SORTIE : Dependency-Aware Symbolic Reasoning for Logical Data-to-text Generation

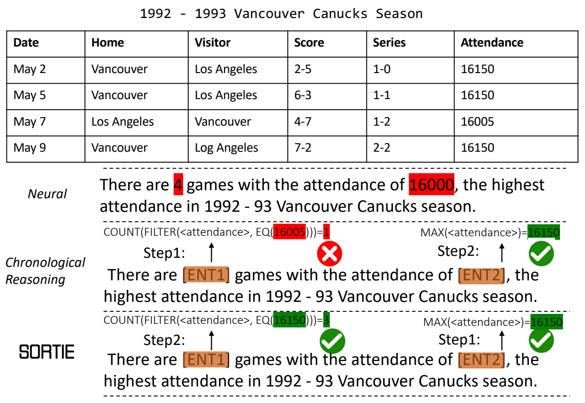
Xueliang Zhao[1]†, Tingchen Fu[2]†, Lemao Liu[3], Lingpeng Kong[1], Shuming Shi[3] Rui Yan[2,4]*
[1]The University of Hong Kong [3]Tencent AI Lab
[2]Gaoling School of Artificial Intelligence, Renmin University of China
[4]Engineering Research Center of Next-Generation Intelligent Search and Recommendation, Ministry of Education
`{xlzhao,lpk}@cs.hku.hk` `{tingchenfu,ruiyan}@ruc.edu.cn`
`{redmondliu,shumingshi}@tencent.com`

Abstract

Logical data-to-text generation is a representative task in measuring the capabilities of both language generation and complex reasoning. Despite the introduction of reasoning skills in generation, existing works still rely on neural language models to output the final table description. However, due to the inefficacy of neural language models in complex reasoning, these methods inevitably have difficulty working out key entities in the description and might produce unfaithful descriptions. To alleviate these issues, we propose a dependency-aware symbolic reasoning framework that reasons out each entity in the table description with our designed table-compatible programming language. To figure out the dependency relationship among entities, we devise an entity scheduling mechanism to determine the order of programme synthesis such that the reasoning of an entity only relies on other "resolved" entities. Experiments on three datasets and three backbones show that ours outperforms previous methods not only in surface-level fidelity but also in logical fidelity. Notably, the proposed framework enhances GPT-2, BART and T5 with an absolute improvement of $5.7\% \sim 11.5\%$ on SP-Acc.

1 Introduction

Generating logical-consistent sentences is an integral part of human intelligence and has attracted broad research interests in the field of natural language processing recently (Chen et al., 2020a,c; Wei et al., 2022; Creswell et al., 2022; Kazemi et al., 2022). One of the most prominent attempts to investigate this capability in neural models is logical data-to-text generation (Chen et al., 2020a), which requires conducting intricate reasoning on the given table to produce a logically-consistent description (Chen et al., 2020a).


1992 - 1993 Vancouver Canucks Season
Neural
There are 4 games with the attendance of 16000, the highest attendance in 1992 - 93 Vancouver Canucks season.
Chronological Reasoning
COUNT(FILTER(<attendance>, EQ(16005)))=1
MAX(<attendance>)=16150
Step1:
Step2:
There are [ENT1] games with the attendance of [ENT2], the highest attendance in 1992 - 93 Vancouver Canucks season.
SORTIE
COUNT(FILTER(<attendance>, EQ(16150)))=3
MAX(<attendance>)=16150
Step2:
Step1:
There are [ENT1] games with the attendance of [ENT2], the highest attendance in 1992 - 93 Vancouver Canucks season.

Date	Home	Visitor	Score	Series	Attendance
May 2	Vancouver	Los Angeles	2-5	1-0	16150
May 5	Vancouver	Los Angeles	6-3	1-1	16150
May 7	Los Angeles	Vancouver	4-7	1-2	16005
May 9	Vancouver	Log Angeles	7-2	2-2	16150

Figure 1: A table from LogicNLG (Chen et al., 2020a). Without symbolic reasoning, neural models fabricate entities that do not appear in the table; with symbolic reasoning in chronological order ([ENT1]→[ENT2]), it is difficult to directly work out [ENT1] since [ENT1] depends on 16150 ([ENT2]); SORTIE perform dependency-aware symbolic reasoning ([ENT2]→[ENT1]) and solves two entities correctly.

To realize intricate reasoning on tables, a wide spectrum of methods have been proposed such as table pre-training with diverse self-supervision tasks (Andrejczuk et al., 2022; Liu et al., 2022; Zhao et al., 2022) or coarse-to-fine text deliberation (Chen et al., 2020a). Generally, existing methods attempt to internalize the reasoning ability with neural model parameters, but they take the direct output from neural models as the final table description, ignoring the fact that neural language models suffer from hallucination when performing open-ended generation task (Maynez et al., 2020). In addition, since neural language models (even the large scale ones) often suffer from limited multi-step reasoning ability, methods based on these thus struggle to reason out key entities in description and thus perform poorly on generation faithfulness (Liu et al., 2022).

With the recent surge of combining contempo-

†Xueliang Zhao and Tingchen Fu contribute equally to this work. This work was done during their internship at Tencent AI Lab.
*Corresponding author: Rui Yan (`ruiyan@ruc.edu.cn`).

rary deep learning methods and symbolic AI, we get inspiration from recent neural symbolic literature (Gao et al., 2022) that decouples complex reasoning with language generation. In specific, we delegate the inference of entities mentioned in the table description to a programme interpreter. The interpreter executes our generated python-like programme, thus working out the entities correctly and alleviating hallucination.

However, synthesizing such a programme to infer entities is not a trivial task due to two major challenges: First, though there are some domain-specific programming languages on natural text (Chen et al., 2020b; Gupta et al., 2019), we need to design a table-compatible and easy-to-execute programming language to support the reasoning over entities in the table. Second, the entities to infer are not independent but have a complex dependency relationship and the inference of one might rely on the others. For instance, as is shown in Figure 1, we can not count the appearance of 16150 unless we work out the 16150 first. Thus, figuring out the synthesis order of the entities is fundamental to the reasoning process. To make it worse, there is no human annotation of programmes or synthesis order for the entities.

To mitigate the aforementioned problems, we propose SORTIE (**S**ymb**O**lic **R**easoning with en**TI**ty sch**E**duling), a framework that reasons out each named entities in the table description with dependency-aware symbolic reasoning. Specifically, (1) we introduce a table-compatible programming language that defines the grammar and operators for reasoning on the tabular data and delegates the reasoning of each entity to the execution of the programme; (2) we devise a new pipeline to predict the dependency relationship between entities and synchronously synthesize the programmes to work out each entity; (3) we heuristically search pseudo labels for both the programmes and synthesis order of entities. We further adjust the sample weight of pseudo labels to alleviate the spurious correlation issue with a self-adaptive training algorithm.

To summarize, our contributions are three-fold: (1) To our best knowledge, we are the first to model the dependency relationship between entities in the table description and propose a new pipeline to synchronously predict the order of entities and reason them out one by one. (2) We successfully apply symbolic reasoning to logical data-to-text generation tasks. To support the reasoning of entities, we design a table-compatible python-like programming language that is more feature-rich and table-friendly compared to previous ones. (3) We empirically validate the efficacy of SORTIE on three benchmarks for logical data-to-text generation, including LogicNLG (Chen et al., 2020a), Logic2Text (Chen et al., 2020c), and SciGen (Moosavi et al., 2021). When applied on GPT-2, BART or T5, our methods substantially enhance the SP-Acc which is a crucial measurement for logical fidelity with an absolute improvement of $5.7\% \sim 11.5\%$.

2 Related Work

2.1 Data-to-text Generation

Early data-to-text generation mainly focuses on surface-level descriptions of the table contents (Lebret et al., 2016; Liu et al., 2018; Ma et al., 2019; Wang et al., 2020). However, in spite of generation fluency, neural-based generation models struggle to perform rich inference based on the facts in table (Chen et al., 2020a,c). To make up for that, logical table-to-text generation is proposed as a new task with the aim of generating logically-consistent descriptions from open-domain tables (Chen et al., 2020a,c).

In recent years, to endow neural models with complex reasoning ability, DCVED (Chen et al., 2021) applies causal intervention methods to reduce the spurious correlation in entities. PLOG (Liu et al., 2022) and TABT5 (Andrejczuk et al., 2022) introduce table-to-logical-form or table denoising as self-supervision tasks in the pre-training stage. Similarly, REASTAP (Zhao et al., 2022) introduces 7 pre-training tasks to mimic the 7 types of reasoning skills of humans. It is worth noting that this line of research is orthogonal to ours since they primarily concentrate on developing training instances that reflect the desired reasoning skills. Similar to the programming language in our proposal, Saha et al. (2022) introduce logic string as an intermediate step to guide generation. However, the surface realization from the logic string to the final description is very prone to hallucinations as it is done purely by neural language models.

2.2 Symbolic Reasoning

The idea of symbolic reasoning has garnered considerable attention in numerous natural language processing and computer vision tasks. Andreas et al. (2016) make the first attempt to combine symbolic reasoning with visual question answering,

Category	Operator	Arguments	Output	Description
Value Operator	`SUM` `DIFF` `DIV`	v0: a numerical value v1: a numerical value	a numerical value	Return the sum, difference or ratio of two values.
	`COUNT`	v: list	a numerical number	Count the number of element.
	`SELECT`	v0: a list. v1: an index	value	Select an element from a list.
	`MAX` `MIN`	v: a list	value	Return the maximum or minimum value of a list.
List Operator	`FILTER`	v0: a list. v1: a condition	list	Filter elements that meets specific conditions from a list.
	`UNIQUE`	v: list	list	Remove the dublicated value.
	`ARGMAX` `ARGMIN`	v: list	index	Return the index of the maximum or minimum value.
	`ARGWHERE`	v0: a list. v1: a condition	list of index	Return the list of index of the elements that meets specific conditions.
Boolean Operator	`EQ` `GE` `GEQ` `LE` `LEQ`	v: a numerical parameter	a condition	Constitute a condition for FILTER and ARGWHERE.

Table 1: The operators used in our symbolic reasoning.

parsing questions into linguistic substructures and constructing question-specific deep networks from smaller modules that each tackle one subtask. Following this work, numerous efforts have been made to directly predict the instance-specific network layouts in an end-to-end manner (Hu et al., 2017), to alleviate the requirement for mediate supervision on semantic parsers (Hu et al., 2018; Mao et al., 2019), to infer the answer with a purely symbolic executor (Yi et al., 2018), and to conduct visual co-reference resolution (Kottur et al., 2018). Very recently, Gupta et al. (2019) and Chen et al. (2020b) concurrently proposed using neural symbolic approaches to answer questions in machine reading comprehension, which demonstrates advantages in numerical reasoning and interpretability. Compared to the previous tasks, which only need to derive a single entity or value, the task of logical table-to-text generation requires the generation of a complete natural language sentence containing multiple entities or logical types.

3 Methodology

3.1 Problem Formulation and Overview

Given a table T, the task of the logical data-to-text generation is to generate a description Y that is both fluent and logically consistent. Following Chen et al. (2020a), we decompose the problem into a two-step pipeline: template generation and entity instantiation. Specifically, we first generate a template $\tilde{Y}$ with n placeholders[1] $P_1, P_2, \cdots, P_n$, and then seek for a sequence of named entities $[e_1, e_2, \cdots, e_n]$ to fill in the placeholders to form a complete description Y. We focus on the second step in this work while following Chen et al. (2020a) for the first step.

Road Map We first introduce our designed table-compatible programming language in § 3.2. The architecture of our model, mostly composed of three components, is illustrated at § 3.3. Finally, the learning algorithm to deal with the scarcity of human annotation labels is described at § 3.4.

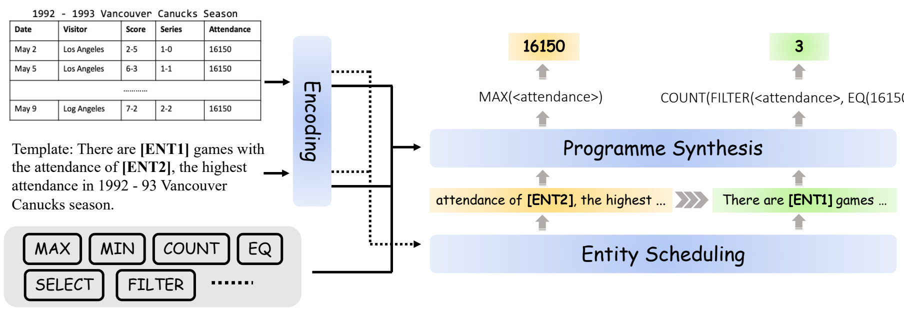
3.2 Table-compatible programming language

To reason out the named entities faithfully from the table, we introduce a programming language composed of a series of specially designed operators and named entities as operands. We list our operators in Table 1.

Based on the type of output, we roughly sort all the operators into three categories: value operators, list operators, and boolean operators. Borrowed from Chen et al. (2020b), the value operators are designed to select a value from the table (`SELECT`, `MAX` and `MIN`) and calculate some simple arithmetic (`SUM`, `DIFF`, `DIV` and `COUNT`). Apart from that, since the layout of the data in a table is in the format of columns with each column including homogeneous information, we design list operators (`FILTER` or `UNIQUE`) and index operators (`ARGMAX`, `ARGMIN`, `ARGWHERE`) directed against a single column [2] to obtain a new list or indies of a list respectively. Finally, we also include boolean operation (`EQ`, `GE`, `LE`, `GEQ`, `LEQ`) as an integral part

[1]Template is a draft table description that all the named entities are replaced temporarily with special "[ENT]" tokens.

[2]In this work, we use the column and the list interchangeably.



1992 - 1993 Vancouver Canucks Season

Date	Visitor	Score	Series	Attendance
May 2	Los Angeles	2-5	1-0	16150
May 5	Los Angeles	6-3	1-1	16150
…………				
May 9	Log Angeles	7-2	2-2	16150

Template: There are **[ENT1]** games with the attendance of **[ENT2]**, the highest attendance in 1992 - 93 Vancouver Canucks season.

Encoding
MAX
MIN
COUNT
EQ
SELECT
FILTER
16150
3
MAX(<attendance>)
COUNT(FILTER(<attendance>, EQ(16150)))
Programme Synthesis
attendance of [ENT2], the highest ...
There are [ENT1] games ...
Entity Scheduling

Figure 2: Working flow of SORTIE . The table and template are first encoded into dense representations, and the entity scheduling mechanism dynamically picks the placeholders (i.e., "[ENT2]" and "[ENT1]") that will be resolved by programme synthesis and execution.

of `FILTER` and `ARGWHERE`.

Compared with the domain-specific language in Chen et al. (2020b) and Gupta et al. (2019), the major novelty lies in its compatibility with structured tabular data, for example, the list operator to accurately pick one or more specific values from a table according to our requirement. We note a concurrent work (Zhou et al., 2022) also puts forward a table-supported programming language. Different from ours, it only operates on linearized tables in natural language form but does not support raw structured tables. Generally speaking, we extrapolate the traditional symbolic reasoning operators in reading comprehension to adapt to a more complex scenario. At the same time, our operators still keep the compositionality, the ability to generate complex programmes by compositionally applying the operators. We leave more detailed discussions about the connections to other domain-specific languages in Appendix A.

3.3 Main Components

The main working flow of the proposed method is illustrated in Figure 2. In a nutshell, it is composed of three parts, (1) encoding, (2) entity scheduling and (3) programme synthesis and execution, which we will elaborate on below respectively.

Encoding. Given a table T, we first linearize the table into a natural language form following Chen et al. (2020a). Then we concatenate the linearized table with the template into a single sequence and transform it into a dense representation with a pre-trained language model (PLM): $\mathbf{H}^{enc} = [\mathbf{h}_1^{enc}, \cdots, \mathbf{h}_l^{enc}]$, where l is the total length of the linearized table and the template. During the training phase, the template is obtained by substituting the entities in the golden description with placeholders. At inference, the template is obtained with the same PLM.

Entity Scheduling. As mentioned before, entities within a description are not isolated and there exists a latent dependency relationship between each other. If the entities are reasoned in chronological order (i.e., from left to right), the programmer may struggle to synthesize a suitable programme when faced with entities whose dependencies are unsolved yet. To this end, we devise an entity scheduling mechanism to dynamically select to-be-solved placeholders that only depends on currently known entities.

In detail, we employ a 1-layer GRU to realize scheduling. At the t-th step, with the entity[3] reasoned out at the last step, we concatenate its word embedding together with the dense representation of the corresponding placeholder as input. The former provides the semantics of the last entity while the dense representation of the placeholder carries the contextual and positional information in the template, which is helpful to reason out the next placeholder. The input is used to update the inner hidden state of GRU $\mathbf{h}_{t-1}^s$.

Then, we calculate the probability of selecting a placeholder in the template $\tilde{Y}$ according to the similarity between $\mathbf{h}_t^s$ and the embeddings of the placeholders:

$$\Pr(P_i) = \frac{\exp(f_{sim}(\mathbf{h}_t^s, \mathbf{h}_i^{plh}))}{\sum_{j=1}^{n} \exp(f_{sim}(\mathbf{h}_t^s, \mathbf{h}_j^{plh}))} \qquad (1)$$

where $[\mathbf{h}_1^{plh}, \cdots, \mathbf{h}_n^{plh}]$ is a slice of $\mathbf{H}^{enc}$ corresponding to the placeholders in the template, and $f_{sim}(\cdot, \cdot)$ is a similarity function implemented as

[3]Strictly speaking, a placeholder is unsolved and is a temporal substitution for an entity. In what follows, we may slightly abuse the word "entity" to refer to an unsolved placeholder.

the dot product. $\Pr(P_i)$ is the probability of selecting the i-th placeholder in the template to solve at the t-th step. We choose the placeholder with the highest probability:

$$\lambda_t = \operatorname*{argmax}_i \Pr(P_i), \tag{2}$$

and use the dense representation of the chosen placeholder to initialize the hidden state of the programmer, which will be illustrated later. To deal with the undifferentiable problem in selecting a single placeholder, we apply gumbel-softmax (Jang et al., 2016) in the training stage.

Programme Synthesis and Execution. Inspired by Gupta et al. (2019) and Chen et al. (2020b), we propose to synthesize programmes in our designed table-compatible programming language to reason out each entity. Specifically, the programme synthesis is conducted by a 1-layer GRU. At the t-th time step, we first update the hidden state $\mathbf{h}_{t-1}^p$ by the embedding of the last generated operator/operand. Next, we calculate the relevance between $\mathbf{h}_t^p$ and all the operator/operand embeddings to predict the next operator/operand op_t. With a generated programme $[op_1, \cdots, op_{l_p}]$, we execute it on the table T to reason out current entity.

To find more details and specific implementation, please refer to Appendix B.

3.4 Learning Strategy

3.4.1 Weak Supervision

Since the human annotation about entity scheduling and programme of entities are absent, we initiate the learning of the proposed model with weak supervision:

Weak Supervision on Programme Synthesis. Heuristically tagging pseudo labels is a common practice to solve the paucity of human annotation. Following previous works (Min et al., 2019; Chen et al., 2020b), we collect a group of the most common programmes as heuristic set $\mathcal{H}$. For every entity e_i that appears in the description, we exhaustively enumerate the heuristic set $\mathcal{H}$ and find a subset, $\mathcal{S}_i^p$, that could derive the target entity as programme candidates for e_i. More details about the $\mathcal{S}_i^p$ could be found in Appendix C.1.

Weak Supervision on Entity Scheduling. To deal with the paucity of annotation on entity dependency relationships, again, we explore constructing supervision signals with pseudo programmes.

Algorithm 1 The proposed learning algorithm.

```
1: Input: A dataset of ${(T, Y)}$ pairs, programme synthesis model $p_\theta$ and entity scheduling model $q_\phi$, moving-average momentum $\alpha$, maximum training step $M$, hyper-parameter $M_0, \beta$.
2: for $m \leftarrow 1$ to $M$ do
3:     $(T, Y) \leftarrow$ batch data.
4:     $\mathcal{D} = \emptyset$.
5:     Replace the entities in $Y$ to obtain a template $\tilde{Y}$.
6:     For all the placeholders $P_1, P_2, \cdots, P_n$ in $\tilde{Y}$, construct their programme candidate set $\mathcal{S}_1^p, \mathcal{S}_2^p, \cdots, \mathcal{S}_n^p$.
7:     for $\mathcal{P} \in \mathcal{S}_1^p \times \mathcal{S}_2^p \times \cdots \times \mathcal{S}_n^p$ do
8:         if A topological order $\mathcal{T}$ exists and $|\mathcal{D}| \le \beta$ then
9:             $\mathcal{D} = \mathcal{D} \cup (T, \tilde{Y}, \mathcal{P}, \mathcal{T})$.
10:        end if
11:    end for
12:    Calculate the likelihood for each suite of programme and scheduling order in $\mathcal{D}$: $[\hat{w}_1, \hat{w}_2, \cdots, \hat{w}_{|\mathcal{D}|}]$,
13:    if $m \ge M_0$ then
14:        Update the pseudo label $w_i \leftarrow \alpha \times w_i + (1-\alpha) \times \hat{w}_i, i \in \{1, 2, \cdots, |\mathcal{D}|\}$.
15:    end if
16:    Optimize $p_\theta$ and $q_\phi$ on $\mathcal{D}$ according to Eq. 3.
17: end for
18: Return: programme synthesis model $p_\theta$ and entity scheduling model $q_\phi$
```

Specifically, we define a n-tuple programme candidates $(p_1, p_2, \cdots, p_n)$ as *a suite of programme* $\mathcal{P}$, where p_i is a programme candidate for the i-th placeholder. In fact, it is an element from the cartesian product $\mathcal{S}_1^p \times \mathcal{S}_2^p \times \cdots \mathcal{S}_n^p$. For any $\mathcal{P}$, we could construct a dependency graph with an edge pointing from entity e_i to entity e_j if the reasoning of entity e_j is dependent on entity e_i. If the dependency graph is a directed acyclic graph (DAG) then we use the topological order $\mathcal{T}$ to serve as a possible candidate for an entity scheduling order.

3.4.2 Self-adaptive Training

Although we could obtain more than one suite of programmes for an entity and many possible scheduling orders through weak supervision, usually only one is correct while others are spurious solutions (Min et al., 2019). Inspired by Huang et al. (2020), we employ the self-adaptive learning algorithm to eliminate the influence of the spurious correlation in the training process.

Given all suites of programmes and corresponding scheduling order $\mathcal{D} = \{(\mathcal{P}_i, \mathcal{T}_i)\}_{i=1}^m$ for a template where m is the number of programme suites with a legal topological order. We consider a soft pseudo label for each suite: $w = [w_1, w_2, \cdots, w_m]$ which satisfy $\sum_{i=1}^m w_i = 1, w_i \in [0, 1]$. w_i is initialized to be $\frac{1}{m}$ at the beginning. For each iteration, we calculate and normalize the likelihood for each suite $[\hat{w}_1, \hat{w}_2, \cdots, \hat{w}_m]$ with the programmer, then we update the pseudo label by $w_i \leftarrow \alpha \times w_i + (1-\alpha) \times \hat{w}_i$. α is a hyper-parameter and serves as the momentum of the exponential-moving-average scheme. The learning objective of the programmer and entity scheduling is then defined as:

$$\max_{\theta} \sum_{i=1}^{m} w_i \log p_\theta(\mathcal{P}_i | T, \tilde{Y}) \\ \max_{\phi} \sum_{i=1}^{m} w_i \log q_\phi(\mathcal{T}_i | T, \tilde{Y}), \tag{3}$$

where p_θ and q_ϕ represents the programme synthesis and the entity scheduling, with trainable parameter θ and ϕ respectively.

A high-level learning algorithm is summarized in Algorithm 1. We leave the specific implementation of the training strategy in Appendix C.3 due to the space constraint.

4 Experimental Setup

4.1 Datasets

We conduct experiments on three benchmark datasets for logical table-to-text generation: LogicNLG (Chen et al., 2020a), Logic2Text (Chen et al., 2020c) and SciGen (Moosavi et al., 2021). The test set of SciGen was split by the data owners into the "Computation and Language" (C&L) domain, and the "Other" domain, which primarily contains examples from "Machine Learning" (ML) papers. More details about these three datasets can be found in Appendix D.

4.2 Evaluation Metrics

Automatic Evaluation. We evaluate the surface-level and logical fidelity of all models, as described in previous works (Chen et al., 2020a, 2021). For surface-level fidelity, we calculate multi-reference BLEU-n (abbrv. B-n, $n = 1, 2, 3$). In terms of logical fidelity, we employ SP-Acc and NLI-Acc following previous works (Chen et al., 2020a, 2021). The former aims to measure the logical consistency through a semantic parser, while the latter evaluates the entailment degree. More specific implementations of the automatic evaluation metrics are provided in Appendix E.1.

Human Evaluation. We conduct the human evaluation by selecting 300 samples randomly from the test set of LogicNLG, Logic2Text and SciGen respectively, and hiring 6 well-educated native speakers to conduct qualitative analysis on the descriptions produced by our model and all competitive baselines. Two criteria are used by the annotators to assess the descriptions' quality: *Language Fluency* and *Factual Correctness*. Each annotator assigns a score from $\{0, 1, 2\}$ (representing "bad", "fair" and "good" respectively) to each description for each aspect, and Fleiss' Kappa (Fleiss, 1971) is used to gauge the level of agreement between all annotators. We leave more details about the setup of human evaluation in Appendix E.2.

4.3 Baseline Models

The following models are selected as baselines: (1) **GPT-Coarse-to-Fine**: A template-based model that first generates a global logical structure of the description with all entities and numbers replaced by "[ENT]", and then conducts surface realization based on the logical structure (Chen et al., 2020a). (2) **DCVED**: A variational auto-encoder model that employs a confounder to represent the spurious entities and a mediator to represent the precisely picked entities (Chen et al., 2021). (3) **PLOG**: Proposed by Liu et al. (2022), the model is first pre-trained on a table-to-logic generation task, and then fine-tuned on downstream table-to-text tasks. (4) **REASTAP**: Zhao et al. (2022) propose 7 table reasoning skill and construct training examples respectively to learn the 7 reasoning skill by pre-training on generative table QA tasks.

5 Results and Discussions

5.1 Main Results

Table 2 and Table 3 show the performance of our model on LogicNLG, Logic2Text and SciGen. From the tables, we can observe that ours substantially outperform previous methods, especially on SP-Acc and NLI-Acc, which proves the effectiveness of the proposed method. When compared with PLOG and REASTAP, two representative methods that learn reasoning skills through pre-training, we conclude that symbolic reasoning as well as our table-compatible programming language is helpful to promote faithfulness.

Human Evaluation. The human evaluation results are shown in Table 4. Although our model performs comparably to other baselines in terms of language fluency, it attains a significant improvement

Model	LogicNLG					Logic2Text				
	Surface-level			Logical Fidelity		Surface-level			Logical Fidelity	
	B-1	B-2	B-3	SP-Acc	NLI-Acc	B-1	B-2	B-3	SP-Acc	NLI-Acc
GPT-small	45.9	26.3	13.0	42.2	73.0	48.7	30.1	19.3	41.2	63.4
GPT-Coarse-to-Fine	46.6	26.8	13.3	42.7	72.2	48.3	31.9	20.8	42.5	68.9
DCVED	49.5	28.6	15.3	43.9	76.9	48.9	32.7	21.4	43.9	73.8
SORTIE (Ours)	**49.8**	**30.1**	**16.9**	**49.3**	**79.9**	**50.4**	**33.0**	**22.7**	**47.2**	**84.3**
T5-large	53.4	34.1	20.4	48.4	85.9	51.8	35.0	24.2	47.8	89.3
PLOG	53.7	34.1	20.4	54.1	89.0	52.2	35.5	24.9	52.8	90.2
SORTIE (Ours)	**54.7**	**34.9**	**21.0**	**58.5**	**89.9**	**53.1**	**36.1**	**25.2**	**55.0**	**91.6**
BART-large	54.5	34.6	20.6	49.6	85.4	51.3	34.5	23.1	47.8	89.0
PLOG	54.9	35.0	21.0	50.5	88.9	52.1	35.2	22.9	51.8	91.1
REASTAP	52.5	32.5	18.9	54.8	89.2	51.6	34.7	24.3	53.3	90.3
SORTIE (Ours)	**56.2**	**35.8**	**21.4**	**57.8**	**89.3**	**52.6**	**35.6**	**24.8**	**59.3**	**94.1**

Table 2: Automatic evaluation results on the test sets of LogicNLG and Logic2Text. From top to bottom, the models in three blocks use GPT2-small, T5-large and BART-large as backbone respectively. The numbers in bold are the best results.

Model	C&L					Other				
	Surface-level			Logical Fidelity		Surface-level			Logical Fidelity	
	B-1	B-2	B-3	SP-Acc	NLI-Acc	B-1	B-2	B-3	SP-Acc	NLI-Acc
T5-large	13.5	4.9	1.7	28.9	93.9	11.9	4.4	1.7	20.7	88.3
PLOG	16.4	5.5	2.0	32.2	97.8	12.1	5.6	2.2	25.0	94.2
SORTIE (Ours)	**17.8**	**7.0**	**2.7**	**34.7**	**99.0**	**18.3**	**7.5**	**3.0**	**27.4**	**96.3**
BART-large	17.1	6.7	2.2	35.6	98.6	17.2	7.1	2.5	33.6	98.4
PLOG	18.2	8.0	3.3	38.3	98.8	18.4	7.6	3.2	34.4	98.5
REASTAP	18.7	8.1	3.2	38.7	98.9	18.6	8.2	3.4	37.3	98.1
SORTIE (Ours)	**21.2**	**9.2**	**4.0**	**41.3**	**99.2**	**21.0**	**9.5**	**4.3**	**39.9**	**98.9**

Table 3: Automatic evaluation results on two test splits of SciGen. From top to bottom, the models in two blocks use T5-large and BART-large as the backbone respectively. The numbers in bold are the best results.

Model	LogicNLG			Logic2Text			SciGen		
	Language Fluency	Factual Correctness	Kappa	Language Fluency	Factual Correctness	Kappa	Language Fluency	Factual Correctness	Kappa
GPT-Coarse-to-Fine	1.61	1.44	0.68	1.54	1.52	0.69	1.51	1.37	0.81
DCVED	1.62	1.47	0.69	1.58	1.50	0.64	1.44	1.36	0.76
PLOG	1.69	1.61	0.74	1.65	1.58	0.77	1.54	1.49	0.68
REASTAP	1.67	1.63	0.71	1.61	1.59	0.68	1.57	1.51	0.71
SORTIE (Ours)	**1.70**	**1.73**	0.65	**1.66**	**1.74**	0.71	1.57	**1.64**	0.79

Table 4: Human evaluation results on LogicNLG, Logic2Text and SciGen. Numbers in bold mean the best performance.

in terms of factual correctness, which is consistent with the automatic evaluation results. All kappa values are more than 0.6, demonstrating agreement between the annotators.

5.2 Ablation Study

Apart from the main experiments, to have a better understanding of how each component and mechanism contribute to surface-level fidelity and logical fidelity, we conduct an ablation study with the following variants: (1)-*symbolic*: The programmer and the discrete symbolic reasoning are removed. Placeholders in the template are filled up with entities whose hidden state is most similar[4] to the hidden state of placeholders. (2)-*scheduling*: The topological order among the entities is disregarded. Instead, we use the embedding of the placeholders $P_1, P_2, \cdots, P_n$ as the initial hidden state for the programmer and perform symbolic reasoning simultaneously. (3)-*both*: Both the programmer and the decoder are discarded. In this case, we use the embeddings of the placeholders to predict the entities simultaneously. (4)-*self*: The self-adaptive training is removed and we optimize our model to marginal maximum likelihood (MML) estimation when there exist multiple pseudo programme labels

[4]measured with dot production

Model	Surface-level Fidelity			Logical Fidelity	
	BLEU-1	BLEU-2	BLEU-3	SP-Acc	NLI-Acc
Ours	49.8	30.1	16.9	49.3	79.9
-symbolic	47.2	29.7	16.3	46.8	73.5
-scheduling	48.2	28.8	16.7	45.2	74.1
-both	46.3	26.5	13.7	42.5	71.1
-self	47.4	26.2	13.3	42.3	68.0

Table 5: Ablation experiment results on the test set of LogicNLG.

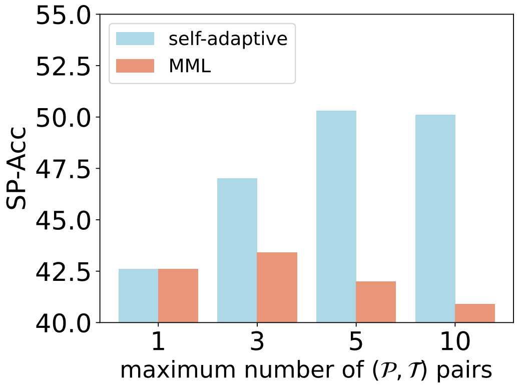


Figure 3: SP-Acc vs. the maximum number of the pseudo labels on LogicNLG.

and topological orders.

The experiment results of ablation are shown in Table 5. We can observe that: (1) Both symbolic reasoning and topological entity decoding is vital for the performance of our approach since the removal of either would cause an evident drop in fidelity. (2) The surface-level fidelity is less sensitive to different variants, and the chief advantage of our approach lies in improving logical fidelity.

5.3 Effect of the Pseudo Label Quantity

To see how the proposed learning algorithm works with respect to the size of $(\mathcal{P}, \mathcal{T})$ pairs, we vary the maximum threshold for the pseudo labels (or the β in Algorithm 1). Performance of the variants *-self*, which is actually maximum marginal likelihood (MML), is also included for comparison and the results are shown in Figure 3.

It is obvious that the fidelity of the MML optimization deteriorates with the size of the pseudo label set increases. We gauge that is because there is usually only one correct topological order and programme for each entity. More candidates would inevitably introduce noise and mislead the model to assign high probabilities to spurious solutions. Notably, our method is immune to spurious solutions, thus exhibiting a different tendency and keeping competitive.

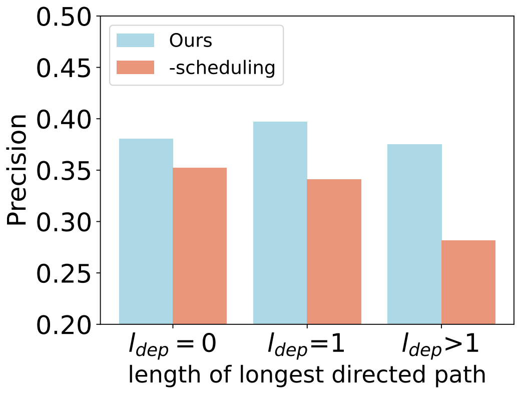


Figure 4: Precision vs. length of the longest directed path in the dependency graph.

5.4 Effect of Entity Scheduling

To have a closer look at how the complexity of the inter-dependency relationship influence the precision of entity reasoning and how the entity scheduling mechanism takes effect, we bin all the test case of the LogicNLG (Chen et al., 2020a) into three buckets according to the length of the longest directed path l_{dep} in the dependency graph. The results are shown in Figure 4. We can see that with entity scheduling, the precision slightly fluctuates with different l_{dep} but does not show an obvious drop in performance. In comparison, when scheduling is removed and the entities are inferred in left-to-right chronological order, the performance of reasoning declines, possibly due to its inability to deal with more complicated dependency scenarios and directly work out all entities without considering their dependency. Take the case in Figure 1 as an example, if we deal with [ENT1] first according to chronological order, it is challenging to directly synthesize a programme like "`[COUNT]([FILTER]`(<attendance>,`[EQ]([MAX]`(<attendance>))))". But if we have reasoned [ENT2] out, the programme for [ENT1] is simplified as "`[COUNT]([FILTER]`(<attendance>,`[EQ]`(16150)))"

6 Conclusion

We propose a neural symbolic approach for logical data-to-text generation. With a table-compatible programming language, our approach automatically synthesizes a programme to reason out each entity. Specifically, to handle the inter-dependency between entities, we propose an entity scheduling mechanism that dynamically predicts the reasoning order of entities such that the entity to be reasoned at each iteration has a minimum dependency on "unseen" entities. In addition, to deal with the paucity of human annotations of both programmes and scheduling order, we put forward a weak supervision method and a self-adaptive learning algorithm to mitigate the spurious correlation issue. Evaluation results on three benchmarks show that our model can significantly outperform state-of-the-art approaches, and considerably boost the performance of a pre-trained language model in terms of logical fidelity.

Ethical Considerations

This paper will not pose any ethical problems. First, logical data-to-text generation is an old task in natural language processing, and several papers about this task are published at ACL conferences. Second, the datasets used in this paper have been used in previous papers.

Limitations

The paper presents a dependency-aware symbolic reasoning approach for logical data-to-text generation. All technologies built upon the large-scale PLM more or less inherit their potential harms (Bender et al., 2021). Besides, we acknowledge some specific limitations within our methods:

1. Data-to-text generation is essentially a one-to-many problem since there is more than one plausible and logically-consistent description given a specific table. Our approach has little control over the diversity and the logical form of the generated template. It is also possible that our approach only generates trivial or naive descriptions if trivial data dominate in the training dataset.

2. Our work mostly focuses on the named entities in the description, but logical consistency is not all about entities. The syntactic structure or other semantic information also has an influence on generation fidelity, and we leave the symbolic reasoning for more complex logical structures or formats as our future work.

3. Our table-compatible programming language is mainly designed for simple flat tables, and extra operators are necessary before it could be applied to all tables, especially hierarchical tables where its header exhibits a multi-level structure (Cheng et al., 2022).

4. Currently, it is difficult to directly integrate GPT-3 (Brown et al., 2020) or other LLMs into SORTIE to substitute the PLM backbones. The reason is that LLM can not be used for encoding since we have no access to the dense representation in an LLM. It might be plausible to only use LLM to generate a template and use another PLM to do encoding, but we leave this exploration to our future work.

Acknowledgement

We thank all the reviewers and chairs for their suggestions and recommendation. This work was supported by National Natural Science Foundation of China (NSFC Grant No. 62122089), Beijing Outstanding Young Scientist Program NO. BJJWZYJH012019100020098, and Intelligent Social Governance Platform, Major Innovation & Planning Inter-disciplinary Platform for the "Double-First Class" Initiative, Renmin University of China. We wish to acknowledge the support provided by Public Policy and Decision-making Research Lab, Renmin University of China and the Public Computing Cloud, Renmin University of China.

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 39–48.

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

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

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

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

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

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V. Le. 2020b. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *International Conference on Learning Representations*.

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

Zhoujun Cheng, Haoyu Dong, Zhiruo Wang, Ran Jia, Jiaqi Guo, Yan Gao, Shi Han, Jian-Guang Lou, and Dongmei Zhang. 2022. HiTab: A hierarchical table dataset for question answering and natural language generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1094–1110, Dublin, Ireland. Association for Computational Linguistics.

Antonia Creswell, Murray Shanahan, and Irina Higgins. 2022. Selection-inference: Exploiting large language models for interpretable logical reasoning. *arXiv preprint arXiv:2205.09712*.

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

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. 2019. Neural module networks for reasoning over text. *arXiv preprint arXiv:1912.04971*.

Ronghang Hu, Jacob Andreas, Trevor Darrell, and Kate Saenko. 2018. Explainable neural computation via stack neural module networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 53–69.

Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. 2017. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 804–813.

Lang Huang, Chao Zhang, and Hongyang Zhang. 2020. Self-adaptive training: beyond empirical risk minimization. In *Advances in Neural Information Processing Systems*, volume 33.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Arthur B Kahn. 1962. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562.

Seyed Mehran Kazemi, Najoung Kim, Deepti Bhatia, Xin Xu, and Deepak Ramachandran. 2022. Lambada: Backward chaining for automated reasoning in natural language. *arXiv preprint arXiv:2212.13894*.

Satwik Kottur, José MF Moura, Devi Parikh, Dhruv Batra, and Marcus Rohrbach. 2018. Visual coreference resolution in visual dialog using neural module networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 153–169.

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

Ao Liu, Haoyu Dong, Naoaki Okazaki, Shi Han, and Dongmei Zhang. 2022. Plog: Table-to-logic pretraining for logical table-to-text generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, page 5531–5546, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Shuming Ma, Pengcheng Yang, Tianyu Liu, Peng Li, Jie Zhou, and Xu Sun. 2019. Key fact as pivot: A two-stage model for low resource table-to-text generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2047–2057.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. 2019. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919, Online. Association for Computational Linguistics.

Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2019. A discrete hard em approach for weakly supervised question answering. *arXiv preprint arXiv:1909.04849*.

Nafise Sadat Moosavi, Andreas Rücklé, Dan Roth, and Iryna Gurevych. 2021. 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)*.

Swarnadeep Saha, Xinyan Velocity Yu, Mohit Bansal, Ramakanth Pasunuru, and Asli Celikyilmaz. 2022. Murmur: Modular multi-step reasoning for semi-structured data-to-text generation. *arXiv preprint arXiv:2212.08607*.

Zhenyi Wang, Xiaoyang Wang, Bang An, Dong Yu, and Changyou Chen. 2020. Towards faithful neural table-to-text generation with content-matching constraints. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1072–1086.

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

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. 2018. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. *Advances in neural information processing systems*, 31.

Yilun Zhao, Linyong Nan, Zhenting Qi, Rui Zhang, and Dragomir Radev. 2022. Reastap: Injecting table reasoning skills during pre-training via synthetic reasoning examples. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, page 9006–9018, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yongwei Zhou, Junwei Bao, Chaoqun Duan, Youzheng Wu, Xiaodong He, and Tiejun Zhao. 2022. Unirpg: Unified discrete reasoning over table and text as program generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, page 7494–7507, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

A Connection to Other Domain-Specific Languages

A.1 Connection to the NMN for Text

Gupta et al. (2019) propose Neural Module Network (NMN) for reasoning over text (in the form of unstructured paragraphs) which also designs a suite of operations for reasoning. Although NMN has achieved remarkable success and enjoys good interpretability, one of its major limitations is that it cannot be deployed with other forms of data, e.g., the structured table in this work. Additionally, since each operation in NMN is implemented as a neural model, the dearth of human-annotated data for each module makes model training more challenging. On the other hand, we follow Chen et al. (2020b) to implement operations with parameter-free modules that can be directly executed through an "interpreter".

A.2 Connection to NeRd

NeRd (Chen et al., 2020b) is a neural symbolic model that integrates discrete reasoning in reading comprehension. In addition to value operations (e.g., `DIFF`/`SUM`, `COUNT`, `MAX`/`MIN`, `ARGMAX`/`ARGMIN`, which are identical to value operators in our programming language), NeRd also designs operations for picking spans or numbers from the passage and question, including `SPAN`, `VALUE` and `KEY-VALUE`. Similar to NMN, NeRd is also incompatible with tabular data. Although we can flatten a structured table into an unstructured natural language form, this simple strategy may lose some of its feasibility when operating on raw table data. For example, we can directly fetch a column of data from a raw table by utilizing the name of a column, but with NeRd, we must repeatedly invoke the `SPAN` operation and predict the start and end indices of each span.

A.3 Connection to UniRPG

We notice that a contemporaneous work UniRPG (Zhou et al., 2022) also proposes a collection of domain-specific operations to carry out discrete reasoning on tables. Although our value operators (e.g., `SUM`, `DIFF`, and `DIV`) and those of UniRPG have some similarities, there still exist some crucial differences. In what follows, we first briefly describe the operations in UniRPG and then go into more detail about how our table-compatible language differs from UniRPG.

A Brief Review of Operations in UniRPG. In general, the operations in UniRPG can be roughly categorized into atomic operations and higher-order operations. For atomic operations, aside from the `SPAN` and `VALUE` introduced in NeRd, UniRPG also introduces `CELL` and `CELL_VALUE` which possess similar functionality to `SPAN` and `VALUE` respectively but operate on the linearized table. In order to perform higher-order operations, UniRPG enriches the original set of arithmetic operations in NeRd by introducing

1. `MULTI_SPANS` which returns all the extracted text fragments or numbers;
2. `TIMES` and `DIV` which compute the product and quotient between two numbers;
3. `AVG` which returns the average value of the argument numbers;
4. `CHANGE_R` that outputs the rate of change between two numbers.

Differences from UniRPG. The core difference is that UniRPG supports complex reasoning on tabular data simply by linearizing a structured table into unstructured natural language, but ours could directly operate on the structured table and thus is able to capture the structural relationship among cell values. To put it more plainly, tabular data is essentially relational data and cell values in the same row (column) share some common feature or attribute. In light of this, our designed programming language can easily fetch a column of cell values and do further analysis (with `MAX`/`MIN`, `COUNT` and so on), or associate two cell values in a row (by the combination of `SELECT` and `ARGWHERE`). On the other hand, when an operation on a whole column is necessary, UniRPG entirely relies on the programmer to predict the start and end index for all cells in an interested column and use `CELL` operation to pick them out one by one.

Another difference lies in searching for a specific list that satisfies some conditions. A number of operations need to take a list of cells as one of their arguments. For instance, in order to count games with the largest number of attendees, we would need to pass a list of games meeting the requirement to the `COUNT` operation. In these cases, UniRPG infers the list by independently deriving elements in it using the `CELL` operation. Such an approach makes it challenging to scale to situations when many more objects meet the requirements and need to be retrieved since it demands the programmer to understand the intricate semantics of the natural language (e.g., the meaning of "largest") to precisely predict the start and end indices of each object. In contrast, our model can directly operate on the raw table and shrink the scope by recursively invoking the `FILTER` and boolean operations, which shifts the responsibility for predicting the indices from the programmer to symbolic execution.

B More Implementation Details of Different Components

B.1 Encoding

The first step of encoding is to flatten a structured table into an unstructured paragraph, or "table linearization". Following Chen et al. (2020a), supposing $T_{i,j}$ is the value of table cell in the i-th row and j-th column, we transform the original table T into a paragraph by horizontally scanning each cell $T_{1,1} \to T_{1,C_T} \to T_{R_T,C_T}$ in the table, where R_T and C_T are the numbers of rows and columns in the table respectively. For example, the table in Figure 2 is linearized as "Given the table titled *1992 - 1993 Vancouver Canucks Season*, in row 1, the Date is May 2, the Visitor is Los Angeles, the Score is 2-5, the Series is 1-0, the Attendance is 16150; in row 5, the Date is May 5, the Visitor is Los Angeles, the Score is 6-3, the Series is 1-1, the Attendance is 16150; ...... in row 9, the Date is May 9, the Visitor is Los Angeles, the Score is 7-2, the Series is 2-2, the Attendance is 16150. Start Describing: "

We recognize all the entities in the golden table description with heuristics:

1. All the cell values that appear in the table;
2. The named entities recognized by spaCy [5] whose entity labels are among {cardinal, date, time, quantity } and not appear in table caption.

We replace all the detected named entities in the table description Y with a special placeholder "[ENT]" to obtain a template $\tilde{Y}$, and finetune a PLM on $(T, \tilde{Y})$ pairs, following previous work (Chen et al., 2020a).

B.2 Entity Scheduling

At the t-th step, with the last reasoned entity $e_{\lambda_{t-1}}$, we obtain its semantic representation $\mathbf{h}^{ent}_{\lambda_{t-1}}$ by looking up and averaging the word embedding of all sub-tokens in the entity. Then, we update the inner hidden state of GRU by:

$$\mathbf{h}^s_t = \mathrm{GRU}_s(\mathbf{h}^s_{t-1}, f_{mlp}([\mathbf{h}^{plh}_{\lambda_{t-1}}; \mathbf{h}^{ent}_{\lambda_{t-1}}])), \quad (4)$$

where $f_{mlp}(\cdot)$ is a multi-layer perceptron network and $[\cdot;\cdot]$ denotes the concatenation of two vectors.

At inference, the next placeholder is chosen with argmax operation. However, argmax is not differentiable and hinders the gradient propagation from subsequent programme synthesis to the scheduling or encoding part. To solve the problem, at training phase, we apply gumbel-softmax (Jang et al., 2016) to sample the next placeholder:

$$\lambda_t \sim \mathrm{Gumbel}(\Pr(P_i), \tau), \quad (5)$$

where τ is the temperature.

B.3 Program Synthesis and Execution

At the t-th time step, we first update the hidden state $\mathbf{h}^p_{t-1}$ of the 1-layer GRU which is responsible for programme synthesis by the embedding of the last generated operator/operand $f_{emb}(op_{t-1})$:

$$\mathbf{h}^p_t = \mathrm{GRU}_p(\mathbf{h}^p_{t-1}, f_{emb}(op_{t-1})), \quad (6)$$

where $f_{emb}(\cdot)$ is an embedding function that converts a programme operator/operand to its embedding, and op_{t-1} is a programme operator/operand generated at $(t-1)$-th step. The definition of f_{emb} is divided into three cases:

- If op_t is from resolved entities, then $f_{emb}(op_t) = \mathbf{E}^{ent}\mathbb{1}_{\omega(op_t)}$, where $\omega(\cdot)$ returns the index of op_t in the resolved entities $[e_{\lambda_1}, \cdots, e_{\lambda_{l_e}}]$ and $\mathbb{1}_\omega$ is a one-hot vector with a one at index ω and zeros otherwise. $\mathbf{E}^{ent}$ is the embedding matrix of the resolved entities and defined as follows:

$$\mathbf{E}^{ent} = \mathbf{W}^{ent}[\mathbf{h}^s_1, \cdots, \mathbf{h}^s_{l_e}], \quad (7)$$

  where $\mathbf{W}^{ent}$ is a trainable parameter, and l_e denotes the number of resolved entities so far;

- If op_t is from Table T or template $\tilde{Y}$, then $f_{emb}(op_t) = \mathbf{E}^{enc}\mathbb{1}_{\tilde{\omega}(op_t)}$, where $\tilde{\omega}(\cdot)$ returns the index of op_t in the linearized table with the template. $\mathbf{E}^{enc}$ serves as the embedding matrix of the table and template, and is defined as follows:

$$\mathbf{E}^{enc} = \mathbf{W}^{enc}\mathbf{H}^{enc}, \quad (8)$$

[5] https://spacy.io

where $\mathbf{W}^{enc}$ is a training parameter and $\mathbf{H}^{enc}$ is the dense representation of linearized table with the template as defined in § 3.3;

- If op_t is from the reserved operators, then $f_{emb}(op_t)$ is defined as $f_{emb} = \mathbf{E}^{res}\mathbb{1}_{\hat{\omega}}(op_t)$, where $\hat{\omega}(\cdot)$ returns the index of op_t in the reserved operators, and $\mathbf{E}^{res}$ is the embedding matrix of reserved operators and implemented a training parameter.

After that, $\mathbf{h}_t^p$ performs attention on $[f_{emb}(op_1), \cdots, f_{emb}(op_{t-1})]$ and $\mathbf{H}^{enc}$ to obtain a context-aware representation $\tilde{\mathbf{h}}_t^p$:

$$\tilde{\mathbf{h}}_t^p = \mathbf{W}^{att}[f_{o-att}(\mathbf{h}_t^p); f_{h-att}(\mathbf{h}_t^p); \mathbf{h}_t^p], \quad (9)$$

where $\mathbf{W}^{att}$ is a trainable parameter, $f_{o-att}(\cdot)$ returns the attended representation of the operator/operand embeddings and is defined as:

$$\begin{aligned} f_{o-att}(\mathbf{h}_t^p) &= \sum_{i=1}^{t-1} \tilde{\alpha}_i f_{emb}(op_i), \\ \tilde{\alpha}_i &= \frac{\exp(\mathbf{h}_t^p \cdot f_{emb}(op_i))}{\sum\limits_{j=1}^{t-1} \exp(\mathbf{h}_t^p \cdot f_{emb}(op_j))}. \end{aligned} \quad (10)$$

The attended representation of the dense representations $\mathbf{H}^{enc}$, $f_{h-att}(\mathbf{h}_t^p)$, is defined in a similar way:

$$\begin{aligned} f_{h-att}(\mathbf{h}_t^p) &= \sum_{i=1}^{l} \hat{\alpha}_i \mathbf{h}_i^{enc}, \\ \hat{\alpha}_i &= \frac{\exp(\mathbf{h}_t^p \cdot \mathbf{h}_i^{enc})}{\sum\limits_{j=1}^{l} \exp(\mathbf{h}_t^p \cdot \mathbf{h}_j^{enc})}. \end{aligned} \quad (11)$$

The following step is to predict the next token op_t using $\tilde{\mathbf{h}}_t^p$. We first compute the similarity score between $\tilde{\mathbf{h}}_t^p$ and each column in $[\mathbf{E}^{ent}; \mathbf{E}^{enc}; \mathbf{E}^{res}]$ where $[\cdot;\cdot;\cdot]$ means concatenating three matrices along the column axis, and then acquire op_t which corresponds to the index with the highest similarity score.

Finally, we execute the generated programme $[op_1, \cdots, op_{l_p}]$ on the table T to reason out the entity $e_{\lambda_{l_e+1}}$.

C Details about Learning Strategy

C.1 Programme Heuristic set

When pruning for the possible programme candidate of an entity, we exhaustively search within a heuristic set $\mathcal{H}$ listed below, which includes the most common and typical "programme templates" in tabular reasoning. Specifically, we fill <list_name> and <value> with all the possible column names and cell values in the table to instantiate each "programme template" into a real programme. If the execution result of the programme is the correct entity, then we add the instantiated programme into the candidate set $\mathcal{S}_i^p$ for an entity e_i.

They are by no means complete or cover all the possible situations, but we find it is sufficient in our experiment.

- `MAX` <list_name>
- `MIN` <list_name>
- `SELECT` <list_name> `ARGMAX` <list_name>
- `SELECT` <list_name> `ARGMIN` <list_name>
- `SELECT` <list_name> `(ARGWHERE` <list_name> `(EQ` <value> `) )`
- `SELECT` <list_name> `(ARGWHERE` <list_name> `(GE` <value> `) )`
- `SELECT` <list_name> `(ARGWHERE` <list_name> `(LE` <value> `) )`
- `SELECT` <list_name> `(ARGWHERE` <list_name> `(GEQ` <value> `) )`
- `SELECT` <list_name> `(ARGWHERE` <list_name> `(LEQ` <value> `) )`
- `COUNT` <list_name>
- `COUNT (UNIQUE` <list_name>`)`
- `COUNT (FILTER` <list_name> `EQ` <value>`)`
- `COUNT (FILTER` <list_name> `GEQ` <value>`)`
- `COUNT (FILTER` <list_name> `LEQ` <value>`)`
- `COUNT (FILTER` <list_name> `GE` <value>`)`
- `COUNT (FILTER` <list_name> `LE` <value>`)`
- `SUM` <value> <value>
- `DIFF` <value> <value>
- `DIV` <value> <value>

C.2 Topological Sorting of Entities

When seeking for weak supervision signals of the entity scheduling order, we enumerate all the possible combinations of programmes $(p_1, p_2, \cdots, p_n) \in \mathcal{S}_1^p \times \mathcal{S}_2^p \times \mathcal{S}_3^p \times \cdots \times \mathcal{S}_n^p$ for $e_1, e_2, \cdots, e_n$ in the table description. We treat every entity as a vertex and add a direct edge pointing from e_i to e_j if e_i appears in p_j to construct a dependency graph $\mathcal{G}$. Kahn's algorithm (Kahn, 1962) is used to judge whether the graph is a DAG and find out a possible topological order of entities if so. Note that a DAG can have more than one topological order since two entities having no interdependency can exchange. In the implementation, we only keep the order that exchangeable entities follow the left-to-right chronological order in the description.

C.3 Self-adaptive Training

When calculating the $\hat{w}$ for a pair of programme suite and scheduling order $(\mathcal{P}, \mathcal{T})$, where $\mathcal{P}$ is a suite of programme $(p_1, p_2, \cdots, p_n)$ and $\mathcal{T}$ in implementation is a sequence $[\lambda_1, \lambda_2, \cdots, \lambda_n]$ where λ_i is the index in left-to-right chronological order for the i-th entity in topological order.

We first calculate the log-likelihood for each $(\mathcal{P}, \mathcal{T})$ in a case:

$$\tilde{w} = \log p_\theta(\mathcal{P}|T, \tilde{Y}) + \log q_\phi(\mathcal{T}|T, \tilde{Y}), \quad (12)$$

where the first part is the likelihood of the programme suite:

$$\begin{aligned} &\log p_\theta(\mathcal{P}|T, \tilde{Y}) \\ &= \sum_{j=1}^{n} \log p_\theta(p_{\lambda_j}|T, \tilde{Y}, e_{\lambda_{1:j-1}}) \\ &= \sum_{j=1}^{n} \sum_{t=1}^{l_p} \log p_\theta(op_t|T, \tilde{Y}, e_{\lambda_{1:j-1}}, op_{1:t-1}), \end{aligned} \quad (13)$$

and the second part is the likelihood of the scheduling order:

$$\begin{aligned} &\log q_\phi(\mathcal{T}|T, \tilde{Y}) \\ &= \sum_{j=1}^{n} \log q_\phi(\lambda_j|T, \tilde{Y}, e_{\lambda_{1:j-1}}) \end{aligned} \quad (14)$$

Finally, we normalize the likelihood among all possible $(\mathcal{P}, \mathcal{T})$ pairs in a case:

$$\hat{w}_i = \frac{\tilde{w}_i}{\sum_{j=1}^{m} \tilde{w}_j} \quad (15)$$

We also endow the PLM to learn predicting template $\tilde{Y}$ given the Table T in the training process and optimize the PLM with maximum likelihood estimation.

D Dataset Statistics

We conduct experiments on the following three benchmarks for logical data-to-text generation:

LogicNLG (Chen et al., 2020a). This dataset is constructed based on the TabFact (Chen et al., 2019), by taking the statements that are entailed by the tabular knowledge as the target text. Tables in this dataset are crawled from Wikipedia and cover a wide range of topics.

Logic2Text (Chen et al., 2020c). This dataset is collected by employing AMT workers to label the statement of each table. Specifically, the workers are encouraged to choose diversified logic types and write descriptions in a creative tone rather than using template-like terminology. Despite the fact that the data owners provide logic forms as well, we only employ the table-description pairs following the setting in prior work (Chen et al., 2021).

SciGen (Moosavi et al., 2021). This dataset is established by collecting tables from scientific articles along with their corresponding descriptions. The tables in SciGen mostly contain numerical values and arithmetic reasoning is required to synthesize the description. The test set was split by the data owners into the "Computation and Language" (C&L) domain, and the "Other" domain, which primarily contains examples from "Machine Learning" (ML) papers. The table-description pairs in the training and development sets are taken from "C&L" articles. We choose the medium-size variant in our experiments.

To facilitate reproducibility, we adopt the datasets shared by the data owners and conduct preprocessing strictly following the released code. The statistics about these three datasets can be found in Table 6.

E More Details about Evaluation Metrics

E.1 Automatic Evaluation

We evaluate the surface-level fidelity and the logical fidelity of all models, as described in previous works (Chen et al., 2020a, 2021). For surface-level fidelity, we calculate multi-reference

	LogicNLG			Logic2Text			SciGen		
	Train	Valid	Test	Train	Valid	Test	Train	Valid	Test
# Statements	28,450	4,260	4,305	8,566	1,095	1,092	13,607	3,452	492(C&L)+546(Other)
# Tables	5,682	848	862	4,549	500	500	13,607	3,452	492(C&L)+546(Other)
Avg. # of words per statement	14.08	14.63	14.77	16.83	16.55	16.54	103.50	107.49	96.39(C&L)/98.81(Other)

Table 6: Statistics of the three datasets.

BLEU-n ($n = 1, 2, 3$) which are based on n-gram matching between the models' generations and gold references. We use B-n as an abbreviation for BLUE-n. Following Chen et al. (2021), we construct the multi-reference test set of the Logic2Text dataset by aggregating the references from the same table into a test data point. In terms of logical fidelity, we employ SP-Acc and NLI-Acc following previous works (Chen et al., 2020a, 2021). Specifically, SP-Acc aims to examine whether the logical representations of the generated descriptions, which are obtained by a semantic parser, are consistent with the table's facts. While NLI-Acc targets evaluating the entailment score between the table and the generated description based on a pre-trained Table-BERT (Chen et al., 2019). All automatic evaluation metrics are calculated using the official code released on `https://github.com/wenhuchen/LogicNLG`.

E.2 Human Evaluation

According to Chen et al. (2020a,c), automatic evaluation scores are not sufficient for precise evaluation of factual and logical correctness. Because of this, we conduct the human evaluation by selecting 300 samples randomly from the test set of LogicNLG, Logic2Text and SciGen respectively, and hiring 6 undergraduates from the department of linguistics in our school to conduct qualitative analysis on the descriptions produced by our model and all competitive baselines. We pay 20 cents for each case. To obscure their sources, the generated descriptions are mixed up at random. Two criteria are used by the annotators to assess the descriptions' quality: (1) *Language Fluency*: whether the description is fluent and free of grammatical errors, and (2) *Factual Correctness*: whether the description is factually supported by the table. Each annotator assigns a score from $\{0, 1, 2\}$ (representing "bad", "fair" and "good" respectively) to each description for each aspect. Each description receives two scores for the aforementioned aspects, and Fleiss' Kappa (Fleiss, 1971) is used to gauge the level of agreement between all annotators.

F More Implementation Details about Experiment and Hyperparameter

For template generation, We perform experiments on three backbones: GPT-2 (117M), BART-large (406M), and T5-large (770M). Theoretically, any pre-trained language model could be our backbone. We employ beam search with a beam size of 5. For entity scheduling and programme synthesis, the dimension of the hidden state in two 1-layer unidirectional GRU are both 512. The temperature for gumbel-softmax is $\tau = 1.0$ and we keep the temperature unchanged through the training process. The f_{mlp} in entity scheduling is a 2-layer MLP network and the hidden sizes are both set to be 512. We apply greedy search when decoding programme tokens. For self-adaptive learning, we set α and β to be 0.9 and 5 respectively, the pseudo labels are kept fixed within the first $M_0 = 500$ steps in the training process. All models are trained with Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We sweep the learning rate from $[5e-6, 1e-5, 2e-5, 4e-5, 6e-6, 8e-5]$ and the best-found learning rate is $1e-5$; We sweep batch size from $[16, 32, 64, 128, 256]$ and the best-found batch size is 32. We set the weight decay as $1e-2$ and sweep the warm-up steps from $[500, 1000, 2000, 4000]$. The best found warm-up step is 1000. Early stopping on validation is adopted as a regularization strategy. All models are trained on an 8×RTX 3090 Ti machine on 5 hours. We report the performance averaged over three repetitive experiments.

G More Experiment Analysis

G.1 More Analysis about Effects of Entity Scheduling

To have a better understanding of how the entity scheduling mechanism promotes the precision of

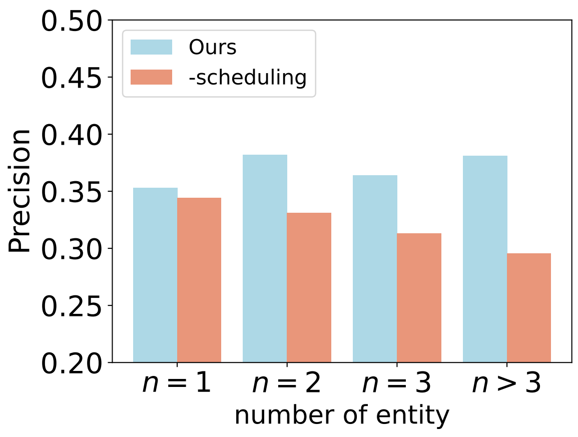


Figure 5: Entity reasoning precision vs. the number of entities.

entity reasoning, we bin all the test cases of LogicNLG (Chen et al., 2020a) into four bins according to the number of entities in the description. The results are shown in Figure 5. We can observe a similar trend to Figure 4. With the number of entities increasing, variant *-scheduling* exhibits evident deterioration. We conjecture the reason is that a table description with more entities is more likely to have complicated dependency relationships among entities, and thus more difficult to reason out. But with entity scheduling, the precision is barely impacted by the number of entities. Note that the templates used in this experiment are derived from golden description, rather than generated by PLM.

G.2 Analysis about Inference Speed

Model	*-symbolic*	*-scheduling*	SORTIE	CTF
Inference Time	394.79	404.29	404.82	1181.40

Table 7: Average inference time (ms) of SORTIE and three other variants or baselines. CTF = Coarse-to-Fine

To investigate whether entity scheduling leads to serious latency at inference, we measure the decoding time of SORTIE in comparison with variant *-scheduling*, *-symbolic* and baseline method Coarse-to-Fine with BART-large backbone. The experiment results are shown in Table 7. From the table, we can see that SORTIE has comparable latency with *-symbolic* and *-scheduling*. Or in other words, programme synthesis and entity scheduling do not enhance generation fidelity at the sacrifice of decoding speed. Besides, SORTIE costs much less time than Coarse-to-Fine, since the latter requires a PLM to first generate a template and then a completed description, which results in low efficiency.

Year	Men's singles	Women's singles
1990	nicholas hall	stephanie spicer
...	...	...
1995	tam kai chuen	song yang
1996	tam kai chuen	li feng
1997	nicholas hall	li feng
1998	geoffrey bellingham	li feng
...	...	...

Template: In 1996, the Women's singles competitor was [ENT1], which appears [ENT2] times.
Topological Order:
[START] → [ENT1] → [ENT2] → [END]
Programme:
[ENT1]: `SELECT` (Women's singles, `ARGWHERE` (Year, `EQ` (1996))) = li feng;
[ENT2]: `COUNT` (`FILTER` (Women's Singles, `EQ` ([ENT1]))) = 3.

Table 8: A table from LogicNLG with caption *New zealand open (badminton)*.

H Case Study

To have an intuitive insight into the strengths of SORTIE , we show the predicted programme and the topological order of several cases from LogicNLG in Table 8, Table 9 and Table 10. We can see that our model is able to compositionally assemble simple operators into a complicated programme sequence. When executed, the programme emits propitiate and faithful entities to fill in the placeholders, which might account for the impressive fidelity.

Nation	Gold	Silver	Bronze
Switzerland	5	5	15
...	...	...	...
Netherlands	3	2	2
West Germany	2	4	2
United States	2	1	3
Italy	2	1	2
Canada	0	2	3

Template: [ENT1] received [ENT2] more gold medal than [ENT3] did.
Topological Order: [START] $\rightarrow$ [ENT3] $\rightarrow$ [ENT1] $\rightarrow$ [ENT2] $\rightarrow$ [END]
Programme:
[ENT3]: `SELECT` (<Nation>, `ARGMIN` (<Gold>)) = Canada;
[ENT1]: `SELECT` (<Nation>, `ARGWHERE` (<Nation>, `NEQ` ([ENT3]))) = Italy;
[ENT2]: `DIFF` (`SELECT` (<Gold>, `ARGWHERE` (Nation, `EQ` ([ENT1]))), `SELECT` (<Gold>, `ARGWHERE` (Nation, `EQ` ([ENT3])))) = 2.

Table 9: A case form LogicNLG with caption *1988 winter Olympics*.

Song	Language	point
In the blue painted blue	Italian	13
The Whole World	Dutch	1
Sleep, My Love	French	27
A Great Love	French	1
Little Start	Swedish	10
I Tore A Page Out of My Diary	Danish	3
Music For Two Pennies	German	5

Template:The Eurovision Song Contest of 1958 consisted of [ENT1] different languages.
Topological Order:
[START] $\rightarrow$ [ENT1] $\rightarrow$ [END].
Programme:
[ENT1]: `COUNT` (`UNIQUE` (<Language>)) = 6.

Table 10: A case form LogicNLG with caption *Eurovision Song Contest 1958*.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
In the "Limitations" section (the last section of the main text)

☑ A2. Did you discuss any potential risks of your work?
In the "Ethical Considerations" section (the second last section of the main text). It is notable that this paper does not pose any ethical problems.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
In the Abstract and Section 1.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

In Section 4.1 and Appendix D.

☑ B1. Did you cite the creators of artifacts you used?
In Section 4.1 and Appendix D.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Not applicable. Left blank.

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
In Section 4.1 and Appendix D.

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
These steps have been conducted by the publishers of the datasets we used. We strictly follow the data preprocessing steps in the original papers or released codes.

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
In Section 4.1 and Appendix D.

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
In Appendix D.

C ☑ Did you run computational experiments?

In Section 4 and Section 5.

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
In Appendix F.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
In Appendix F.

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
In Appendix F.

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
In Appendix B.

D ☑ Did you use human annotators (e.g., crowdworkers) or research with human participants?

In Section 4.2 and Appendix E.2.

☑ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
In Section 4.2 and Appendix E.2.

☑ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
In Appendix E.2.

☑ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
In Appendix E.2.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Not applicable. Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Not applicable. Left blank.