Neural Rule-Execution Tracking Machine For Transformer-Based Text Generation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Sequence-to-Sequence (Seq2Seq) neural text generation models, especially the pre-trained ones (e.g., BART and T5), have exhibited compelling performance on various natural language generation tasks. However, the black-box nature of these models limits their application in tasks where specific rules (e.g., controllable constraints, prior knowledge) need to be executed. Previous works either design specific model structures (e.g., Copy Mechanism corresponding to the rule "the generated output should include certain words in the source input") or implement specialized inference algorithms (e.g., Constrained Beam Search) to execute particular rules through the text generation. These methods require the careful design case-by-case and are difficult to support multiple rules concurrently. In this paper, we propose a novel module named Neural Rule-Execution Tracking Machine, i.e., NRETM, that can be equipped into various transformer-based generators to leverage multiple rules simultaneously to guide the neural generation model for superior generation performance in an unified and scalable way. Extensive experiments on several benchmarks verify the effectiveness of our proposed model in both controllable and general text generation tasks.

1 Introduction

Transformer-based neural language models (LMs), such as GPT/BART [1, 2, 3], have led a wave of new trends in natural language generation, producing texts of prominent quality. Such LMs are usually trained roughly on huge amounts of text corpora to maximize the likelihoods of predicting next tokens. Despite their success in varieties of NLP tasks, we argue that the black-box nature of these models leads to inefficiently learning to follow constraints and incorporating prior knowledge.

In controllable text generation, most relevant studies [4, 5, 6] focus on controlling high-level text attributes (e.g., topic, sentiment) or simply keyword/phrase. More complex fine-grained control constraints such as "generate a text with 'apple' in the second sentence which has 15 words and 'orange' or 'oranges' in the sixth sentence" are less explored. A very recent work [7] reveals that large-scale LMs do not learn to obey the underlying constraints reliably, even in a quite simple constrained generation task (cover all the given keywords without hallucinating new ones). It is conceivable that how such LMs will behave when expected to follow the propositional control constraints mentioned above. In general text generation, existing works on various tasks reveal the benefit of incorporating task-specific prior knowledge: machine translation [8] (e.g., the coverage constraint that each source phrase should be translated into exactly one target phrase), text summarization [9] (e.g., the lead bias: front loading the most salient information), dialogue generation [10] (e.g., humans tend to repeat entity names or even long phrases in conversation). However, they either need designing specific model architectures (e.g., Coverage Mechanism and Copy Mechanism) or devising well-designed learning objectives (e.g., GSG [11]). These methods require careful design case-by-case and are difficult to combine multiple arbitrary prior knowledge simultaneously.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

The dilemma of the above two research lines motivates us to explore unifying the constrained generation and prior knowledge integration into one single generation framework that could effectively execute multiple rules (defined in predicate logic form) simultaneously. Thus, we propose to equip the transformer-based sequence-to-sequence architecture with a novel module named Neural Rule-Execution Tracking Machine (`NRETM`) [1], which effectively enforces the satisfaction of given constraints by dynamically monitoring the expression progress of each constraint during the decoding process. We leverage executable programs as the logic checking operator, thus these constraints can be any predicate logic formula and a variety of predicate functions can be defined to support flexible expansion. `NRETM` consists of a state matrix and a logical tracker where the former records status of constraint expression and the latter update the state matrix according to the latest decoding result. The state representations aggregated from the state matrix are injected into the decoder as the relative positions between encoder output and current decoder input in the cross-attention module. Our approach reconciles symbolic computing (that has precise logic and numerical calculation capabilities) with neural language generation (that has an exceptional ability of wording and phrasing), which results in both the precisely controllable ability and the superior generation performance.

We conduct experiments on three benchmarks: ROCStories [12] consisting five-sentence stories; Commonsense Generation [13], target explicitly test machines for the ability of generative commonsense reasoning; TED15 Zh-En [14], a common document-level machine translation benchmark. We design various of controllable and general text generation settings in these benchmarks and our `NRETM` model significantly outperforms Seq2Seq pre-trained baselines.

Our contributions in this work are three-fold: (1) proposal of unifying constrained generation and prior knowledge incorporation in a highly customizable predicate logic controllable generation framework; (2) proposal of a neural rule-execution tracking machine that effectively guides the generation following any predicate logic constraints; and (3) empirical verification of the effectiveness of the proposed approach on three benchmarks.

2 Approach

In this work, we formalize the execution of rules during text generation as generating sentences that conform to certain predicate logic constraints. We start from a definition of predicate logic constraint in our framework, an overview of the model, and then dive into details of each component.

2.1 Predicate Logic Constraint

We define a predicate U with zero or more arguments. For example, $U(\mathbf{a}, \mathbf{b})$ is an atomic formula with predicate U and two arguments, here occupied by the variable **a** and **b**. The predicate $U(\mathbf{a}, \mathbf{b})$ is a boolean function asserting a relationship between variables **a** and **b**, e.g., $\text{Copy}(\mathbf{a}, \mathbf{b})$ means the occurrence of keyword **a** in a sequence **b**, where Copy is a specific kind of predicate and **a** can be either unigram or multi-gram. In general, an argument is either a variable or a constant. Further, our method accepts predicate logic constraint P_c in propositional logical formula like:

$$\big(U_1 \vee U_2 \cdots \vee U_i\big) \wedge \cdots \wedge \big(U_k \cdots \vee U_n\big)$$

where each U_i represents a single positive or negative constraint, e.g., $U(\mathbf{a}, \mathbf{b})$ or $\neg U(\mathbf{a}, \mathbf{b})$, restricting whether a and b satisfy the relationship defined by U or not, respectively. Our method seeks optimal generated sequence $\mathbf{y}$ conditioned on the input $\mathbf{x}$ in which P_c are satisfied:

$$\mathbf{y} = \arg\max_{\mathbf{y} \in \mathcal{Y}} P(\mathbf{y}|\mathbf{x}) \quad \text{where } P_c = 1 \tag{1}$$

With the help of custom predicate logic functions, various control constraints and prior knowledge can be converted to the propositional logical formula, and thus handled by `NRETM`. For example, the control constraint "generate a text with 'apple' in the second sentence which has 15 words and 'orange' or 'oranges' in the sixth sentence" can be transformed to "$\text{InSen}(\text{apple}, 2) \wedge \text{SenLen}(2, 15) \wedge (\text{InSen}(\text{orange}, 6) \vee \text{InSen}(\text{oranges}, 6))$"; the prior knowledge "each source phrase should be translated into exactly one target phrase" can be transformed to "$\forall\, s_i \in \mathbf{S}, \text{TranslatedOnce}(s_i)$", s_i is the ith phrase in source. We define six kinds of predicates that used by `NRETM` in this work, details please refer to the Supplementary Material A.2.

[1] We will make the code public available later to facilitate reproducing the results

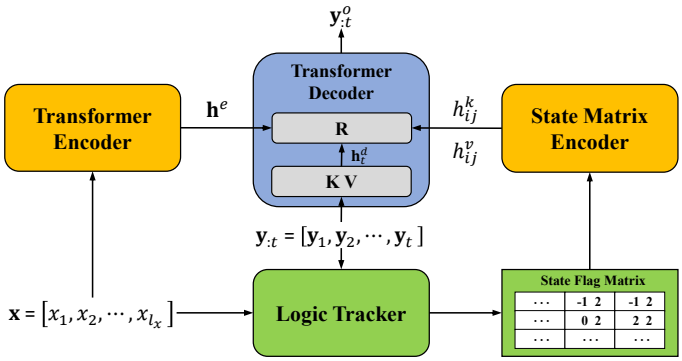

Figure 1: An overview of our NRETM model. Given logic constraints, input and current generated sequences, our *Logic Tracker* maintains the current status of all logic constraints in State Flag Matrix, which is further processed by the State Matrix Encoder and fed into the Transformer-based text decoder to guide the text generation procedure.

2.2 Model Overview

Figure 1 illustrates the architecture of our neural rule-execution tracking machine (NRETM). NRETM is able to be equipped into any sequence-to-sequence transformer-based text generator, for the sake of simplicity, we take T5 for an example. Before output decoding, the input tokens are fed into transformer encoder, and the state matrix is initialized by the initial value of all predicates in constraints. Then, during response decoding, the state matrix tracks the expression of each predicate and gets updated by the Logic Tracker. The Logic Tracker manages the process of decoding at each step by checking the logical satisfaction of each predicate, and updating the status and corresponding intermediate value of each predicate into the state matrix. Based on the information provided by the state matrix, and the generated word sequence, the decoder predicts the next word of the output.

2.3 Constrained Text Generation

In the sequence-to-sequence constrained text generation tasks, we are given encoder inputs $\mathbf{x} = [x_1, \ldots, x_{l_m}, \ldots, x_{l_x}] \in \mathcal{X}$, where $[x_{l_{m+1}}, \ldots, x_{l_x}]$ corresponds to predicate logic constraints that should be satisfied in the generated outputs. At generation step t, the decoder take $\mathbf{y}_{:t} = [y_1, \cdots, y_t] \in \mathcal{Y}$ as input and generate $\mathbf{y}_{t+1}$.

2.4 Neural Rule-Execution Tracking Machine

The NRETM dynamically controls text generation according to the given constraints via the cooperation of the state matrix and the Logic Tracker. It informs the decoder at the first time to what extend the constraints have been expressed. For the example shown in 1, the dynamic control strategy is more reasonable than static strategies such as concatenating the logic constraints with the input text to the encoder. This is because if the whole constraints are composed of five predicates and two predicates have been satisfied during decoding, then the decoder needs to know which three predicates have not been met and the intermediate status of each unfulfilled predicate rather than always memorizing that the five predicates should be all satisfied.

State Flag: Suppose that the given constraint $\mathcal{M}$ consists of n predicates $\mathcal{D} = \{U_i\}_{i=1}^n$. At generation step t, there is a set of state flags indicates the expression progress of each predicate in $\mathcal{M}$ up to this step (i.e., in the decoder input sequence $\mathbf{y}_{:t}$). Formally, they can be defined as $\mathrm{s} : \mathcal{X} \times \mathcal{Y} \rightarrow \{q_{it}\}_{i=1}^{l_x}$ where $|\mathrm{s}(\mathbf{x}, \mathbf{y}_{:t})| = |\mathbf{x}|$, q_{it} is the state flag in a string form. Specifically, state flag $\mathrm{s}(\mathbf{x}, \mathbf{y}_{:t})_i = q_{it}$ is for the input token x_i in $\mathbf{x}$ at the decoding step t. As q_{it} records the satisfaction of $\mathcal{D}$, q_{it} is the concatenation of $\{q^k_{it}\}_{k=1}^n$, where q^k_{it} is the ith expression state string of U_k returned by Logic Tracker.

Logic Tracker: Suppose that NRETM consist of l unique predicates $\{\mathcal{U}_i\}_{i=1}^l$, then the Logic Tracker $\mathcal{L}$ is made up of l logical operator $\{\mathcal{O}_i\}_{i=1}^l$, where $\mathcal{O}_i(\cdot, \cdot)$ is an executable program corresponding to $\mathcal{U}_i$, which takes the arguments of $\mathcal{U}_i$ and a default argument $\mathbf{y}_{:t}$ as inputs, and outputs expression state strings. Specifically, at step t given a predicate U_k, $\mathcal{L}.\mathcal{O}_{\bar{k}}(U_k, \mathbf{y}_{:t}) = \{q^k_{it}\}_{i=1}^{l_x}$, where q^k_{it} is the

State Flag Matrix

$\mathbf{x}$ \ $\mathbf{y}_{:t}$	The	woman	cannot	drive	her	car	in	the	snow	EOS
Copy(car)	0 N (1,9)	0 N (1,8)	0 N (1,7)	0 N (1,6)	0 N (1,5)	0 N (1,4)	2 N (1,3)	2 N (1,2)	2 N (1,1)	2 N 2
&	N N (1,9)	N N (1,8)	N N (1,7)	N N (1,6)	N N (1,5)	N N (1,4)	N N (1,3)	N N (1,2)	N N (1,1)	N N 2
Copy(snow)	N 0 (1,9)	N 0 (1,8)	N 0 (1,7)	N 0 (1,6)	N 0 (1,5)	N 0 (1,4)	N 0 (1,3)	N 0 (1,2)	N 0 (1,1)	N 2 2
&	N N (1,9)	N N (1,8)	N N (1,7)	N N (1,6)	N N (1,5)	N N (1,4)	N N (1,3)	N N (1,2)	N N (1,1)	N N 2
Len(9)	N N (1,9)	N N (1,8)	N N (1,7)	N N (1,6)	N N (1,5)	N N (1,4)	N N (1,3)	N N (1,2)	N N (1,1)	N N 2

Figure 2: A running example of our NRETM model with three logic constraints (i.e., copy(car) & copy(snow) & len(9); copy car and snow in the output with 9 tokens). Each State Flag is a string that records the current status of all given logic constraints. Three tokens in each State Flag correspond to the constraints with the same color. Length constraint ("len(9)") tracks intermediate values in purple color. Copy constraints only track the final satisfaction status. All constraints are satisfied in the end.

concatenation of default state status $\hat{q}_{it}^k$ and additional intermediate values $\bar{q}_{it}^k$ (it is optional, maybe empty). $\hat{q}_{it}^k$ is formulated as:

$$\hat{q}_{it}^k = \begin{cases} \text{N} & \text{variables of } U_k \text{ do not contain } x_i \\ 0 & U_k \text{ is not satisfied} \\ 1 & U_k \text{ is in progress} \\ 2 & U_k \text{ is satisfied} \end{cases} \tag{2}$$

The values 0 and 2 represent the status of predicate constraint satisfaction. At the start, for all tokens the value of default state status is set to 0. Once $\mathbf{y}_{:t}$ satisfies the constraints, value of the corresponding $\hat{q}_{it}^k$ is updated to 2. Value 1 is a state customized by each logical operator indicates that at current step t the predicate is not satisfied but triggered the beginning condition. $\bar{q}_{it}^k$ is another value that customized by each logical operator for better tracking, for example in the predicate $\text{Len}(\mathbf{y}, 20)$ (means the generated output should have a length of 20), $\bar{q}_{it}^k$ could be 20 minus the current length of $\mathbf{y}$ difference informing decoder how many words are left to satisfy the constraint. Each variable in U_k takes one form from the following set $\{\mathbf{x}_{i:j}, \mathbf{y}_{i:j}, \mathbf{x}^i, \mathbf{y}^i\}$, where $\mathbf{x}_{i:j}$ and $\mathbf{y}_{i:j}$ are the sub-sequences of $\mathbf{x}$ and $\mathbf{y}$ from i^{th} token to j^{th} token respectively, $\mathbf{x}^i$ and $\mathbf{y}^i$ refer to i^{th} sentence in $\mathbf{x}$ and $\mathbf{y}$ respectively. In a special case (i.e., variables in U_k only have tokens from $\mathbf{y}$), $\hat{q}_{it}^k$ only have three states: 0,1,2.

Example: In Figure 1, we are given three logic constraints, *a)* copy "car"; *b)* copy "snow" and *c)* generate 9 tokens. Each State Flag has three values indicating the status of the three logic constraints. We assign "N" to the second constraints in the first row and the first constraints in the third row because *a)* and *b)* only associate with their corresponding words. Length control maintains a set of intermediate values (e.g., the residual length). While copy controls do not have intermediate values and they are updated from Value 0 to 2 only when the corresponding words (at step 7 and 10 in our example) are generated in the $\mathbf{y}_{:t}$.

State Matrix: Given $\mathbf{x}$, $\mathbf{y}_{:t}^i$, we define the two-dimensional state matrix $\mathcal{S} \in \{q_{ij}\}^{l_x \times t}$ as follows:

$$\mathcal{S} = [\text{s}(\mathbf{x}, \varepsilon); \text{s}(\mathbf{x}, \mathbf{y}_{:1}); \cdots; \text{s}(\mathbf{x}, \mathbf{y}_{:t})] \tag{3}$$

where ε is the empty string. During training, given $\mathbf{x}$ and the golden output $\mathbf{y}^{gt}$ (with l_{gt} tokens), we can construct the ground-truth state matrix $\mathcal{S}^{gt} \in \{q_{ij}\}^{l_x \times l_{gt}}$ by utilizing each step of $\mathbf{y}^{gt}$ to get the state flag from the Logic Tracker $\mathcal{L}$. $\mathcal{S}^{gt}$ follows the same masking strategy as the decoder input tokens $\mathbf{y}_{:t}$. For the tokens whose corresponding predicate constraints having no alignment with $\mathbf{y}^{gt}$, their state flags are also assigned value N. During inference, we build the state matrix incrementally, starting from $\mathcal{S}^{inf,0} = [\text{s}(\mathbf{x}, \varepsilon)] \in \{\text{N}, 0\}^{l_x \times 1}$. In step t, we add a new column $\text{s}(\mathbf{x}, \mathbf{y}_{:t}^i)$ to $\mathcal{S}^{inf,t-1} \in \{q_{ij}\}^{l_x \times (t-1)}$ and obtain the new state matrix $\mathcal{S}^{inf,t} \in \{q_{ij}\}^{l_x \times t}$.

2.5 Integration with Transformer-Based Generator

In a standard transformer-based encoder-decoder framework, given the input tokens $\{x_i\}_{i=1}^{l_x}$ and the generated tokens $\mathbf{y}_{:t}$, the probability of the next token y_{t+1} can be calculated by:

$$\mathbf{H}^e = \mathrm{TransformerEncoder}(\{\mathbf{E}(x_i)\}_{i=1}^{l_x}) \tag{4}$$

$$\mathbf{H}_t^d = \mathrm{KV}(\mathbf{W}_q^s\mathbf{E}(\mathbf{y}_{:t}), \mathbf{W}_k^s\mathbf{E}(\mathbf{y}_{:t}), \mathbf{W}_v^s\mathbf{E}(\mathbf{y}_{:t})) \tag{5}$$

$$o_{t+1} = \mathrm{KV}(\mathbf{W}_q^c\mathbf{H}_t^d, \mathbf{W}_k^c\mathbf{H}^e, \mathbf{W}_v^c\mathbf{H}^e) \tag{6}$$

$$p(y_{t+1}|x_{1:l_x}, y_{1:t}) = \mathrm{softmax}(\mathbf{W}_o\, o_{t+1}) \tag{7}$$

where $\mathbf{E} \in \mathbb{R}^{d_e \times |V|}$ is the word embedding matrix with d_e the dimension and $|V|$ the vocabulary size, $o_{t+1} \in \mathbb{R}^{d_c}$ is the hidden state at step t with d_c the hidden size, and $\mathbf{W}_o \in \mathbb{R}^{|V| \times d_c}$ is a parameter matrix that maps the hidden state o_{t+1} to a logit vector in the size of $|V|$, and KV is the standard key-value self-attention proposed in [15].

State Matrix Encoder: For each element q_{ij} in the state matrix $\mathcal{S}$, we use the same tokenizer of the transformer-based encoder-decoder framework to obtain tokenized sequence $\{q_{ij}^k\}_{k=1}^{l_{ij}^q}$. Then we get the representation of q_{ij} by feeding the tokenized sequence into a shallow transformer-based encoder (only one-layer encoder using same architecture as $\mathrm{TransformerEncoder}$), the detailed formulation is:

$$\mathbf{h}_{ij} = \mathrm{ShallowEncoder}(\{\mathbf{E^q}(q_{ij}^t)\}_{t=1}^{l_{ij}^q}) \tag{8}$$

$$\bar{\mathbf{h}}_{ij} = \mathrm{MeanPooling}(\mathbf{h}_{ij}) \tag{9}$$

where $\mathbf{E^q} \in \mathbb{R}^{d_k \times |V|}$, which is frozen word embeddings initialized with the parameters of the first d_k dimension of $\mathbf{E}$, $\mathbf{h}_{ij} \in \mathbb{R}^{l_{ij}^q \times d}$, $\bar{\mathbf{h}}_{ij} \in \mathbb{R}^d$ and d is the hidden size of $\mathrm{ShallowEncoder}$.

State Matrix as Relative Position: Inspired by [16] which incorporates token relative positions into the self-attention (Eq. 5) module, we propose to inject State Matrix as the "relative positions" between encoder output $\mathbf{H}^e$ and current decoder input $\mathbf{y}_{:t}^i$ in the cross-attention (Eq. 6) module. Following this approach, we linearly project each $\bar{\mathbf{h}}_{ij}$ into *State Flag key* $\bar{\mathbf{h}}_{ij}^k = \mathbf{W}_k^f \cdot \bar{\mathbf{h}}_{ij} + \mathbf{b}_k^f$ and *State Flag Value* $\bar{\mathbf{h}}_{ij}^v = \mathbf{W}_v^f \cdot \bar{\mathbf{h}}_{ij} + \mathbf{b}_v^f$. All transformer decoder layers share the same State Flags representations. Eq. 6 is changed to:

$$o_{t+1} = \mathrm{R}(\mathbf{W}_q^c\mathbf{H}_t^d, \mathbf{W}_k^c\mathbf{H}^e, \mathbf{W}_v^c\mathbf{H}^e, \bar{\mathbf{h}}^k, \bar{\mathbf{h}}^v) \tag{10}$$

where $\bar{\mathbf{h}}^k$ and $\bar{\mathbf{h}}^v \in \mathbb{R}^{l_x \times t \times d}$ and R is the Self-Attention function with relative position, defined as follows:

$$R(\mathbf{q}, \mathbf{k}, \mathbf{v}, \mathbf{m}^k, \mathbf{m}^v)_j = \sum_{i=1}^{l_x} \mathbf{a}_{i,j}(\mathbf{v}_i + \mathbf{m}_{i,j}^v) \tag{11}$$

$$\mathbf{a}_{*,j} = \mathit{Softmax}(\mathbf{e}_{*,j}) \tag{12}$$

$$\mathbf{e}_{i,j} = \frac{\mathbf{q}_j(\mathbf{k}_i + \mathbf{m}_{i,j}^k)^T}{\sqrt{d}} \tag{13}$$

2.6 Why NRETM Could Satisfy Constraints

Intuitively, we just make the neural generator aware of the expression state of each predicate logic constraint rather than forcing the model to execute these logical expressions explicitly. However, a powerful implicit compulsion comes from the combined force of two aspects: (1) before generating the EOS token (i.e., End-Of-Sequence Token), all the predicate constraints should be satisfied (you can imagine, in the training data at the position of EOS token in the ground-truth output, all elements in the state matrix must be set to "satisfied"); (2) The EOS token also should be generated within a limited decoding steps (any training data is of finite length). Such a soft way of combining symbolic operators (good at logical and mathematical calculations) and neural operators (good at wording and phrasing) can retain their respective strengths to the utmost extent.

3 Experiment

We test the proposed method on two kinds of generation tasks: controllable text generation and general text generation. For controllable text generation, we choose ROCStories to verify that our method could generate a text under complex fine-grained control instruction. We leverage commonsense generation and document-level machine translation to prove that NRETM can efficiently integrate prior knowledge into seq2seq models towards superior generation performance.

3.1 Controllable ROC Stories

In order to fully tap the potential of our framework in executing complex control sentences (defined in the form of predicate logic), we choose an open-domain text generation benchmark ROCStories [12] which is a corpus of five-sentence stories that capture a rich set of causal and temporal commonsense relations between daily events, making them a good resource for training text generation models.

Dataset The ROCstories dataset consists of 98,162 five-line stories for training, and 1871 stories each for the development and test sets. Following [17], we construct storyline, which is a (un)ordered sequence of phrases that are mentioned in the story, using RAKE algorithm [18].

Predicate Logic Formulation As shown in table 1, We have experimented with various storyline constraints with increasing difficulties: (1) Generate a story in the order of the storyline w_1, w_2, w_3, w_4. (2) Generate a story with storyline w_1 in the ith sentence and w_2 in the jth sentence. (3) Generate a story with storyline w_1 in the ith sentence which has l_i words and w_2 in the jth sentence whose length is l_j. (4) Generate a storyline w_1 in the ith sentence which has l_i words or s_i stop words and w_2 in the jth sentence that does not mention w_3 (5) Generate a storyline w_1 in the ith sentence which has l_i words or s_i stop words and w_2 in the jth sentence which has l_j words or s_j stop words.

Baselines and Metrics We use the T5-Base model as baseline which takes the given predicate logic constraints as inputs and generates the five-sentence stories. We combine our proposed `NRETM` module with the T5-Base model to track the execution of the constraints. We report *Constraints Success Ratio* (CSR) and ROUGH-L (R-L) on the ROCStories Test Split to evaluate our `NRETM` model.

Main Results As shown in Table 1, in all five predicate logic constraints, compared to the T5 model, the `NRETM` model achieves higher Constraint Success Ratio and maintains a similar level of ROUGH-L, showing that the `NRETM` model can be flexibly controlled without loss of generated text quality. The gap in CSR between the T5 and `NRETM` model is moderate in the first two constraints with simple token permutations. However, the success ratio of T5 model drops significantly given constraints that requires long-range numerical tracking (e.g., sentence length and the count of stop words).

Table 1: Controllable ROCStories Experiment Results.

Predicate Logic Constraint	Model	CSR	R-L
$\text{InSen}(w_1, y^i) \wedge \text{InSen}(w_2, y^j)$	T5	98.7	33.1
	`NRETM`	98.8	33.3
$\text{Order}(w_1, w_2) \wedge \text{Order}(w_2, w_3) \wedge \text{Order}(w_3, w_4)$	T5	97.0	49.6
	`NRETM`	97.4	49.5
$\text{InSen}(w_1, y^i) \wedge \text{InSen}(w_2, y^j) \wedge \text{Len}(y^j, l_j) \wedge \text{Len}(y^i, l_i)$	T5	18.5	32.9
	`NRETM`	84.5	32.6
$\text{InSen}(w_1, y^i) \wedge \text{InSen}(w_2, y^j) \wedge (\neg\text{InSen}(w_3, y^j))$ $\wedge(\text{Len}(y^i, l_i) \vee \text{StopWordCount}(y^i, s_i))$	T5	23.5	33.0
	`NRETM`	49.4	33.1
$\text{InSen}(w_1, y^i) \wedge \text{InSen}(w_2, y^j) \wedge (\text{Len}(y^i, l_i) \vee \text{StopWordCount}(y^i, s_i))$ $\wedge(\text{Len}(y^j, l_j) \vee \text{StopWordCount}(y^j, s_j))$	T5	11.0	32.9
	`NRETM`	35.9	33.0

3.2 Commonsense Generation

COMMONGEN is a generation benchmark dataset target explicitly test machines for the ability of generative commonsense reasoning. Given a set of common concepts the task is to generate a coherent sentence describing an everyday scenario using these concepts.

Dataset The COMMONGEN dataset consists of 35,141 concept-sets (32,651 in *train*, 993 in *val*, 1,497 in *test*) associated with 77,449 sentences.

Predicate Logic Formulation The input is an unordered set of n concepts $\mathbf{x} = \{x_i\}_{i=1}^n$. The expected output is a coherent, grammatical sentence $\mathbf{y}$ that describes a common scenario using all given concepts in $\mathbf{x}$. From the expectation of COMMONGEN, one easily obtained prior knowledge is that each x_i must appear in output $\mathbf{y}$. The corresponding predicate logic constraint P_c is:

$$P_c = \wedge_{i=1}^n \big(\text{Copy}(x_i)\big)$$

where $\mathbf{y}$ will appear by default, for the sake of brevity, we have omitted $\mathbf{y}$ in predicate Copy. Another prior knowledge comes from the observation that generating $\mathbf{y}$ requires giving the correct morphological inflections of the concept word rather than copy it as it is. Let $\tilde{x}_i = \{\tilde{x}_k^i\}_{k=1}^{|\tilde{x}_i|}$ denote all inflections of x_i. $\mathbf{y}$ covers concept x_i, if at least one of $\{\tilde{x}_k^i\}_{k=1}^{|\tilde{x}_i|}$ appears. The constraint $\hat{P}_c$ is:

$$\hat{P}_c = \wedge_{i=1}^n \big(\vee_{j=1}^{|\tilde{x}^i|} \text{Copy}(\tilde{x}_j^i)\big)$$

Baselines and Metrics The standard pipeline of approaching this problem is to consider it as a conditional sentence generation task. We experiment with the recent pretrained language models, including T5-Base and T5-Large[19]. All models are finetuned with their default hyperparameters. We equip `NRETM` into the T5-Large and T5-Base model to incorporate P_c and $\hat{P}_c$ respectively (+ `NRETM` P_c) (+ `NRETM` $\hat{P}_c$). Grid Beam Search (GBS) [20] is a well-designed decoding method that ensures the generation model satisfies the lexical constraints. We only apply GBS to the T5-Base model due to the memory constraint (+ G). Following the suggestions in [13], we use CIDEr [21] and SPICE [22] to automatically assess the quality of generated texts. We calculate constraint satisfaction for all constraints (ALL), novel constraints (Novel) and seen constraints (Seen).

Main Results Table 2 shows that the `NRETM` model improves the constraint satisfaction over the baselines for all cases, achieving close to 100% (i.e., 99.3% and 99.4%). While GBS achieves perfect constraint satisfaction (i.e., 100%), doing so significantly degrades the output text quality (more than 50 CIDEr), indicating the necessity integrating piror knowledge in training rather than inference. In addition, both prior knowledge P_c and $\hat{P}_c$ have a positive effect on our model, improving our T5-large baseline by 3.7 and 11.4 CIDEr score, respectively. Finally, our *T5-Large* + *`NRETM`* $\hat{P}_c$ model outperforms the previous state-of-the-art result [23], which integrates the *ConceptNet* [24] into the BART model, suggesting that our incorporated task-specific prior knowledge could be as powerful as knowledge from large-scale hand-crafted corpus. We calculate the average growth of CIDEr score for the four improvement directions: (1) double the parameters (from *T5-Base* to *T5-Large*) (2) add more piror knowledge (from P_c to $\hat{P}_c$) (3) add symbolic operator (only increase 0.9% parameters by adding `NRETM`). The growth from (1) to (3) is 4.06, 6.55, 4.3. Unexpectedly, adding `NRETM` into T5-Base model is better than adding another T5-base, only adding a new piror knowledge in `NRETM` significantly outperforms doubling the parameters. All of the above shows how potential it is to find a method that could execute multiple rules effectively.

3.3 Document-Level Machine Translation

Document-level machine translation tasks is a general text generation task, where the goal is to translate segments of text that contain more than one sentence (up to an entire document).

Dataset We conduct experiments on the common document-level MT dataset: TED15 Zh-En which is from IWSLT 2014 and 2015 [25, 26]. Following [14], we use 2010-2013 TED as the test set.

Predicate Logic Formulation The input is an ordered set of n sentences in the source language that form a document $\mathbf{x} = \{x^i\}_{i=1}^n$, the expected output is a translated document $\mathbf{y} = \{y^i\}_{i=1}^n$ = in the target language. We observed that neural model is prone to sentence correspondence confusion (the ith sentence in source document is translated as the jth sentence in target document) when doing

Table 2: Experiment Results on Commonsense.

Method	CIDEr	SPICE	Constraint		
			Seen	Novel	ALL
T5-Base	159.5	31.9	95.4	93.3	94.0
T5-Base + G	110.7	27.8	100	100	100
T5-Base + NRETM P_c	164.4	32.1	95.7	94.6	95.6
T5-Base + NRETM $\hat{P}_c$	169.8	32.7	99.5	99.1	99.3
T5-Large	163.6	32.4	94.4	93.0	93.9
T5-Large + NRETM P_c	167.3	33.0	93.9	93.8	93.9
T5-Large + NRETM $\hat{P}_c$	175.0	33.8	99.6	99.2	99.4
KGBART [23]	168.3	32.7	-	-	98.6

document-level translation. To alleviate this problem, we propose incorporating Doc-mBART25 with prior knowledge: each source sentence should be translated only once:

$$\text{TranslatedOnce}(x^i) = \begin{cases} 2 & \theta(\mathbf{y}_{:t}) > i \\ 1 & \theta(\mathbf{y}_{:t}) = i \\ 0 & \theta(\mathbf{y}_{:t}) < i \end{cases} \tag{14}$$

where $\theta(\cdot)$ returns the line number of y_t in $\mathbf{y}$, as t is monotonic during generation, the status only set to be 2 once. The predicate logic constraint P_c of this task can be formulated as:

$$P_c = \wedge_{i=1}^{n} \big(\text{TranslatedOnce}(x^i)\big)$$

Baselines and Metrics We combine our NRETM P_c component with the Doc-mBART25 model proposed in [3] which is a state-of-the-art multilingual pre-trained language model. We compare this model with the state-of-the-art non-pretraining and pretraining approaches, including HAN (Hierarchical Attention Networks) [14], Doc-mBART25 and Sen-mBART25 [3]. When implementing our model, we use the same pre-processing method, blocks segmentation strategy and beam search setting as [3]. Unlike [3], We use both document-level (d-BLEU) and sentence-level (s-BLEU) to measure the similarities between generated target document and the source document. Besides, we also calculate the Sentence Aligned Ratio (SAR) between the source and target documents.

Main Results Table 3 shows that the NRETM P_c component helps the Doc-mBART25 model to better capture the sentence-level corresponding relationship between the source and target documents. In particular, sentence-level alignment ratio is improved from 98.7% to 100%. The improvement in s-BLEU (+ 1.1 BLEU) also confirms that our final Doc-mBART25 + NRETM P_c model learns to translate sentences based on the sentence order in source documents.

Table 3: Model Performance on TED15 Zh-En Test.

Model	s-BLEU	d-BLEU	SAR
Doc-mBART25 + NRETM P_c	24.9	30.4	100 %
Doc-mBART25 [3]	23.8	29.6	98.7 %
Sen-mBART25 [3]	-	28.4	-
HAN [14]	-	24.0	-

3.4 Discussion

Zero-Shot Execution In Table 1, we show that simple sequence-to-sequence pre-trained language model (i.e., T5) cannot handle complicated and fine-grained constraints even after fine-tuning. Here, we further demonstrate that NRETM model is capable to handle zero-shot rule execution. We train the T5 and NRETM model to only mention storyline in the 3rd, 4th and 5th sentence and test these models to mention storyline in the first and second sentence of the whole story. As shown in Table 4, although both T5 and NRETM model mention most of the storyline (95.7% and 98.3% respectively) in the generated story,

Table 4: Novel Sentence Index Experiment. MR for Mention Ratio.

model	CSA	MR	R-L
T5	19.7	95.7	30.0
NRETM	97.7	98.3	33.3

the T5 model only mention 19.7% of storyline in the correct sentence and the NRETM model makes 97.7% of storyline correct. This is becuase the seuqence-to-sequence T5 model cannot recognize the novel sentence index "1" and "2" during the generation. The logic tracker helps the NRETM model to generalize to handle these cases.

Case Study More generation results of our models can be found in the Supplementary Material A.1.

4 Related Work

We are mainly related to two lines of research work in the field of text generation: constrained generation and prior knowledge integration. Some of the earliest works [27, 28, 29] involve controllable generation methods where the generators are trained on text data with the labeled target attributes. CTRL[4], PPLM[6] and CoCon[30] are recent approaches that built on the transformer-based large-scale pretrained LMs, they pay more attention on controlling high-level attributes,phrases and keywords. While NRETM focuses on controlling text generation to follow arbitrary logical constraints, leading to a fine-grained control. Recently, GDC[31] permits to specify both pointwise and distributional constraints over the target LMs. In the field of constrained generation closest to our work is NEUROLOGIC[7] which seeks to generate fluent text while satisfying complex lexical constraints (in a predicate logic form). While NEUROLOGIC's design also enables controlled generation with logical constraints, our approach leverages executable programs as logical operators and the dynamic tracking mechanism to realize a highly customizable and scalable predicate logic function system rather than simply controlling what words should and should not appear in the output. Existing efforts [32, 33, 34, 35, 36] to incorporate prior knowledge into sequence-to-sequence framework either resort to modifying model architectures or designing training objectives. To the best of our knowledge, we first attempt to formalize the prior knowledge integration in seq2seq generation as text generation that conforms to predicate logic constraints.

5 Conclusion and Future Work

In this paper, we propose a unified controllable generation framework that leverages predicate logic constraints to implement efficient complex fine-grained control and scalable prior knowledge integration. We explore and compare two controllable strategies: dynamic tracking and static strategy, and show that the proposed dynamic tracking mechanism significantly outperforms the static ones. Empirical results on three benchmarks indicate that NRETM could achieve accurate control and exhibits a superior generation ability over different tasks. Pre-trained models have been the dominant paradigm in natural language processing, and researchers resort to massive data and large-scale models to improve performance. We unify the rules used in various tasks into the form of predicate logic, provide the possibility to pretrain models on massive rules. In the future, we will explore pre-training large-scale neural rule-execution machine with massive rules and data.

Broader Impact

Our work proposes a unified and scalable approach to efficiently perform fine-grained controllable text generation and incorporate multiple prior knowledge for superior text generation performance. This work uses story generation, machine translation, commonsense generation as applications to verify the effectiveness. However, it could be used in a broader range of more complex scenes (e.g., financial report generation) due to its advantages of combining multiple rules to generate high-quality texts. On the positive side, more efficient text generation can make these technologies more available to the general public. Our model verified on machine translation can benefit commercial and humanitarian translation services to help overcome language barriers. With these advantages also come potential downsides: powerful language models might be used to generate abuse, faked or misleading content in the news or on social media; they might pose safety concerns if they are used to generate harassing or hateful materials. In order to mitigate these risks, AI systems could be leveraged to fight against misleading content and harassing material.

References

[1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[2] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

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

[4] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

[5] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

[6] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. *arXiv preprint arXiv:1912.02164*, 2019.

[7] Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Neurologic decoding:(un) supervised neural text generation with predicate logic constraints. *arXiv preprint arXiv:2010.12884*, 2020.

[8] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. Modeling coverage for neural machine translation. *arXiv preprint arXiv:1601.04811*, 2016.

[9] Ziyi Yang, Chenguang Zhu, Robert Gmyr, Michael Zeng, Xuedong Huang, and Eric Darve. Ted: A pretrained unsupervised summarization model with theme modeling and denoising. *arXiv preprint arXiv:2001.00725*, 2020.

[10] Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany, August 2016. Association for Computational Linguistics.

[11] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR, 2020.

[12] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849, San Diego, California, June 2016. Association for Computational Linguistics.

[13] Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. CommonGen: A constrained text generation challenge for generative commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1823–1840, Online, November 2020. Association for Computational Linguistics.

[14] Lesly Miculicich, Dhananjay Ram, Nikolaos Pappas, and James Henderson. Document-level neural machine translation with hierarchical attention networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2947–2954, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017.

[16] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[17] Nanyun Peng, Marjan Ghazvininejad, Jonathan May, and Kevin Knight. Towards controllable story generation. In *Proceedings of the First Workshop on Storytelling*, pages 43–49, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[18] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. Automatic keyword extraction from individual documents. *Text mining: applications and theory*, 1:1–20, 2010.

[19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[20] Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. *arXiv preprint arXiv:1704.07138*, 2017.

[21] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575, 2015.

[22] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Spice: Semantic propositional image caption evaluation. In *European conference on computer vision*, pages 382–398. Springer, 2016.

[23] Ye Liu, Yao Wan, Lifang He, Hao Peng, and Philip S. Yu. Kg-bart: Knowledge graph-augmented bart for generative commonsense reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[24] Robyn Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 4444–4451. AAAI Press, 2017.

[25] Mauro Cettolo, Christian Girardi, and Marcello Federico. Wit3: Web inventory of transcribed and translated talks. In *Conference of European Association for Machine Translation*, pages 261–268, 2012.

[26] Mauro Cettolo, Niehues Jan, Stüker Sebastian, Luisa Bentivogli, Roldano Cattoni, and Marcello Federico. The iwslt 2015 evaluation campaign. In *International Workshop on Spoken Language Translation*, 2015.

[27] Jessica Ficler and Yoav Goldberg. Controlling linguistic style aspects in neural language generation. *arXiv preprint arXiv:1707.02633*, 2017.

[28] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[29] Yuta Kikuchi, Graham Neubig, Ryohei Sasano, Hiroya Takamura, and Manabu Okumura. Controlling output length in neural encoder-decoders. *arXiv preprint arXiv:1609.09552*, 2016.

[30] Alvin Chan, Yew-Soon Ong, Bill Pung, Aston Zhang, and Jie Fu. Cocon: A self-supervised approach for controlled text generation. *arXiv preprint arXiv:2006.03535*, 2020.

[31] Muhammad Khalifa, Hady Elsahar, and Marc Dymetman. A distributional approach to controlled text generation. *arXiv preprint arXiv:2012.11635*, 2020.

[32] Jiacheng Zhang, Yang Liu, Huanbo Luan, Jingfang Xu, and Maosong Sun. Prior knowledge integration for neural machine translation using posterior regularization. *arXiv preprint arXiv:1811.01100*, 2018.

[33] Christos Baziotis, Barry Haddow, and Alexandra Birch. Language model prior for low-resource neural machine translation. *arXiv preprint arXiv:2004.14928*, 2020.

[34] Jingyi Zhang, Masao Utiyama, Eiichro Sumita, Graham Neubig, and Satoshi Nakamura. Improving neural machine translation through phrase-based forced decoding. *arXiv preprint arXiv:1711.00309*, 2017.

[35] Yaohua Tang, Fandong Meng, Zhengdong Lu, Hang Li, and Philip LH Yu. Neural machine translation with external phrase memory. *arXiv preprint arXiv:1606.01792*, 2016.

[36] Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Minimum risk training for neural machine translation. *arXiv preprint arXiv:1512.02433*, 2015.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Broader Impact
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Broader Impact
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.1
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 4.1
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.1
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4.1
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] See Section 4.1
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [Yes] See Section 4.1
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [No]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [No]