electronics

MDPI

Article

SP-NLG: A Semantic-Parsing-Guided Natural Language Generation Framework

Tongliang Li, Shun Zhang and Zhoujun Li *

State Key Lab of Software Development Environment, Beihang University, Beijing 100191, China; tonyliangli@buaa.edu.cn (T.L.); shunzhang@buaa.edu.cn (S.Z.)
* Correspondence: lizj@buaa.edu.cn

Abstract: We propose SP-NLG: A **s**emantic-**p**arsing-**g**uided **n**atural **l**anguage **g**eneration framework for logical content generation with high fidelity. Prior studies adopt large pretrained language models and coarse-to-fine decoding techniques to generate text with logic; while achieving considerable results on automatic evaluation metrics, they still face challenges in terms of logical fidelity based on human evaluation. Inspired by semantic parsing, which translates natural language utterances into executable logical forms, we propose to guide the generation process with a semantic parser. Different from semantic parsing and QA tasks, of which the logical correctness can be evaluated based on the execution result, the logic information of the generated content is implicit. To leverage a semantic parser for generation, we propose a slot-tied back-search algorithm. We follow the coarse-to-fine generation scheme, but instead of filling the slots with model predictions, which is less uncontrolled, the slot values are offline searched by the algorithm. The slot-tied back-search algorithm ties the parameters of a logic form with the slots of a template in one-to-one correspondence. We back-search the arguments that correctly execute the logic form and fill the arguments (as slot values) into the textual template to generate the final target, which ensures the logical correctness. Experiment results of a model built on SP-NLG demonstrates that our framework achieves high fidelity on logical text generation.

Keywords: data-to-text generation; logical natural language generation; semantic parsing

Citation: Li, T.; Zhang, S.; Li, Z. TiSP-NLG: A Semantic-Parsing-Guided Natural Language Generation Frameworktle. *Electronics* **2023**, *12*, 1772. https://doi.org/10.3390/electronics12081772

Academic Editor: Arkaitz Zubiaga

Received: 19 March 2023
Revised: 4 April 2023
Accepted: 4 April 2023
Published: 8 April 2023

1. Introduction

Natural language generation (NLG) based on neural network models has developed remarkable progress recently. Large pretrained language models with an encoder–decoder structure, such as BERT2BERT [1], BART [2], and T5 [3], are more than capable of generating fluent and coherent text, gaining high performance on various NLG tasks. However, such developments are mainly made on the surface realization level: how to "say" the words. For what to "say", formally called content planning, many problems remain unsolved, one of which is low fidelity. Generation fidelity refers to how faithful the generated target is to the underlying facts, data, knowledge, or meaning representation, i.e., for the data-to-text task, the generated text should only describe what the data explicitly describe or imply. Hence, the concept of fidelity can be further interpreted as (1) the surface-level factual correctness (explicit) and (2) the logical correctness (implicit) of the generated target. Such a concept was brought up by a recent study of LogicNLG [4].

For the first type of fidelity, a series of recent studies adopt the copy mechanism [5,6] or model content planning as an individual stage [7–9] to gain higher surface-level fidelity, though recent encoder–decoder-based deep neural network models tend to be trained in an end-to-end fashion [10–12] instead of a cascade of two stages. Most of these approaches simply copied or restated the underlying facts from the input data and gained considerable results on factual correctness [13]. However, humans are capable of discovering implicit information by inspecting the data and inferring from the underlying facts (e.g., "Daniel was

the Governor for 4 years." in Figure 1). We believe such entailed information is important for a deeper level of understanding and generating more valuable targets. Therefore, the main focus of this work is high-fidelity logical content generation. Some factual and logical statements are shown in Figure 1.

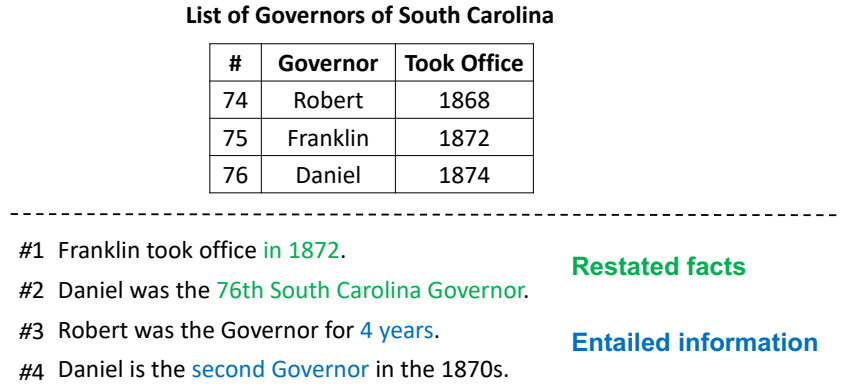

Figure 1. Examples of factual and logical statements.

A logical natural language generation (logical NLG) task was already brought up by LogicNLG [4]. The task definition is to generate natural language statements that can be logically inferred from the given data (i.e., the table). To avoid confusion, the logic here refers to the inferences with symbolic operations over the input data [14], which is mainly numerical reasoning. The generated text is then automatically evaluated on three different metrics specifically designed for logical fidelity evaluation: the parsing-based, NLI-based, and adversarial evaluation. More details of the evaluation metrics can be found in their original paper. Results of recent studies on the logical NLG task are still primitive, which is mainly due to that (1) it is challenging for existing NLG models to perform reasoning and symbolic calculations, and (2) the search space of logically entailed statements is exponentially large due to a vast number of combinations of different operations and arguments from the table. To partially address these challenges, LogicNLG proposes a model using the coarse-to-fine generation scheme: the model generates a template that determines the logical structure globally first and fills the slots conditioned on the template. The coarse-to-fine generation schema is shown in Figure 2. To construct a logical template, the numbers and entities are replaced by a placeholder "[ENT]".

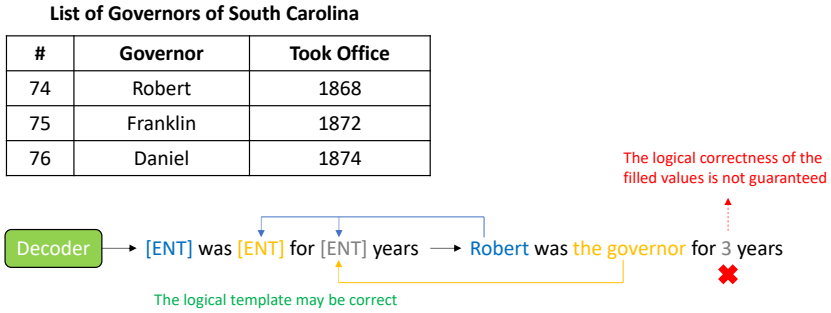

Figure 2. The coarse-to-fine generation scheme proposed by LogicNLG.

The decoder-based model predicts both the the template Y_T and the slot-filled target Y using a left-to-right decoding style. The template and the final text is separated with the special token "[SEP]", denoted by $\hat{Y} = [Y_T; [SEP]; Y]$. The goal is to maximize the overall likelihood of $P(\hat{Y} \mid \mathbf{T}; \beta)$, where $\mathbf{T}$ is a sequence of table headers, table captions, and table content.

Though such a coarse-to-fine generation scheme could partially improve the logical fidelity by constraining the generated target to a specific logical structure, we argue that the slot-filling stage remains uncontrolled. The model lacks reasoning abilities to fill the correct values that fit the logical template. Suppose that the logical template "[ENT] was [ENT] for [ENT] years" in Figure 2 is correctly generated. The model may not fully understand the logic information implied by the word "for" and predict the wrong slot value: Robert was actually the governor for "4" years instead of "3" years, based on the given table. A recent work, TAPAS [15], suggests that pretrained models are fairly accurate in "guessing" numbers that are close to the target. Even the state-of-the-art large language model ChatGPT [16] may make mistakes in solving math equations, which is a natural problem for data-driven probabilistic models. We argue that surface-realization (how to say it) can be flexible, but the logic behind the words should be absolutely correct. To address the "guessing" problem, we suggest that the logic verification should be completed by a computer program, as models are "creative writers" while computer programs are deterministic. Hence, we propose to guide the slot-filling stage with semantic parsing techniques.

Another prior work, Logic2Text [17], formulates the logical NLG task as a logic-form-to-text problem. The model is provided with both the source table and a logic form, which represents the semantics of the target text. The training objective is

$$\arg\max_{\beta} P(Y \mid \mathbf{T}, S; \beta) \tag{1}$$

where Y is the final target, $\mathbf{T}$ is a sequence of table headers, table captions, and the table content, and S is the logic form. With the logic form included, the logic behind the word "for" from "Robert was the governor for 4 years" can now be represented by a subfunction $diff\{param_1, param_2\}$ belonging to the logic form. $param_1$, $param_2$ denote the two parameters of the logic function, and $diff\{param_1, param_2\}$ calculates the difference between two arguments. The main problem for Logic2Text is the approach they propose to leverage the logic forms, which is to (1) consider the logic form as a known variable, and (2) feed the model with both the source data (table) and the logic form linearly as pure text. Such an approach poses great challenges for the model's ability to understand the semantics behind the logic forms, as the logic forms are mainly diversified graph structures. The limited amount of annotated logic-form-to-text pairs is insufficient for fully training a model to understand the semantics.

To address the aforementioned problems and further encourage research in this direction, we propose SP-NLG: a **s**emantic-**p**arsing-**g**uided **n**atural **l**anguage **g**eneration framework to generate content with high logical fidelity. The framework consists of three modules: (1) logical template generation, (2) semantic parsing, and (3) target ranking. Figure 3 is an overview of our framework. The modules are presented from bottom to top.

Similar to the coarse part of LogicNLG, we generate a logical template Y_T based on the source data (table) $\mathbf{T}$, which can be formally described as $P(Y_T \mid \mathbf{T}; \beta)$, where β is the learnable parameter. Instead of removing entities and numbers from the target to construct the logical templates, we tie the parameters of the logic form with the slots of the logical template using heuristic rules. We repurpose the annotated logic-form–text pairs from Logic2Text train a semantic parser with text–logic-form pairs. The training objective is to maximize the likelihood of $P(S \mid \mathbf{T}, Y_T; \theta)$, where S is the logic form with slot-tied parameters to the logical template Y_T, and θ is the the learnable parameter. Different from Logic2Text, instead of requiring the model to understand the logic form, we transform the understanding problem into a translation problem: the model only needs to "translate" the logical template Y_T into a logic form S. Such transformation significantly reduces the difficulty of model learning. Different from LogicNLG, which predicts the slot-filled target next to the template in sequence, we design a slot-tied back-search algorithm on the logic form to find correct arguments in real time. As the the parameters of the logic form are tied to the slots of the logical template in one-to-one correspondence, these arguments are also slot values for the template. The correct argument for a single logic form (also a single

template) may not be distinct. We rank all slot-filled templates (candidate targets) and consider the one with the highest ranking score as the final generated target. The ranking label is based on the BLEU score of the candidate target against the golden target. With SP-NLG, the logical correctness of the generated target can be guaranteed.

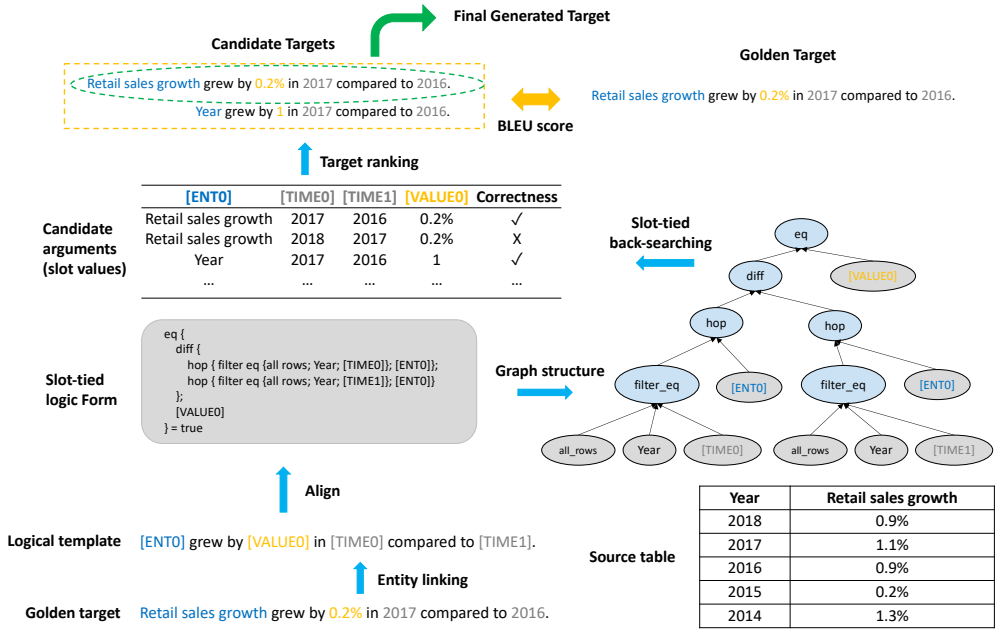

Figure 3. An overview of our SP-NLG framework.

To conclude, the contribution of this work can be summarized as follows:

(1) A semantic-parsing-guided natural language generation framework to generate logical content. The design of slot-tied back-searching allows a generation model to leverage the arguments and results of the logic form in real time, which guarantees the logical correctness of the output.

(2) Experiment results of a model built on SP-NLG demonstrates the effectiveness of our framework on generating text with high logical fidelity.

2. Related Work

2.1. Data-to-Text Generation

Data-to-text generation refers to the task of generating natural language text based on structured data or meaning representations, which has been widely studied for many years. Applications of data-to-text generation include automatically generating sports reports [12], weather reports [18], health reports [19], and dialog responses [20,21].

Prior data-to-text datasets mainly focus on surface-level generation within a particular schema or domain, for example, ROBOCUP [22], WEATHERGOV [18], E2ENLG [23], and WebNLG [24]. Several metrics are proposed to evaluate the factual correctness of surface-level generations, e.g., the entity-centric metric [13] and the overall slot-filling metric [25]. The faithfulness metrics are roughly calculated as $|\mathbb{F}_p \cap \mathbb{F}_g|/|\mathbb{F}_g|$, where $\mathbb{F}_g$ is a set of aligned facts of the golden target and the input data, and $\mathbb{F}_p$ is that for the generated target. Based on such definition, the factual correctness for open-ended generated targets will be significantly lower than faithful generations. Most datasets require the generation model to describe all records of the input data. WikiBio [6] expects the generated text to cover key records only, with no explicit guidance on the topic. Moving a step further, ToTTo [26] argues that the existing models tend to generate more fluent, but more unfaithful, targets to the source than faithful ones. Tasks with open-ended targets may lead to subject generations and challenges for evaluation. Hence, to ensure that the generated text is both natural and faithful to the source, ToTTo proposes to guide the topic of the target text with a

set of highlighted table cells. More intuitively, TWT [27] requires the model to generate the next words based on a given prefix. The prefix mocks a piece of written text by a human and also serves as a control factor on what to generate next.

Recent studies have brought attention to logical natural language generation. RotoWire [11] is a slightly challenging dataset on generating basketball game results from multirow tables; while involving a few logical inferences, most targets remain surface-level factual restatements. Chen et al. [28] studied fact verification on semistructured data by labeling the statements as entailed or refuted based on linguistic and symbolic reasoning. Chen et al. [4] proposed a new task to generate natural language text with numerical reasoning and designed a coarse-to-fine model to address the mismatch problem between sequence order and logical order. The task was mainly designed for testing model abilities of generating logically inferred content solely based on the given source. Chen et al. [17] argued that such task formulation is not yet appropriate for practical NLG applications due to low logical fidelity and a lack of control on what to generate. To address such a problem, Logic2Text formulated high-logical-fidelity natural language generation as a logic-form-to-text task. The generation model was provided with an additional underlying logic form along with the table; however, such underlying logic forms barely exist in the real world. A more practical scenario is to consider the logic form as an unknown argument and reverse the logic-form–text pair to formulate a text-to-logic-form semantic parsing task. The semantic parser equips the generation model with reasoning abilities to generate high-logical-fidelity targets.

Traditional NLG approaches [29] adopt a pipeline of two individual stages: content planning (what to say) and surface realization (how to say it). Recent studies tend to train generation models in an end-to-end fashion using the encoder–decoder structure [10–12]. More recently, large pretrained language models achieved state-of-the-art results on NLG tasks and became the new paradigm [3,30,31]. Another line of work brought a sense of separating content planning as a individual stage again to reduce hallucinated content which cannot be aligned to any input table record [7,9]. Another hot research topic in NLG is controlled text generation. Such a task introduces control of the tense [32], sentiment [33], politeness [34], text length [35], the identity of the speaker [36], and even the desired content [27] with control factors. The control factor can be a word, a phrase, or a piece of text. Our work also belongs to controlled text generation as the the generation process is controlled by the semantic parser, producing higher-logical-fidelity and less-open-ended targets.

2.2. Semantic Parsing

Our work is also related to semantic parsing, which refers to the task of mapping natural language utterances into machine executable programs. The executable program is a formal, symbolic, complete meaning representation of a sentence [37]. Commonly used forms of executable programs include a combination of first-order logic (FOL) and λ-calculus such as CCG [38] and λ-DCS [39]. Text-to-SQL tasks [40,41] can also be considered as semantic parsing by translating the natural language text into SQL. For natural language generation, Logic2Text [17] presents a Python-like program including seven common coarse logic types specifically designed to represent the semantics of the target text.

Similar to NLG tasks, semantic parsing tasks have achieved remarkable progress in recent years by adopting neural sequence-to-sequence models. To train a semantic parser under the fully supervised setting [42,43], a large amount of utterance–program pairs is required, which can be expensive and labor-intensive. When there is a lack of labeled data, weakly-supervised approaches and data augmentation methods should be considered first. OVERNIGHT [44] demonstrates how to build a domain-specific semantic parser from scratch with no training data. Recent studies tend to train a semantic parser with indirect supervision [45]: only the correct output of a program is labeled, instead of the program which produced the output. For example, for the knowledge base question-answering

(KBQA) task [46], only the knowledge base, question, and answer are provided. The model is required to learn the possible logical expressions from the indirect supervisions.

Different from QA tasks, the NLG task has no direct answer. The target is more open-ended, which is the main difficulty in introducing semantic parsing into generation. As the answer (target) is not unique, training a semantic parser under semisupervised settings can be difficult. The labeled logic-form–target-text pairs from Logic2Text are insufficient for fully training a semantic parser to provide the generation model with reasoning abilities. One way to address this problem is to adopt data augmentation methods similar to OVERNIGHT. For SP-NLG, the logic template generation module and the semantic parser from the semantic-guided slot-filling module can be trained either jointly or separately, leaving the flexibility to plug in/out different semantic parsers and logic template generation modules. In other words, each of these modules can be trained offline with the best resources possible. A strong logical NLG model can be built by combining the best trained modules in cascade, along with a few more steps of joint tuning.

3. Task Definition and Dataset

3.1. Task Definition

We study the logical table-to-text generation task proposed by LogicNLG [4]. The input of the task is a table **T** along with its title and caption. **T** consists of R_T rows and C_T columns, with T_{ij} being the (i, j)-th cell:

$$\mathbf{T} = \{T_{ij} \mid i \leq R_T, j \leq C_T\} \tag{2}$$

T_{ij} could be a word, a phrase, a number, or even a piece of natural language text. The golden target is a sentence:

$$Y = y_1, y_2, ..., y_n \tag{3}$$

where y_n is a word from the sentence. The logic form S is a symbolic meaning representation representing the semantics of the golden target Y and can be executed on the table **T**:

$$S = s_1, s_2, ..., s_m \tag{4}$$

where s_m is a token from the logic form. The task objective is to train a neural generation model $p(Y \mid \mathbf{T})$ that generates a sentence $\hat{Y}$ which is both fluent and faithful (factually and logically) to the given table **T**. For the fully supervised setting, where logic form L is annotated, the training objective is $p(Y \mid \mathbf{T}, S)$, and for the semisupervised setting, where the golden S is unknown, the training objective would be $p(Y \mid \mathbf{T}, \hat{S})$.

3.2. Dataset

We study logical natural language generation on two datasets.

- **LogicNLG** [4] is based on the positive statements of TabFact [28], a table-based fact verification dataset including both positive and negative statements. The statements are collected from the "complex" channel, of which the sentences require annotation with reasoning.
- **Logic2Text** [17] is built on WikiTables [47], a set of open-domain tables crawled from Wikipedia. The over-complicated tables are filtered out, and only tables with fewer than 20 rows and 10 columns are adopted. Seven common logic types are designed to describe multi-row tables: *count*, *superlative*, *comparative*, *aggregation*, *majority*, *unique*, and *ordinal*.

The Logic2Text dataset includes annotated logic forms while the LogicNLG dataset only provides table–sentence pairs. Table 1 shows statistics of the two datasets. It should be noted that according to LogicNLG, using the whole table greatly compromises the performance. Therefore, only a partial table P_T is adopted, which is built with entity-linking-based alignment between the whole table **T** and the target sentence Y. We adopt the same strategy for our work.

Table 1. Dataset statistics.

Dataset	Train	Validation	Test	Tables	Logic Form
LogicNLG	28,450	4260	4305	7392	Not included
Logic2Text	8566	1095	1092	5554	Included

4. Methodology

By adopting transformer-based [48] structures and fine-tuning pretrained language models on task-specific data, NLG models can be easily applied to data-to-text generation tasks. For example, on Logic2Text, both the table and the logic form are fed into the encoder and the target text is fed into the decoder for training. During inference, the model generates targets conditioned on both the table and the logic form.

Based on similar structures, we propose the SP-NLG framework. The framework structure with all module designs are presented in Figure 4. The framework consists of three modules: (1) the logical template generation module, (2) the semantic-guided slot-filling module, and (3) the target ranking module. We use a transformer-based encoder–decoder structure to generate a template based on the table, which determines the global logic structures. Then, a different transformer-based model is employed to generate the corresponding logic form conditioned on both the table and the template, which serves the semantic parsing purpose. As the table is already encoded for template generation, the semantic parser can either share the table representation from the template generation module or encode the table separately. We further design a strategy to tie the slots of the template and the parameters of the logic form in one-to-one correspondence. Based on the generated logic form, we propose a back-search algorithm to search possible arguments that execute the correct result. Such arguments are considered as candidate slot values for the template. We use the same logic form proposed by Logic2Text, where the correct execution result is always "true". As the parameters of the logic form are slot-tied to the template, the candidate slot values can be directly filled into the template to build candidate targets. The slot values are only correct in terms of logical correctness, but are not necessarily the best for making up a whole sentence from the aspects of fluency and coherence. Thus, a target ranking module is required to select the best candidate target which is both logically faithful to the source and fluent as a natural language text. We use the BLEU metric score as supervision to train the target ranking module. The higher the BLEU score, the higher the rank for a candidate target. The target with the highest ranking score is considered as the final generated target. We describe the details of module designs in the following subsections.

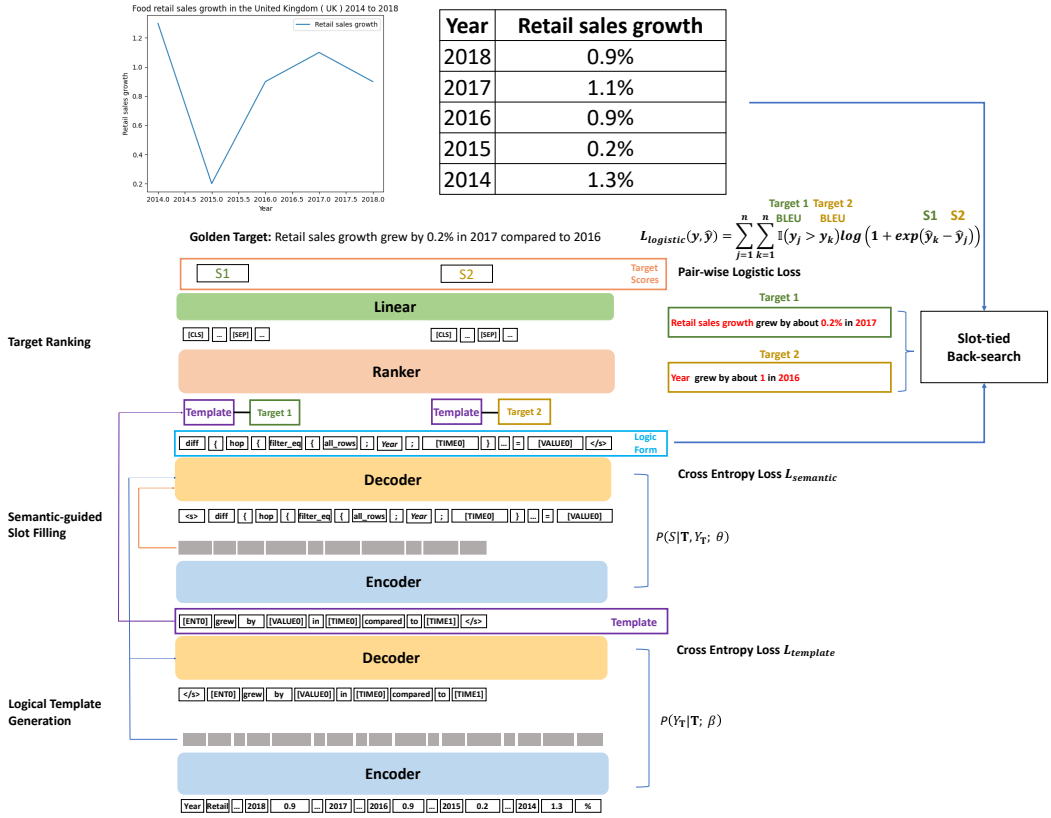

Figure 4. The SP-NLG framework structure.

4.1. Logical Template Generation

The logical template generation module here is similar to the coarse part of the coarse-to-fine model proposed by LogicNLG [4]. It determines the global logical structures of what to generate. We use the same entity linker to identify aligned entities and numbers between the table **T** and the target sentence Y. LogicNLG replaces the aligned items with a fixed placeholder "[ENT]" to construct the template Y_T. For our work, we need to assign each aligned item a unique placeholder so that each slot can be uniquely identified to fill the corresponding argument of the logic form. To fully leverage language models and reduce the difficulty of learning, we use the slot type with id as the placeholder for uniqueness, in case the target sentence Y contains more than one slot with the same type. For example, given the sentence "Retail sales grew by 0.2% in 2017 compared to 2016", the template with unique placeholders would be "[ENT0] grew by [VALUE0] in [TIME0] compared to [TIME1]". The placeholders are predefined and considered as single tokens for the model vocabulary. Supported slot types and placeholders are listed in Table 2.

Table 2. Slot types and placeholders.

Entity	Value	Time
ENT[0–4]	VALUE[0–4]	TIME[0–4]

The training object for the logical template generation module is to maximize the overall likelihood of

$$P(Y_T \mid \mathbf{T}; \beta) \tag{5}$$

where β is the learnable parameter of the logical template generation module. The training objective is equivalent to minimizing the cross-entropy loss (denoted as CE) between the generated and golden logical template.

$$L_{template} = CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{\mathrm{d}} y_i \log(\hat{y}_i) \tag{6}$$

It should be noted that for simplicity, the slots can be assigned with only slot types (without ids) or even "[ENT]" only for template generation. For the same example, the template would be "[ENT] grew by [VALUE] in [TIME] compared to [TIME]" or "[ENT] grew by [ENT] in [ENT] compared to [ENT]". The unique slot placeholders shall be assigned before semantic parsing.

To encode the source table, we follow previous studies on linearizing structured tables as natural language text [49]. In detail, we adopt a global (table level) template "Given the table of [TABLE NAME], in the 1st row, the [COLUMN0 NAME] is [COLUMN0 VALUE], in the 2nd row, the [COLUMN1 NAME] is [COLUMN1 VALUE], ..." to flatten the whole table **T** as a passage:

$$P_T = w_1, w_2, ..., w_{|\mathbf{T}|} \tag{7}$$

where $w_{|\mathbf{T}|}$ is a word from the constructed template. $|\mathbf{T}|$ denotes the length of the paragraph. The table cells are scanned row by row horizontally to construct the paragraph. P_T is fed into the model as the proxy of **T** to generate the template Y_T. As the language models are pretrained on natural language text, converting structured information to a natural language passage fully leverages the pretrained parameters.

Another approach to encode the table **T** is to flatten the table cells directly while introducing additional embeddings to the model. Each cell is represented by a local (cell level) template: $h_c \mid h_r \mid v$, where h_c and h_r are the row and column names of cell v. According to TAPAS [15], the following embeddings are required to identify the structural information: the row embedding **r**, the column embedding **c**, and the type embedding **t**. **r** and **c** are built with the coordinates of the cell. **t** represents the input type, which could be the table title, table caption, or table cell. Letting **w** be the word embedding, **p** be the positional embedding, we have the final input representation:

$$\mathbf{e} = \mathbf{w} + \mathbf{t} + \mathbf{p} + \mathbf{r} + \mathbf{c} \tag{8}$$

4.2. Semantic-Parsing-Guided Slot-Filling

4.2.1. Semantic Parsing

The semantic parsing module takes both the template and the table as input and outputs the logic form that can be correctly executed on the table and represents the semantics of the target text. Logic2Text and other similar studies require the model to understand the structural semantics in diversified graph structures, leading to learning difficulties and challenges for model design. We transform the semantic understanding problem into a "translation" problem, which is the main insight of this work. The translation direction is from a natural language text template (with table information) to a program language logic form. The constructed templates and their corresponding logic forms are considered as parallel corpus. The model is only required to complete the translation task, which reduces the difficulty significantly compared to understanding the semantics. We leave the semantic understanding part to an offline interpreter, which is a computer program that checks the grammar validity and executes the logic form.

Building a semantic parser matching the translation goal is simple. The transformer-based encoder–decoder structure is a natural fit for tasks with parallel corpus. We adopt a heuristic approach to align the template slot values with the logic form parameters. The aligned logic form parameters are assigned with the same corresponding slot placeholders of the template. As shown in Figure 3, the overall process to construct a logical template and its corresponding logic form is that we first align the annotated target sentence with the table using entity-linking methods to build slots with unique placeholders. Then, we align the arguments of the annotated logic form with the slot values of the template using heuristic approaches. Finally, the logic form with the same slots as the logical template is

constructed. For example, the logic form of the target sentence "Retail sales growth grew by 0.2% in 2017 compared to 2016" is

```
eq{
    diff{
        hop{filter_eq{all_rows; Year; 2017}; Retail sales growth
            }};
        hop{filter_eq{all_rows; Year; 2016}; Retail sales growth
            }}
    };
    0.2%
} = true
```

The definitions of the functions of the above logic form are explained in Table 3. All functions defined by Logic2Text can be found in the appendix of their original paper.

The logical template built for the example is "[ENT0] grew by [VALUE0] in [TIME0] compared to [TIME1]", which suggests that "retail sales growth", "0.2%", "2017", and "2016" are the aligned slot values. Hence, these slot values are employed to align with the annotated logic form, constructing the following logic form with the same slots corresponding to the logical template.

```
eq{
    diff{
        hop{filter_eq{all_rows; Year; [TIME0]; Retail sales
            growth};
        hop{filter_eq{all_rows; Year; [TIME1]; Retail sales
            growth}
    };
    [VALUE0]
} = true
```

The slots between the logical template and the logic form are tied with the same unique placeholders, so that the arguments (slot values) of the logic form that execute the correct result can be directly filled into the corresponding template slots. We explain how to search the correct arguments in the next subsection.

Table 3. Logic form function definitions.

Function	Arguments	Output	Description
eq	object, Object	bool	Returns if equal
diff	object, Object	object	Returns the difference
hop	Row, table header	object	Returns the column value of the row
filter_eq	View, header string, object	view	Returns the subview with column values equal to object

To train a semantic parser, we use both the table and the logical template as input and the corresponding logic form as output. The training objective is to maximize the overall likelihood of

$$P(S \mid \mathbf{T}, Y_T; \theta) \tag{9}$$

where S is the logic form and θ is the learnable parameter. We minimize the cross-entropy loss $L_{semantic}$ between the generated and golden logic form. Note that the golden logic form is the one with tied slots.

We propose two methods to encode the table information for the semantic parsing module. The first method is the same as the one adopted by the logical template generation module. We feed both the linearized table and the template to the encoder to generate the corresponding logic form. Considering that the linearized table is already encoded in the

logical template generation module, and encoding the table again for the semantic parsing module can be redundant and resource-costly, we allow the semantic parsing module to share the encoded table representation. Specifically, letting h_t^T denote the last encoder layer output (the table representation) of the template generation module and h_t^S denote the last encoder layer output (the template representation) of the semantic parsing module, the final h_t^S would be

$$h_t^S = h_t^T \oplus h_t^S \tag{10}$$

where $\oplus$ denotes vector concatenation. The shared representation saves computing resources and improves the model performance on table encoding.

4.2.2. Slot-Tied Back-Search Algorithm

Given the logic form with undetermined parameters (unfilled slots), we need to search groups of arguments (slot values) that execute the logic form correctly. Such arguments are considered as candidate slot values. We design a dynamic-programming-like breadth-first search program to find correct arguments. The search program can be described as Algorithm 1. We transform the flattened logic form into a graph structure, denoted by $S = \{N_1, N_2, ..., N_m\}$, where N_m is a node of the logic form graph. There are two types of nodes: a function node denotes a subfunction of the logic form, which is also a subgraph, while a text node is always a leaf node (an actual value). The slots can be constructed on text nodes only. The blue and gray nodes in Figure 3 denote function and text nodes, respectively.

The goal of the back-search algorithm is to build a collection of logic forms $C = \{S_1, S_2, ..., S_n\}$, where the slots of each S are filled with positive arguments $P = \{p_1, p_2, ..., p_r\}$. Here the "back-search" concept refers to finding possible arguments based on the given execution result of the logic form. For the logic form defined by Logic2Text, the correct execution result is always "true". The main idea of the back-search algorithm is to breadth-first search the logic form graph and traverse the nodes in reverse order, which is from text nodes to function nodes, from inner functions to outer functions. If the node is a determined text node (an unfilled slot) N_i, we search and verify positive arguments P of N_i.

The search process is to generate a number of possible arguments based on the parameter type. Only the following parameter types can make up a slot: "row", "header", "bool", "num", and "str". Depending on the parameter type, the parameter position, and the function, the possible arguments are listed in Table 4. The search space of possible arguments cannot be reduced for conditions that are not listed in the table. For these conditions, the search space is the full table: $\{row_1, row_2, ...row_n, col_1, col_2, ..., col_n, header_1, header_2, ..., header_n\}$. The possible arguments of some parameter types can not be inferred, such as a "str" or "obj". We assign a unique placeholder *any_obj* as the possible argument and leave it to be determined during recursion.

Table 4. Possible arguments for different parameter types, positions, and functions.

Parameter Type	Position	Function	Possible Arguments
row	any	hop, only	$\{row_1, row_2, ..., row_n\}$
row	any	any but hop, only	*any_obj*
header	any	any	$\{header_1, header_2, ..., header_n\}$
bool	any	any	$\{true, false\}$
num	any	any	$\{1, 2, ..., len(rows)\}$
str, obj	3	any with 3 params and param 2 is header	$\{cell\_1, cell\_2, ..., cell\_n\} \in col_{header}$
str, obj	1, 2	avg, sum, diff	*any_obj*

Algorithm 1 Back-search logic form arguments.

Require: $S = \{N_1, N_2, ..., N_m\}$
Ensure: $C = \{S_1, S_2, ..., S_n\}$
$C \Leftarrow \{\}$
$i \Leftarrow m$
while $i > 0$ **do**
  if N_i is an text node **then**
    if N_i is undetermined (an unfilled slot) **then**
      $P \Leftarrow$ search arguments of N_i
      $P \Leftarrow$ verify arguments of (P, N_i)
      if $P \neq \{\}$ **then**
        for each parameter $p \in P$ **do**
          $L_{clone} = \{N_1^{clone}, N_2^{clone}, ..., N_m^{clone}\} \Leftarrow$ clone L
          assign p to all N_m^{clone} with the same placeholder as N_i
          $C \Leftarrow C \vee \{L_{clone}\}$
        end for
      else
        break search
      end if
    end if
  else[N_i is a function node]
    if the execution result of N_i is unknown **then**
      if N_i is not executable (includes unfilled slots) **then**
        $P \Leftarrow$ search arguments of N_i
        $P \Leftarrow$ verify arguments of (P, N_i)
        if $P \neq \{\}$ **then**
          for each parameter $p \in P$ **do**
            $L_{clone} = \{N_1^{clone}, N_2^{clone}, ..., N_m^{clone}\} \Leftarrow$ clone L
            assign p to the execution result of N_i^{clone}
            $C \Leftarrow C \vee \{L_{clone}\}$
          end for
        else
          break search
        end if
      else[N_i is executable]
        $p \Leftarrow$ execute N_i
        assign p to the execution result of N_i
      end if
    end if
    Back-search logic form arguments of N_i recursively
  end if
  $i \Leftarrow i - 1$
end while

The search process only provides possible arguments which are legal for the function. The verify process checks if a possible argument is correct by executing the function with the argument and verifying the result. Through such a process, we limit the search space as possible with the result of a subfunction to verify if an argument is correct in a reverse way. The result of the subfunction is gathered either from the saved result during recursive searching or by executing the function in real time, if the subfunction is executable. Whether the argument can be verified depends on many conditions, such as if the child or parent nodes are undetermined and if the argument is *any_obj*. As the recursive search carries on, most arguments can be finally determined.

Now we have the verified arguments P for node N_1. For each p of P, a separate branch starting from the current node is formed. In others words, each positive argument of each slot matches an individual branch of the same logic form with different arguments. For a

function node, we check if the function is executable and the result of the subfunction is known. The function is executable only if any subnode of the function is not undetermined. Subfunctions with unknown results need to be processed. If the function is not executable, we search and verify the positive arguments of N_i using the same algorithms. The only difference is that the positive arguments are saved as possible results of this function to reduce search space for its undetermined subnodes. The result of the outer function (node) serves as a constraint on the possible parameters of its inner function (subnode). We also need to maintain multiple branches for each possible function result. If the function is executable, we save the execution result directly. The back-search algorithm needs to be called recursively for each function node. To reduce time cost, we limit the maximum number of logic forms to five. Finally, we obtain a collection of five logic forms with correct arguments. Based on the tied-slots, the logic form arguments can be filled into the corresponding slots of the logical template, building candidate targets which may become the final target.

During inference, the back-search algorithm works in coordination with the model. First, the semantic parsing module generates the logic form S with undetermined arguments based on the table $\mathbf{T}$ and the logical template Y_T. Then, the back-search algorithm takes S as the input and produces a collection of possible logic form arguments. As the logic form and the logical template are slot-tied, the determined arguments produced by the back-search algorithm can be filled into the corresponding slots of the logical template directly. Compared to the search algorithms adopted by weakly-supervised semantic parsing [50] and SP-Acc [4], the search algorithm we proposed works more efficiently with less time cost. The comparison of different search algorithms is shown in Table 5.

Table 5. Comparison of different search algorithms.

	Weakly-Supervised SP	**SP-Acc**	**Our Search Algorithm**
Input	Table & answer	Table & target	Table & logic form
Search target	Logic forms & args	Logic forms & args	Args
Time cost	Medium	High	Low

4.3. Target Ranking

For n logic forms with correct arguments, now we have n candidate targets. Though these logic forms can be correctly executed, which guarantees logical fidelity, the arguments may not match the logical template in terms of fluency and coherence as a natural language sentence. Therefore, a ranking model is required to determine which group of arguments is the best for the logical template. We follow the learning to rank (LTR) task and adopt a pairwise logistic loss function similar to RankNet [51] to rank the candidate targets according to their BLEU scores. The loss function is defined as

$$L_{logistic} = log(1 + e^{-pairwise\_label * pairwise\_logits}) \tag{11}$$

where

$$pairwise\_logits = f(x_i) - f(x_j) \tag{12}$$

$$pairwise\_label = \begin{cases} 1, & label_i > label_j, \\ -1, & label_i \leq label_j. \end{cases} \tag{13}$$

We only need to consider positive labels when $label_i > label_j$, which simplifies the loss function as

$$\begin{aligned} L_{logistic} &= log(1 + e^{-pairwise\_logits}) \\ &= log(1 + e^{-(x_i - x_j)}) \\ &= max(f(x_j) - f(x_i), 0) + log(1 + e^{-|f(x_j) - f(x_i)|}) \end{aligned} \tag{14}$$

To acquire the pairwise logits, we use BERT [52] with a projection layer on the pooled representation "[CLS]" to score the candidate–golden target pairs. Supposing that $x_i = [\texttt{[CLS]}\, \mathrm{T}_{\mathrm{golden}}\, \texttt{[SEP]}\, \mathrm{T}_{\mathrm{candidate}}]$, $\mathrm{T}_{\mathrm{golden}}$ and $\mathrm{T}_{\mathrm{candidate}}$ denote the golden and candidate target, respectively, the score model is defined as

$$f(x_i) = sigmoid(\mathrm{MLP}(\mathrm{BERT}(\mathbf{z}))) \in (0, 1) \tag{15}$$

We calculate the BLEU score for $\mathrm{T}_{\mathrm{candidate}}$ against $\mathrm{T}_{\mathrm{golden}}$ as $label_i$. The training objective is to minimize $L_{logistic}$ between the *pairwise_logits* and the *pairwise_label*. We sum all three losses as the final loss for training:

$$L = L_{template} + L_{semantic} + L_{logistic} \tag{16}$$

During inference, the ranking model predicts a score for each candidate–golden target pair. We choose the candidate target with the highest score as the final generated target.

5. Experiments and Results

We examine the SP-NLG framework with BART [2] (except the ranking module, which is based on BERT [52]), a pretrained language model targeted on NLG tasks with both encoder–decoder structures. It should be noted that our framework can be built on any pretrained language model with a decoder, such as BERT2BERT [1] and T5 [3].

For the baseline model, we use the pretrained model "GPT-TabGen" and a nonpretrained model "Field-Infusing + Trans". Both methods are from the original work of LogicNLG [4]. The motivation of this work is to propose a new perspective of logical natural language generation. The idea of combining semantic parsing techniques with generation approaches provides new insights and possibilities on generating logical inferred content instead of simply restating facts. Thus, comparing the performance of a specific model built on our framework with other models is not a priority of this work. The BART-based model we built on the SP-NLG framework should be considered as a baseline. In fact, each module of this framework can be further optimized with additional model designs or a larger amount of annotated data. We hope the SP-NLG framework may encourage further endeavors on the logical NLG task. As only the Logic2Text dataset provides annotated logic forms, we use the LogicNLG dataset here for data augmentation purposes. The experiments are conducted on the Logic2Text dataset.

5.1. Setup

We follow the same data preprocessing steps adopted by Chen et al. [4]. First, the target is aligned with the given table with a entity parser to build a logical template. Then, we identify the arguments matching the same aligned entities of the target and replace them with unique placeholders. We save the correspondence between the slots of the logical template and the placeholders in order to fill the arguments searched from Algorithm 1.

Both the logical template generation module and the semantic parsing module are initialized with BART, and the embedding weights of the two modules can be either shared or not shared. We conduct experiments on both settings and compare the difference. The target ranking model is built on BERT. The max input and output tokens are set to 1024 and 128, respectively based on the top 90% token length of the dataset examples. We pad the input tokens to the max length with a special "[PAD]" token. Tokens that exceed the maximum length will be truncated. The learning rate is set to 3×10^{-5}, combined with a linear learning rate scheduler. We further employ the Adam optimizer with the epsilon set to 1×10^{-6}. Decoding is conducted via beam search with a beam size of four. Due to limited computing resources, we train our model with eight NVIDIA Tesla V100 32G GPUs. The final batch size combined with gradient accumulation and multiple GPUs is 64. We evaluate both the baseline and our model on the following metrics: BLEU-1/2/3, SP-Acc, and NLI-Acc. The SP-Acc and NLI-Acc are two automatic metrics proposed by

LogicNLG to evaluate the logical fidelity of the generated text from the aspect of parsing and entailment. The metrics are computed with their official released scripts.

5.2. Results

Table 6 shows the comparison between different settings of our approach and the baseline. We observe that (1) the SP-NLG model with back-search outperforms all methods on almost all metrics, (2) shared embedding weights does not further improve the model performance, and (3) due to the best model checkpoint being selected based on the best BLEU-3, the SP-NLG model tends to be optimized for maximizing the BLEU-3 score.

Table 6. Experiment results.

	Surface-Level Fidelity			**Logical Fidelity**	
Model	**BLEU-1**	**BLEU-2**	**BLEU-3**	**SP-Acc**	**NLI-Acc**
Field-Infusing + Trans	37.7	21.0	10.5	38.5	42.4
GPT-TabGen	46.5	30.9	19.9	42.4	66.5
SP-NLG	44.4	30.3	21.5	31.4	56.2
SP-NLG (shared embeddings)	43.1	28.9	19.5	34.1	52.9
SP-NLG (back-search & rank)	84.0	78.9	74.8	48.7	64.6
Golden Target (oracle)	1.0	1.0	1.0	64.3	72.8

Note: The logical template generation module and semantic parsing module are initialized with BART and the target ranking module is initialized with BERT. "Shared embeddings" refers to the embedding weights being shared between the logical template generation module and the semantic parsing module. "Back-search & rank" denotes that only the slot-tied back-search algorithm and the ranking module are employed while the logical template and logic form with slots are given.

Based on the results of the oracle setting, which is evaluated on the golden target, we discover that the upper bounds for SP-Acc and NLI-Acc are 64.3 and 72.8, respectively. The SP-NLG (with back-search) setting assumes that the logical template and the corresponding logic form with slots are given, and only the slot-tied back-search algorithm and the target ranking module are employed for SP-NLG. Comparison between these results with the oracle results suggests that the slot-tied back-search algorithm is more than capable of searching correct arguments (template slot values), which contributes greatly to the logical fidelity of the final generated target. Target ranking is also an effective design to determine the best arguments for the logical template in terms of surface-level fidelity.

Almost all metrics except SP-Acc dropped when sharing the embedding weights of the logical template module with the semantic parser, which suggests that the logical template module and the semantic parser should be considered as individual components. In fact, this could be an advantage. As each module is replaceable, the model is not required to be trained jointly. All modules can be trained offline separately and plugged in/out of the framework.

We choose the best checkpoint of our model based on the BLEU-3 score. To achieve a better result on SP-Acc or NLI-Acc, the evaluation strategy can be changed to select the best checkpoint based on SP-Acc or NLI-Acc. As these metrics are model-based, which is time-costly, we did not try these strategies. Another approach is to optimize the semantic parser. The original work of Logic2Text annotated around 10,000 logic forms; while this may be sufficient for their approach by considering the logic form as a given input, the scale of annotated data is far from enough for fully training a semantic parser. To address this problem, the following methods should be considered:

- **Simplify the logic form grammar.** The grammar of the logic form defined by Logic2Text is complicated. For a fixed amount of annotated data, the model trained on simple logic forms usually gains higher performance than the one trained on complex logic forms. Guo et al. [53] compared the performance of semantic parsing models trained on different types of logic forms denoting the same semantics.

- **Data augmentation.** Data augmentation methods such as OVERNIGHT [44] should be considered by changing slot positions or replacing them with different entities from both the template and logic form to extend the scale of data for training. Data augmentation methods could be limited as the diversity of logic frameworks is usually unchanged.
- **Annotate more data.** The most effective approach to boost model performance is to annotate more data with high quality, though it can be time- and effort-costly, especially for logic form annotation.

6. Conclusions

We propose SP-NLG, a semantic-parsing-guided natural language generation framework. The framework ensures high logical fidelity of the generated target. Instead of requiring the model to understand the logic, we transform the understanding problem into a translation problem by "translating" a textual logical template into a logic form with tied-slots. We further design a back-search algorithm: a offline program to search arguments that correctly executes the logic form in real time. The arguments with the highest ranking score are filled into the logical template slots to form the final generated target. The framework leaves the "calculation" part for the computer program and the "translation" part for the model, which makes the most of each component. Experiment results show that SP-NLG is effective for generating high-logical-fidelity text.

The logical natural language generation task also faces a number of challenges. One of the challenges is determining how to improve the quality of automatic metrics to assist human judgment on logical fidelity. The NLI-based and SP-based metrics are model-based metrics trained on a different dataset, which inevitably introduces errors. Another challenge is allowing the model to truly understand the logic. The reasoning ability of SP-NLG mainly relies on translating natural language text into logic forms and the back-search algorithm. The model itself may not fully understand the logic. Therefore, further effort should be made on this research topic.

For future work, we will focus on improving the semantic parser. Another problem is that the arguments of the logic form are filled directly into the slots of the logical template, which may require rephrasing for better fluency.

Author Contributions: T.L.: Conceptualization, formal analysis, investigation, methodology, software, writing—original draft. S.Z.: Formal analysis, investigation, writing—review and editing. Z.L.: Project administration, conceptualization, writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Publicly available datasets were analyzed in this study. These data can be found here: https://github.com/wenhuchen/LogicNLG, (accessed on 15 March 2021) and here: https://github.com/czyssrs/Logic2Text, (accessed on 15 March 2021).

Acknowledgments: This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 62276017, U1636211, 61672081), the 2022 Tencent Big Travel Rhino-Bird Special Research Program, and the Fund of the State Key Laboratory of Software Development Environment (Grant No. SKLSDE-2021ZX-18).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SP Semantic parsing

References

1. Rothe, S.; Narayan, S.; Severyn, A. Leveraging Pre-trained Checkpoints for Sequence Generation Tasks. *Trans. Assoc. Comput. Linguist.* **2020**, *8*, 264–280. [CrossRef]
2. Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A.; Levy, O.; Stoyanov, V.; Zettlemoyer, L. 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, Online, 5–10 June 2020; pp. 7871–7880.
3. Raffel, C.; Shazeer, N.; Roberts., A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* **2020**, *21*, 1–67.
4. Chen, W.; Chen, J.; Su, Y.; Chen, Z.; Wang, W.Y. Logical Natural Language Generation from Open-Domain Tables. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Online, 5–10 June 2020; pp. 7929–7942.
5. Dzmitry, B.; Cho, K.; Yoshua, B. Neural machine translation by jointly learning to align and translate. In Proceedings of the 3rd International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
6. Lebret, R.; Grangier, D.; Auli, M. Neural Text Generation from Structured Data with Application to the Biography Domain. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, TX, USA, 1–4 November 2016; pp. 1203–1213.
7. Ratish, P.; Li, D.; Mirella, L. Data-to-Text Generation with Content Selection and Planning. In Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; pp. 6908–6915.
8. Puduppully, R.; Dong, L.; Lapata, M. Data-to-text Generation with Entity Modeling. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, 28 July–2 August 2019; pp. 2023–2035.
9. Gong, H.; Bi, W.; Feng, X.; Qin, B.; Liu, X.; Liu, T. Enhancing Content Planning for Table-to-Text Generation with Data Understanding and Verification. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2020, Online, 16–20 November 2020; pp. 2905–2914.
10. Liu, T.; Wang, K.; Sha, L.; Chang, B.; Sui, Z. Table-to-Text Generation by Structure-Aware Seq2seq Learning. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; pp. 4881–4888.
11. Wiseman, S.; Shieber, S.; Rush, A. Challenges in Data-to-Document Generation. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Copenhagen, Denmark, 9–11 September 2017; pp. 2253–2263.
12. Wiseman, S.; Shieber, S.; Rush, A. Learning Neural Templates for Text Generation. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 31 October–4 November 2018; pp. 3174–3187.
13. Liu, T.; Zheng, X.; Chang, B.; Sui, Z. Towards Faithfulness in Open Domain Table-to-text Generation from an Entity-centric View. In Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, Online, 2–9 February 2021; pp. 13415–13423.
14. Pasupat, P.; Liang, P. 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), Beijing, China, 26–31 July 2015; pp. 1470–1480.
15. Herzig, J.; Nowak, P.K.; Müller, T.; Piccinno, F.; Eisenschlos, J. TaPas: Weakly Supervised Table Parsing via Pre-training. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Online, 5–10 July 2020; pp. 4320–4333.
16. OpenAI. ChatGPT: Optimizing Language Models for Dialogue. *OpenAI Blog* **2022** , *1*, 1–20.
17. Chen, Z.; Chen, W.; Zha, H.; Zhou, X.; Zhang, Y.; Sundaresan, S.; Wang, W.Y. Logic2Text: High-Fidelity Natural Language Generation from Logical Forms. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2020, Online, 16–20 November 2020; pp. 2096–2111.
18. Liang, P.; Jordan, M.; Klein, D. 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, Singapore, 2–7 August 2009; pp. 91–99.
19. Lee, S. Natural language generation for electronic health records. *NPJ Digit. Med.* **2018**, *1*, 63. [CrossRef] [PubMed]
20. Wen, T.H.; Gašić, M.; Mrkšić, N.; Su, P.H.; Vandyke, D.; Young, S. Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, 17–21 September 2015; pp. 1711–1721.
21. Budzianowski, P.; Wen, T.H.; Tseng, B.H.; Casanueva, I.; Ultes, S.; Ramadan, O.; Gašić, M. MultiWOZ—A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 31 October–4 November 2018; pp. 5016–5026.
22. Chen, D.L.; Mooney, R.J. Learning to Sportscast: A Test of Grounded Language Acquisition. In Proceedings of the 25th International Conference on Machine Learning, Helsinki, Finland, 5–9 June 2008; pp. 128–135.
23. Novikova, J.; Dušek, O.; Rieser, V. The E2E Dataset: New Challenges For End-to-End Generation. In Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Saarbrücken, Germany, 15–17 August 2017; pp. 201–206.
24. Gardent, C.; Shimorina, A.; Narayan, S.; Perez-Beltrachini, L. The WebNLG Challenge: Generating Text from RDF Data. In Proceedings of the 10th International Conference on Natural Language Generation, Santiago de Compostela, Spain, 4–7 September 2017; pp. 124–133.
25. Wang, Q.; Pan, X.; Huang, L.; Zhang, B.; Jiang, Z.; Ji, H.; Knight, K. Describing a Knowledge Base. In Proceedings of the 11th International Conference on Natural Language Generation, Tilburg University, The Netherlands, 5–8 November 2018; pp. 10–21.

26. Parikh, A.; Wang, X.; Gehrmann, S.; Faruqui, M.; Dhingra, B.; Yang, D.; Das, D. ToTTo: A Controlled Table-To-Text Generation Dataset. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online, 16–20 November 2020; pp. 1173–1186.
27. Li, T.; Fang, L.; Lou, J.G.; Li, Z. TWT: Table with Written Text for Controlled Data-to-Text Generation. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2021, Punta Cana, Dominican Republic, 7–11 November 2021; pp. 1244–1254.
28. Chen, W.; Wang, H.; Chen, J.; Zhang, Y.; Wang, H.; Li., S.; Zhou, X.; Wang, W.Y. TabFact: A Large-scale Dataset for Table-based Fact Verification. In Proceedings of the 7th International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
29. Reiter, E.; Dale, R. Building applied natural language generation systems. *Nat. Lang. Eng.* **1997**, *3*, 57–87. [CrossRef]
30. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
31. Song, K.; Tan, X.; Qin, T.; Lu, J.; Liu, T.Y. MASS: Masked Sequence to Sequence Pre-training for Language Generation. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; pp. 5926–5936.
32. Hu, Z.; Yang, Z.; Liang, X.; Salakhutdinov, R.; Xing, E.P. Toward Controlled Generation of Text. In Proceedings of the 34th International Conference on Machine Learning, Sydney, NSW, Australia, 6–11 August 2017; pp. 1587–1596.
33. Dou, L.; Qin, G.; Wang, J.; Yao, J.G.; Lin, C.Y. Data2Text Studio: Automated Text Generation from Structured Data. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Brussels, Belgium, 31 October–4 November 2018; pp. 13–18.
34. Sennrich, R.; Haddow, B.; Birch, A. Controlling Politeness in Neural Machine Translation via Side Constraints. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, San Diego, CA, USA, 12–17 June 2016; pp. 35–40.
35. Kikuchi, Y.; Neubig, G.; Sasano, R.; Takamura, H.; Okumura, M. Controlling Output Length in Neural Encoder-Decoders. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, Austin, TX, USA, 1–4 November 2016; pp. 1328–1338.
36. Li, J.; Galley, M.; Brockett, C.; Spithourakis, G.; Gao, J.; Dolan, B. A Persona-Based Neural Conversation Model. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Berlin, Germany, 7–12 August 2016; pp. 994–1003.
37. Wong, Y.W.; Mooney, R. Learning for Semantic Parsing with Statistical Machine Translation. In Proceedings of the Human Language Technology Conference of the NAACL, Main Conference, New York, NY, USA, 4–9 June 2006; pp. 439–446.
38. White, M. Efficient Realization of Coordinate Structures in Combinatory Categorial Grammar. *Res. Lang. Comput.* **2006**, *4*, 39–75. [CrossRef]
39. Liang, P.; Jordan, M.; Klein, D. Learning Dependency-Based Compositional Semantics. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Portland, OR, USA, 19–24 June 2011; pp. 590–599.
40. Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; et al. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 31 October–4 November 2018; pp. 3911–3921.
41. Wang, B.; Shin, R.; Liu, X.; Polozov, O.; Richardson, M. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Online, 5–10 July 2020; pp. 7567–7578.
42. Zettlemoyer, L.; Collins, M. Online Learning of Relaxed CCG Grammars for Parsing to Logical Form. In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), Prague, Czech Republic, 28–30 June 2007; pp. 678–687.
43. Kwiatkowksi, T.; Zettlemoyer, L.; Goldwater, S.; Steedman, M. Inducing Probabilistic CCG Grammars from Logical Form with Higher-Order Unification. In Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, MIT Stata Center, Cambridge, MA, USA, 9–11 October 2010; pp. 1223–1233.
44. Wang, Y.; Berant, J.; Liang, P. Building a Semantic Parser Overnight. 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), Beijing, China, 26–31 July 2015; pp. 1332–1342.
45. Guu, K.; Pasupat, P.; Liu, E.; Liang, P. From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vancouver, BC, Canada, 30 July–4 August 2017; pp. 1051–1062.
46. Das, R.; Zaheer, M.; Thai, D.; Godbole, A.; Perez, E.; Lee, J.Y.; Tan, L.; Polymenakos, L.; McCallum, A. Case-based Reasoning for Natural Language Queries over Knowledge Bases. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Punta Cana, Dominican Republic, 7–11 November 2021; pp. 9594–9611.
47. Bhagavatula, C.S.; Noraset, T.; Downey, D. Methods for Exploring and Mining Tables on Wikipedia. In Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics, Chicago, IL, USA, 11–14 August 2013; p. 18–26.

48. Ashish, V.; Noam, S.; Niki, P.; Jakob, U.; Llion, J.; Aidan, N.G.; Ł ukasz, K.; Illia, P. Attention is All you Need. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 6000–6010.
49. Yin, P.; Neubig, G.; Yih, W.T.; Riedel, S. TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Online, 5–10 July 2020; pp. 8413–8426.
50. Berant, J.; Chou, A.; Frostig, R.; Liang, P. Semantic Parsing on Freebase from Question-Answer Pairs. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Grand Hyatt Seattle, Seattle, WA, USA, 18–21 October 2013; pp. 1533–1544.
51. Burges, C.; Shaked, T.; Renshaw, E.; Lazier, A.; Deeds, M.; Hamilton, N.; Hullender, G. Learning to Rank Using Gradient Descent. In Proceedings of the 22nd International Conference on Machine Learning, Bonn, Germany, 7–11 August 2005; pp. 89–96.
52. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 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), Minneapolis, MN, USA, 2–7 June 2019; pp. 4171–4186.
53. Guo, J.; Liu, Q.; Lou, J.G.; Li, Z.; Liu, X.; Xie, T.; Liu, T. Benchmarking Meaning Representations in Neural Semantic Parsing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online, 16–20 November 2020; pp. 1520–1540.