MURMUR: Modular Multi-Step Reasoning for Semi-Structured Data-to-Text Generation

Swarnadeep Saha\textsuperscript{1} Xinyan Velocity Yu\textsuperscript{2} Mohit Bansal\textsuperscript{1} Ramakanth Pasunuru\textsuperscript{2} Asli Celikyilmaz\textsuperscript{2}
\textsuperscript{1}UNC Chapel Hill \textsuperscript{2}Meta AI
\{swarna, mbansal\}@cs.unc.edu
\{velocityyu, rpasunuru, aslic\}@meta.com

Abstract

Prompting large language models has enabled significant recent progress in multi-step reasoning over text. However, when applied to text generation from semi-structured data (e.g., graphs or tables), these methods typically suffer from low semantic coverage, hallucination, and logical inconsistency. We propose MURMUR, a neuro-symbolic modular approach to text generation from semi-structured data with multi-step reasoning. MURMUR is a best-first search method that generates reasoning paths using: (1) neural and symbolic modules with specific linguistic and logical skills, (2) a grammar whose production rules define valid compositions of modules, and (3) value functions that assess the quality of each reasoning step. We conduct experiments on two diverse data-to-text generation tasks like WebNLG and LogicNLG. The tasks differ in their data representations (graphs and tables) and span multiple linguistic and logical skills. MURMUR obtains significant improvements over recent few-shot baselines like direct prompting and chain-of-thought prompting, while also achieving comparable performance to fine-tuned GPT-2 on out-of-domain data. Moreover, human evaluation shows that MURMUR generates highly faithful and correct reasoning paths that lead to 26\% more logically consistent summaries on LogicNLG, compared to direct prompting.\textsuperscript{1}

1 Introduction

Data-to-text generation (McKeown, 1992; Reiter and Dale, 1997; Wen et al., 2015; Dušek and Jurcicek, 2015; Mei et al., 2016; Novikova et al., 2017; Gatt and Krahmer, 2018) is the task of generating fluent, faithful, and consistent summaries of semi-structured data. Recent works have introduced different data-to-text generation tasks wherein the data is represented in diverse structures, like meaning representations (Novikova et al., 2017), graphs (Gardent et al., 2017), or tables (Lebret et al., 2016; Parikh et al., 2020; Chen et al., 2020a). Text generation from such data is challenging because it extends surface realization of the input content and requires various reasoning and compositionality skills, such as filtering a table based on a certain criterion, retrieving the maximum value from a table column, etc.

Existing works fine-tune pre-trained language models (Radford et al., 2019; Raffel et al., 2020) as the de-facto standard for building supervised

\textsuperscript{1}Supporting code available at https://github.com/swarnaHub/MURMUR

Table Topic: Reinhold Roth

<table>
<thead>
<tr>
<th>Year</th>
<th>Class</th>
<th>Team</th>
<th>Points</th>
<th>Wins</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>250cc</td>
<td>yamaha</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>1982</td>
<td>250cc</td>
<td>yamaha</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1983</td>
<td>250cc</td>
<td>yamaha</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>1984</td>
<td>500cc</td>
<td>honda</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>1985</td>
<td>250cc</td>
<td>romer-juchem</td>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>1986</td>
<td>250cc</td>
<td>hh - honda</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>1987</td>
<td>250cc</td>
<td>hh - honda</td>
<td>108</td>
<td>1</td>
</tr>
<tr>
<td>1988</td>
<td>250cc</td>
<td>hh - honda</td>
<td>158</td>
<td>0</td>
</tr>
<tr>
<td>1989</td>
<td>250cc</td>
<td>hh - honda</td>
<td>190</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 1: Sample table from LogicNLG and two logical summaries generated by MURMUR and Direct Prompting baseline. Direct Prompting summaries include logical inconsistencies and hallucinations (marked in red) while MURMUR generates reasoning paths (composed of modules) and converts them to logically consistent summaries (marked in green). Each color code highlights part of the table relevant to a MURMUR summary.
data-to-text generation systems (Kale and Rastogi, 2020; Agarwal et al., 2021). However, this requires a large amount of domain-specific parallel data, which is expensive to obtain, and training models on such data also affects out-of-domain generalization (Laha et al., 2020; Dušek et al., 2020).

Motivated by the recent success of few-shot prompting in multi-step reasoning over text (Wei et al., 2022; Nye et al., 2021; Wang et al., 2022a; Dohan et al., 2022), we pose data-to-text generation as multi-step reasoning over data.\(^2\) Reasoning over data for text generation brings its own set of challenges: (1) **Generation Quality**: Firstly, directly prompting large language models (LLMs) can cause models to suffer from low semantic coverage, hallucinations, and logically inconsistent generations (see red marked phrases for the Direct Prompting summaries in Fig. 1). Other prompting methods like Chain-of-Thought (CoT) encourage LLMs to also generate intermediate reasoning steps (Wei et al., 2022) but it compromises the *transparency, faithfulness,\(^3\) and correctness* of the reasoning process due to the lack of explicit conditioning between the reasoning steps (Creswell and Shanahan, 2022). (2) **Transformation-invariance**: Text is a sequence of tokens while data is typically represented as a *set* of elements (e.g., a graph is a set of edges, a table is a set of rows, etc.). Hence, a model that reasons over data must be transformation-invariant (Wang et al., 2022a). For instance, the summary generated from a table should be invariant to randomly shuffling the rows of the table. Thus, prompting methods that linearize the data in an arbitrary order, can be prone to some variance (see Table 3 and 6).

We propose MURMUR, a few-shot Modular Multi-step Reasoning approach to text generation from data (§3). It is a best-first search algorithm (§3.4) that generates reasoning paths (see examples in Fig 1) with three features: (1) **Modularity** (§3.1): MURMUR defines a set of few-shot neural and symbolic modules with diverse input/output data types that constitute multiple steps in a reasoning path. Neural modules perform linguistic skills that LLMs are good at (e.g., the *Surface Realization* module in Fig. 1 converts a reasoning path to a natural language summary) and symbolic modules perform logical skills that they mostly struggle with (Wang et al., 2022b; Gao et al., 2022) (e.g., the *argmin* module in Fig. 1 finds the row with the minimum points); (2) **Grammar** (§3.2): MURMUR introduces a grammar whose production rules specify valid compositions of modules. For instance, in the second path of Fig. 1, MURMUR first generates the module *filter_eq* followed by the *avg* module, because the former outputs a *table* data type which is also the input data type to the latter; (3) **Value functions** (§3.3): To evaluate the quality of each plausible reasoning step and choose the best modules at each step, MURMUR defines value functions that score, rank, and select the best steps. For example, in the second path of Fig. 1, an *avg* module is perhaps more salient than a *max* or *min* module (which only finds the maximum or minimum points).

Our findings are: MURMUR can perform multi-step generative reasoning on simple to complex semi-structured data-to-text generation tasks including WebNLG (Gardent et al., 2017), a graph-to-text task (§5) and LogicNLG (Chen et al., 2020a), a table-to-text task (§6). We compare MURMUR with state-of-the-art supervised (end-to-end and pipeline) and few-shot prompting methods. On WebNLG, MURMUR obtains significant improvements in semantic coverage and hallucinations of generated summaries over other few-shot baselines like direct prompting and CoT prompting. Additionally, MURMUR demonstrates good out-of-domain generalizability by obtaining comparable performance to fine-tuned LMs like GPT-2. On LogicNLG, human evaluation demonstrates that MURMUR significantly improves the logical consistency of summaries over direct prompting (by up to 26%), showcasing the strength of a neuro-symbolic approach for data-to-text generation.

### 2 Definitions: Reasoning Step and Path

A *Reasoning Step* is a triple \((M, X, y)\) where a module \(M\) performs a certain skill by conditioning on an input \(X\) to generate an output \(y\). For example, in Fig. 2, the module *argmin* takes a table and a column (*points*) as input and outputs the row with the minimum points. A *Reasoning Path* is defined as a sequence of such reasoning steps \(\{(M_i, X_i, y_i)\}_{i=1}^\nu\). Fig. 2 shows an example of a reasoning path, represented as a nested structure. It consists of three reasoning steps for three modules (*argmin, hop, and eq*). The *argmin* module outputs

\(^2\)By data, we mean semi-structured data such as graphs or tables. Henceforth, we will refer to it as just ‘data’.

\(^3\)Faithful reasoning refers to an underlying causal structure in the reasoning process. This is different from a text’s faithfulness to an input context which will be called hallucinations.
<table>
<thead>
<tr>
<th>Year</th>
<th>Class</th>
<th>Team</th>
<th>Points</th>
<th>Wins</th>
</tr>
</thead>
<tbody>
<tr>
<td>1979</td>
<td>350cc</td>
<td>yamaha</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>1980</td>
<td>250cc</td>
<td>yamaha</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1982</td>
<td>250cc</td>
<td>yamaha</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1982</td>
<td>500cc</td>
<td>suzuki</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table Topic: Reinhold Roth

**Figure 2:** Illustration of MURMUR generating a reasoning path and then converting it into a logically consistent summary, supported by the input table. The reasoning path consists of three reasoning steps. At each step, MURMUR chooses a set of plausible modules (according to a grammar) and then selects the best module, with the best input according to a value function. The output generated at each step serves as the input to the next step.

3 MURMUR Approach

MURMUR consists of four components: (1) a set of modules, (2) a grammar, (3) value function(s), and (4) a search algorithm that brings all the previous three components together. The search algorithm constructs reasoning paths by first identifying plausible modules at each reasoning step according to the grammar and then determining the best modules (and their corresponding inputs) with the help of value functions. Fig. 2 shows a working example of MURMUR, in which given an input table, it searches for a reasoning path (of three steps), and finally converts it into a summary. The specifics of MURMUR’s components vary based on the task at hand. As case studies, we consider two data-to-text generation tasks: WebNLG (Gardent et al., 2017), a graph-to-text generation task and LogicNLG (Chen et al., 2020a), a complex table-to-text generation task where the goal is to generate logical summaries from salient parts of the table.

3.1 MURMUR Modules

MURMUR defines a set of modules \( \{ M_i \}_{i=1}^{m} \) that perform specialized reasoning skills for the corresponding task. Formally, each module \( M_i \) is defined as a multi-variate function \( M_i : X \to y \) that maps an \( n \)-tuple input \( X = (x_1, \ldots, x_n) \) to an output \( y \). Each input variable \( x_i \) and output \( y \) can have their own expected data types \( d_i \) and \( d_y \) respectively. These data types could be user-defined\(^5\) like Table, Triple, etc or standard ones like String, Number, Bool, etc. For example, in Fig. 2, the module \( M_{\text{argmin}} : (t, c) \to r \) takes a table \( t \) (with data type Table) and a column \( c \) (with data type String) as input and outputs a row \( r \) (with data type row) with the minimum value in column \( c \). The modules are implemented as few-shot neural models or symbolic functions. We choose few-shot neural modules for linguistic skills that LLMs typically excel at and symbolic modules for logical operations that LLMs mostly struggle with (Wang et al., 2022b; Gao et al., 2022). Reasoning over semi-structured data allows us to implement symbolic modules with PYTHON functions. Below we provide examples of neural and symbolic modules for the two tasks.

**Neural Linguistic Modules.** In any modular data-to-text generation approach, one of the modules is responsible for the transition from structured data to unstructured text. We call it Surface Realization. In particular, for WebNLG, we define it as \( M_{\text{sr}} : t \to s \) that converts a triple \( t \) (with data type Triple) into a short sentence \( s \) (with data type String). For LogicNLG, we define it as \( M_{\text{sr}} : (t, p) \to s \) that takes a table \( t \) (with data type Table) and a reasoning path \( p \) as input and converts it into a summary \( s \) (with data type String). As we show later, in WebNLG, Surface Realizations are the first reasoning steps, while in LogicNLG, it

---

\(^5\) The modules are analogous to function definitions with expected IO types. Similarly, user-defined data types can be thought of as class definitions. For instance, a data type `Triple` can be implemented as a class consisting of a subject, a relation, and an object (all with data type `String`).
Module Description

Filtering $M_{\text{filter}} : (t, cr) \rightarrow t'$: takes a table $t$ and a filtering criterion $cr$ as input and outputs a table $t'$ with rows where the criterion $cr$ is satisfied.

Aggregation Performs aggregation operations on a table. For example, $M_{\text{max}} : (t, c) \rightarrow n$ is a max module that takes a table $t$ and a column $c$ as input and outputs the maximum number $n$ in column $c$.

Boolean $M_{\text{bool}} : (t, cr) \rightarrow b$: takes a table $t$ and a criterion $cr$ as input and outputs a boolean $b$ based on whether the criterion is satisfied.

Hop $M_{\text{hop}} : (r, c) \rightarrow e$: takes a row $r$ and a column $c$ as input; outputs the element $e$ in $(r, c)$ cell.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filtering</td>
<td>$M_{\text{filter}} : (t, cr) \rightarrow t'$: takes a table $t$ and a filtering criterion $cr$ as input and outputs a table $t'$ with rows where the criterion $cr$ is satisfied.</td>
</tr>
<tr>
<td>Aggregation</td>
<td>Performs aggregation operations on a table. For example, $M_{\text{max}} : (t, c) \rightarrow n$ is a max module that takes a table $t$ and a column $c$ as input and outputs the maximum number $n$ in column $c$.</td>
</tr>
<tr>
<td>Boolean</td>
<td>$M_{\text{bool}} : (t, cr) \rightarrow b$: takes a table $t$ and a criterion $cr$ as input and outputs a boolean $b$ based on whether the criterion is satisfied.</td>
</tr>
<tr>
<td>Hop</td>
<td>$M_{\text{hop}} : (r, c) \rightarrow e$: takes a row $r$ and a column $c$ as input; outputs the element $e$ in $(r, c)$ cell.</td>
</tr>
</tbody>
</table>

Table 1: MURMUR Symbolic Modules to perform logical operations over tables in Table-to-Text generation.

is the last step. For WebNLG, we also define a Text Fusion module $M_{\text{tf}} : (s_1, s_2) \rightarrow s$ that combines two strings $s_1$ and $s_2$ into a coherent text $s$. Text Fusion iteratively combines intermediate generations at each step, enabling more controllability in generation (Tan et al., 2021).

Symbolic Logical Modules. For LogicNLG, drawing motivation from prior work (Chen et al., 2020c), we define different categories of symbolic modules that perform logical operations over tables (see Table 1 and refer to Table 8 for the detailed list). WebNLG requires summarizing an input graph and hence, does not involve any logical modules.

3.2 Grammar over Modules

The role of the grammar is to determine a set of plausible modules in a reasoning step and how they should be composed. The production rules of the grammar capture possible transitions from an input data type to an output data type(s) (see Fig. 2 and Table 2). Each production rule thus defines multiple permissible modules. For example, the production rule ‘Table → Number’ (meaning that a number can be generated from a table) is valid for both max and min modules. When MURMUR searches for reasoning paths, the grammar reduces the search space (over all possible modules) by only selecting the ones that can be composed at each reasoning step. We provide examples below of how such grammars are constructed.

Grammar for Graph-to-Text Generation. Table 2 shows the grammar for Graph-to-Text generation. It consists of two production rules, one for Surface Realization and another for Text Fusion. Past pipeline approaches for graph-to-text generation (Xiang et al., 2022) also perform surface realization followed by fusion, as explained through the grammar.

Grammar for Table-to-Text Generation. Generating logical summaries from a table is a more challenging task. Based on the types of modules introduced previously, we define a grammar, as shown in Table 2. As an instance, the first rule encodes the knowledge that given an input of type Table, one can output a Table, a Row of the table, a Number, or a Boolean.

3.3 Value Functions

While the grammar helps reduce the search space by defining permissible compositions of modules, each reasoning step can still have multiple plausible modules and each module can also have multiple plausible inputs to choose from. Thus, MURMUR introduces value functions (see Fig. 2) that assess the quality of each plausible reasoning step by scoring, ranking, and selecting the best step(s).

Value Function for Graph-to-Text Generation. In a Graph-to-Text generation task, each intermediate reasoning step $r$ generates a summary $y_r$ for a subset of edges (triples) $G_r$ from the input graph (see Fig. 7 for an illustration). The value functions evaluate the following two aspects of the generated summary $y_r$. First, Fluency is measured by log-likelihood of the generated text similar to BARTScore (Yuan et al., 2021):

$$S_f(y_r) = \exp \left\{ \frac{1}{T} \sum_{t=1}^{T} \log p_{\theta}(y_r^t | y_r^{<t}) \right\}$$

Second, Semantic Consistency measures the average logical entailment probability $P_e(\cdot)$ between the generation $y_r$ and triples $G_r$ and vice-versa:

$$S_{sc}(G_r, y_r) = 0.5 \times (P_e(G_r, y_r) + P_e(y_r, G_r))$$

We use an NLI model to compute entailment probabilities. The both-way entailment scores capture

---

We concatenate the surface realizations of the triples to construct the sequence for the NLI model.
equivalence between the triples and the generation, ensuring that the latter not only covers all the triples but also does not hallucinate any new information. Overall score is an ensemble of the two scores, given by $\alpha S_f(y_r) + (1 - \alpha) S_{sc}(G_r, y_r)$.

**Value Function for Table-to-Text Generation.** Our value function chooses the best module(s) at each reasoning step, as well as the best input(s) for the corresponding module(s).\(^7\) For instance, if a reasoning step generates a number from a table (according to the grammar), the value function should determine the best module(s) between $\text{max}$, $\text{min}$, etc., as well as which column the $\text{max}$ or $\text{min}$ module should be computed on. Taking inspiration from past work on verifying intermediate reasoning traces over text (Creswell and Shanahan, 2022; Yang et al., 2022), we train a value function $S : (T, P_r) \rightarrow p$ that judges the correctness of a partial reasoning path $P_r$ for an input table $T$. In particular, we train a binary classifier on samples with correct and incorrect partial reasoning paths. We call this value function a saliency metric because it selects the best reasoning steps that reason over salient parts of the table. We discuss the model and training data for our saliency metric in §4.2.

### 3.4 Search Algorithm

We now describe how all the three components discussed above come together in generating reasoning paths for MURMuR (see Fig. 2). We propose a best-first search algorithm that operates as follows. It takes as input a set of $m$ modules $\{M_i\}_{i=1}^m$, a grammar $G$, a value function $V$, and number of reasoning paths or summaries to generate $p$. Additionally, it considers a hyperparameter, the beam size $b$ of the search ($b \geq p$). The search begins by initializing an empty priority queue that maintains the beam (best $b$ partial reasoning paths to be explored anytime during the search). Next, at each step, MURMuR (1) pops an element from the queue, (2) identifies the data type of the element (e.g., Table), (3) looks up the grammar to find all possible transitions from that data type (e.g., Row, Number), (4) selects all modules for each such transition (e.g., argmax and argmin for ‘Table → Row’, $\max$ and $\min$ for ‘Table → Number’), and (5) constructs all plausible reasoning steps consisting of modules and their corresponding inputs (e.g., all numerical columns for argmax). It then scores all these reasoning steps using the value function, ranks them, and only keeps the top-$b$ paths in the queue. For WebNLG, the search terminates when all triples have been iterated. For LogicNLG, a reasoning path is complete when the current module outputs a boolean variable that evaluates to true (e.g., the eq module). Upon termination of the search, we return the top-$p$ paths and the corresponding summaries.

### 4 Experimental Setup

#### 4.1 Graph-to-Text Generation

We report results on both seen and unseen splits of the test set of WebNLG (Gardent et al., 2017).\(^8\)

**Modules.** We implement both modules, *Surface Realization* and *Text Fusion* as few-shot neural models by prompting OPT-175B (Zhang et al., 2022) with skill-specific prompts (see Appendix D) and greedy decoding.

**Value Function.** As defined in §3.3, we compute fluency using the log probabilities estimated by OPT-175B. The entailment probability for the semantic scorer is based on a DeBERTa-base model (He et al., 2020) trained on a collection of eight NLI datasets.\(^9\) The mixing ratio $\alpha$ is set to 0.05. At each reasoning step, MURMuR scores and ranks the intermediate generations in the queue using the value function. Subsequently, it only explores the highest scoring intermediate generation in the next step of the search and prunes the rest.\(^10\)

#### 4.2 Table-to-Text Generation

**Modules.** We implement all logical modules, as described in §3.1, with PYTHON functions. We again prompt OPT-175B for the *Surface Realization* module (see Appendix D for the prompt).

**Value Function.** Our saliency metric is a binary classifier. Specifically, we train a BERT-base model that takes a table (as a sequence of rows) and a partial reasoning path as input and classifies it as correct or incorrect. During inference, we consider the correct class probability as the saliency score. We obtain training data from the Logic2Text dataset (Chen et al., 2020c) that annotates open-domain tables with gold reasoning paths. Given a

---

\(^7\)For Graph-to-Text, we only need to choose the best inputs because at each step there is only one plausible module.

\(^8\)The 2017 version of the dataset is available at [https://webnlg-challenge.loria.fr/challenge_2017/](https://webnlg-challenge.loria.fr/challenge_2017/).

\(^9\)https://huggingface.co/MoritzLaurer/DeBERTa-v3-base-mnli-fever-docnli-ling-2c.

\(^10\)This is equivalent to performing greedy search. We experimented with larger beams for a more exhaustive search without observing any noticeable improvement in performance.
Table 3: Comparison of supervised and few-shot approaches on the WebNLG Seen and Unseen splits of the test set. † = Supervised with 7k in-domain samples. ‡ = Supervised with a synthetic corpus of 934k samples. ⋆ = Few-shot with k demonstrations. We report mean and variance for all few-shot methods with three random triple orderings.

5 Experiments on Graph-to-Text

5.1 Comparison of MURmUR with supervised and few-shot methods

Baselines. We compare with both supervised and few-shot baselines.

• Supervised. We compare with MELBOURNE, a non-pretrained encoder-decoder model (Gardent et al., 2017) and two fine-tuned LMs, GPT-2-large (Radford et al., 2019) and T5-large (Raffel et al., 2020). We also compare with a SOTA modular pipeline approach, Neural Pipeline (Kasner and Dušek, 2022) that first converts triples to sentences using hand-designed templates and subsequently orders and fuses the sentences by fine-tuning on a large synthetic corpus of 934k samples.

• Few-shot. For direct comparisons, we consider two few-shot baselines, Direct Prompting (DP) that directly prompts the OPT-175B model to generate a summary of the graph, and Chain-of-Thought Prompting (CoT) (Wei et al., 2022) that prompts the model to generate the summary step-by-step (see Appendix D for prompts). We choose the demonstrations randomly from the training data and keep them consistent across all few-shot methods.

Metrics. Following prior work, we perform automatic evaluation using BLEU (Papineni et al., 2002) and METEOR (Banerjee and Lavie, 2005).

Results. For all few-shot methods, we report mean and variance of three random triple orderings. Table 3 shows the results. MURmUR significantly outperforms DP and CoT by up to 8 points in BLEU and METEOR ($p < 0.001$), when using a single demonstration ($k=1$). MURmUR even outperforms DP with five demonstrations ($k=5$). Prompting an LLM by simply concatenating the intermediate steps for CoT does not work well for text generation. MURmUR also outperforms a supervised baseline MELBOURNE and obtains comparable performance to fine-tuned models like GPT-2 on the unseen test split. Through its modular treatment, MURmUR generates outputs with more coverage of triples and lesser hallucinations, as reflected in the improved scores and further demonstrated in §5.2 through human evaluation. Finally, MURmUR is transformation-invariant because it treats the graph as a set (not sequence) of triples. Refer to Appendix B for experiments studying the number and variation in demonstrations.
DP | MURmUR | % Improve
---|---|---
Omissions↓ | 1.64±0.06 | 0.73±0.01 | +24%
Hallucinations↓ | 0.77±0.03 | 0.43±0.03 | +9%
Disfluencies↓ | 0.14±0.05 | 0.30±0.04 | -4%

Table 4: Average count of omissions, hallucinations, and disfluencies in WebNLG summaries.

5.2 Human Evaluation of Final Summaries and Intermediate Reasoning Steps

Evaluation of Final Summaries. We compare the summaries generated by DP (our best baseline) and MURmUR. Two NLP experts take part in the study with 50 randomly chosen test samples (having an average of 3.8 triples). They count the number of omissions, hallucinations, and disfluencies in the generated outputs. Our results in Table 4 demonstrate that MURmUR benefits significantly from a step-wise generative process and reduces omissions by 24% and hallucinations by 9%. We do observe a slight drop in fluency in MURmUR’s generations because of its iterative fusion process.

Evaluation of Intermediate Reasoning Steps. We also evaluate the quality of the individual reasoning steps of MURmUR. For every reasoning step of a data point, we provide the annotators with the (1) generation, and (2) the previous steps that the current step is conditioned on. We conduct this study on six randomly chosen test examples, spanning 50 reasoning steps (28 Surface Realization and 22 Text Fusion). Annotators judge the generations for their grammaticality, module faithfulness (i.e., if the module is doing what it is supposed to do), and correctness (e.g., whether the fusion is correct). From Table 5, we conclude that both modules are almost always grammatical, and highly faithful and 64% of fusion operations are also fully correct.

6 Experiments on Table-to-Text

6.1 Comparison of MURmUR with supervised and few-shot methods

Baselines. We compare with several non-pretrained and pretrained supervised methods as well as few-shot methods.
- Non-pretrained Supervised. We compare MURmUR with a non-pretrained transformer model, Field-Infusing + Trans (Chen et al., 2020a).

<table>
<thead>
<tr>
<th>Module</th>
<th>Grammatical</th>
<th>Faithful</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surface Realization</td>
<td>1.00</td>
<td>1.00</td>
<td>0.82</td>
</tr>
<tr>
<td>Text Fusion</td>
<td>0.90</td>
<td>0.72</td>
<td>0.64</td>
</tr>
</tbody>
</table>

Table 5: Fraction of grammatical, module faithful, and correct intermediate reasoning steps generated by the two modules in MURmUR for WebNLG.

| Field-Infusing† | 43.7 / 20.9 / 8.4 |
| BERT-TabGen† | 49.1 / 27.7 / 13.5 |
| GPT-TabGen† | 49.6 / 28.2 / 14.2 |
| GPT-Coarse-to-Fine† | 49.0 / 28.3 / 14.6 |
| DCVED† | 49.5 / 28.6 / 15.3 |

Table 6: Comparison of supervised non-pretrained, pretrained, and few-shot approaches on the LogicNLG test set. † = Supervised with 37k in-domain samples. * = Few-shot with 1 demonstration. ‡ = Few-shot with 1 demonstration and 221 gold (table, path) pairs. We report mean and variance for all few-shot methods with three random orderings of the input table rows.

- Pretrained Supervised. Next, we compare with three pre-trained LMs based on BERT and GPT-2, BERT-TabGen, GPT-TabGen, GPT-Coarse-to-Fine (Chen et al., 2020a) and a deconfounded variational encoder-decoder model, DCVED (Chen et al., 2021).
- Few-shot. We also compare with Direct Prompting (DP) and CoT Prompting. Additionally, we evaluate the effect of our search algorithm and saliency metric. First, in BART + SR, instead of searching for reasoning paths, we fine-tune a BART model that generates reasoning paths in one go. As training data, we leverage the (table, gold reasoning path) pairs that are used for training the saliency metric. The surface realization (SR) step is left unchanged. Second, we remove the saliency metric by randomly selecting a module at each step (but according to the grammar). All few-shot methods use one random demonstration.

Metrics. Following Chen et al. (2020a), we compare all methods with BLEU scores. They also propose metrics to evaluate logical consistency but we found such learned metrics do not correlate well with humans. Instead, we conduct more reliable human evaluations of logical correctness in §6.2.
Table 7: Human evaluation of logical correctness for LogicNLG. ‘Is Logical’ denotes the percentage of correct generations that also involve some underlying logical computations.

<table>
<thead>
<tr>
<th></th>
<th>Correct</th>
<th>Partial</th>
<th>Incorrect</th>
<th>Ungrammatical</th>
<th>Is Logical?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct Prompting</td>
<td>28.7±3.7</td>
<td>20.0±2.5</td>
<td>38.8±8.7</td>
<td>12.5±2.5</td>
<td>62.0±0.5</td>
</tr>
<tr>
<td>MURMUR</td>
<td>55.0±2.5</td>
<td>1.2±1.2</td>
<td>38.8±3.7</td>
<td>5.0±2.5</td>
<td>95.4±0.2</td>
</tr>
</tbody>
</table>

Results. Table 6 shows the results on the test set of LogicNLG. MURMUR significantly improves upon DP and CoT prompting by up to 2.4 points in BLEU-3 \((p < 0.001)\). We attribute this to two factors: (1) leveraging symbolic modules for logical skills that ensure their correctness, (2) delegating the task of converting a path to natural language to an LLM. Both CoT and BART+SR, while generating intermediate reasoning paths, do not use executable modules and hence cannot guarantee valid compositionality or logical correctness of the reasoning steps. MURMUR also improves upon the supervised Field-Infusing model. Finally, MURMUR obtains some improvement with the saliency metric, indicating that it helps in choosing more salient paths. Refer to Appendix C for studies on the number and variation in demonstrations.

6.2 Human Evaluation of Logical Correctness

Next, we conduct human evaluation to compare the logical correctness of the generations from DP and MURMUR. Two NLP experts annotate 40 randomly chosen generations from eight different tables. In particular, they take part in two studies. First, they classify each generation into whether it is (a) ungrammatical, (b) grammatical but incorrect, (c) grammatical but partially correct, or (d) grammatical and also fully correct. Next, for each fully correct generation, they annotate whether it involves any underlying logical operation (like counting, summation, etc) or are mere surface realizations of the table content. We observe from Table 7 that MURMUR not only generates 26% more correct outputs, but about 95% of those generations also involve some logical operations. In summary, MURMUR is most beneficial in two scenarios: (1) generations that require many steps of reasoning, (2) generations that require logical reasoning. The first capability comes from the fact that MURMUR is specifically designed to compose multiple steps of reasoning through its grammar and value functions. The second benefit is because of the presence of symbolic modules that ensure logical correctness. These two capabilities are specifically required in long complex tables involving numerical columns where there is a need to summarize content (e.g., by filtering, averaging a numerical column, etc). Generating reasoning paths through logical modules ensures that almost all generations are logical derivations from the table, an ability that is significantly harder to achieve through direct prompting. See Fig. 8 in the appendix for an illustrative example of the generations of MURMUR for long complex tables.

7 Related Work

Multi-step Reasoning over Text. Recent developments in LLMs (Brown et al., 2020; Zhang et al., 2022; Thoppilan et al., 2022; Chowdhery et al., 2022) have enabled significant progress in few-shot methods for logical reasoning tasks (Wei et al., 2022; Creswell et al., 2022; Nye et al., 2021; Wang et al., 2022; Zhou et al., 2022; Dasgupta et al., 2022; Kojima et al., 2022; Dohan et al., 2022). Representative methods like CoT prompting output intermediate reasoning steps before generating the final answer. However, the reasoning steps are all generated in one go from a single model, potentially leading to unfaithful reasoning due to the lack of explicit conditioning between the steps (Creswell and Shanahan, 2022). MURMUR overcomes this issue by developing granular modules that are capable of performing specialized skills by explicitly conditioning on the outputs from previous reasoning steps. Conceptually, MURMUR bears similarity with the Selection-Inference modular architecture (Creswell et al., 2022; Creswell and Shanahan, 2022). However, their focus is on QA and reasoning over textual context (Saha et al., 2020, 2021b; Tafjord et al., 2021; Dalvi et al., 2021; Bostrom et al., 2022). A few concurrent works have also proposed neuro-symbolic approaches for reasoning over text (Gao et al., 2022; Wang et al., 2022b; Chen et al., 2022; Cheng et al., 2022). Different from these, we tackle a more challenging setup of multi-step reasoning for controlled generation from semi-structured data.

Modular Reasoning over Text. Neural Module Networks learn and execute compositional programs over modules (Andreas et al., 2016; Jiang and Bansal, 2019; Gupta et al., 2020; Subramanian...
et al., 2020; Saha et al., 2021a). While their modules typically output attention maps, prior works have also used text-in text-out modules whose input/output data types are strings (Khot et al., 2021, 2022; Saha et al., 2022). MUR’s modules are a generalization of text-in text-out modules since they can capture operations involving complex data types (like tables and strings), among others. The data to text transition is also clearly represented through the compositions of our modules, unlike attention maps-based modules whose interpretability has often been debated (Serrano and Smith, 2019).

Data-to-Text Generation. Existing methods for data-to-text generation include (1) supervised methods that finetune seq2seq LMs (Kale and Rastogi, 2020; Chen et al., 2020b; Ribeiro et al., 2021; Ke et al., 2021; Xiang et al., 2022), (2) pipeline modular methods (Reiter and Dale, 1997; Reiter, 2007; Laha et al., 2020; Kasner and Dušek, 2022), and (3) few-shot methods that assume access to a large corpus of unlabeled examples for data augmentation or retrieving similar examples (Puduppully et al., 2019; Zhao et al., 2020; Trisedya et al., 2020; Su et al., 2021). Unlike prior modular methods, MUR uses few-shot neural or symbolic modules. Unlike past few-shot methods, MUR works well with as few as one demonstration, without requiring access to any unlabeled corpus.

8 Conclusion

We presented MUR, a neuro-symbolic modular reasoning approach for data-to-text generation. MUR shows the benefits of building interpretable modular text generation systems by breaking a task down into sub-problems and then solving them through separate modules, without requiring module-specific supervision. It utilizes the power of LLMs in solving linguistic sub-tasks through in-context learning while delegating the logical sub-tasks to symbolic modules. MUR generalizes the concept of modules by treating them as functions and defining their behaviors through expected input/output data types and compositions with a grammar (analogous to function compositions).

Limitations

MUR relies on large language models for few-shot linguistic skills like surface realization and text fusion. It is probable that smaller models do not work as well, in which case one may curate additional training data to train these modules. We also note that our choice of logical modules is motivated by the characteristics of the task. Hence, it is conceivable that other data-to-text generation tasks might benefit from incorporating additional modules. MUR does not make any assumptions about the type or implementation of the modules and it should be straightforward to extend our method to other data-to-text generation tasks.

We limit our experiments to English datasets. We also adopt a simple prompting strategy for converting a reasoning path to a natural language summary by representing the path as a string. This works well in practice and OPT is typically able to resolve the module names and their arguments correctly. However, more future work is needed to understand when this fails so that better prompting methods can be developed. Despite the known limitations of standard automatic metrics like BLEU and METEOR, we use them to compare our method to previous works. While this is not ideal, we have performed comprehensive human evaluation for both tasks to further verify our claims.

Ethics Statement

Large Language Models can be prone to generating toxic and unwanted content (Weidinger et al., 2021). Since MUR uses focused modules to accomplish specific skills, we believe that this might help limit inadvertent negative impacts. Furthermore, the presence of specific modules should provide users with more trust and control in real-world scenarios, allowing one to verify, debug, and improve the capabilities of these modules.

References


Kaiyu Yang, Jia Deng, and Danqi Chen. 2022. Generating natural language proofs with verifier-guided search. In EMNLP.


<table>
<thead>
<tr>
<th>Module Name</th>
<th>Input Data Type</th>
<th>Output Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>filter_eq</code></td>
<td>table, string, string</td>
<td>table</td>
<td>Returns a table with the rows where entry in the input column (second argument) is equal or not equal to the input value (third argument).</td>
</tr>
<tr>
<td><code>filter_not_eq</code></td>
<td>table, string, string</td>
<td>table</td>
<td>Returns a table with the rows where entry in the input column (second argument) is equal or not equal to the input value (third argument).</td>
</tr>
<tr>
<td><code>filter_greater</code></td>
<td>table, string, number</td>
<td>table</td>
<td>Returns a table with the rows where a numerical column (second argument) is greater than or less than (or equal to) the input number (third argument).</td>
</tr>
<tr>
<td><code>filter_lesser</code></td>
<td>table, string, number</td>
<td>table</td>
<td>Returns a table with the rows where a numerical column (second argument) is greater than or less than (or equal to) the input number (third argument).</td>
</tr>
<tr>
<td><code>filter_all</code></td>
<td>table, string</td>
<td>table</td>
<td>Returns the whole table.</td>
</tr>
<tr>
<td><code>arg_max</code></td>
<td>table, string</td>
<td>row</td>
<td>Returns the row with the minimum or maximum value for the input column (second argument).</td>
</tr>
<tr>
<td><code>arg_min</code></td>
<td>table, string</td>
<td>row</td>
<td>Returns the row with the minimum or maximum value for the input column (second argument).</td>
</tr>
<tr>
<td><code>max</code></td>
<td>table, string</td>
<td>number</td>
<td>Returns the maximum, minimum, average or sum of numbers in the input column (second argument).</td>
</tr>
<tr>
<td><code>min</code></td>
<td>table, string</td>
<td>number</td>
<td>Returns the maximum, minimum, average or sum of numbers in the input column (second argument).</td>
</tr>
<tr>
<td><code>avg</code></td>
<td>table, string</td>
<td>number</td>
<td>Returns the maximum, minimum, average or sum of numbers in the input column (second argument).</td>
</tr>
<tr>
<td><code>sum</code></td>
<td>table, string</td>
<td>number</td>
<td>Returns the maximum, minimum, average or sum of numbers in the input column (second argument).</td>
</tr>
<tr>
<td><code>count</code></td>
<td>table</td>
<td>number</td>
<td>Returns the number of rows in the table.</td>
</tr>
<tr>
<td><code>all_eq</code></td>
<td>table, string, string</td>
<td>bool</td>
<td>Returns whether all entries in the input column are equal (or not equal to) the input value.</td>
</tr>
<tr>
<td><code>all_not_eq</code></td>
<td>table, string, string</td>
<td>bool</td>
<td>Returns whether all entries in the input column are equal (or not equal to) the input value.</td>
</tr>
<tr>
<td><code>all_greater</code></td>
<td>table, string, number</td>
<td>bool</td>
<td>Returns whether all entries in the input column are greater than or less than (or equal to) the input number.</td>
</tr>
<tr>
<td><code>all_lesser</code></td>
<td>table, string, number</td>
<td>bool</td>
<td>Returns whether all entries in the input column are greater than or less than (or equal to) the input number.</td>
</tr>
<tr>
<td><code>most_eq</code></td>
<td>table, string, string</td>
<td>bool</td>
<td>Returns whether most entries in the input column are equal (or not equal to) the input value.</td>
</tr>
<tr>
<td><code>most_not_eq</code></td>
<td>table, string, string</td>
<td>bool</td>
<td>Returns whether most entries in the input column are equal (or not equal to) the input value.</td>
</tr>
<tr>
<td><code>most_greater</code></td>
<td>table, string, number</td>
<td>bool</td>
<td>Returns whether most entries in the input column are greater than or less than (or equal to) the corresponding number.</td>
</tr>
<tr>
<td><code>most_lesser</code></td>
<td>table, string, number</td>
<td>bool</td>
<td>Returns whether most entries in the input column are greater than or less than (or equal to) the corresponding number.</td>
</tr>
<tr>
<td><code>only</code></td>
<td>table</td>
<td>bool</td>
<td>Returns whether the table has exactly one row.</td>
</tr>
<tr>
<td><code>hop</code></td>
<td>row, string</td>
<td>string</td>
<td>Returns the entry corresponding to the input column in the row.</td>
</tr>
<tr>
<td><code>eq</code></td>
<td>string, string</td>
<td>bool</td>
<td>Returns whether the two inputs are equal or not.</td>
</tr>
</tbody>
</table>

Table 8: List of modules for LogicNLG with their corresponding input / output data types and descriptions.
A Modules for Table-to-Text Generation (Cont. from §3.1)

Table 8 shows the list of all modules for Logic-NLG. Our choice of modules is motivated from prior work (Chen et al., 2020c) that defines similar modules for generating logical summaries from open-domain tables.

B Additional Experiments on WebNLG (Cont. from §5)

B.1 Effect of Number of Demonstrations

Figure 3: Comparison of METEOR scores for Direct Prompting (DP) versus MURMUR with varying number of demonstrations on WebNLG test set. DP shows improved performance with more demonstrations while MURMUR’s gains are marginal.

In Fig. 3, we compare the METEOR scores of DP and MURMUR by varying the number of demonstrations. DP shows improved performance with more demonstrations, while MURMUR’s improvements are marginal. In the process of providing more demonstrative examples, DP implicitly learns the underlying step-wise reasoning process, while such phenomenon is explicitly captured through one demonstration in MURMUR.

B.2 Effect of Variations of Demonstrations

In Table 9, we compare the performance of few-shot baselines on the WebNLG validation set. We report mean and variance of BLEU and METEOR scores with three different random seeds for choosing demonstrations from the training set.

In Table 9, we compare the performance of few-shot baselines on the validation set of WebNLG and analyze the effect of different choices of random demonstrations on in-content learning. Using three different random seeds, we show that all methods are fairly robust to randomness in demonstrations.

C Additional Experiments on LogicNLG (Cont. from §6)

C.1 Effect of Number of Demonstrations

In Fig. 4, we compare BLEU-3 scores of DP and MURMUR by varying the number of demonstrations from 1 to 3. Unlike WebNLG, we do not observe any noticeable improvements in in-context learning capabilities with more demonstrations, possibly because of the inherent difficulty of generating logical summaries from tables.

C.2 Effect of Variations in Demonstrations

In Table 10, we study the effect of randomness in the choice of a single demonstration for LogicNLG. We report mean and variance of BLEU scores for each method with a randomly chosen demonstration from the training examples. Similar to WebNLG, all methods are fairly robust to the choice of demonstrations and exhibit comparable variance in performance.

C.3 Effect of Different Beam Sizes in Best-first Search of MURMUR

At each step of the search, MURMUR keeps track of the highest scoring reasoning paths. Table 11 compares the effect of the beam size for our search algorithm on the LogicNLG validation set. Perhaps unsurprisingly, maintaining a bigger beam i.e., conducting a more exhaustive search leads to some
improvements in BLEU scores, however, the gain mostly saturates with beam sizes of around 50-100.

### Table 10: Effect of beam size in M\textsubscript{UR}'s search algorithm on BLEU scores of LogicNLG validation set.

<table>
<thead>
<tr>
<th>Beam Size</th>
<th>BLEU-1 / BLEU-2 / BLEU-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>39.7 / 21.2 / 10.3</td>
</tr>
<tr>
<td>20</td>
<td>40.2 / 21.8 / 10.7</td>
</tr>
<tr>
<td>50</td>
<td>40.5 / 22.2 / 10.8</td>
</tr>
<tr>
<td>100</td>
<td>40.7 / 22.5 / 10.9</td>
</tr>
</tbody>
</table>

### Table 11: Effect of beam size in M\textsubscript{UR}'s search algorithm on BLEU scores of LogicNLG validation set.

**D Prompts (Cont. from §4)**

**WebNLG.** Table 13 shows an example of direct prompting (Zhang et al., 2022) for WebNLG. In Table 15 and Table 16, we show the prompts for the *surface realization* and *text fusion* modules in M\textsubscript{UR}. Note that the single demonstration for direct prompting is decomposed into individual reasoning steps for the two modules in M\textsubscript{UR}.

Let’s convert triples to sentences step-by-step

### Triples: A.S._Gubbio_1910 | league | Serie_D | Italy | leader | Pietro_Grasso | # capital | Rome | # A.S._Gubbio_1910 | ground | Italy | # Serie_D | champions | S.S._Robur_Siena
Output: A.S. Gubbio 1910 plays in Serie D. # Pietro Grasso is the leader of Italy. # S.S. Robur Siena are champions of Serie D in which AS Gubbio 1910 also play. This latter club have their home ground in Italy where the capital city is Rome and the leader is Pietro Grasso.

### Table 14: Example of Chain-of-Thought Prompting for WebNLG. The intermediate reasoning steps (truncated for clarity) are concatenated together and we consider the last step as the final summary.
Table 15: Example of the Surface Realization prompt for MUR in WebNLG.

LogicNLG. Table 17 shows an example of direct prompting for LogicNLG. Table 18 shows an example prompt for the surface realization module in LogicNLG. We only provide the table topic, table header, and the reasoning path in the prompt. We do not add the table content to the prompt because all the information needed by the model to generate the summary is typically present in the reasoning path. Any other contextual information about the table can also be inferred from the table header and topic. We observe that adding the table content makes the model more prone to hallucinations because it may not limit its generation to the information provided in the reasoning path alone.

E Examples of MUR Summaries

In Fig. 6, we show representative examples of summaries generated by Direct Prompting and MUR for WebNLG. Fig. 7 shows the step-wise summary generation process of MUR for WebNLG. In Fig. 8 and 9, we show representative examples of the reasoning paths and summaries generated for two tables in LogicNLG.
Texas is the second largest state in the United States of America. Its capital is Austin and the largest city is Houston.

Houston is the largest city in Texas where English is the official language and Arlington, Texas is a city in Texas.

Hays County, Texas is a county in the U.S. state of Texas. Spanish is the official language of Texas. San Marcos is the county seat of Hays County, Texas where Houston is the largest city in Texas where Austin, Texas is part of Texas.

Alvah Sabin was born on March 3, 1857 in Vermont. Burlington is the largest city in Vermont where Alvah Sabin is from and he played from 1857-03-03 to 1857-03-03. This latter club have their home ground in Italy where the capital city is Rome and the leader is Pietro Grasso.

Let’s combine two sentences

First Sentence: S.S. Robur Siena are champions of Serie D in which AS Gubbio 1910 also play. This latter club have their home ground in Italy. Second Sentence: Rome is the capital of Italy where Pietro Grasso is the leader.

Let’s generate a logically entailed statement from the table

Table Topic: 1938 U.S. Open (golf)
Table Header: place # player # country # score # to par # money
Table Content: 1 # ralph guldahl # united states # 74 + 70 + 71 + 69 = 284 # e # 1000 | ... | 10 # gene sarazen # united states # 74 + 74 + 75 + 73 = 296 # + 12 # 106

Generation: The majority of the players in the 1938 US Open scored at least 9 over par or above.

Table 16: Example of the Text Fusion prompt for MURMUR in WebNLG.
### Input Graph:

- **Antwerp International Airport** | **operatingOrganisation** | Flemish Government
- **Antwerp International Airport** | **elevationAboveTheSeaLevel (in metres)** | 12.0
- **Antwerp International Airport** | **owner** | Flemish Region
- **Antwerp International Airport** | **runwayLength** | 600.0

<table>
<thead>
<tr>
<th>#Step</th>
<th>Module</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Surface Realization</td>
<td>Antwerp International Airport</td>
<td>runwayLength</td>
</tr>
<tr>
<td>2</td>
<td>Surface Realization</td>
<td>Antwerp International Airport</td>
<td>elevationAboveTheSeaLevel (in metres)</td>
</tr>
<tr>
<td>3</td>
<td>Surface Realization</td>
<td>Antwerp International Airport</td>
<td>operatingOrganisation</td>
</tr>
<tr>
<td>4</td>
<td>Text Fusion</td>
<td>Antwerp International Airport is owned by the Flemish Region.; Antwerp International Airport is operated by the Flemish Government.</td>
<td>Antwerp International Airport is owned by the Flemish Region and operated by the Flemish Government.</td>
</tr>
<tr>
<td>5</td>
<td>Text Fusion</td>
<td>Antwerp International Airport is 12.0 metres above sea level.; Antwerp International Airport is owned by the Flemish Region and operated by the Flemish Government.</td>
<td>Antwerp International Airport is 12.0 metres above sea level and is owned by the Flemish Region and operated by the Flemish Government.</td>
</tr>
<tr>
<td>6</td>
<td>Text Fusion</td>
<td>The runway at Antwerp International Airport is 600.0 metres long.; Antwerp International Airport is 12.0 metres above sea level and is owned by the Flemish Region and operated by the Flemish Government.</td>
<td>The runway at Antwerp International Airport is 600.0 metres long and is 12.0 metres above sea level and is owned by the Flemish Region and operated by the Flemish Government.</td>
</tr>
</tbody>
</table>

**Figure 7:** Illustration of the step-wise summary generation process of MURMUR for WebNLG. Each step consists of a module (Surface Realization or Fusion), the input to the module (a triple or a pair of texts) and the output summary (Cont. from §3.3).
Table Topic: 2002 Colorado Rockies Season

<table>
<thead>
<tr>
<th>Date</th>
<th>Opponent</th>
<th>Score</th>
<th>Loss</th>
<th>Attendance</th>
<th>Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 1</td>
<td>giants</td>
<td>5 - 4</td>
<td>hernández (5 - 5)</td>
<td>40893</td>
<td>30 - 26</td>
</tr>
<tr>
<td>June 2</td>
<td>giants</td>
<td>9 - 2</td>
<td>thomson (6 - 4)</td>
<td>40551</td>
<td>30 - 27</td>
</tr>
<tr>
<td>June 3</td>
<td>dodgers</td>
<td>11 - 5</td>
<td>jímenez (1 - 3)</td>
<td>30150</td>
<td>30 - 28</td>
</tr>
<tr>
<td>June 4</td>
<td>dodgers</td>
<td>10 - 4</td>
<td>jones (6 - 1)</td>
<td>30195</td>
<td>30 - 29</td>
</tr>
<tr>
<td>June 5</td>
<td>dodgers</td>
<td>8 - 6</td>
<td>duál (4 - 2)</td>
<td>31793</td>
<td>31 - 28</td>
</tr>
<tr>
<td>June 7</td>
<td>blue jays</td>
<td>8 - 0</td>
<td>hampton (5 - 7)</td>
<td>20322</td>
<td>31 - 30</td>
</tr>
<tr>
<td>June 8</td>
<td>blue jays</td>
<td>3 - 1</td>
<td>thomson (6 - 3)</td>
<td>21298</td>
<td>31 - 31</td>
</tr>
<tr>
<td>June 9</td>
<td>blue jays</td>
<td>3 - 2</td>
<td>jímenez (1 - 4)</td>
<td>20128</td>
<td>31 - 32</td>
</tr>
<tr>
<td>June 10</td>
<td>red sox</td>
<td>7 - 3</td>
<td>neagle (4 - 3)</td>
<td>33558</td>
<td>31 - 33</td>
</tr>
<tr>
<td>June 11</td>
<td>red sox</td>
<td>3 - 1</td>
<td>forham (2 - 1)</td>
<td>32340</td>
<td>32 - 33</td>
</tr>
<tr>
<td>June 12</td>
<td>red sox</td>
<td>7 - 5</td>
<td>hampton (5 - 8)</td>
<td>31583</td>
<td>32 - 34</td>
</tr>
<tr>
<td>June 14</td>
<td>indians</td>
<td>5 - 3</td>
<td>thomson (6 - 6)</td>
<td>40156</td>
<td>32 - 35</td>
</tr>
<tr>
<td>June 15</td>
<td>indians</td>
<td>7 - 4</td>
<td>parroto (1 - 1)</td>
<td>41870</td>
<td>33 - 35</td>
</tr>
<tr>
<td>June 16</td>
<td>indians</td>
<td>5 - 4</td>
<td>neagle (4 - 4)</td>
<td>47702</td>
<td>33 - 36</td>
</tr>
<tr>
<td>June 18</td>
<td>yankees</td>
<td>8 - 5</td>
<td>jennings (8 - 3)</td>
<td>48738</td>
<td>33 - 37</td>
</tr>
<tr>
<td>June 19</td>
<td>yankees</td>
<td>26 - 10</td>
<td>white (1 - 5)</td>
<td>48521</td>
<td>33 - 38</td>
</tr>
<tr>
<td>June 20</td>
<td>yankees</td>
<td>44 - 11 (10)</td>
<td>karsen (3 - 3)</td>
<td>48916</td>
<td>34 - 38</td>
</tr>
<tr>
<td>June 21</td>
<td>devil rays</td>
<td>8 - 7 (9)</td>
<td>yan (3 - 3)</td>
<td>32284</td>
<td>35 - 38</td>
</tr>
<tr>
<td>June 22</td>
<td>devil rays</td>
<td>6 - 5 (8)</td>
<td>kent (6 - 9)</td>
<td>31950</td>
<td>35 - 38</td>
</tr>
<tr>
<td>June 23</td>
<td>devil rays</td>
<td>6 - 5</td>
<td>kenedy (5 - 6)</td>
<td>31043</td>
<td>37 - 38</td>
</tr>
<tr>
<td>June 24</td>
<td>dodgers</td>
<td>4 - 1</td>
<td>ishii (11 - 3)</td>
<td>34641</td>
<td>38 - 38</td>
</tr>
<tr>
<td>June 25</td>
<td>dodgers</td>
<td>4 - 0</td>
<td>thomson (6 - 7)</td>
<td>23535</td>
<td>38 - 39</td>
</tr>
<tr>
<td>June 26</td>
<td>dodgers</td>
<td>5 - 2</td>
<td>chaico (3 - 2)</td>
<td>25013</td>
<td>38 - 40</td>
</tr>
<tr>
<td>June 27</td>
<td>dodgers</td>
<td>7 - 6</td>
<td>neagle (4 - 3)</td>
<td>41278</td>
<td>38 - 41</td>
</tr>
<tr>
<td>June 28</td>
<td>mariners</td>
<td>6 - 2</td>
<td>jennings (8 - 4)</td>
<td>45118</td>
<td>38 - 42</td>
</tr>
<tr>
<td>June 29</td>
<td>mariners</td>
<td>8 - 1</td>
<td>hampton (4 - 9)</td>
<td>45760</td>
<td>38 - 43</td>
</tr>
<tr>
<td>June 30</td>
<td>mariners</td>
<td>4 - 3</td>
<td>sasaki (2 - 2)</td>
<td>49218</td>
<td>39 - 43</td>
</tr>
</tbody>
</table>

Figure 8: Sample table from LogicNLG dataset and five diverse logical summaries generated by MURMUR. Each color code in the table cells highlights parts of the table relevant to a MURMUR summary. All generated summaries are logically correct.
<table>
<thead>
<tr>
<th>year (ceremony)</th>
<th>film title used in nomination</th>
<th>original title</th>
<th>director</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1992 (65th)</td>
<td>the child of man</td>
<td>cilvēka bērns</td>
<td>jānis streits</td>
<td>not nominated</td>
</tr>
<tr>
<td>2008 (81st)</td>
<td>defenders of riga</td>
<td>rigas sargi</td>
<td>aigars grauba</td>
<td>not nominated</td>
</tr>
<tr>
<td>2010 (83rd)</td>
<td>hong kong confidential</td>
<td>amaya</td>
<td>māris martinsons</td>
<td>not nominated</td>
</tr>
<tr>
<td>2012 (85th)</td>
<td>gulf stream under the iceberg</td>
<td>golfa straume zem ledus kalna</td>
<td>Evgeni Pashkevich</td>
<td>not nominated</td>
</tr>
<tr>
<td>2013 (86th)</td>
<td>mother, i love you</td>
<td>mammas, es tevi milu</td>
<td>jānis nords</td>
<td>tbd</td>
</tr>
</tbody>
</table>

**Table Topic:** List of Latvian submissions for the academy award for best foreign language film

**Reasoning Path-1:**

\[
\text{eq} \{ \text{count} \{ \text{filter} \_\_eq \{ \text{all.rows} ; \text{result} ; \text{not nominated} \} \}; 4 \} \\
\]

**Summary-1:** Four of the films submitted by Latvia for the Academy Award for Best Foreign Language Film were not nominated.

**Reasoning Path-2:**

\[
\text{eq} \{ \text{count} \{ \text{filter} \_\_not\_\_eq \{ \text{all.rows} ; \text{result} ; \text{not nominated} \} \}; 1 \} \\
\]

**Summary-2:** 1 film was not nominated for the academy award for best foreign language film.

**Reasoning Path-3:**

\[
\text{eq} \{ \text{count} \{ \text{filter} \_\_all \{ \text{all.rows} ; \text{year (ceremony)} \} \}; 5 \} \\
\]

**Summary-3:** There have been 5 Latvian submissions for the Academy Award for Best Foreign Language Film.

**Reasoning Path-4:**

\[
\text{eq} \{ \text{count} \{ \text{filter} \_\_all \{ \text{all.rows} ; \text{year (ceremony)} \} \}; 5 \} \\
\]

**Summary-4:** There have been 5 Latvian submissions for the Academy Award for Best Foreign Language Film.

**Reasoning Path-5:**

\[
\text{only} \{ \text{filter} \_\_eq \{ \text{all.rows} ; \text{film title used in nomination} ; \text{defenders of riga} \} \} \\
\]

**Summary-5:** defenders of riga was the only Latvian submission for the academy award for best foreign language film.

**Figure 9:** Sample table from LogicNLG dataset and five diverse logical summaries generated by MURMUR. Each color code in the table cells highlights parts of the table relevant to a MURMUR summary. The red marked blocks are incorrect summaries generated by MURMUR.
ACL 2023 Responsible NLP Checklist

A  For every submission:

✔ A1. Did you describe the limitations of your work?  
   9

✔ A2. Did you discuss any potential risks of your work?  
   10

✔ A3. Do the abstract and introduction summarize the paper’s main claims?  
   1

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

B  ✔ Did you use or create scientific artifacts?

5, 6

✔ B1. Did you cite the creators of artifacts you used?  
   5, 6

□ 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)?  
   Not applicable. Left blank.

□ 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?  
   Not applicable. Left blank.

✔ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?  
   4

✔ 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.  
   5, 6

C  ✔ Did you run computational experiments?

5, 6

✔ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?  
   4, 5, 6

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?

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?

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.)?

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

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.?

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)?

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?

D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?

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.