Triple-to-Text Generation with an Anchor-to-Prototype Framework

Ziran Li[1,2*], **Zibo Lin**[1,2*], **Ning Ding**[1,2], **Hai-Tao Zheng**[1,2†] and **Ying Shen**[3†]
[1]Department of Computer Science and Technology, Tsinghua University
[2]Tsinghua ShenZhen International Graduate School, Tsinghua University
[3]School of Intelligent Systems Engineering, Sun Yat-Sen University
{lizr8, lzb18, dingn18}@mails.tsinghua.edu.cn, zheng.haitao@sz.tsinghua.edu.cn,
sheny76@mail.sysu.edu.cn

Abstract

Generating a textual description from a set of RDF triplets is a challenging task in natural language generation. Recent neural methods have become the mainstream for this task, which often generate sentences from scratch. However, due to the huge gap between the structured input and the unstructured output, the input triples alone are insufficient to decide an expressive and specific description. In this paper, we propose a novel anchor-to-prototype framework to bridge the gap between structured RDF triples and natural text. The model retrieves a set of prototype descriptions from the training data and extracts writing patterns from them to guide the generation process. Furthermore, to make a more precise use of the retrieved prototypes, we employ a triple anchor that aligns the input triples into groups so as to better match the prototypes. Experimental results on both English and Chinese datasets show that our method significantly outperforms the state-of-the-art baselines in terms of both automatic and manual evaluation, demonstrating the benefit of learning guidance from retrieved prototypes to facilitate triple-to-text generation.

1 Introduction

We consider the problem of automatically generating appropriate natural sentences for a given set of RDF triples, namely triple-to-text generation. Defined by the W3C standard [Miller, 2001], a Resource Description Frameworks (RDF) triple is in the form of $\langle subject, predicate, object \rangle$, which can be used to represent the relation between resources (the subject and object) in a structured knowledge base. Generating textual description from RDF triples can be applied to many scenarios for better user interaction such as question answering [Bordes *et al.*, 2014; Fader *et al.*, 2014], search engines [Ding *et al.*, 2004] and profile summarizing [Chisholm *et al.*, 2017].

Great efforts have been devoted to the sequence-to-sequence [Sutskever *et al.*, 2014] based neural models for triple-to-text generation, which often generate sentences from scratch in a left-to-right manner. [Vougiouklis *et al.*, 2018; Distiawan *et al.*, 2018; Zhu *et al.*, 2019; Koncel-Kedziorski *et al.*, 2019; Moryossef *et al.*, 2019]. However, due to the huge gap between the structured input and the unstructured output, the input triples alone are insufficient to decide and acquire an informative and specific description. As a result, these models prefer generic utterances and tend to produce repeated and irrelevant words.

To alleviate such problems, given the input triples, we can retrieve similar triples and the associated description, termed as prototype triples and prototype description, to guide triple-to-text generation. Intuitively, a human-written prototype description is informative and expressive, which can provide concrete patterns that faithfully represent the information en-

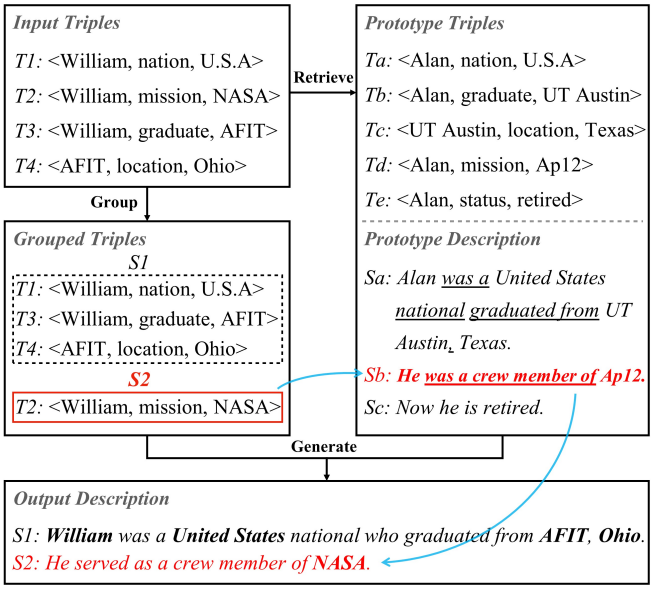

Figure 1: An example of writing a description from triples with retrieved prototypes. We manually divide each description into several sentences for easy understanding. After retrieving, we could obtain a similar triple set with a corresponding description (prototype triples and prototype description) to guide generation. After grouping, the input triples will be segmented into groups to better match the prototype. E.g., *T2* refers to *Sb*, and they will generate *S2* collaboratively.

*indicates equal contribution.
†Corresponding authors.

coded in the prototype triples. For example, to describe ⟨*William, mission, NASA*⟩, a prototype triple ⟨*Alan, mission, Apollo12*⟩ and its description "*Alan was a crew member of Apollo12*" are first retrieved from the training data. Then a model can imitate the prototype to use the pattern "*a crew member of*" to express the *mission* relation and write its own description as "*William served as a crew member of NASA*".

Despite the benefits of leveraging prototypes, as the number of triplets in the input increases, both the reference and the retrieved prototypes will become more complex and verbose. In this case, the model is unable to precisely learn writing guidance from the complicated prototypes. Consequently, the model tends to overlook the retrieved prototypes and degrades to a vanilla seq2seq model [Cai *et al.*, 2019b]. As we made statistic on the WebNLG dataset [Gardent *et al.*, 2017], samples with multiple triples accounts for over 76%. Such proportion cannot be ignored.

To this end, we propose a novel framework, anchor-to-prototype, to facilitate triple-to-text generation. Given the input triples along with the retrieved prototype triples and description, we first employ a transformer-based triple encoder to represent the input triples. Then, to extract writing patterns from the retrieved prototypes, a guidance extractor is used to extract a guidance vector based on the difference among the input triples, the prototype triples and the prototype description.

As mentioned before, with multiple triples as input, the retrieved prototypes could become too complex to be made precise use. Thus, before incorporating the guidance vector into our decoder, we additionally apply a triple anchor to align the input triples into groups to better match the retrieved prototypes. A concrete example illustrating our idea is shown in Fig 1. Finally in the decoding stage, the hidden state of the triple anchor is used to revise the guidance vector via a gate mechanism, so as to dynamically control the information flows from the prototypes to the decoder.

Our experiments are conducted on both English and Chinese datasets towards triple-to-text generation. Experimental results show that our model outperforms state-of-the-art approaches in automatic metrics as well as human evaluation. In addition, we also conduct extensive ablation studies to quantify the improvement from each designed component of our model.

2 Related Work

Recently, neural models have become the mainstream for triple-to-text generation. As a pioneer, Vougiouklis *et al.* [2018] first employ a feed-forward neural network to encode the input triples and add them into a LSTM decoder to generate words. Based on Vougiouklis's work, various techniques have been developed to make further improvement, including encoding relationships among triples [Distiawan *et al.*, 2018; Koncel-Kedziorski *et al.*, 2019; Cai and Lam, 2020], modifying training objective [Zhu *et al.*, 2019] and explicitly modeling text planning [Moryossef *et al.*, 2019]. The main drawback of these models is that they often generate sentences from scratch, while the input triples alone are insufficient to decide an expressive and specific description.

Several studies design templates (prototypes) to guide the text generation from triples [Kukich, 1983; Duma and Klein, 2013; Wiseman *et al.*, 2017], but they mostly use either fixed or hard templates, which are limited in scalability and flexibility. By contrast, our model is guided by retrieved prototypes that served as soft templates. Additionally, the idea of utilizing retrieved prototypes has also been exploited in other generation task such as unconditional text generation [Guu *et al.*, 2018], dialogue response generation [Wu *et al.*, 2019; Cai *et al.*, 2019a; Cai *et al.*, 2019b] and abstractive summarization [Cao *et al.*, 2018; Wang *et al.*, 2019]. Different from these methods, to precisely learn writing guidance from complicated prototypes, our framework models a grouping stage to control the information flows from the prototypes.

3 Models

The input to our model is a set of RDF triples $x = \{t_1, t_2, ..., t_N\}$, where $t_i = \langle s_i, p_i, o_i \rangle$ is the i-th triple in x (elements s_i, p_i and o_i correspond to subject, predicate and object respectively). Each element can contain multiple words. While the output $y = y_1 y_2 ... y_M$ (y_j is the j-th word) is a paragraph that faithfully represents the information encoded in x.

To facilitate the generation from x to y, a similar set of triples x' and its associated description y' are retrieved from the training data, which are called prototype triples and prototype description respectively. Then the goal of our model is to maximize $p(y|x, x', y')$.

As shown in Fig 2, our model consists of three main components: (a) **triple encoder**, which encodes the input triples into vector representation via a set of multi-head attention layers. (b) **guidance extractor**, which learns a guidance vector from the retrieved prototype to guide the generation. (c) **triple anchor**, which groups the input triples to match the prototype so that the decoder can make a better use of the learnt guidance.

3.1 Triple Encoder

The triple encoder converts the input triples x into embedding vectors and then encodes the vectors via multi-head attention layers. We first flatten the input triples $x = \{t_1, t_2, ..., t_N\}$ into a sequence of words $\bar{x} = \{w_{1,1}, w_{1,2}, ..., w_{1,|t_1|}, ..., w_{N,|t_N|}\}$, where $w_{i,j}$ is the j-th word in the i-th triple. To facilitate input representation, a special token $[\texttt{CLS}]$ is added into the beginning of $\bar{x}$.

For each word in the input, its input embedding is the sum of its word and position embeddings. Unlike the conventional position embedding that only records word indices in the input sequence, we employ local and global position to encode both the intra and inter positional information among triples.

Specifically, for a word $w_{i,j}$ in $\bar{x}$, its local position $p'_{i,j} = j$ is the word index within the associated triple while the global position $p_{i,j} = i$ corresponds to the triple index in the input. Then we embed the word, local position and global position into vector representations (denoted as $\bar{\boldsymbol{x}}$, $\boldsymbol{p'}$ and $\boldsymbol{p}$) and computed the embedding vectors of x as the summation of them:

$$\boldsymbol{x} = \bar{\boldsymbol{x}} + \boldsymbol{p'} + \boldsymbol{p}. \quad (1)$$

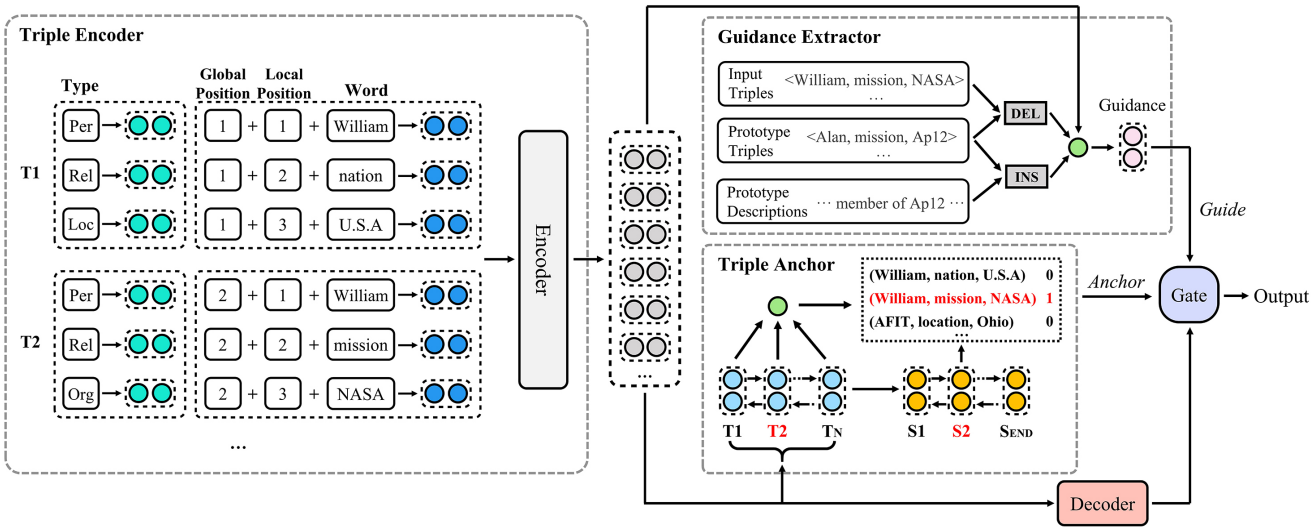

Figure 2: The overall architecture of the anchor-to-prototype framework.

Besides, the entity type information of each element in the input is also considered. We obtain a sequence of type labels $x_{type} = \{l_{1,1}, ..., l_{N,|t_N|}\}$ provided by the datasets and map x_{type} into embedding vectors $\boldsymbol{x}_{type}$, which will be utilized to enhance representation in the encoder layers. The input embedding $\boldsymbol{x}$ and type embedding $\boldsymbol{x}_{type}$ share the same number of dimensions d^k and both are optimized as parameters during the training process.

After obtaining the embedding vectors of the input, we feed the vectors into our triple encoder. The encoder is based on transformer [Vaswani *et al.*, 2017], which consists of a stack of identical multi-head attention layers. Each layer takes queries $\boldsymbol{Q}$, keys $\boldsymbol{K}$, and values $\boldsymbol{V}$ as input and is formulated as:

$$\text{MultiAttn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = [\boldsymbol{h}_1; ...; \boldsymbol{h}_H]\boldsymbol{W}, \tag{2}$$

$$\boldsymbol{h}_i = \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V), \tag{3}$$

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d^k}}\right)\boldsymbol{V}, \tag{4}$$

where $\boldsymbol{W}$, $\boldsymbol{W}_i^Q$, $\boldsymbol{W}_i^K$ and $\boldsymbol{W}_i^V$ are trainable parameters, $[;]$ represents the concatenation operation and H is the number of attention heads. A fully connected feed-forward network with residual connection and a layer normalization are then added behind the MultiAttn to form an encoder layer $\text{Block}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V})$.

Finally, based on the above calculating process, we stack L layers to encode the input from both the input embedding $\boldsymbol{x}$ and type embedding $\boldsymbol{x}_{type}$, which can be shorten as:

$$\boldsymbol{z}^L = \begin{cases} \boldsymbol{z}^i = \text{Block}_i(\boldsymbol{z}^{i-1}, \boldsymbol{z}^{i-1}, \boldsymbol{z}^{i-1}) & i \geq 2 \\ \boldsymbol{z}^1 = \text{Block}_1(\boldsymbol{x}, \boldsymbol{x}_{type}, \boldsymbol{x}_{type}) & i = 1 \end{cases}, \tag{5}$$

where $\boldsymbol{z}^L$ is the final representation of the input x, which will be fed into the following modules of our model.

3.2 Guidance Extractor

The guidance extractor aims to extract a guidance vector based on the difference among the input triples x, the prototype triples x' and the prototype description y'. According to [Wu *et al.*, 2019], the extracted vector should absorb appropriate content from the prototype while discarding irrelevant words regarding the current input.

Following this idea, to construct the guidance vector, we first define two word sets: insertion words $I = \{w | w \in y' \cap w \notin x'\}$ and deletion words $D = \{w | w \in x' \cap w \notin x\}$. The insertion words I consider words in the writing patterns derived from the prototype description y', while the deletion words D correspond to words that only specific to the prototype triples x'. Then, we concatenate the representation of the two word sets to form a textual difference vector $\boldsymbol{g}^*$:

$$\boldsymbol{g}^* = \sum_{w_I \in I} \alpha_{w_I} \boldsymbol{e}(w_I) \oplus \sum_{w_D \in D} \beta_{w_D} \boldsymbol{e}(w_D), \tag{6}$$

where $\boldsymbol{e}(\cdot)$ converts a word into its word embedding. The value α_{w_I} is the weight of a word w_I in the insertion words, which is derived by an attention mechanism with $\boldsymbol{v}_I$ and $\boldsymbol{W}_I$ as parameters:

$$\alpha_{w_I} = \frac{\exp(s_{w_I})}{\sum_{w \in I} \exp(s_w)}, \tag{7}$$

$$s_{w_I} = \boldsymbol{v}_I^\top \tanh(\boldsymbol{W}_I[\boldsymbol{e}(w_I); \boldsymbol{z}_0^L]). \tag{8}$$

Here, $\boldsymbol{z}_0^L$ is the hidden state of the [CLS] token in Eq. 5, which can be regarded as a sentence representation of the input x. In addition, β_{w_D}, the weight of a deletion word w_D in D, can be obtained with a similar process:

$$\beta_{w_D} = \frac{\exp(s_{w_D})}{\sum_{w \in D} \exp(s_w)}, \tag{9}$$

$$s_{w_D} = \boldsymbol{v}_D^\top \tanh(\boldsymbol{W}_D[\boldsymbol{e}(w_D); \boldsymbol{z}_0^L]). \tag{10}$$

Finally, we use a dense layer with parameters $\boldsymbol{W}_G$ and $\boldsymbol{b}_G$ to transform $\boldsymbol{g}^*$ to the guidance vector $\boldsymbol{g}$:

$$\boldsymbol{g} = \tanh(\boldsymbol{W}_G \cdot \boldsymbol{g}^* + \boldsymbol{b}_G), \tag{11}$$

then the guidance vector $\boldsymbol{g}$ will be fed into the sentence decoder.

3.3 Triple Anchor

As discussed in Section 1, the retrieved prototypes could be too complex for the decoder. Thus, before moving on to the decoding stage, the triple anchor splits input triples into groups to align the prototypes, where a group corresponds to a sentence in the description. Then the decoder can make a more precise use of the extracted guidance vector.

Inspired by the content selection applied in [Hua and Wang, 2019; Shao *et al.*, 2019], we extend the Memory Network [Sukhbaatar *et al.*, 2015] to model the grouping of triples. The whole process can be regarded as selecting a subset of the input triples to be covered for the current group in each time step.

Given the representation of the input triples $\boldsymbol{z}^L$ from Eq. 5, we first sum up all the elements in the same triple in $\boldsymbol{z}^L$ to obtain triple representations $T = \{\boldsymbol{t}_1, \boldsymbol{t}_2, ..., \boldsymbol{t}_N\}$, where $\boldsymbol{t}_i = \sum_j \boldsymbol{z}^L_{i,j}$ is the representation of the i-th triple. To control the start and end of the selection process, two special entries [START] and [END] are also added into T. Then, we transform T to hidden vectors $\{\boldsymbol{h}^e_i | \boldsymbol{h}^e_i = [\overrightarrow{\boldsymbol{h}}^e_i; \overleftarrow{\boldsymbol{h}}^e_i]\}_{i=1}^{N+2}$ through a biLSTM, where $\boldsymbol{h}^e_i$ is the hidden state of the entry $\boldsymbol{t}_i$.

After obtaining the hidden state of each entry in T, we use a LSTM decoder to make selection of triples. Starting with the first group that only contains the [START] entry, the decoder recurrently selects triples for the next group until the [END] entry is selected. Formally, at time step t, the hidden state of the decoder $\boldsymbol{h}^d_t$ is computed as:

$$\boldsymbol{h}^d_t = \mathrm{LSTM}(\boldsymbol{h}^d_{t-1}, \boldsymbol{s}_t), \tag{12}$$

$$\boldsymbol{s}_t = \sum_{i=1}^{N+2} sel_{t,i} \boldsymbol{h}^e_i, \tag{13}$$

where $sel_{t,i} \in \{0, 1\}$ is the selection for the i-th entry in the t-th group, and $\boldsymbol{s}_t$ is the summation of all the currently selected entries. Then, the hidden state $\boldsymbol{h}^d_t$ along with all the previously selected entries $\boldsymbol{p}_t$ are used to calculated the selection for the next group, which is treated as a binary prediction for each input entry:

$$P(sel_{t+1,i} = 1) = \sigma(\boldsymbol{v}_A^\top \tanh(\boldsymbol{W}_A[\boldsymbol{h}^d_t; \boldsymbol{h}^e_i; \boldsymbol{p}_t])), \tag{14}$$

$$\boldsymbol{p}_t = \sum_{\tau=1}^{t} \boldsymbol{s}_\tau, \tag{15}$$

where σ is the sigmoid function. $\boldsymbol{p}_t$ records entries that have been selected before, which can prevent the model from repeatedly picking the same triples.

3.4 Sentence Decoder

Our sentence decoder basically follows the same spirit of the Transformer decoder, in which L decoder layers are stacked. Each decoder layer has the similar structure with the block in the encoder, while with an addition of masked multi-attention layer before the feed-forward layer.

At time step t, the decoder is fed with the input representation $\boldsymbol{z}^L$ learnt from the encoder and the embedding of the predicted output $\boldsymbol{y}_{<t}$, and outcomes a sequence of hidden states $\boldsymbol{y}^L$. Then the probability of generating the next target word y_t is jointly decided by the t-th hidden state $\boldsymbol{y}^L_t$, the guidance vector $\boldsymbol{g}$ and the triple anchor's state $\boldsymbol{h}^d$. Specifically, we first calculate the weighted sum of $\boldsymbol{h}^d$ to form an anchor representation $\boldsymbol{c}$ based on the guidance vector $\boldsymbol{g}$, which is given as:

$$\boldsymbol{c} = \sum\nolimits_i a^*_i \boldsymbol{h}^d_i, \tag{16}$$

$$a^*_i = \mathrm{softmax}(\boldsymbol{g} \boldsymbol{W}_T \boldsymbol{h}^d_i). \tag{17}$$

Then we use the anchor representation $\boldsymbol{c}$ to calculate a gate probability that determines how much information of the guidance vector should flow into the output sequence:

$$g_a = \sigma(\boldsymbol{W}_H[\boldsymbol{c}; \boldsymbol{g}; \boldsymbol{y}^L_t] + \boldsymbol{b}_H), \tag{18}$$

$$\boldsymbol{y}^*_t = g_a * \boldsymbol{g} + (1 - g_a) * \boldsymbol{y}^L_t, \tag{19}$$

where $\boldsymbol{W}_H$ and $\boldsymbol{b}_H$ are trainable parameters. Finally, the output probability of the next word y_t is estimated by $\boldsymbol{y}^*_t$ via a softmax function over the target vocabulary.

3.5 Training Objective

There are two optimization objectives in our work:

1) $loss_1$ is the cross entropy for sequence generation between the predicted token distribution and the reference distribution.
2) $loss_2$ is the loss function of the triple anchor which utilizes the binary cross-entropy loss with gold-standard selection as criterion over the training set. The gold-standard selection is manually annotated so that the grouping of triples can be trained with supervision.

The total loss training objective is given by:

$$L = \gamma * loss_1 + (1 - \gamma) * loss_2, \tag{20}$$

where γ is a hyper-parameter for trade-off between $loss_1$ and $loss_2$ and controls the weight of the explicit supervision.

4 Experiments

4.1 Datasets

We conduct experiments on two benchmark datasets: WebNLG [Gardent *et al.*, 2017] and Baidu SKE [1]. The WebNLG dataset is driven from DBPedia and contains 25,298 triple-text pairs, with 18,102 samples for training, 2,268 for validating and 4,928 for testing. The Baidu SKE

[1] http://ai.baidu.com/broad/introduction

Model	WebNLG				SKE			
	BLEU	Corr.	Expr.	Gram.	BLEU	Corr.	Expr.	Gram.
NMT [Bahdanau *et al.*, 2015]	39.3	2.82	3.36	3.44	19.6	2.12	2.56	2.57
Transformer [Vaswani *et al.*, 2017]	45.6	3.22	3.75	3.94	25.0	2.16	3.52	3.5
GTR-LSTM [Distiawan *et al.*, 2018]	42.1	3.20	3.77	3.93	23.4	2.72	3.36	3.38
GraphWriter [Knocel *et al.*, 2019]	34.1	1.48	2.42	2.36	19.1	1.48	2.42	2.36
Step-by-Step [Moryossef *et al.*, 2019]	47.4	3.35	3.78	3.84	-	-	-	-
EditVec [Wu *et al.*, 2019]	42.6	2.60	3.09	3.23	21.1	3.17	3.28	3.22
Anchor-to-Prototype (Ours)	**49.9**	**3.52**	**3.88**	**4.05**	**29.8**	**3.18**	**3.78**	**3.74**

Table 1: Result of different methods on WebNLG and Baidu SKE on both automatic metrics and human evaluation. The human evalution includes Corr. (Correctness), Expr.(Expression) and Gram. (Grammar).

is a large-scale human annotated dataset based on real-world Chinese corpora. We follow Zhu *et al.* [2019] who only used samples related to the film domain. The number of samples for training, validating and testing are 28,597, 3,177 and 3,972 respectively.

4.2 Implementation Details

We implement the triple anchor based on a bidirectional recurrent neural network with 512 LSTM units. The trade-off γ between $loss_1$ and $loss_2$ is 0.9. We use the Adam optimizer and the learning rate of it is 1e-5. For the encoder and decoder, the number of multi-head attention layer L is 4 and the rest parameters are followed by the setting of the basic Transformer [Vaswani *et al.*, 2017].

For better comparison, we use the similar strategy as Wu *et al.* [2019] to retrieve prototypes for each sample. Specifically, we first employ Lucene [2] to construct a pre-defined index based on description similarity[3]. For each training pair (x, y), top 20 similar descriptions with their associated triples are retrieved, denoted as $\{(x'_i, y'_i)\}_{i=1}^{20}$. Furthermore, to avoid the input and the prototype being too similar or too different, only the pairs that satisfy $0.25 \leqslant Jaccard(y, y'_i) \leqslant 0.75$ can be reserved, where $Jaccard$ measures the Jaccard similarity. If no prototype is found or no pair is reserved, a special prototype [DEFAULT] will be provided.

4.3 Compared Methods

We compare our model with several approaches:

NMT [Bahdanau *et al.*, 2015] is the standard attention-based Seq2Seq baseline.

Transformer [Vaswani *et al.*, 2017] puts forward an encoder-decoder model based on multi-head attention layers.

GTR-LSTM [Distiawan *et al.*, 2018] improves the LSTM encoder by considering relationships between the triples.

Step-by-Step [Moryossef *et al.*, 2019] explicitly models content selection and order planning for triple-to-text generation by manually designing features.

GraphWriter [Koncel-Kedziorski *et al.*, 2019] is a transformer model with graph encoding mechanism to encode both the entities and relations from the input.

EditVec [Wu *et al.*, 2019] is a prototype-based model proposed for response generation.

[2]https://lucene.apache.org/core/

[3]For test data, the pre-defined index is based on triples similarity

4.4 Evaluation Metrics

We calculate scores on both automatic and manual evaluation. For automatic evaluation, we adopt BLEU [Papineni *et al.*, 2002]. For human evaluation, we evaluate the quality of a generated description on three criteria: *Grammar* (whether the output is Grammatically correct), *Expression* (whether the output is fluent and uses appropriate words to express) and *Correctness* (whether the output correctly represents the information in the input). The score for each criterion takes an integer from 0 to 5 (the higher the better). We randomly select 200 sets of input triples in test set from both the WebNLG and Baidu SKE datasets, along with corresponding descriptions produced by our model and other baselines. Five well-educated graduate student are asked to score the descriptions. Notice that method identifiers are masked during the human evaluation.

4.5 Overall Results

The experimental results on WebNLG and Baidu SKE are summarized in Table 1. There are multiple observations as follows:

(1) Generally, our model substantially and consistently outperforms the existing methods by a noticeable margin on both automatic and human evaluation metrics. The main strength of our model comes from its capability of leveraging anchor to prototype that are complementary to each other for triple-to-text generation.

(2) Our model outperforms prototype-based model (EditVec) in term of Expression metric, verifying the benefits of incorporating triple anchor to better match the prototypes. The advantage of Step-by-Step is that it employs manual design rules to make use of the standard Seq2Seq framework. Nevertheless, the performance of our model performs better with less labor intensity and time costs.

(3) The performance of baseline models in terms of Grammar metric cannot perform as well as our model, which shows that our model can produce descriptions close to human writing. With the combined exploration of guidance extractor and triple anchor, our model effectively bridges the gap between structured input and unstructured output, and overcomes the issue of complex and verbose prototype, thereby providing fluent and informative description.

Input	**a:** <Buzz Aldrin, birth date, 1930 01 20> **b:** <Buzz Aldrin, status, Retired> **c:** <Buzz Aldrin, mission, Apollo11> **d:** <Apollo11, backup pilot, William Anders> **e:** < Apollo11, operator, NASA>
Reference	Buzz Aldrin was born on the 20th October 1930. He was a crew member of Apollo 11, but is now Retired. William Anders was a backup pilot on the Apollo 11 mission, which was operated by NASA.
Step-by-Step	Buzz Aldrin was born on the 20th of January, 1930. Buzz Aldrin has retired. Buzz Aldrin was a member on Apollo 11 operated by NASA and the backup pilot William Anders.
EditVec	Buzz Aldrin was born on January 20th, 1930, he performed as a crew member in Apollo 11 which is now retired before retiring by the backup pilot and William Anders of his credit dimitia mission there.
Ours	Buzz Aldrin was born on Jan 20, 1930. He was a member of Apollo 11's crew operated by NASA, where William Anders was a backup pilot. Aldrin is now retired.
Prototype Triples	**I:** <William Anders, occupation, fighter pilot> **II:** <William Anders, mission, Apollo8> **III:** <Apollo8, operator, NASA> **IV:** <Apollo8, crew, Frank Borman> **V:** <Apollo8, backup pilot, Buzz Aldrin> **VI:** <William Anders, status, Retired>
Prototype Description	William Anders served as a Fighter pilot and was a member of Apollo 8's crew. He is now Retired. Frank Borman was also a crew member of the NASA operated Apollo 8. Buzz Aldrin was a back up pilot for Apollo 8.

Figure 3: An example with multiple triples from WebNLG, as well as generated results from different models and retrieved prototype in our model. Writing patterns learnt from the prototype are in colored background.

4.6 Ablation Study

In this section, we conduct ablation study to systematically evaluate the effect of our guidance extractor and triple anchor, which is reported in Table 2. In order to perform more comprehensive evaluation, we split the test set based on the number of input triples: single triple and multiple triples. In addition to the BLEU score, we additionally introduce two extra metrics: (1) Coverage (Cov.) measures the overlap of words (entities and relations) between input triples and output descriptions. (2) Originality (Ori.) calculates the proportion of words that are not in the input but both in the output and references.

The results from Table 2 show that equipped with both the guidance extractor and triple anchor, the complete version achieves almost the best performance on both data splits. In the case of input with single triple, the coverage and originality drop without the guidance extractor, which suggests that the guidance extractor could maintain the provided words from the input and create new words to add the diversity. However, since the input is too simple, the triple anchor has little influence in this situation.

In the situation of multiple triples with more information, the coverage and originality drop dramatically in comparison with the single one. In this case, although the guidance extractor still have impact, it has become smaller than the single input case. Meanwhile, thanks to the triple anchor which groups triples to better match the prototypes, the model with the triple anchor performs better in terms of all metrics.

4.7 Case Study

An example with multiple triples in WebNLG along with generated results from different models are shown in Fig 3, from which we can observe that: although the Step-by-Step outputs acceptable description, there are some grammatical errors and the sentences appear mechanically and not fluently. Due to the complexity of the input, the EditVec mixes all the triples into a single sentence with logical confusion ("*is now retired before retiring*") and groundless information ("*credit dimitia mission*"). By contrast, our model precisely learns writing patterns (e.g., "*was a member of _'s crew*") from the retrieved prototype to describe the input triples and the generated result is fluent, expressive and faithful to the input.

Model	Single			Multiple		
	BLEU	Cov.	Ori.	BLEU	Cov.	Ori.
Ours	**55.6**	**37.8**	**35.0**	**56.8**	34.8	**33.0**
- anchor	55.2	**37.8**	34.8	55.8	**34.9**	32.3
- guide	55.0	37.2	34.6	56.1	34.5	32.5
- both	54.2	34.8	34.7	51.5	28.8	29.6

Table 2: Ablation study of our model on WebNLG datasets. "Single" and "Multiple" correspond to data splits with single triples input and multiple triples input respectively

5 Conclusion

In this paper, we present a novel framework, Anchor-to-Prototype, to facilitate triple-to-text generation. To bridge the information gap between the structured input and a natural text, the model retrieves sets of human-written descriptions as prototypes from the training data to guide the text generation from the input triples. What's more, we employ a triple anchor that aligns the input triples into groups to better match the prototypes, so that the model can make a more precise use of the retrieved prototypes.

Acknowledgments

This research is supported by National Natural Science Foundation of China (Grant No. 61773229 and 61972219), Shenzhen Giiso Information Technology Co. Ltd., the Basic Research Fund of Shenzhen City (Grand No. JCYJ20190813165003837), Tencent AI Lab Rhino-Bird Focused Research Program (No. JR202032), Overseas Cooperation Research Fund of Graduate School at Shenzhen, Tsinghua University (Grant No. HW2018002) and the Shenzhen General Research Project (No. JCYJ20190808182805919).

References

[Bahdanau *et al.*, 2015] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[Bordes *et al.*, 2014] Antoine Bordes, Sumit Chopra, and Jason Weston. Question answering with subgraph embeddings. In *EMNLP*, pages 615–620, 2014.

[Cai and Lam, 2020] Deng Cai and Wai Lam. Graph transformer for graph-to-sequence learning. In *AAAI*, 2020.

[Cai *et al.*, 2019a] Deng Cai, Yan Wang, Wei Bi, Zhaopeng Tu, Xiaojiang Liu, Wai Lam, and Shuming Shi. Skeleton-to-response: Dialogue generation guided by retrieval memory. In *NAACL-HLT*, pages 1219–1228, 2019.

[Cai *et al.*, 2019b] Deng Cai, Yan Wang, Wei Bi, Zhaopeng Tu, Xiaojiang Liu, and Shuming Shi. Retrieval-guided dialogue response generation via a matching-to-generation framework. In *EMNLP-IJCNLP*, pages 1866–1875, 2019.

[Cao *et al.*, 2018] Ziqiang Cao, Wenjie Li, Sujian Li, and Furu Wei. Retrieve, rerank and rewrite: Soft template based neural summarization. In *ACL*, pages 152–161, 2018.

[Chisholm *et al.*, 2017] Andrew Chisholm, Will Radford, and Ben Hachey. Learning to generate one-sentence biographies from wikidata. In *EACL*, pages 633–642, 2017.

[Ding *et al.*, 2004] Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R Scott Cost, Yun Peng, Pavan Reddivari, Vishal Doshi, and Joel Sachs. Swoogle: a search and metadata engine for the semantic web. In *CIKM*, pages 652–659, 2004.

[Distiawan *et al.*, 2018] Bayu Distiawan, Jianzhong Qi, Rui Zhang, and Wei Wang. Gtr-lstm: A triple encoder for sentence generation from rdf data. In *ACL*, pages 1627–1637, 2018.

[Duma and Klein, 2013] Daniel Duma and Ewan Klein. Generating natural language from linked data: Unsupervised template extraction. In *IWCS 2013*, pages 83–94, 2013.

[Fader *et al.*, 2014] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. Open question answering over curated and extracted knowledge bases. In *KDD*, pages 1156–1165, 2014.

[Gardent *et al.*, 2017] Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. The webnlg challenge: Generating text from rdf data. In *ICNLG*, pages 124–133, 2017.

[Guu *et al.*, 2018] Kelvin Guu, Tatsunori B Hashimoto, Yonatan Oren, and Percy Liang. Generating sentences by editing prototypes. *Transactions of the Association for Computational Linguistics*, 6:437–450, 2018.

[Hua and Wang, 2019] Xinyu Hua and Lu Wang. Sentence-level content planning and style specification for neural text generation. In *EMNLP-IJCNLP*, pages 591–602, 2019.

[Koncel-Kedziorski *et al.*, 2019] Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. Text generation from knowledge graphs with graph transformers. In *NAACL*, pages 2284–2293, 2019.

[Kukich, 1983] Karen Kukich. Design of a knowledge-based report generator. In *ACL*, pages 145–150, 1983.

[Miller, 2001] Eric Miller. An introduction to the resource description framework. *Journal of library administration*, 34(3-4):245–255, 2001.

[Moryossef *et al.*, 2019] Amit Moryossef, Yoav Goldberg, and Ido Dagan. Step-by-step: Separating planning from realization in neural data-to-text generation. In *NAACL*, pages 2267–2277, 2019.

[Papineni *et al.*, 2002] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318, 2002.

[Shao *et al.*, 2019] Zhihong Shao, Minlie Huang, Jiangtao Wen, Wenfei Xu, and Xiaoyan Zhu. Long and diverse text generation with planning-based hierarchical variational model. In *EMNLP-IJCNLP*, pages 3255–3266, 2019.

[Sukhbaatar *et al.*, 2015] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In *NIPS*, pages 2440–2448, 2015.

[Sutskever *et al.*, 2014] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

[Vougiouklis *et al.*, 2018] Pavlos Vougiouklis, Hady Elsahar, Lucie-Aimée Kaffee, Christophe Gravier, Frederique Laforest, Jonathon Hare, and Elena Simperl. Neural wikipedian: Generating textual summaries from knowledge base triples. *Journal of Web Semantics*, 52:1–15, 2018.

[Wang *et al.*, 2019] Kai Wang, Xiaojun Quan, and Rui Wang. Biset: Bi-directional selective encoding with template for abstractive summarization. In *ACL*, pages 2153–2162, 2019.

[Wiseman *et al.*, 2017] Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. Challenges in data-to-document generation. In *EMNLP*, pages 2253–2263, 2017.

[Wu *et al.*, 2019] Yu Wu, Furu Wei, Shaohan Huang, Yunli Wang, Zhoujun Li, and Ming Zhou. Response generation by context-aware prototype editing. In *AAAI*, volume 33, pages 7281–7288, 2019.

[Zhu *et al.*, 2019] Yaoming Zhu, Juncheng Wan, Zhiming Zhou, Liheng Chen, Lin Qiu, Weinan Zhang, Xin Jiang, and Yong Yu. Triple-to-text: Converting RDF triples into high-quality natural languages via optimizing an inverse KL divergence. In *SIGIR*, pages 455–464, 2019.