2, the additional time cost of our models mainly comes from the inference and training on additional pseudo data, which is much larger than the supervision data, i.e., $N_{ut} > N_s$ and $N_{ul} > N_s$. However, this is a common trade-off of using semi-supervised learning approaches, where large-scale unlabeled data are exploited to boost the performance. As shown in the table, the time complexity of our full model is asymptotically close to the standard self-training strategy GPT-2+ST, where re-training student models is also used to avoid overfitting on pseudo-parallel data.

3.6 Summary

In this chapter, we introduce our first approach, data augmentation for logical NLG with explicit logical form control. We herein proposed a topic-conditioned data augmentation method to generate logical forms and textual descriptions with auxiliary tasks. We also introduced logical form generation as a dual task of logical-level NLG, and proposed a joint semi-supervised learning approach to

Model	Time Complexity
GPT-2	$2 \times M \times T(N_s)$
GPT-2+ST	$R \times (I(N_{ut} + N_{ul}) + M_p \times T(N_{ut} + N_{ul}) + 2 \times M_f \times T(N_s))$
Our full model	$R \times (2 \times I(N_{ut} + N_{ul}) + 2 \times M_p \times T(N_{ut} + N_{ul}) + 2 \times M_f \times T(N_s))$
-RTW	$R \times (I(N_{ut} + N_{ul}) + 2 \times M_p \times T(N_{ut} + N_{ul}) + 2 \times M_f \times T(N_s))$
-RTW&PL	$R \times (I(N_{ut} + N_{ul}) + M_p \times T(N_{ut} + N_{ul}) + 2 \times M_f \times T(N_s))$

Table 3.7: The time complexity of the models. R is the number of iterations ($R \leq 3$ in practice) . M is the number of training epochs for GPT-2. In practice, we set $M_p = 4, M_f = 7, M = 50$ (early-stopping is not considered here). We also omit the time of warm-up pretraining of the semi-supervised methods because it is relatively trivial.

improve these two tasks with augmented data. The experimental results demonstrated the effectiveness of the proposed method in data augmentation, which requires no external resources. Our approach shows a novel pipeline from augmenting unpaired LFs/texts to generating pseudo-parallel data with semi-supervised learning, effectively addressing the data scarcity problem of using explicit LFs for logical NLG.

4 Augmenting Logical Forms to Improve Logical NLG via Implicit Knowledge Transfer

The method introduced in the last chapter treats LFs as explicit control features. With LFs as mediators conveying accurate logical-level facts, models can focus on surface realization from associated LFs and achieve high fidelity. However, such a task setting requires parallel <LF, text> data, while the annotation of LFs aligned with logical descriptions requires intensive human labor. While our approach alleviates the data scarcity problem by augmenting additional parallel data from tables, the requirement of a readily available parallel corpus cannot be avoided. While Chen et al. [20] annotated the LOGIC2TEXT dataset that enables our research, such parallel data may not be available in real-world scenarios, which limits the potential applicability of our method. Another concern is about the model's interface. For each generation, an LF must be provided in advance, while providing such LFs is non-trivial and implausible for end users as it also requires content selection and logical inference over tables. Therefore, the model for this kind of LF-to-text generation may only serve as a sub-module in a real NLG system.

In this chapter, we propose a second approach to address logical NLG in the direct table-to-text manner. We avoid the explicit usage of LFs but treat LFs as implicit mediators to improve the learning of logical inference.

Specifically, we propose a **P**retrained **Lo**gical Form **G**enerator (**PLOG**) framework, in which the model 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 natural language generation. Our insights are three-fold.

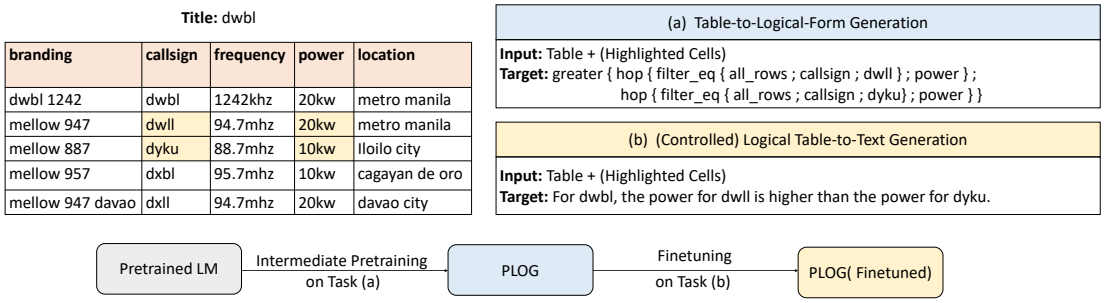


Figure 4.1: Examples of the tasks and the training procedure of our proposed PLOG model. Task (a) is the table-to-logic pretraining task we propose; task (b) is the downstream logical table-to-text task we target. The yellow-colored table cells are annotated as control features for the ContLog task, 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.

1. Unlike natural language sentences, LFs 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.

2. It is viable to collect large-scale logical form corpora via rule-based sampling over tables without the efforts of human annotators.

3. Via pretraining on large amounts of table-to-logic data, the proposed model can better understand the table and organize the content planning with logical inference, leading to faithful table-to-text generation.

Here, we treat logical forms as intermediate meaning representations of logical-level texts, while we do not need them when performing the downstream task. This setting averts the explicit alignment of the LFs used in pretraining and the texts used in finetuning. This kind of task transfer learning techniques have been widely-explored in the representation learning of tabular data [69, 49, 33, 28]. The basic idea is to reuse the PLMs pretrained on unstructured text corpora, and perform an intermediate pretraining for these PLMs on well-designed pretraining

tasks to enhance the model's understanding of structured data [28]. Synthetic pretraining data is collected via self-supervision or data augmentation. Such table pretraining models are applicable to various downstream tasks such as table question answering [33, 69, 49], but have not been explored well in the table-to-text task. Our approach is also the first trial of using formal language in pretraining models for table-to-text generation.

To realize the table-to-logic pretraining, it is crucial to construct a table-to-logic dataset, comprising of <table, LF> pairs, where each LF is factually supported by the table. In Chapter 3, we developed an auxiliary table-to-logic model, designed to generate augmented LFs from existing tables. This model underwent pretraining on the LOGIC2TEXT dataset to encourage the production of diversified logical forms given a table and assigned logic types. However, the model's effectiveness was limited, primarily because the generated LFs were prone to factual inaccuracies. In fact, as highlighted in Chapter 3 (Table 3.5), the factual accuracy was only 13.47%. This limitation can be attributed to the constrained size of the training data within the LOGIC2TEXT dataset.

Such factual inaccuracies is acceptable for the research in Chapter 3, as logical consistency between LFs and texts is prioritized when using LFs for explicit guidance. However, in this chapter, our focus shifts to training a table-to-logic model that serves as the foundation for the table-to-text model. Overcoming the scarcity of LF data is crucial, necessitating the collection of large-scale, factually accurate LFs. This allows the table-to-logic model to learn precise logical reasoning knowledge. Therefore, using the model-based data augmentation method from the previous chapter is not suitable.

To address this, we introduce a rule-based method for LF data augmentation, comprising a *templatization* phase and an *instantiation* phase. This process allows us to collect factually correct LFs via sampling from tables. For the templatization phase, we manually construct a set of abstract templates for LFs. These templates comprise function nodes, with several placeholder nodes allocated for future instantiation based on specific tables. These templates are extracted from the LFs in the LOGIC2TEXT dataset to ensure a distribution closely aligned with downstream tasks. In the instantiation phase, we propose an execution-guided bottom-up sampling method to fill in the placeholder nodes of the LF templates with concrete objects and operations. This sampling is achieved through a post-order execution of the LF tree, thereby ensuring the accuracy of the instantiated

LFs.

Compared to the model-based method in Chapter 3, our rule-based method offers several advantages:

- It ensures 100% factual accuracy of the augmented LFs through the execution-guided sampling strategy, addressing the issue of factual errors in LFs compared to the model-based method.
- It enables the generation of diverse LFs. We can select LF templates with various logic types and combinations of logical operations, thereby promoting logic-diversified augmented data. Additionally, randomness during the sampling phase allows for the creation of various candidate LFs from the same table and LF template. In contrast, the model-based method does not solve the problem of diversifying outputs from the same input, making its diversity constrained by the control features during DA model training.
- It does not necessitate existing parallel datasets for training a DA model. Although we extract LF templates from the LOGIC2TEXT dataset, such templates can also be seen as constructed based on the LF definition. Notably, the DA model is essentially an outcome of this chapter's proposed table-to-logic pretraining framework. From this aspect, training a table-to-logic model with LFs augmented via another pretrained table-to-logic model is unreasonable.

For the model training, 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 NLG, LOGICNLG, lacks control features, leading to uncontrollable content selection, we collect 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 4.1 presents examples of the table-to-logic pretraining task and the (controlled) logical NLG task.

On the two benchmarks, LOGICNLG and CONTLOG, PLOG outperforms the strong pretrained baselines 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 NLG.

The rest of this chapter is organized as follows. In section 4.1, we introduce the related work on table pretraining techniques. In Section 4.2, we introduce the downstream task formulation of logical NLG and the construction of the CONTLOG dataset. Section 4.3 describes the formulation of the proposed table-to-logic pretraining task, including the task design and the collection of pretraining data. Section 4.4 introduce the design of PLOG models, such as model architectures, the model input and output. Section 4.5 describes the experiments on LOGICNLG and CONTLOG with the proposed method. Then, we employ a comparison experiment to analyze the model performance with different model interfaces in Section 4.6.

4.1 Related Work

Table pretraining [33, 69, 28, 49] has been popular for table understanding tasks such as Table Question Answering (TableQA) [140, 88] and Table Fact Verification (TableFV) [16]. 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 [69] 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. Another line of related works adopts pretraining techniques to solve the text-to-SQL parsing [134, 109] 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 pretraining 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 (PLMs) by flattening structured tables into sequences and finetuning on in-domain data [40, 52, 130]. For logical NLG, the prior works [17, 125] are also based on the direct finetuning of PLMs that were pretrained on textual corpora. Compared to these studies, our approach shows a novel way to enhance the logical reasoning ability of PLMs with transfer learning. Andrejczuk et al. [3] have incorporated structural positional embeddings of tables into T5 [98] and performs intermediate pretraining in a similar way to TaPas [33]. Similarly, PLOG can also be seen as an intermediate pretraining of PLMs for table-to-text generation, but it is the first trial of using LFs as the pretraining resource.

4.2 Downstream Tasks

In this work, we focus on table-to-text generation without explicit LFs. The previous benchmark LogicNLG aims at generating sentences from a full table[1] without control features, which causes uncontrollable content selection and hinders faithful generation [20]. 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 [87, 22], we incorporate highlighted table cells as additional supervision signals in ContLog (Figure 4.1) to narrow down the scope of content selection. From the view of real-world applications, the highlighted cells are easy to specify and directly reflect the region of interest to end users [48]. Unlike the Logic2Text task setting, the models trained on ContLog still need to reason over the table for content planning, which shows a reasonable balance between controllability and flexibility.

[1] In practice, we follow [17] to detect the relevant columns with the target text and only include these columns in the input. While this brings some controllability, the columns still lead to a large search space for the content selection.

4.2.1 CONTLOG Dataset Construction

We reuse the LOGIC2TEXT dataset to build CONTLOG. In LOGIC2TEXT, there is an annotated LF for each target sentence. As described in Section 2.2.1, each LF can be regarded as an executable program composed of combinations of multiple logical operations and the LFs can be divided into seven logic types. All the relevant table columns and entities involved in the target sentence are also explicitly given in the LF. Therefore, we seek to reuse the annotation of LFs in LOGIC2TEXT to automatically extract highlighted table cells without human annotation. A simple way for this is to directly extract the corresponding columns with a simple string match between the LF and the table's column headers. However, highlighted cells may be random local regions of a table rather than a full column. Hence, we aim to design more fine-grained control features for this task, for which we execute each LF in LOGIC2TEXT against the context table and extract the table cells relevant to the execution process. The selection of final table cells is based on manually designed rules. The highlighted cells may comprise multiple columns, a single row, or scattered table cells. Finally, we obtain CONTLOG, a re-purposed version of LOGIC2TEXT with the same examples but varied control features. Figure 4.1 shows an example of CONTLOG.

4.2.2 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 $P(y|T; H)$. It also contains a logic type c for each example, which will serve as an optional feature in our experiments.

4.3 Table-to-Logic Pretraining

To achieve faithful logical table-to-text generation, we propose the table-to-logic pretraining task that involves generating an LF from an input table. In this task, the model needs to mine logical-level facts from tables and organize the facts into formally defined LFs. Each logical form can be regarded as an abstract content plan of a logical-level description.

4.3.1 Pretraining Task Formulation

The input of the pretraining task is the same (sub-) table as we introduced in Section 4.2.2, while the target is an LF instead of a sentence. The definition of LFs is the same as we introduced in Section 2.2.1. An LF can be represented as a tree-structured program or just a linearized string form language, as shown in Figure 1.2. We adopt the string form as the generation target to enable sequence-to-sequence generation.

4.3.2 Evaluation of Table-to-Logic

The pretraining task test the ability of models to generate a logical form string that can be parsed into a tree-structured program, and the program can be executed on the table and returns a *true* output that indicates the table entails it. Such an evaluation implies both grammatical and semantic correctness:

1. The generated LF string must not have syntactical errors. This requires correct operation names and structural indicators such as '{', '}', and ';'. A correct LF must be successfully parsed back to a tree structure.

2. The tree-structured LF must pass the execution verifier.

Therefore, we adopt the Execution Accuracy of generated LFs as the evaluation metric for the pretraining task, the same as the metric used for LG evaluation in Chapter 3.

4.3.3 Pretraining Data Collection

To perform table-to-logic pretraining, we must collect enough <table, LF> pairs. The formal definition of logical forms allows us to automatically collect a large

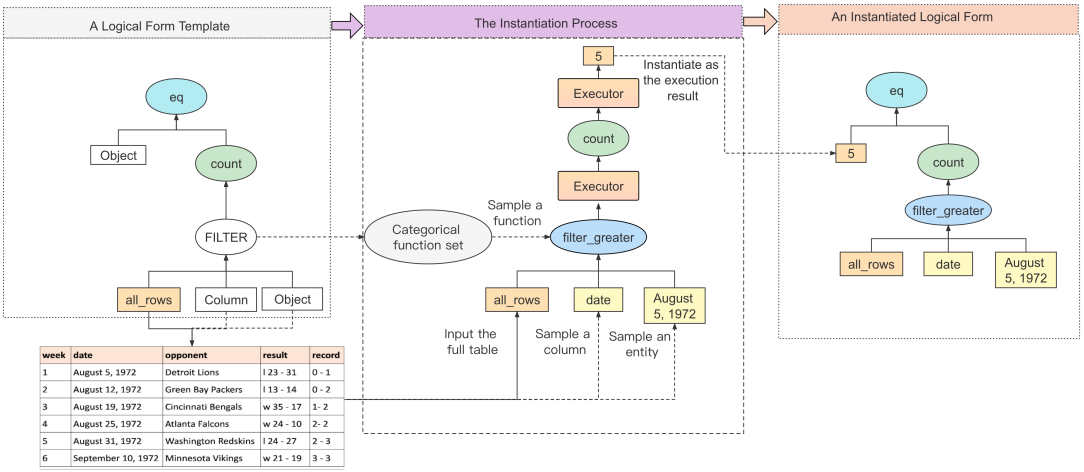


Figure 4.2: An example of instantiating a logical form template. The colored nodes in the template indicate nodes that do not need instantiation, while the white-background nodes are typed placeholders in the template. The dotted arrows indicate instantiation. We employ instantiation of these white nodes 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.

amount of LFs from tables via rule-based sampling. Here, we propose instantiating existing logical form templates to sample logical forms similarly to how prior studies collect SQL programs [141, 69]. Specifically, we extract abstract templates from the logic schema used in LOGIC2TEXT. Then we adopt an execution-guided bottom-up sampling method to instantiate the templates based on the context tables.

Templatization

We first extract the templates based on the LF 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

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

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 4.1 shows the list of these function placeholders. The other functions that cannot be categorized need not instantiation. Finally, we obtained 35 templates, an average of 5 for each logic type.

Instantiation

We propose an execution-guided bottom-up sampling strategy to instantiate the template trees. An example of template instantiation is depicted in Figure 4.2. We handcraft a set of complete rules to instantiate different placeholder nodes. For example, we uniformly sample a column from the table to instantiate a `Column` placeholder (e.g. `date` in Figure 4.2). For a function placeholder such as `FILTER`, we sample a specific function from the corresponding category it represents (e.g. `filter_greater` in Figure 4.2).

The instantiation is performed in a post-order traversal of the LF tree. 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. Here, we list all the instantiation rules regarding different placeholders. Note that each placeholder is sampled by considering the input view (a full table or sub-table returned by child functions)

to the current function node.

1. For placeholder type `Column`, we randomly sample a column header from the current input view.

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 view. 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 judgments.

3. The instantiation of a function-type placeholder depends on its function category, as listed in Table 4.1. If the placeholder belongs to the function category `COMPARE` or `MAJORITY`, we choose the specific function name based on the actual 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 will be uniformly sampled from the function set.

We show more examples of the templates and instantiated LFs in Table 4.2.

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.

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 LFs that have appeared in LOGIC2TEXT since they are semantically equal to some of

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 4.2: 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 4.1, and a textual explanation of the instance.

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 LFs into train/validation/test sets. The statistics of the pretraining data and their corresponding downstream datasets are shown in Table 4.3. Although we can sample more logical forms with more trials, we find the current pretraining data enough to obtain ideal experimental results.

4.4 Model Details

In this section, we introduce the details of our proposed model PLOG and how we conduct the sequence-to-sequence generation for the pretraining and downstream tasks.

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

4.4.1 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 [96], BART, and T5. We test different backbone models, including BART-large, T5-base, and T5-large.

4.4.2 Model Input

Similarly to prior work on table-to-text generation [52, 87], we employ a template-based method to serialize the input table. For the LogicNLG task, we follow [17] 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 [87]. This is to avoid the over-length issue with pretrained models and the negative impacts caused by irrelevant table information.

4.4.3 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 preprocessing step by pre-computing some potentially useful numerical values. Similar approaches have also been proposed to improve the fidelity in table summarization [116] and text-to-SQL tasks [138]. 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>.

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. The templates for the two strings are as follows.

c_j^{sum} : <sum_cell> sum_val <col_header> Col_j </col_header> </sum_cell>.

c_j^{avg} : <avg_cell> avg_val <col_header> Col_j </col_header> </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>.

Figure 4.3 shows an example of the input serialization.

4.4.4 Model Output

We linearize each logical form into a string via a pre-order traversal of the logic tree following [20]. Special punctuations such as semicolons and braces are used to indicate the structural relationships between functions. For example, the logical form instance in Figure 4.2 can be linearized into `eq { count { filter_greater { all_rows ; date ; August 5, 1972 } } ; 5 }`. 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.

4.5 Experiments

4.5.1 Experimental Settings

Evaluation Metrics

Following prior works [17, 19] on LogicNLG, we evaluate our models on both surface-level matching metrics and logical fidelity scores. Surface-level metrics include BLEU-1/2/3, and the logical fidelity metrics include SP-Acc, NLI-Acc,

Title: Medal Table from Tournament

Nation	Gold Medal	Silver Medal	Bronze Medal	Sports
Canada	3	1	2	Ice Hockey
Mexico	2	3	1	Baseball
Colombia	1	3	0	Roller Skating

Input serialization

<table> <caption> Medal Table from Tournament </caption> <sum_cell> 6 <col_header> Gold Medal </col_header> </sum_cell> <avg_cell> 2 <col_header> Gold Medal </col_header> </avg_cell> ... <cell> Canada <col_header> Nation </col_header> <row_idx> 1 </row_idx> </cell> <cell> 3 <col_header> Gold Medal </col_header> <row_idx> 1 </row_idx> <max_rank> 1 </max_rank> <min_rank> 3 </min_rank> </cell> ... </table>

Figure 4.3: An example of input serialization in PLOG. The red-colored words represent aggregation cells obtained via numerical pre-computation. The green-colored words indicate the numerical ranks obtained via numerical pre-computation.

TaPas-Acc and TaPEx-Acc. 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. The details of these metrics are described in Section 2.2.2.

Models for Comparison

For LogicNLG, we compare our method with the following previous models:

1. BERT-TabGen [17], a pretrained generative model based on BERT, where the input table is serialized into a sequence via a text template. Two variants are built on BERT-base (sm) and BERT-large (lg), respectively.

2. GPT-TabGen [17], similarly to BERT-TabGen, but with GPT-2 as the base model. Two variants adopt GPT-2-small (sm) and GPT-2-medium (med), respectively.

3. GPT-Coarse-to-Fine [17], a coarse-to-fine generation model based on GPT-TabGen. The difference is that the model first generates a masked template of the target text, then realizes the template into the final surface form. The templates are constructed via masking entities in the target text that can be linked with the table. Two variants adopt GPT-2-small (sm) and GPT-2-medium (med), respectively.

4. DCVED + GPT-TabGen [19], a de-confounded variational encoder-decoder model with GPT-TabGen as the base model. It introduce a latent variable as the mediator between the table and text.

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 preprocessing method introduced in Section 4.4. 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 [128] and PyTorch [89]. 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 ContLog, 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.

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 [71].
- 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 [108].
- Learning rate: 2×10^{-4} for both pretraining and finetuning.
- Batch size: 10 (2×5 gradient accumulation steps, the same below) 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 finetuning on both datasets.
- Batch size: 256 ($4\times$ 64) for pretraining on LogicNLG and 32 (4×8) on ContLog; 32 (4×8) for finetuning on both datasets.

Model	Surface-level Evaluation			Logical Fidelity			
	BLEU-1	BLEU-2	BLEU-3	SP-Acc	NLI-Acc	TAPEX-Acc	TAPAS-Acc
BERT-TabGen (sm)	47.8	26.3	11.9	42.2	68.1	–	–
BERT-TabGen (lg)	49.1	27.7	13.5	44.4	73.9	–	–
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
GPT-TabGen (med)	49.6	28.2	14.2	44.7	74.6	–	–
GPT-Coarse-to-Fine (med)	49.0	28.3	14.6	45.3	76.4	–	–
DCVED + GPT-TabGen(sm)	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 4.4: The experimental results of different models on the test split of LOGICNLG. The previous models adopt different sizes of PLMs as the base model, including small (sm) models and medium-large models (med/lg). 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.

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. The pretraining is 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 about five 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.

4.5.2 Automatic Evaluation

Table 4.4 presents the results on LOGICNLG. We can observe that the BART and T5 models with our preprocessing strategies outperform all the previous models

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 4.5: 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.

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.

The results on CONTLOG are shown in Table 4.5. As observed, PLOG models outperform their base counterparts consistently on both 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 demonstrate the general effectiveness of our approach. The reason for the improvements in BLEU and ROUGE scores may be attributed to the fact that the table-to-logic pretraining brings extra effects in improving the table understanding ability of the models.

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 4.6: 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 [35].

4.5.3 Human Evaluation

To further investigate whether the models can generate faithful sentences, we perform a human evaluation 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 [17]:

- 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 from 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

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

as correct[2]. The evaluation is only based on the context table without considering gold references, because the generated sentences may not describe the same fact as the references do but still present high quality in terms of fidelity and fluency.

As shown in Table 4.6, 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.

4.5.4 Table-to-Logic Results

We report the Execution Accuracy of our pretrained models on the table-to-logic pretraining task in Table 4.7. 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 were 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 brackets. 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 [121] may al-

[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.

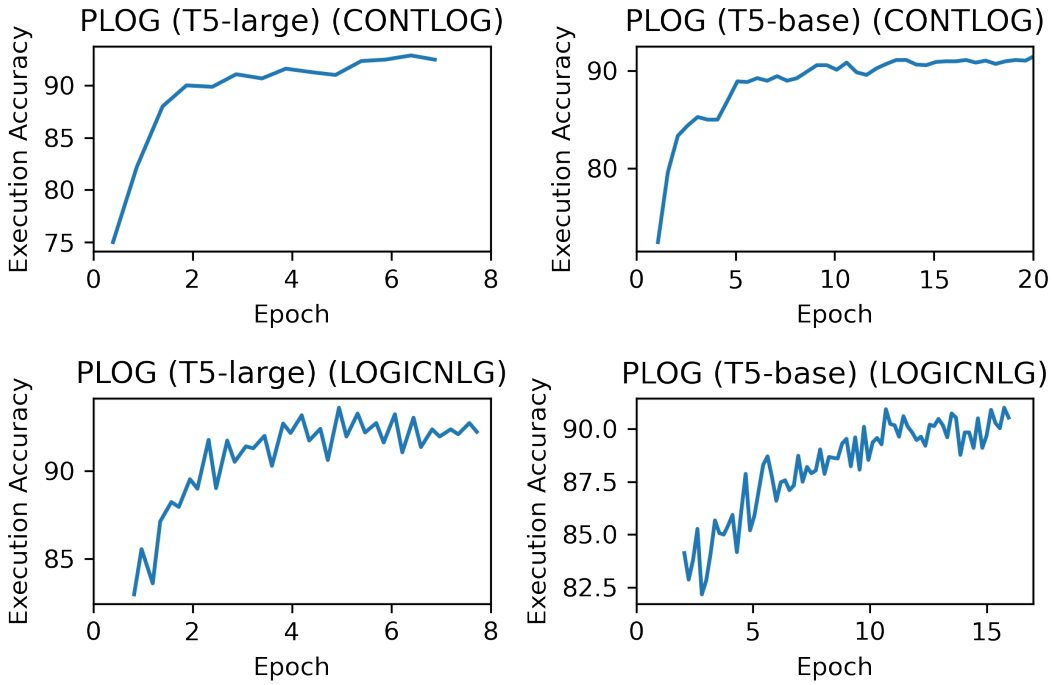


Figure 4.4: Validation results of table-to-logic pretraining with T5-base and T5-large as the backbones. The results of LogicNLG pretraining and ContLog pretraining are shown at different intervals for better illustration. The results within the first 160k steps are not computed. The results of PLOG (BART-large) is not shown here due to the unstable performance of BART-large in generated executable LFs.

leviate 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. Figure 4.4 presents the trend of validation results of pretraining during the training process. We can observe that the models achieve higher accuracy when trained for more epochs.

4.5.5 Verification Experiments

To demonstrate the robustness of our models and the credibility of experimental results, we conduct verification experiments for the experimental results on the

Model	Surface-level Evaluation					Logical Fidelity	
	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-4	ROUGE-L	TAPEX-ACC	TAPAS-ACC
T5-base	31.0	61.6	38.1	17.8	50.9	66.8	71.0
PLOG (T5-base)	**31.8**	**63.0**	**39.3**	**18.4**	**52.7**	**77.6**	**74.9**
T5-large	31.6	62.5	38.7	**18.1**	51.8	75.9	71.4
PLOG (T5-large)	**31.9**	**63.8**	**39.2**	18.0	**52.6**	**83.1**	**78.8**
BART-large	30.2	61.4	37.5	17.0	50.5	70.4	67.2
PLOG (BART-large)	**32.1**	**63.7**	**39.7**	**18.3**	**53.1**	**83.1**	**78.3**

Table 4.8: The verification results on the test split of CONTLOG. We rerun both the pretraining and finetuning, and show an average of two runs on the finetuning.

CONTLOG dataset in Section 4.5.2. We re-run the table-to-logic pretraining with different hyperparamters[3] and re-select the pretrained checkpoints for finetuning. For finetuning both the base models and the PLOG models, we run two times of experiments with different random seeds and report the average scores. Due to the computational cost of pretraining large models, we early-stop the pretraining within only two epochs, leading to slightly lower results of PLOG models on some metrics than the results in Table 4.5. However, the results of PLOG models still surpass the base models by a large margin on fidelity, demonstrating the robustness of our framework with varied pretrained checkpoints and hyperparameters.

4.5.6 Analysis on Different Logic Types

In CONTLOG, each target sentence belongs to a pre-defined logic type inherited from LOGIC2TEXT, allowing us to analyze the performance of models on different logical reasoning types. In Figure 4.5, we can observe that our PLOG models generally improves the performance of their base models on most logic types, especially on *superlative* and *ordinal*. The models perform closely on the *unique* and *majority* types, but improvements still can be observed. However, we still observe a considerable amount of incorrect generations of all the models, suggesting the potential room for improvement in the future. In particular, on *comparative*,

[3]This is a post-hoc experiment for reproduction and verification, so the experiments are done on different devices compared to the original experiments.

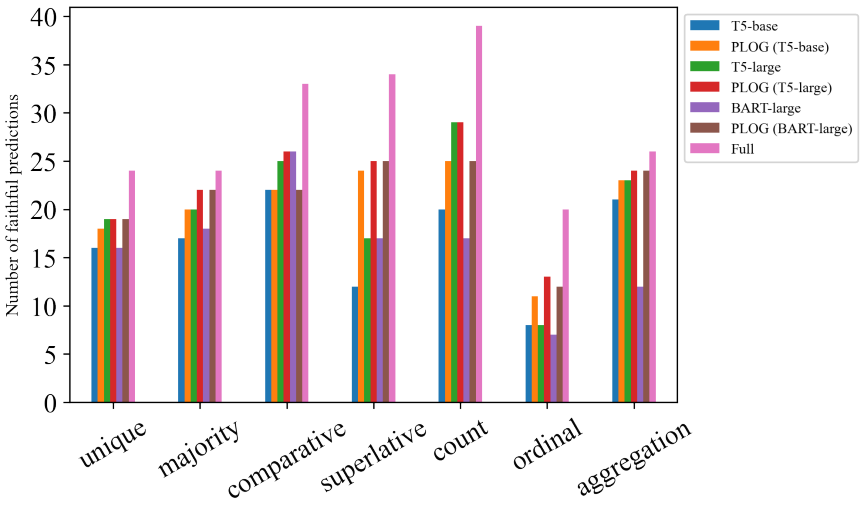


Figure 4.5: The human evaluation results of different models on the different logic types of ConTLog. The y-axis indicates the number of samples scored as correct. *Full* indicates the number of samples of each logic type in the 200 human evaluation samples.

superlative, *count* and *ordinal* types, there are more wrong generations.

4.5.7 Qualitative Examples

We further conduct a case study by showing some qualitative examples of model generations. As presented in Figure 4.6, the base models without table-to-logic pretraining are prone to factual errors in the generations. As can be observed in Figure 4.6a, the models are expected to detect the correct ranks of the `area (km 2)` column and associate the ranks with corresponding countries. However, all the base models fail to detect `tana river` as the country with the highest area. By contrast, the PLOG models all produce faithful descriptions. PLOG (BART-large) even describes a distinct fact from the others, which involves an ordinal operation on the ranks. In Figure 4.6b, the example is even more difficult in that the `location attendance` column contains mixed data types of both entities and numerical values. The base models T5-base and BART-large have made similar mistakes in misunderstanding the meaning of the attendance numbers. T5-large fails to detect the highest attendance. The PLOG models all present perfect faithfulness, while PLOG (BART-large)'s generation is less fluent than

the others as it does not understand the role of `Miami`. This showcases the difficulty of surface realization in logical NLG.

4.6 Comparison of Model Interfaces for Logical NLG

To gain more insights about the difficulties of different interfaces of logical NLG models, we conduct a comparative experiment of three model interfaces with varying control features, including:

- No control: the input is the full table without any specification for the generation.
- Highlighted cells: the input is a table with highlighted table cells.
- Logical forms: the input is a table and a logical form.

We use LOGIC2TEXT as the evaluation dataset, and for highlighted cells, we use the annotation of CONTLOG. We finetune a T5-large model with the three interfaces, denoted as *Table (T5)*, *Table+HL (T5)* and *Table+LF (T5)* in Table 4.9. We also report the performance of using highlighted cells but with the pretrained PLOG (T5-large) model as the basis (*Table+HL (PLOG)*). We can observe that the interface with explicit LF control eases the generation and leads to the best performance on both surface-level and fidelity metrics with the same model as the other interfaces. By contrast, the table-only interface is the most difficult, leading to the lowest results, which can be attributed to the uncontrollable content selection given only a table. The Table+HL interface (Table+HL (T5)) results in a much better performance than the table-only interface but is obviously more difficult than the LF-control interface.

The three interfaces are not fairly compared since additional supervisory signals are included in the control features. However, we can obtain insights into the relationship between the controllability and the difficulty of the interfaces. In real-world NLG applications, it is crucial to design appropriate interfaces for end users. While LFs are superior in controllability and fidelity, they require expert knowledge to construct beforehand. Therefore, this interface may be more adequate as a submodule of surface realization, similar to the traditional pipeline

Model interface	Surface-level Evaluation					Logical Fidelity	
	BLEU-4	ROUGE-1	ROUGE-2	ROUGE-4	ROUGE-L	TAPEX-ACC	TAPAS-ACC
Table (T5)	17.1	41.6	21.8	9.1	34.5	64.2	64.6
Table+HL (T5)	31.3	62.2	38.2	17.8	51.7	75.5	71.4
Table+LF (T5)	**36.9**	**68.5**	**44.2**	**21.4**	**58.0**	**86.4**	**81.5**
Table+HL (PLOG)	31.9	62.7	39.1	18.3	51.9	83.4	78.4

Table 4.9: Experiments on the comparison of model interfaces. HL indicates highlighted cells and LF means logical forms.

systems [8, 34], which is also the case for our approach in Chapter 3. By contrast, our second approach, PLOG, solves logical NLG with the two interfaces without explicit LFs, implying its applicability to more general cases. As observed in Table 4.9, the performance of Table+HL (PLOG) shows a comparable performance on faithfulness scores with the Table+LF (T5) interface. This indicates that our approach can significantly improve the logical reasoning ability of PLMs, leading to a close effect to that of leveraging ground-truth LFs.

There still exists a large gap between these interfaces and real-word NLG systems, where manually designed modules may still be necessary to convey accurate user intentions. We leave the investigation to more usable logical NLG systems for future work.

4.7 Summary

In this chapter, we introduce the second approach, augmenting LFs for implict task transfer learning. We propose a table-to-logic pretraining task to enhance the fidelity of logical table-to-text generation and a PLOG model to transfer the knowledge learned from table-to-logic to the table-to-text task. In addition, we constructed a controlled logical NLG dataset by re-purposing the LOGIC2TEXT dataset. To realize the pretraining on large-scale corpora, we proposed a template-based sampling method to extract accurate logical forms from tables automatically. With table-to-logic pretraining, our table-to-text model could significantly improve logical fidelity. The results on table-to-logic generation quantitatively demonstrate the model's ability to reason over tables. Further analyses have shown the robustness of our model and its general effectiveness with different backbones. We also conduct a comparative analysis on the effects

of model interfaces, to lay the ground for future works towards more reliable and applicable systems.

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

5 Conclusion

This research presented two unique approaches aimed at boosting the faithfulness of logical natural language generation (NLG) from structured tables by using augmented logical forms (LFs). Both methods were designed to tackle the key challenges in logical NLG: the difficulty in learning an end-to-end model for joint logical reasoning and surface realization, and the scarcity of training data. Nevertheless, they use LFs differently, leading to different model interfaces and task settings. As such, they concentrate on distinct facets of logical NLG's difficulties.

Our first method employs LFs as explicit control features to guide text generation. The model's input includes both a table and an LF, with the latter offering a complete logical fact for description. This strategy relieves the model from the necessity of logical reasoning over the table, thereby allowing it to primarily focus on transforming the LF into natural language. To counterbalance the shortage of aligned LFs and texts, this method augments additional LFs and texts from existing tables using pretrained table-to-text and table-to-LF generators. A semi-supervised dual learning framework is then proposed to convert these new LFs and texts into parallel data. The effectiveness of this approach is demonstrated through experimental results using the GPT-2 pretrained language model as the base model.

Conversely, the second method addresses table-to-text generation without explicit LF control. Its input is solely a table or a partial table, necessitating logical reasoning for the model. This method does not demand LF-text alignment, and instead uses LFs as pretraining resources for task transfer learning. Specifically, a pretraining task, table-to-logic, involving correct LF generation from a table is proposed. The model is first pretrained on this task, then finetuned on the logical NLG task. This framework can be applied to any sequence-to-sequence generation models, and we utilized multiple pretrained language models to assess our method. To provide a better testbed for logical NLG, we introduced a controlled logical NLG benchmark equipped with annotated highlighted table

cells as control features. Experimental results have proven that the table-to-logic pretraining enriches the models with superior logical reasoning capabilities over tables and improves their faithfulness on two logical NLG benchmarks. The reliability of this method is confirmed through its general effectiveness with several base models. Moreover, experimental results on the table-to-logic task proved that the models indeed learn accurate logical reasoning through the pretraining.

The first approach effectively augments data logical NLG but is limited to cases where parallel data of LFs and texts are readily available. However, such data might be scarce in some domains, especially in real-world scenarios. In addition, providing such LFs already involves extracting correct logical facts via logical reasoning, which is not required for the model. Therefore, the first approach mainly addresses the surface realization part of logical NLG. By contrast, the second approach addresses the challenges in both logical reasoning and surface realization, with a focus on the former. In essence, the two approaches have shown two parallel directions for improving logical NLG.

Motivated by this, we further conduct a comparative analysis of the different model interfaces and task settings of logical NLG. Specifically, we analyzed the effects of using different control features in the model input, including LFs, highlighted cells, and no control feature. The LF-controlled variant is the same as the task setting addressed by approach 1, while the latter two are addressed by approach 2. We use the same dataset but vary the control features to compare the same model's performance. Experimental results revealed that a model can perform the best with LF control, showing the simplicity of this task setting. Using highlighted cells achieved moderate results, but the performance would degrade significantly if no control is specified, showing the difficulty of model interfaces without LF control. However, LFs may not be appropriate as a direct control specified by end users in real-world applications, because of the domain knowledge required for deriving a correct logical fact. Therefore, the LF-controlled interface is more potential for a sub-module of an NLG system. In contrast, with no control features alongside the input table, the model would suffer from uncontrollable content selection. Under this setting, the preference for describing facts can only be learned from the target texts, leading to varied and unpredictable generations. Therefore, highlighted cells seem a compromising control feature that enhances the controllability but allows easy specification. The analysis of the interfaces of logical NLG may inspire future studies on more applicable systems.

Finally, we discuss the limitations of this study and potential future directions.

More realistic table types This study only considers the NLG from open-domain Wikipedia tables, which are typically multi-row database-like tables. The contents in the tables are mostly attributes of certain items, e.g., numerical values, dates, or entities. However, real-world tables are often heterogenous, which store data with different data types or structures within the same table. Addressing this kind of table poses challenges for data manipulation and information extraction, which are worthy of future investigation.

Extend the scope of logical reasoning The types of logic considered in this study are limited to the most common symbolic operations on tables. In other domains, such as financial and medical, the required reasoning types may be out of the scope of this study. Generalizing the reasoning types to other domains may be achievable by expanding the logical form schema to support new operations.

Document-level logical NLG This study only addresses sentence-level generation, while the advanced goal of NLG is to generate document-level summaries or reports. This poses additional challenges for document planning, which involves more difficult content determination and information organization across sentences. Prior studies [116, 79] have investigated reasoning-aware generation from scientific tables, which involves mostly arithmetic reasoning. A recent study [38] also explores the generation of analytical text with multiple reasoning types. We anticipate more studies on this direction.

Effects of large language models This study follows a common pipeline to address NLG, that is, finetuning pretrained models on large-scale in-domain data. However, the recent boost of large language models (LLMs) containing over hundreds of billions of parameters has promoted a new research paradigm to solve NLG, i.e., zero-shot or few-shot prompting with LLMs, where the model receives a task description and zero or only a few examples. The chain-of-thought prompting [122, 18] further improves the reasoning ability of LLMs. A concurrent study [139] has evaluated the performance of LLMs such as ChatGPT on the LogicNLG dataset, the results showing that the latest GPT-4 version of ChatGPT can achieve over 90% faithfulness with chain-of-thought prompting, which is beyond the performance of finetuning small-size PLMs as proposed in this study. Our approach is still competitive with the lower cost and much smaller model size. In fact, the second approach in this study encompasses one of the core ideas of ChatGPT, i.e., pretraining on formal language to enhance reasoning ability in

NLP tasks. How to balance the cost and effects of models must be considered for real-world applications. One potential direction may be model distillation to augment the small-size models with LLMs [41].

Bibliography

[1] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saeid Safaei, Elizabeth D Trippe, Juan B Gutierrez, and Krys Kochut. Text summarization techniques: a brief survey. *arXiv preprint arXiv:1707.02268*, 2017.

[2] Iñigo Alonso and Eneko Agirre. Automatic logical forms improve fidelity in table-to-text generation. *Available at SSRN 4432475*.

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

[4] Tatsuya Aoki, Akira Miyazawa, Tatsuya Ishigaki, Keiichi Goshima, Kasumi Aoki, Ichiro Kobayashi, Hiroya Takamura, and Yusuke Miyao. Generating market comments referring to external resources. In *INLG*, pages 135–139, 2018.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[6] Regina Barzilay and Mirella Lapata. Collective content selection for concept-to-text generation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 331–338. Citeseer, 2005.

[7] Regina Barzilay and Mirella Lapata. Collective content selection for concept-to-text generation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 331–338. Citeseer, 2005.

[8] John A Bateman. Enabling technology for multilingual natural language generation: the kpml development environment. *Natural Language Engineering*, 3(1):15–55, 1997.

[9] Anja Belz. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering*, 14(4):431–455, 2008.

[10] Anja Belz. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering*, 14(4):431–455, 2008.

[11] Samuel Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, 2015.

[12] 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.

[13] Thiago Castro Ferreira, Chris van der Lee, Emiel van Miltenburg, and Emiel Krahmer. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 552–562, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1052. URL `https://aclanthology.org/D19-1052`.

[14] Ernie Chang, Vera Demberg, and Alex Marin. Jointly improving language understanding and generation with quality-weighted weak supervision of automatic labeling. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 818–829, 2021.

[15] Ernie Chang, Xiaoyu Shen, Dawei Zhu, Vera Demberg, and Hui Su. Neural data-to-text generation with lm-based text augmentation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 758–768, 2021.

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

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

[18] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

[19] Wenqing Chen, Jidong Tian, Yitian Li, Hao He, and Yaohui Jin. Deconfounded 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, 2021.

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

[21] Zhiyu Chen, Harini Eavani, Wenhu Chen, Yinyin Liu, and William Yang Wang. Few-shot nlg with pre-trained language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 183–190, 2020.

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

[23] Emilie Colin and Claire Gardent. Generating text from anonymised structures. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 112–117, 2019.

[24] Robert Dale. Generating referring expressions in a domain of objects and processes. *Annexe Thesis Digitisation Project 2018 Block 20*, 1989.

[25] 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.

[26] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, 2019.

[27] Nicholas Diakopoulos. *Automating the news: How algorithms are rewriting the media*. Harvard University Press, 2019.

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

[29] Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 1383–1392, 2018.

[30] Pablo Duboue and Kathleen McKeown. Content planner construction via evolutionary algorithms and a corpus-based fitness function. In *Proceedings of the International Natural Language Generation Conference*, pages 89–96, 2002.

[31] Pablo A Duboue and Kathleen McKeown. Statistical acquisition of content selection rules for natural language generation. 2003.

[32] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, 2018.

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

[34] Michael Elhadad and Jacques Robin. An overview of surge: A reusable comprehensive syntactic realization component. 1996.

[35] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[36] Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. Creating training corpora for nlg micro-planning. In *55th annual meeting of the Association for Computational Linguistics (ACL)*, 2017.

[37] Sebastian Gehrmann, Falcon Z Dai, Henry Elder, and Alexander M Rush. End-to-end content and plan selection for data-to-text generation. *arXiv preprint arXiv:1810.04700*, 2018.

[38] Deepanway Ghosal, Preksha Nema, and Aravindan Raghuveer. Stoat: Structured data to analytical text with controls. *arXiv preprint arXiv:2305.11826*, 2023.

[39] Heng Gong, Xiaocheng Feng, Bing Qin, and Ting Liu. Table-to-text generation with effective hierarchical encoder on three dimensions (row, column and time). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3143–3152, 2019.

[40] Heng Gong, Yawei Sun, Xiaocheng Feng, Bing Qin, Wei Bi, Xiaojiang Liu, and Ting Liu. 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, 2020.

[41] Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*, 2023.

[42] Qipeng Guo, Zhijing Jin, Xipeng Qiu, Weinan Zhang, David Wipf, and Zheng Zhang. Cyclegt: Unsupervised graph-to-text and text-to-graph generation via cycle training. *CoRR*, abs/2006.04702, 2020. URL `https://arxiv.org/abs/2006.04702`.

[43] Hamza Harkous, Isabel Groves, and Amir Saffari. Have your text and use it too! end-to-end neural data-to-text generation with semantic fidelity. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2410–2424, 2020.

[44] Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. Revisiting self-training for neural sequence generation. In *International Conference on Learning Representations*, 2019.

[45] Vu Cong Duy Hoang, Philipp Koehn, Gholamreza Haffari, and Trevor Cohn. Iterative back-translation for neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 18–24, 2018.

[46] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[47] Eduard H Hovy. Automated discourse generation using discourse structure relations. *Artificial intelligence*, 63(1-2):341–385, 1993.

[48] Hanxu Hu, Yunqing Liu, Zhongyi Yu, and Laura Perez-Beltrachini. Improving user controlled table-to-text generation robustness. *arXiv preprint arXiv:2302.09820*, 2023.

[49] Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 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, 2021.

[50] Aizhan Imankulova, Takayuki Sato, and Mamoru Komachi. Improving low-resource neural machine translation with filtered pseudo-parallel corpus. In *Proceedings of the 4th Workshop on Asian Translation (WAT2017)*, pages 70–78, 2017.

[51] Harsh Jhamtani and Taylor Berg-Kirkpatrick. Truth-conditional captions for time series data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 719–733, 2021.

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

[53] Nikiforos Karamanis. Entity coherence for descriptive text structuring. 2004.

[54] Zdeněk Kasner, Ekaterina Garanina, Ondřej Plátek, and Ondřej Dušek. Tabgenie: A toolkit for table-to-text generation. *arXiv preprint arXiv:2302.14169*, 2023.

[55] Jyotsana Khatri and Pushpak Bhattacharyya. Filtering back-translated data in unsupervised neural machine translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4334–4339, 2020.

[56] Joohyun Kim and Raymond Mooney. Generative alignment and semantic parsing for learning from ambiguous supervision. In *Coling 2010: Posters*, pages 543–551, 2010.

[57] Flip Korn, Xuezhi Wang, You Wu, and Cong Yu. Automatically generating interesting facts from wikipedia tables. In *Proceedings of the 2019 International Conference on Management of Data*, pages 349–361, 2019.

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

[59] Varun Kumar, Ashutosh Choudhary, and Eunah Cho. Data augmentation using pre-trained transformer models. In *Proceedings of the 2nd Workshop on Life-long Learning for Spoken Language Systems*, pages 18–26, 2020.

[60] Rémi Lebret, David Grangier, and Michael Auli. Neural text generation from structured data with application to the biography domain. In *Pro-*

ceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1203–1213, 2016.

[61] Leo Leppänen, Myriam Munezero, Mark Granroth-Wilding, and Hannu Toivonen. Data-driven news generation for automated journalism. In *Proceedings of the 10th international conference on natural language generation*, pages 188–197, 2017.

[62] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 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, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL `https://aclanthology.org/2020.acl-main.703`.

[63] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 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, 2020.

[64] Liang Li, Can Ma, Yinliang Yue, and Dayong Hu. Improving encoder by auxiliary supervision tasks for 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 5979–5989, 2021.

[65] Ziran Li, Zibo Lin, Ning Ding, Hai-Tao Zheng, and Ying Shen. Triple-to-text generation with an anchor-to-prototype framework. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 3780–3786, 2021.

[66] Percy Liang, Michael I Jordan, and Dan Klein. Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference*

of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, pages 91–99, 2009.

[67] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[68] Chengyuan Liu, Leilei Gan, Kun Kuang, and Fei Wu. Investigating the robustness of natural language generation from logical forms via counterfactual samples. *arXiv preprint arXiv:2210.08548*, 2022.

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

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

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

[72] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

[73] Fuli Luo, Peng Li, Jie Zhou, Pengcheng Yang, Baobao Chang, Zhifang Sui, and Xu Sun. A dual reinforcement learning framework for unsupervised text style transfer. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019*, 2019.

[74] Shuming Ma, Pengcheng Yang, Tianyu Liu, Peng Li, Jie Zhou, and Xu Sun. Key fact as pivot: A two-stage model for low resource table-to-text generation. *arXiv preprint arXiv:1908.03067*, 2019.

[75] François Mairesse, Milica Gasic, Filip Jurcicek, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. Phrase-based statistical language generation using graphical models and active learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1552–1561, 2010.

[76] Kathleen McKeown. *Text generation*. Cambridge University Press, 1992.

[77] Kathleen McKeown and Pablo A Duboue. Empirically estimating order constraints for content planning in generation. 2001.

[78] Chris Mellish, Alistair Knott, Jon Oberlander, and Mick O'Donnell. Experiments using stochastic search for text planning. In *Natural Language Generation*, 1998.

[79] Nafise Sadat Moosavi, Andreas Rücklé, Dan Roth, and Iryna Gurevych. Scigen: a dataset for reasoning-aware text generation from scientific tables. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[80] Amit Moryossef, Yoav Goldberg, and Ido Dagan. Step-by-step: Separating planning from realization in neural data-to-text generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2267–2277, 2019.

[81] Soichiro Murakami, Akihiko Watanabe, Akira Miyazawa, Keiichi Goshima, Toshihiko Yanase, Hiroya Takamura, and Yusuke Miyao. Learning to generate market comments from stock prices. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1374–1384, 2017.

[82] Feng Nie, Jinpeng Wang, Jin-ge Yao, Rong Pan, and Chin-Yew Lin. 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, 2018.

[83] Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. The e2e dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206, 2017.

[84] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[85] Yannis Papanikolaou and Andrea Pierleoni. Dare: Data augmented relation extraction with gpt-2. *arXiv preprint arXiv:2004.13845*, 2020.

[86] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[87] Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 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, 2020.

[88] Panupong Pasupat and Percy Liang. 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, 2015.

[89] 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 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., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[90] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[91] Steffen Pauws, Albert Gatt, Emiel Krahmer, and Ehud Reiter. Making effective use of healthcare data using data-to-text technology. *Data science for healthcare: methodologies and applications*, pages 119–145, 2019.

[92] François Portet, Ehud Reiter, Albert Gatt, Jim Hunter, Somayajulu Sripada, Yvonne Freer, and Cindy Sykes. Automatic generation of textual summaries from neonatal intensive care data. *Artificial Intelligence*, 173 (7-8):789–816, 2009.

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

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

[95] Raheel Qader, François Portet, and Cyril Labbé. Semi-supervised neural text generation by joint learning of natural language generation and natural language understanding models. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 552–562, 2019.

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

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

[98] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*, 21(140): 1–67, 2020. URL `http://jmlr.org/papers/v21/20-074.html`.

[99] Ehud Reiter and Robert Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

[100] Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems*. Studies in Natural Language Processing. Cambridge University Press, 2000. doi: 10.1017/CBO9780511519857.

[101] Ehud Reiter, Somayajulu Sripada, Jim Hunter, Jin Yu, and Ian Davy. Choosing words in computer-generated weather forecasts. *Artificial Intelligence*, 167(1-2):137–169, 2005.

[102] Jacques Pierre Robin. *Revision-based generation of natural language summaries providing historical background: corpus-based analysis, design, implementation and evaluation*. Columbia University, 1995.

[103] Martin Schmitt, Sahand Sharifzadeh, Volker Tresp, and Hinrich Schütze. An unsupervised joint system for text generation from knowledge graphs and semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7117–7130, 2020.

[104] Donia Scott and Clarisse Sieckenius de Souza. Getting the message across in rst-based text generation. *Current research in natural language generation*, 4:47–73, 1990.

[105] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, 2016.

[106] Mandar Sharma, John S Brownstein, and Naren Ramakrishnan. T 3: Domain-agnostic neural time-series narration. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 1324–1329. IEEE, 2021.

[107] Mandar Sharma, Ajay Gogineni, and Naren Ramakrishnan. Innovations in neural data-to-text generation. *arXiv preprint arXiv:2207.12571*, 2022.

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

[109] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. 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, 2021.

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

4414–4426, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.388. URL https://aclanthology.org/2021.findings-acl.388.

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

[112] Somayajulu Sripada, Ehud Reiter, Jim Hunter, and Jin Yu. A two-staged model for content determination. In *Proceedings of the ACL 2001 Eighth European Workshop on Natural Language Generation (EWNLG)*, 2001.

[113] Shang-Yu Su, Chao-Wei Huang, and Yun-Nung Chen. Towards unsupervised language understanding and generation by joint dual learning. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 671–680, 2020.

[114] Yixuan Su, Zaiqiao Meng, Simon Baker, and Nigel Collier. Few-shot table-to-text generation with prototype memory. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 910–917, 2021.

[115] Yixuan Su, David Vandyke, Sihui Wang, Yimai Fang, and Nigel Collier. Plan-then-generate: Controlled data-to-text generation via planning. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 895–909, 2021.

[116] Lya Hulliyyatus Suadaa, Hidetaka Kamigaito, Kotaro Funakoshi, Manabu Okumura, and Hiroya Takamura. 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, 2021.

[117] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of NAACL-HLT*, pages 4149–4158, 2019.

[118] Kumiko Tanaka-Ishii, Kôiti Hasida, and Itsuki Noda. Reactive content selection in the generation of real-time soccer commentary. In *COLING 1998 Volume 2: The 17th International Conference on Computational Linguistics*, 1998.

[119] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[120] Anthony J Viera, Joanne M Garrett, et al. Understanding interobserver agreement: the kappa statistic. *Fam med*, 37(5):360–363, 2005.

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

[122] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[123] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*, 2015.

[124] Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016.

[125] Chen Wenqing, Tian Jidong, Li Yitian, He Hao, and Jin Yaohui. Deconfounded variational encoder-decoder for logical table-to-text generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, pages 5532–5542, 2021.

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

[127] Sam Wiseman, Stuart M Shieber, and Alexander M Rush. Learning neural templates for text generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3174–3187, 2018.

[128] 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. 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, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

[129] 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. 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, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

[130] 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. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *arXiv preprint arXiv:2201.05966*, 2022.

[131] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, and Vadim Sheinin. Sql-to-text generation with graph-to-sequence model. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 931–936, 2018.

[132] Weijia Xu, Xing Niu, and Marine Carpuat. Dual reconstruction: a unifying objective for semi-supervised neural machine translation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 2006–2020, 2020.

[133] Rong Ye, Wenxian Shi, Hao Zhou, Zhongyu Wei, and Lei Li. Variational template machine for data-to-text generation. *arXiv preprint arXiv:2002.01127*, 2020.

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

[135] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SkeHuCVFDr`.

[136] Zhirui Zhang, Shujie Liu, Mu Li, Ming Zhou, and Enhong Chen. Joint training for neural machine translation models with monolingual data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[137] Zhirui Zhang, Shuo Ren, Shujie Liu, Jianyong Wang, Peng Chen, Mu Li, Ming Zhou, and Enhong Chen. Style transfer as unsupervised machine translation. *arXiv preprint arXiv:1808.07894*, 2018.

[138] Chen Zhao, Yu Su, Adam Pauls, and Emmanouil Antonios Platanios. 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, 2022.

[139] Yilun Zhao, Haowei Zhang, Shengyun Si, Linyong Nan, Xiangru Tang, and Arman Cohan. Large language models are effective table-to-text generators, evaluators, and feedback providers. *arXiv preprint arXiv:2305.14987*, 2023.

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

[141] Victor Zhong, Mike Lewis, Sida I Wang, and Luke Zettlemoyer. 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, 2020.

Publication List

Refereed Journals

1. Ao Liu, Congjian Luo, and Naoaki Okazaki. Improving Logical-level Natural Language Generation with Topic-conditioned Data Augmentation and Logical Form Generation. Journal of Information Processing, vol. 31, pages 332-343, May 2023.

Refereed International Conference Papers

2. Ao Liu, Haoyu Dong, Naoaki Okazaki, Shi Han, and Dongmei Zhang. PLOG: Table-to-logic Pretraining for Logical Table-to-text Generation. In Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 5531–5546, December 2022.

3. Ao Liu, An Wang, and Naoaki Okazaki. Semi-Supervised Formality Style Transfer with Consistency Training. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, pages 4689–4701, May 2022.

4. Ao Liu, Shuai Yuan, Chenbin Zhang, Congjian Luo, Yaqing Liao, Kun Bai, and Zenglin Xu. Multi-Level Multimodal Transformer Network for Multimodal Recipe Comprehension. In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, pages 1781-1784, July 2020.

5. Ao Liu, Lizhen Qu, Junyu Lu, Chenbin Zhang, and Zenglin Xu. Machine Reading Comprehension: Matching and Orders. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pages 2057-2060, November 2019.

Co-authored Publications

7. An Wang, Junfeng Jiang, Youmi Ma, Ao Liu, and Naoaki Okazaki. Generative Data Augmentation for Aspect Sentiment Quad Prediction. In Proceedings of the 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023), pages 128-140, July 2023.

8. Yidong Wang, Hao Wu, Ao Liu, Wenxin Hou, Zhen Wu, Jindong Wang, Takahiro Shinozaki, Manabu Okumura, and Yue Zhang. Exploiting Unlabeled Data for Target-Oriented Opinion Words Extraction. In Proceedings of the 29th International Conference on Computational Linguistics, pages 7075-7085, October 2022.

9. Yu Pan, Zeyong Su, Ao Liu, Wang J Q, Nannan Li, and Zenglin Xu. A Unified Weight Initialization Paradigm for Tensorial Convolutional Neural Networks. In International Conference on Machine Learning, pages 17238-17257, June 2022.

10. Haoyu Dong, Zhoujun Cheng, Xinyi He, Mengyu Zhou, Anda Zhou, Fan Zhou, Ao Liu, Shi Han and Dongmei Zhang. Table Pretraining: A Survey on Model Architectures, Pretraining Objectives, and Downstream Tasks. In Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22, Survey Track), pages 5426-5435, July 2022.

11. Chenbin Zhang, Congjian Luo, Junyu Lu, Ao Liu, Bing Bai, Kun Bai and Zenglin Xu. Read, Attend, and Exclude: Multi-Choice Reading Comprehension by Mimicking Human Reasoning Process. In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, pages 1945-1948, July 2020.

12. Junyu Lu, Xiancong Ren, Yazhou Ren, Ao Liu, Zenglin Xu. Improving Contextual Language Models for Response Retrieval in Multi-Turn Conversation. In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, pages 1805-1808, July 2020.