TransCP: A Transformer Pointer Network for Generic Entity Description Generation with Explicit Content-Planning

Bayu Distiawan Trisedya, Jianzhong Qi, Haitao Zheng, Flora D. Salim, and Rui Zhang

Abstract— We study neural data-to-text generation to generate a sentence to describe a target entity based on its attributes. Specifically, we address two problems of the encoder-decoder framework for data-to-text generation: i) how to encode a non-linear input (e.g., a set of attributes); and ii) how to order the attributes in the generated description. Existing studies focus on the encoding problem but do not address the ordering problem, i.e., they learn the content-planning implicitly. The other approaches focus on two-stage models but overlook the encoding problem. To address the two problems at once, we propose a model named **TransCP** to explicitly learn content-planning and integrate them into a description generation model in an end-to-end fashion. We propose a novel Transformer-based Pointer Network with *gated residual attention* and *importance masking* to learn a content-plan. To integrate the content-plan with a description generator, we propose a tracking mechanism to trace the extent to which the content-plan is exposed in the previous decoding time-step. This helps the description generator select the attributes to be mentioned in proper order. Experimental results show that our model consistently outperforms state-of-the-art baselines by up to 2% and 3% in terms of BLEU score on two real-world datasets.

Index Terms—Knowledge base, natural language generation, entity description, content planning.

1 Introduction

Natural language generation is an important and challenging task. In this paper, we study *entity description generation* from structured data (i.e., data-to-text generation), which is essential for various applications such as question answering [1], summarization [2], and knowledge graph enrichment [3], [4]. Specifically, we aim to generate a description from a set of attributes $\mathcal{A}$ of a target entity; the attributes are in the form of pairs of key and value, i.e., $\mathcal{A} = \{\langle k_1; v_1\rangle, \langle k_2; v_2\rangle, \ldots, \langle k_n; v_n\rangle\}$, where k_n is the key of the attribute and v_n is the value of the attribute.

Table 1 illustrates the input and output of the task. In this example, the attributes are `name`, `birth_place`, etc., and their values are `"Keanu Reeves"`, `"Beirut, Lebanon"`, etc. Here, the attributes may have been extracted from a table, which makes the task *table-to-text generation* [2], [5], or a knowledge graph (KG), which makes the task *RDF-to-text generation* [6], [7]. In table-to-text generation, the attributes are extracted from a two-column table (e.g., Wikipedia infobox) where the first column indicates the key and the second column indicates the value of the attributes. In RDF-to-text generation, the attributes are extracted by querying a KG for RDF triples (i.e., $\langle$`subject, predicate, object`$\rangle$) that contain the target entity as the subject. In both cases, the input will form a star-shaped graph with the target entity as the center of the graph, the attribute values as the points of the star, and the attribute keys as the edges.

TABLE 1: Data-to-text Generation.

Input	⟨name; Keanu Reeves⟩ ⟨birth_place; Beirut, Lebanon⟩ ⟨occupation; actor⟩ ⟨occupation; musician⟩ ⟨birth_date; September 2, 1964⟩ ⟨residence; California, U.S.⟩ ⟨birth_name; Keanu Charles Reeves⟩ ⟨citizenship; Canada⟩ ⟨citizenship; United States⟩
Output	"Keanu Charles Reeves (born September 2, 1964, in Beirut, Lebanon) is an American actor who lives in California, U.S."

Recent studies propose end-to-end models by adapting the encoder-decoder framework. The encoder-decoder framework is a sequence-to-sequence model that has been successfully used in many tasks, including machine translation [8], text segmentation [9], sequence labelling [10], and entity and relation extraction [11], [12]. There are two problems to address in such a framework for description generation from a set of attributes:

1. Encoding problem: How to capture the relationships between the attributes in the input. The encoder in the framework aims to compute the representation of the input by computing an embedding (i.e., the vector representation)

- *B. D. Trisedya is with Universitas Indonesia. E-mail: bayudt@gmail.com*
- *J. Qi is with The University of Melbourne. E-mail: jianzhong.qi@unimelb.edu.au*
- *H. Zheng is with Tsinghua University. E-mail: zheng.haitao@sz.tsinghua.edu.cn*
- *F. D. Salim is with University of New South Wales (UNSW) Sydney. E-mail: flora.salim@unsw.edu.au*
- *R. Zhang (www.ruizhang.info) E-mail: rayteam@yeah.net*

Manuscript received XXXXXX XX, XXXX; revised XXXXXX XX, XXXX.

for each attribute. To compute the embeddings, the encoder needs to capture the relationships between attributes, which is challenging. This is because, as mentioned above, the set of attributes and the target entity form a star-shaped graph. There is no link between the attributes in the graph except through the center node (i.e., the target entity). Capturing the relationships between the attributes in this kind of structure is non-trivial.

2. Ordering problem: How to decide the order of attributes in the generated description. This problem is closely related to two typical issues in neural text generation models: i) *repetition* (i.e., the same attribute is mentioned twice or more in the generated description); and ii) *incoherent text* (i.e., close attributes, e.g., `birth_date` and `birth_place`, are mentioned far away) [6], [13]. In traditional text generation models, these issues are handled by exploiting a *content-plan*. A content-plan is a reasonable order of attributes in a well-organized sentence. For example, ⟨`birth_name`, `birth_date`, `birth_place`, `occupation`, `residence`⟩ is a content-plan for the output sentence in Table 1. Exploiting content-plan in a neural data-to-text generation model is understudied, particularly in addressing the challenges to integrate content-planning into an end-to-end text generation model.

Existing studies have not well addressed the above problems. Early studies of neural data-to-text generation [5], [14]–[16] linearize the input set of triples and use LSTM [17] to encode the input, which is sub-optimal since there may not be any sequential (linear) relationships between attributes in a set. Other studies focus on developing a graph-based encoder. For example, GraphWriter [18] uses Graph Attention Networks [19] and Transformer [20] to capture the relationships between attributes. These models do not exploit any content-planning mechanism, making them prone to repetition and incoherent text errors. Meanwhile, recent studies that exploit content-planning in description generation, such as those by Trisedya et al. [6], [21], are prone to producing an overly generalized content-plan, because their method learns the content-plan from the typical entity orders in a corpus (e.g., Wikipedia articles) that may miss the local context for different input graph.

Content-planning is also exploited by Pudupully et al. [22] and Chen et al. [23], where two-stage models are used instead of end-to-end ones. However, the two-stage models are prone to error propagation between the stages and may suffer from the **attribute-missing error**: the generated content-plan may miss one or more attributes that should be mentioned in the description. Tree-PLAN [24] uses a Hierarchical Attention Pointer Network [25] to generate a content-plan and uses a Tree-like encoding mechanism to combine close attributes. To address the attribute-missing error, Tree-PLAN appends the unselected attributes to the end of the generated content-plan. This strategy is sub-optimal since it spoils the content-plan.

We address the problems above by proposing **TransCP**, a joint learning (and end-to-end) model with explicit content-planning for description generation. To address the encoding problem, we use a Transformer-based attribute encoder. To address the ordering problem, we integrate content-planning into description generation. We first propose a novel Transformer-based Pointer Network with *gated residual attention* and *importance masking* to learn a content-plan given a set of attributes. The gated residual attention aims to exploit all the attention (all the multi-head attention in all encoder layers) of the Transformer model and use such attention signals as the Pointer Network. The importance masking helps the Pointer Network avoid generating repetitive attributes by providing a signal on which attributes have and have not been selected in previous time steps of the content-plan generation process. The attributes selected previously are given lower importance scores, while the unselected ones are given higher scores. We further propose a *content-plan tracking* mechanism to integrate the learned content-plan into the description generator. This tracking mechanism helps the Transformer-based decoder capture the most salient attribute at each time-step of the description generation phase in a proper order. It is achieved by using a cross-attention layer between the learned content-plan and the decoder state, which can track the attributes in a content-plan that have been exploited in the previous decoder states.

Our proposed Transformer-based Pointer Network model differs from the existing Hierarchical Attention Pointer Network (HAN) [25] in two aspects. First, HAN uses the decoder state to predict the pointer (i.e., the input order), while our model follows the original Pointer Network [26] and uses the attention weights to decide the pointer. The attention weights are more appropriate for computing the pointer since they provide supervision signals for the decoder to learn the more important parts of the input. In comparison, the decoder state (used by HAN) is a combination of input representations weighted by the attention, which contains noises when being used to predict the pointer. Second, our model integrates all attention (the multi-head attention in multiple layers) in the decoder with our proposed gated residual attention. In contrast, HAN only uses the signal from the last layer of the decoder to make a prediction, which may not exploit all attentions.

Our contributions are summarized as follows:

- **C1**: We propose TransCP, a jointly learned (and end-to-end) model with explicit content-planning for description generation. Our model handles both the encoding and the ordering problems, which has not been achieved by existing models.
- **C2**: We propose a novel Transformer-based Pointer Network to learn a content-plan given a set of attributes. We propose gated residual attention and importance masking mechanisms for the network to optimize the attention computation and to avoid attribute repetition in generated content-plan.
- **C3**: We propose a tracking mechanism to seamlessly integrate content-planning into the Transformer decoder, which helps the decoder to effectively capture salient attributes in a proper order based on a content-plan.
- **C4**: We evaluate the proposed model over two real-world datasets. The experimental results show that our model consistently outperforms state-of-the-art models for data-to-text generation [14], [18], [21], [22].

This paper is an extension of our previous conference paper [27]. In the conference paper, we propose *content-plan-based bag-of-tokens* attention to integrate the content-planner with the text generator. This approach has three limitations.

First, it treats the input attributes as a bag of tokens and uses a simple linear transformation to encode them, which may be sub-optimal in capturing the relationships between attributes. Second, this approach uses LSTM as a Pointer Network to learn a content-plan. LSTM may not be optimal in handling a set of input attributes, because the attributes may not be properly ordered. Third, this approach uses a coverage mechanism to track the content-plan in the description generator. It uses the accumulation of attention weights from the previous decoding steps, which may not accurately capture the actual decoding states.

In this journal extension, we substantially extend the conference paper to address the limitations above. On the encoder side, we substantially improve the Pointer Network used in the previous model by proposing a novel Transformer-based Pointer Network to learn a content-plan (**C2**, detailed in Section 4.2). Together with this new model, we propose a gated residual attention mechanism that optimizes the attention computation process based on multi-head attentions and multi-layer attentions; we also propose an importance masking mechanism that helps our model avoid generating repeated attributes in the content-plan. On the decoder side, we propose a new content-plan tracking mechanism in the description generator to address the coverage mechanism limitation (**C3**, detailed in Section 4.3). We also conducted a comprehensive experimental study on our new model, including comparisons with the state-of-the-art data-to-text generation models (**C4**, detailed in Section 5).

2 Related Work

2.1 Traditional Approach

Traditional approaches for text generation [28] consist of three components: (1) a *content-selector* that selects the data to be expressed, (2) a *content-planner* that decides the order of the selected data to be mentioned, and (3) a *surface-realizer* that generates the final output based on the content-plan. Early studies on content-planning employed handcrafted rules [29] or a machine learning model as a content classifier [30]. For sentence planning and surface realization, early studies proposed template-based models [31], machine learning models using various linguistic features [32], [33], ordering constrained models [34], and tree-based models [35]. These approaches use handcrafted rules or shallow statistical models, which mostly cannot deal with unseen and complex cases.

2.2 Neural Approach

Early efforts on neural data-to-text generation. Earlier studies on neural data-to-text generation are driven by the success of deep neural networks for natural language processing – specifically, the success of the encoder-decoder framework [36] for machine translation. Using the encoder-decoder framework, Serban et al. [37] generated questions from facts in a KG, while Wiseman et al. [38] generated NBA game summaries. Mei et al. [39] proposed an aligner model that integrates the content selection mechanism into the encoder-decoder framework for generating weather forecasts from a set of database records. Lebret et al. [2] proposed a conditional language model for biography summarization. Follow-up studies on biography summarization employed the encoder-decoder framework. Sha et al. [14] proposed a link-based attention model to capture the relationships between attributes. Liu et al. [14] proposed a field-gating LSTM and dual attention mechanism to encode the attributes and inter-attribute relevance.

The aforementioned efforts focus on adapting the encoder-decoder framework for data-to-text generation. The adaptation includes representing the input as a sequence and using recurrent neural networks (e.g., LSTM [17]) as the encoder. However, when the input is in the form of a star-shaped graph, there may not be sequential relations in the attributes. Thus, capturing the relationships between attributes using such an adaptation is sub-optimal [40]. Moreover, Liu et al. [14] reported a decrease in the performance of such an adaptation when experimenting on disordered input. This confirms that simply linearizing the input in data-to-text generation may fail the encoder capturing the proper relationships between the attributes.

Improving the encoder to better capture the relationships between attributes. As discussed in Section 1, one of the problems in building an end-to-end neural data-to-text generation is how to capture the relationships between attributes properly. The original encoder-decoder model uses an LSTM-based encoder, which may not properly handle this problem. To address this problem, previous studies propose to use more suitable encoders to exploit the input structure. Marcheggiani et al. [41] applied a Graph Convolutional Network [42] as the encoder to capture the input structure. Trisedya et al. [6], [21] proposed a graph-based encoder to exploit the input structure for generating sentences from a knowledge base. Their model encodes an input graph using a topological traversal. The traversal algorithm requires further supervision signals from word-entity embeddings to decide the entity mentioning order in a sentence, which can be seen as an implicit content-plan. However, the word-entity embeddings may be overly generalized since they only reflect typical entity orders learned from a corpus (e.g., Wikipedia articles). Hence, this model is prone to ignoring the local context of the input. Another recent work, GraphWriter [18], combines Graph Attention Networks [19] and Transformer [20] to capture the relationships between attributes.

Most data-to-text generation models above only focus on the encoding problem. They are not designed to address the ordering problem (i.e., they do not exploit any explicit content-planning mechanism). Thus, these models are prone to generating text with repeated and/or incoherent content as discussed in Section 1.

Content-planning for neural data-to-text generation. Recently, Puduppully et al. [22] proposed Neural Content Planning (NCP), which is a two-stage model that includes content-planning to handle disordered input. First, NCP uses Pointer Network [26] as a content-planner. Then, the generated content-plan is used as the input of the encoder-decoder model (i.e., text generator) [36] to generate a description. This two-stage model suffers from error propagation between the content-planner and the text generator. The generated content-plan may contain errors (e.g., the attribute-missing error) that lead the text generator to produce an incomplete description. Another model that exploits content-planning is Tree-PLAN [24], which uses a

Hierarchical Attention Pointer Network [25] to generate a content-plan and uses a Tree-like encoding mechanism to combine close attributes, e.g., birth date and birth place. To address the attribute-missing error, Tree-PLAN [24] appends the unselected attributes to the ends of the generated content-plan. This strategy is sub-optimal because it disrupts the content-plan and spoils the content-planning.

3 Preliminary

We start with the problem definition. Let $\mathcal{A}$ be a set of n attributes of an entity in the form of pairs of key and value in any order, i.e., $\mathcal{A} = \{\langle k_1; v_1\rangle, \langle k_2; v_2\rangle, \ldots, \langle k_n; v_n\rangle\}$, where k_n is the key of an attribute and v_n is the value of the attribute. We consider $\mathcal{A}$ as the input and aim to generate a sentence $\mathcal{S} = \langle t_1, t_2, \ldots, t_l\rangle$ as the description of an entity, where t_l is a token at position l in the sentence. Table 1 illustrates the input and output of the task.

Most data-to-text generation models are built on top of an encoder-decoder framework [14], [16], [38], [39]. We first discuss the encoder-decoder framework [36] and its limitation when generating text from a set of attributes.

3.1 Encoder-Decoder Framework

The encoder-decoder framework is a sequence-to-sequence learning model that takes a variable-length input $\mathcal{T}$ and generates a variable-length output $\mathcal{T}'$ where the length of $\mathcal{T}$ and $\mathcal{T}'$ may differ. Typically, both the encoder and the decoder use a recurrent neural network, such as LSTM. The encoder reads each token of the input sequentially and computes a hidden state of each token. The last token's hidden state represents a summary of the input sequence in the form of a fixed-length vector (i.e., context vector $\boldsymbol{c}$). The decoder is trained to generate a sequence by predicting the next token given the decoder's previous hidden state and the context vector $\boldsymbol{c}$. This framework has been successfully applied in machine translation [8] to translate a sequence of words from one language to another.

In data-to-text generation, the encoder-decoder is used to generate text (e.g., entity description) from structured data (e.g., a set of attributes of an entity). Here, the encoder learns to encode the attributes into a fixed-length vector representation, which will be used as a context vector by the decoder to generate a description. Unlike machine translation, in data-to-text generation, the input is a set instead of a sequence, where the attributes may be randomly ordered (i.e., disordered input). Simply linearizing the input may not yield a proper order of the attributes. The encoder thus may fail to capture the relationships between the attributes, leading to an improper context vector.

Two other problems in using the encoder-decoder framework for data-to-text generation are repetition and incoherent text [6], [13] in the generated description. Repetition refers to that the same attribute is mentioned twice or more, and incoherent text refers to that close attributes, e.g., `birth_date` and `birth_place`, are mentioned in separate sentences. These problems occur because the input is in the form of a set of attributes, which does not contain information about the correct order of these attributes. Hence, the encoder cannot learn from the input and provide a proper ordering signal for the decoder to generate an accurate and concise target description. The two problems become more challenging when there are attributes that have similar values (e.g., `name` and `birth_name`) or an attribute has multiple values (e.g., `occupation`).

Recently, the Transformer model achieves state-of-the-art results in sequence-to-sequence modeling and has been used to replace LSTM in the encoder-decoder framework. In data-to-text generation, Transformer handles the encoding problem via its attention-based encoding mechanism to capture the relationships between attributes. Still, a Transformer-based encoder-decoder framework is prone to the problems of repetition and incoherent text. Next, we detail our solution to address these limitations.

4 Proposed Model

4.1 Solution Framework

Figure 1 illustrates the overall solution framework. Our framework consists of two components: a *content-plan generation module* and a *description generation module*.

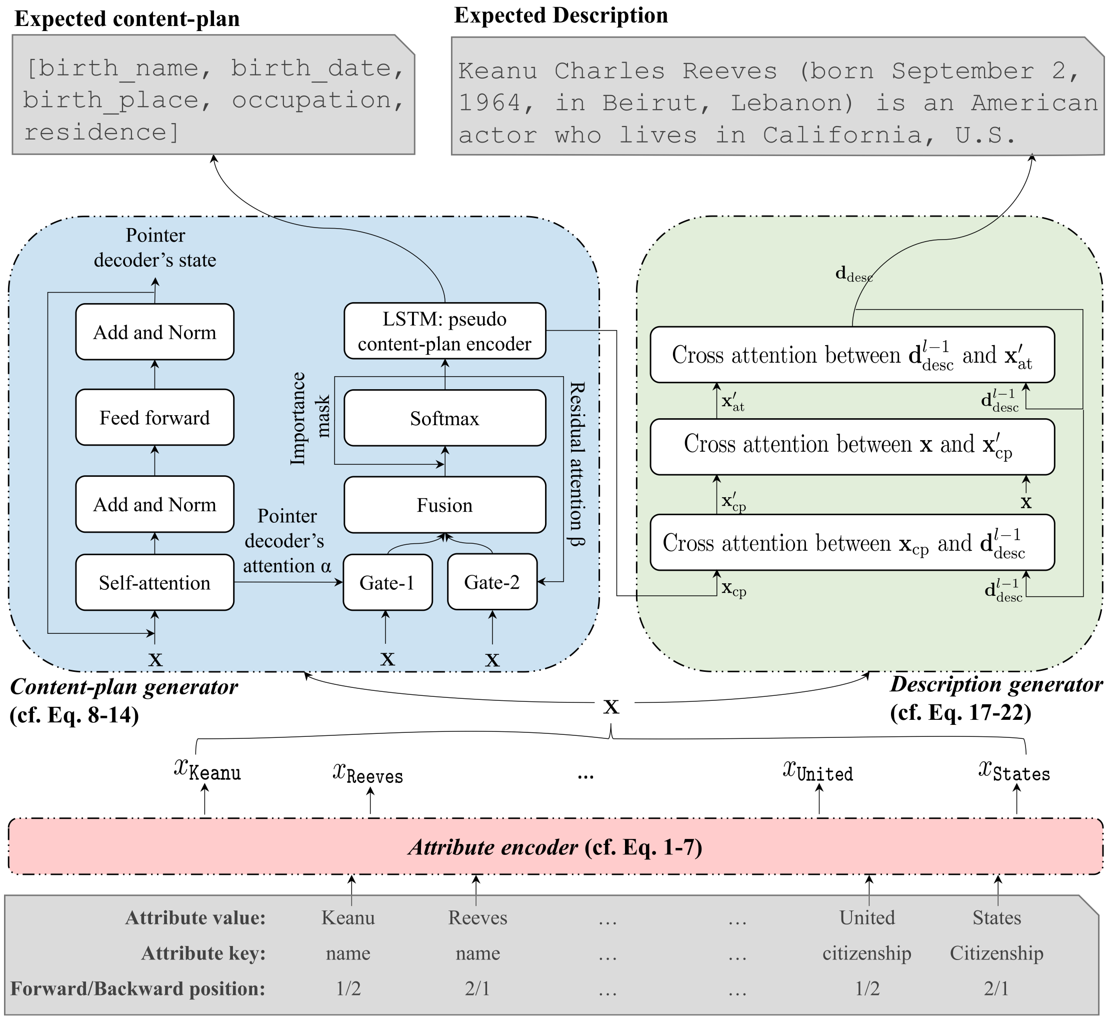
In the content-plan generation module (content-planner, Section 4.2), we propose a Transformer-based Pointer Network to learn a content-plan, which is later used to help the description generation module to highlight the salient attributes in a proper order. This module consists of four components:

1) An **attribute encoder** that encodes a set of attributes using the Transformer encoder (Section 4.2.1).
2) A **pointer generator** that generates a sequence of indexes (pointers) representing the order of attributes in the description. For the pointer generator, we propose a novel Transformer-based Pointer Network with *gated residual attention* and *multi-head dilation* to handle the multiple attention problem. We also propose *importance masking* to handle the repetition problems (Section 4.2.2).
3) A **content-plan generator** that generates the content-plan based on the learned pointers (Section 4.2.3).
4) A **content-plan encoder** that encodes the learned content-plan to be used in the description generation module (Section 4.2.3).

In the description generation module (description generator, Section 4.3), we integrate the content-plan into the Transformer decoder. We use the same Transformer encoder as in the content-plan generation module that treats the input (i.e., attributes) as a set of tokens. To integrate the content-plan into the Transformer decoder, we use a *content-plan tracking* mechanism by adding a cross-attention layer between the learned content-plan and the decoder states. This tracking mechanism traces attributes in a content-plan that have been exposed in the previous decoder states and helps our proposed decoder select the salient attributes conditioned by the content-plan. Hence, it can provide a better context (i.e., attention of salient attributes in an ordered fashion) for each decoding time-step. We also apply the *copy mechanism* [43] to handle the out-of-vocabulary problem.

4.2 Content-plan Generation

For the content-plan generation module, we propose a Transformer-based Pointer Network to learn a content-plan



Expected content-plan
[birth_name, birth_date, birth_place, occupation, residence]
Expected Description
Keanu Charles Reeves (born September 2, 1964, in Beirut, Lebanon) is an American actor who lives in California, U.S.
Pointer decoder's state
Add and Norm
Feed forward
Add and Norm
Self-attention
Pointer decoder's attention α
LSTM: pseudo content-plan encoder
Importance mask
Softmax
Fusion
Gate-1
Gate-2
Residual attention β
x
Content-plan generator (cf. Eq. 8-14)
$\mathbf{d}_{\text{desc}}$
Cross attention between $\mathbf{d}^{l-1}_{\text{desc}}$ and $\mathbf{x}'_{\text{at}}$
Cross attention between $\mathbf{x}$ and $\mathbf{x}'_{\text{cp}}$
Cross attention between $\mathbf{x}_{\text{cp}}$ and $\mathbf{d}^{l-1}_{\text{desc}}$
$\mathbf{x}_{\text{cp}}$
Description generator (cf. Eq. 17-22)
x_{Keanu} x_{Reeves} ... x_{United} x_{States}
Attribute encoder (cf. Eq. 1-7)
Attribute value: Keanu Reeves … … United States
Attribute key: name name … … citizenship Citizenship
Forward/Backward position: 1/2 2/1 … … 1/2 2/1

Fig. 1: Overview of our proposed model. Our model has two main components: the content-plan generator (denoted by the blue box) and the description generator (denoted by the green box). The red box is an attribute encoder based on Transformer shared by the content-plan and description generators. It takes the *attribute-tokens* that consists of key-value pairs of the attributes and the forward/backward positional encoding as input and learns the attribute-tokens vector representation. The content-plan generator takes attribute-token representation as input and generates the expected sequence of attributes (i.e. the content-plan). The description generator takes the attribute-token vector representation and the content-plan as input and generates the target description. The grey boxes indicate the input (attribute-tokens) and output (the expected content-plan and the expected final description) of the model.

given a set of attributes. We use the pairs of attributes and content-plan from a given training dataset to train the network. A Pointer Network uses an attention mechanism to generate a sequence of pointers that represents the order of attributes. Different from the original Pointer Network [26] that uses LSTM to encode the input and generate the output, our proposed Pointer Network is entirely based on the Transformer model, which has its benefit and challenges (cf. Section 4.2.2). The content-plan generation module consists of four components, which are detailed next.

4.2.1 Attribute Encoder.

The attribute encoder takes a set of attributes $\mathcal{A} = \{\langle k_1; v_1\rangle, \langle k_2; v_2\rangle, \ldots, \langle k_n; v_n\rangle\}$ as input. Here, the value of an attribute may consist of multiple tokens (i.e., $v_n = \langle v_n^1, v_n^2, \ldots, v_n^j\rangle$). We transform the multiple tokens into a single token representation and add positional encoding to maintain its internal order. Thus, the attributes are represented as $\mathcal{A} = [\langle k_1^1, v_1^1, f_1^1, r_1^1\rangle, \langle k_1^2, v_1^2, f_1^2, r_1^2\rangle, \ldots, \langle k_n^j, v_n^j, f_n^j, r_n^j\rangle]$ where f_n^j and r_n^j are the forward and reverse positions, respectively. We call the quadruple of key, value, forward and reverse position as *attribute-token*. The representation of each attribute-token is computed as follows.

$$\boldsymbol{z_k}_n^j = \tanh(\mathbf{W_{key}}[\boldsymbol{k}_n^j; \boldsymbol{f}_n^j; \boldsymbol{r}_n^j] + \boldsymbol{b_k}) \tag{1}$$

$$\boldsymbol{z_v}_n^j = \tanh(\mathbf{W_{val}}[\boldsymbol{v}_n^j; \boldsymbol{f}_n^j; \boldsymbol{r}_n^j] + \boldsymbol{b_v}) \tag{2}$$

$$\boldsymbol{x}_n^j = \tanh(\boldsymbol{z_k}_n^j + \boldsymbol{z_v}_n^j) \tag{3}$$

where $[;]$ denotes vector concatenation, $\boldsymbol{b}$ denotes a bias vector, and $\mathbf{W_{key}}$ and $\mathbf{W_{val}}$ are learned parameters. Vectors $\boldsymbol{z_k}$ and $\boldsymbol{z_v}$ represent the attributes' key and value.

Capturing the relationships between attributes is an essential task for the attribute encoder. By capturing these relationships, the encoder can provide a signal that helps the pointer generator to generate a correct content-plan, e.g., the encoder should capture that the attributes `birth_date` and `birth_place` are closely related such that the pointer generator should put them closely in the generated content-plan. To achieve this aim, we use the Transformer encoder to encode the attribute-token as follows:

$$\mathbf{x} = \text{softmax}\left(\frac{\mathbf{Q}_a\mathbf{K}_a^\mathsf{T}}{\sqrt{d}}\right)\mathbf{V}_a \tag{4}$$

$$\mathbf{Q}_a = [\boldsymbol{x}_1^1, \boldsymbol{x}_1^2, ..., \boldsymbol{x}_n^j]^\mathsf{T}\mathbf{W}_{\text{Qa}} + \boldsymbol{b}_{\text{Qa}} \tag{5}$$

$$\mathbf{K}_a = [\boldsymbol{x}_1^1, \boldsymbol{x}_1^2, ..., \boldsymbol{x}_n^j]^\mathsf{T}\mathbf{W}_{\text{Ka}} + \boldsymbol{b}_{\text{Ka}} \tag{6}$$

$$\mathbf{V}_a = [\boldsymbol{x}_1^1, \boldsymbol{x}_1^2, ..., \boldsymbol{x}_n^j]^\mathsf{T}\mathbf{W}_{\text{Va}} + \boldsymbol{b}_{\text{Va}} \tag{7}$$

where d is the dimensionality of the attribute-token vector and $\mathbf{W}$ denotes learned parameters. The output of the attribute encoder is the attribute-token vectors $\mathbf{x} = [\boldsymbol{x_a}_1^1, \boldsymbol{x_a}_1^2, \ldots, \boldsymbol{x_a}_n^j]$ that contain rich information about the relationships between attributes. These vectors are used as context for the pointer generator and the description generator since the attribute encoder is shared with the description generation module (cf. Section 4.4).

4.2.2 Pointer Generator.

Given a sequence of attribute-token vectors $\mathbf{x} = [\boldsymbol{x_a}_1^1, \boldsymbol{x_a}_1^2, \ldots, \boldsymbol{x_a}_n^j]$, the pointer generator aims to generate a sequence of pointer-indexes $I = [i_1, \ldots, i_g]$ (g is the number of attribute-tokens in the target content-plan). Here, i_g indicates an index that points to an attribute-token. To this end, we propose a Transformer-based Pointer Network. This model is different from the original Pointer Network [26] as our model is fully based on the Transformer, while the original one uses LSTM. We discuss the challenges in building such a model and our solutions.

Multiple attention problem. A Pointer Network exploits the encoder-decoder framework's attention mechanism to select attributes from the input set and arrange them as a content-plan. The original Pointer Network uses a single attention mechanism since it is built on top of an LSTM-based encoder-decoder framework [36]. In the Transformer-based encoder-decoder framework, there are multiple attention mechanisms since it uses multi-head attention, and the encoder and the decoder consist of multiple layers. Different attention heads (and different layers) may highlight different items in the input set, which may confuse the pointer generator when selecting an item as its output. These multiple attention mechanisms carry different important information. It is challenging to integrate all of this information to decide the selected item.

To address this challenge, we propose a gated residual attention to aggregate all the attention information. The aggregator takes two attention weights. The first is the aggregated attention from the previous layer β^{l-1} (l denotes the current layer), which serves as residual attention (this is disabled for the first layer since there is no attention computed by the aggregator yet). The second is the attention from the pointer generator's decoder state α (Eq. 8), which is computed based on the previous output $\mathbf{Q}_{\text{ptr}}$ and the encoded input $\mathbf{K}_{\text{ptr}}$. Here, $\mathbf{Q}_{\text{ptr}} = \text{lookup}(\mathbf{x}, [i_1, i_2, \ldots, i_{t-1}])$, where t is the current pointer generator time-step and $\text{lookup}(\cdot)$ is an embedding lookup function, while $\mathbf{K}_{\text{ptr}} = \mathbf{x}$. We update the attribute-token representation based on both attention weights (Eqs. 9-10). Then, we merge these two updates via a gating mechanism (Eqs. 11-12) to adaptively aggregate the attention information in the current layer. To compute the context vector $\mathbf{u}_{\text{ptr}}$ for the final attention, we multiply the current decoder state $\mathbf{d}_{\text{ptr}}$ with the updated attribute-token $\mathbf{x}_{\text{ptr}}$, and then we use softmax to compute the attention probability distribution (Eqs. 13-14).

$$\alpha = \text{softmax}\left(\frac{\mathbf{Q}_{\text{ptr}}\mathbf{K}_{\text{ptr}}^\mathsf{T}}{\sqrt{d}}\right) \tag{8}$$

$$\mathbf{x}_1 = \alpha[\boldsymbol{x_a}_1^1, \boldsymbol{x_a}_1^2, ..., \boldsymbol{x_a}_n^j] \tag{9}$$

$$\mathbf{x}_2 = \beta^{l-1}[\boldsymbol{x_a}_1^1, \boldsymbol{x_a}_1^2, ..., \boldsymbol{x_a}_n^j] \tag{10}$$

$$p_{\text{att}} = \text{sigmoid}([\mathbf{x_1}; \mathbf{x_2}]^\mathsf{T}\mathbf{w}_{\text{att}}) \tag{11}$$

$$\mathbf{x}_{\text{ptr}} = p_{\text{att}}\mathbf{x}_1 + (1 - p_{\text{att}})\mathbf{x}_2 \tag{12}$$

$$\mathbf{u}_{\text{ptr}} = \mathbf{d}_{\text{ptr}}^\mathsf{T}\mathbf{x}_{\text{ptr}} \tag{13}$$

$$\hat{\imath}_{\text{ptr}} = \beta^l = \text{softmax}(\mathbf{u_{ptr}}) \tag{14}$$

Here, $\hat{\imath}_{\text{ptr}}$ is the pointer-index output probability distribution over the vocabulary (i.e., the attribute-token input), p_{att} is a soft switch to aggregate the current and the residual attentions, and $\mathbf{w}_{\text{att}}$ denotes learned parameters.

Note that the attention from the transformer decoder consists of multi-head attention. To handle this issue, we use two strategies. First, we propose to use multi-head dilation, which gradually reduces the number of heads in each layer so that the last layer of the decoder only uses a single-head attention. This mechanism forces the decoder to learn a single-attention from the multi-head attention in multiple layers, which helps the aggregator integrate it with the residual attention. Second, for each layer that uses multi-head attention, we sum all the attention heads to produce a single-attention used for updating the attribute-token representations in the aggregator, i.e., $\alpha = \sum_{i=0}^{|h|} \alpha^i$, assuming h heads in a layer.

Repetition problem. Similar to the LSTM-based encoder-decoder framework, the Transformer-based framework is also subject to the repetition problem, i.e., generating multiple mentions of the same attribute in the output sentence. We address this problem with a final goal to generate a concise sentence as the entity description.

To alleviate the repetition problem, we propose an *importance masking* mechanism. We use an importance mask, which is a binary value that indicates whether an attribute has been selected in the previous time-steps. This mask serves as an additional signal to the pointer generator when computing the context vector $\mathbf{u}_{\text{ptr}}$. Hence, Eq. 13 is rewritten as $\mathbf{u}_{\text{ptr}} = \mathbf{d}_{\text{ptr}}^\mathsf{T}\mathbf{x}_{\text{ptr}}m_{\text{cov}}$, where m_{cov} is the importance mask. The attributes that have been selected in the previous time-steps will be given a very low attention score. They will not be selected in the current time-step, which avoids repetition.

To learn the content-plan, the pointer generator is trained to maximize the conditional log-likelihood:

$$p(I_d \mid A_d) = \sum_g \sum_{j=1}^{j=m} i'_{\text{ptr},j,g} \cdot \log \hat{\imath}_{\text{ptr},j,g} \tag{15}$$

$$\mathcal{L}_{\text{ptr}} = \frac{1}{D}\sum_{d=1}^{D} -\log p(I_d \mid A_d) \tag{16}$$

where (A_d, I_d) is a pair of attributes and target pointer-index (generated by finding the position of the attribute-token of the target content-plan in the original input) given for training, i' is the matrix of target pointer-indexes over the vocabulary, m is the number of attribute-tokens in the input, g is the number of content-plan time steps, D is the number of records in the dataset and $\mathcal{L}_{\text{ptr}}$ is the objective function of the pointer generator.

4.2.3 *Content-plan Generator and Encoder.*

The pointer-index I is a sequence of indexes that refers to the attribute-tokens $\mathbf{x}$ in a proper order. The content-plan generator uses the pointer-index to rearrange the sequence of attribute-tokens into a content-plan $\mathbf{x}'$. In the content-plan encoder, we use LSTM to encode the learned content-plan $\mathbf{x}'$ to capture the relationships between attributes in the content-plan as it already has a proper order over the attributes. The hidden states of the content-plan encoder $\mathbf{x}_{\text{cp}} = [\boldsymbol{x_{cp}}_1, \ldots, \boldsymbol{x_{cp}}_g]$ are forwarded to the text generator to help its attention model select attributes in a proper order.

4.3 Description Generation

We adapt the Transformer decoder to generate entity descriptions by integrating a content-plan. This integration helps the decoder generate a concise description by following the order of the attributes in the content-plan. The decoder takes three input: the attribute token representation $\mathbf{x}$, the content-plan representation $\mathbf{x}_{\text{cp}}$, and the text generator′ decoder state (the output of the previous timestep $\mathbf{d}_{\text{desc}}^{l-1}$).

The attribute-token representation $\mathbf{x}$ comes from the attribute encoder (Section 4.2.1), which contains rich information about the relationships between entities. The content-plan representation $\mathbf{x}_{\text{cp}}$ comes from the content-plan encoder (Section 4.2.3), where we use LSTM to capture the sequential (i.e., the ordering) relationships between attributes in the content-plan. The description generator state $\mathbf{d}_{\text{desc}}$ comes from a self-attention layer between tokens (in the target description) generated in the previous time-step.

The integration of the content-plan in the description generator is done as follows. First, we use a *cross-attention* between the content-plan and the description generator state to update the content-plan representation to mark the part of the content-plan that has been used in the target description (Eq. 17). Next, we update the attribute-token representation by using a cross-attention between the attribute-token and the updated content-plan (Eq. 18). This cross-attention is used to highlight the attributes to be selected next based on the content-plan. We use a gated fusion residual connection between the original and the updated attribute-token representation (i.e., $\mathbf{x}$ and $\mathbf{x}_{\text{at}}$) (Eqs. 19-20). This connection is used to automatically populate the two representations by their weights (γ). Finally, we update the description generator state using a cross-attention between the generator state and the updated attribute-token (Eq. 21). The updated generator state is used as a context vector $\mathbf{c}_{\text{desc}}$ to predict the final output.

$$\mathbf{x}'_{\text{cp}} = \text{softmax}\left(\frac{(\mathbf{x}_{\text{cp}}^{\mathsf{T}}\mathbf{W}_{\texttt{Qd}}^{1})({\mathbf{d}_{\text{desc}}^{l-1}}^{\mathsf{T}}\mathbf{W}_{\texttt{Kd}}^{1})}{\sqrt{d}}\right)\left({\mathbf{d}_{\text{desc}}^{l-1}}^{\mathsf{T}}\mathbf{W}_{\texttt{Vd}}^{1}\right) \quad (17)$$

$$\mathbf{x}_{\text{at}} = \text{softmax}\left(\frac{(\mathbf{x}^{\mathsf{T}}\mathbf{W}_{\texttt{Qd}}^{2})({\mathbf{x}'_{\text{cp}}}^{\mathsf{T}}\mathbf{W}_{\texttt{Kd}}^{2})}{\sqrt{d}}\right)\left({\mathbf{x}'_{\text{cp}}}^{\mathsf{T}}\mathbf{W}_{\texttt{Vd}}^{2}\right) \quad (18)$$

$$\gamma = \text{sigmoid}\left(\mathbf{W}_{\texttt{fusion}}[\mathbf{x};\mathbf{x}_{\text{at}}]\right) \quad (19)$$

$$\mathbf{x}'_{\text{at}} = \gamma\mathbf{x} + \left(1-\gamma\right)\mathbf{x}_{\text{at}} \quad (20)$$

$$\mathbf{c}_{\text{desc}} = \text{softmax}\left(\frac{({\mathbf{d}_{\text{desc}}^{l-1}}^{\mathsf{T}}\mathbf{W}_{\texttt{Qd}}^{3})({\mathbf{x}'_{\text{at}}}^{\mathsf{T}}\mathbf{W}_{\texttt{Kd}}^{3})}{\sqrt{d}}\right)\left({\mathbf{x}'_{\text{at}}}^{\mathsf{T}}\mathbf{W}_{\texttt{Vd}}^{3}\right) \quad (21)$$

$$\mathbf{d}_{\text{desc}} = \mathbf{w}_{\texttt{out}}\mathbf{c}_{\text{desc}} \quad (22)$$

Here, $\mathbf{x}'_{\text{cp}}$, $\mathbf{x}_{\text{at}}$, and $\mathbf{c}_{\text{desc}}$ are the updated content-plan representation, the updated attribute-token representation, and the context vector of the current generator state, respectively. $\mathbf{W}$ denotes learned parameters, and d is the dimensionality of the hidden state.

Our proposed model differs from the existing entity description generation models that also exploit content-planning. Unlike the two-staged model by Puduppully et al. [22], which only uses the content-plan as input for the description generator, our proposed model combines the content-plan and the original input (i.e., the attribute-tokens) to compute the context vector of the current generator state. Using this mechanism, our model can reduce the error propagation caused by incomplete content-plan (i.e., the generated content-plan may miss some salient attributes). Our model can recover the attribute-missing error by referring to the original input. We also devise an improved integration mechanism over the coverage mechanism presented in our previous conference version [27]. The coverage mechanism uses the accumulation of attention weights from the previous decoding steps, which may not accurately capture the actual decoding states (i.e., the actual part of the target sentence that has been generated). In contrast, our integration mechanism uses a cross-attention between the learned content-plan and the description generator state. It captures the actual description generator state more accurately than the coverage mechanism.

4.3.1 *Description Generator Training.*

The description generator aims to predict the next token of the description conditioned on the current context vector of the generator $\mathbf{c}_{\text{desc}}$. To handle the out-of-vocabulary problem, we apply the copy mechanism [43]. We update the probability distribution of the final vocabulary based on a learned probability p_{desc} (Eq. 23), which is used to select between predicting the output from the predefined vocabulary or the input token. Then, we combine the input attribute-token attention probability and the probability distribution of the predefined vocabulary as the final vocabulary probability distribution (Eq. 24).

$$p_{\text{desc}} = \text{sigmoid}\left({\mathbf{w}_{\text{copy}}^{1}}^{\mathsf{T}}\mathbf{c}_{\text{desc}} + {\mathbf{w}_{\text{copy}}^{2}}^{\mathsf{T}}\mathbf{d}_{\text{desc}}\right) \quad (23)$$

$$\hat{t} = \left[\text{softmax}(\mathbf{V}\mathbf{c}_{\text{desc}})p_{\text{desc}}; \boldsymbol{a}_{\text{desc}}(1-p_{\text{desc}})\right] \quad (24)$$

Here, $\hat{t}$ is the output probability distribution over the final vocabulary, $\mathbf{d}_{\text{desc}}$ is the previous description generator state, $\boldsymbol{a}_{\text{desc}}$ is the attention weight of the input, $\mathbf{w}_{\text{copy}}$ denotes learnable parameters, and $\mathbf{V}$ is the hidden-to-output weight matrix. The description generator is trained to maximize the conditional log-likelihood:

$$p(S_d \mid A_d) = \sum_{l}\sum_{j=1}^{j=|V|} {t'}_{l,j} \times \log \hat{t}_{l,j} \quad (25)$$

$$\mathcal{L}_{\text{dec}} = -\frac{1}{D}\sum_{d=1}^{D}\log p(S_d \mid A_d) \quad (26)$$

$$\mathcal{L} = \mathcal{L}_{\text{ptr}} + \mathcal{L}_{\text{dec}} \quad (27)$$

where (A_d, S_d) is a pair of attributes and entity description given for training, t' is the matrix of the target token description over the final vocabulary V, l is number of output time steps, D is the number of training samples, $\mathcal{L}_{\text{dec}}$ is the objective function of the description generator, and $\mathcal{L}$ is the overall objective function of our proposed model.

5 Experiments

5.1 Dataset

We aim to generate a description of an entity from its attributes where the attributes may be disordered. To handle disordered input, we propose a model that performs joint learning of content-planning and text generation. To train such a model, we need labeled training data in the form of triples of attributes, content-plan, and entity description.

Following Lebret et al. [2], we extract the first sentence of a Wikipedia page of a target entity as the description. Different from Lebret et al., who collected a specific type of entities (e.g., `Person`), we do not restrict the type of entities to be collected. We extract the attributes of a target entity by querying Wikidata for RDF triples that contain the target entity as the subject. In other words, we extract the direct relationships of an entity that form a star-shaped graph. We query the attributes from Wikidata instead of extracting from Wikipedia infobox to avoid additional processing (e.g., HTML tag removal, normalization, etc.).

We use string matching to find the order of attributes that are mentioned in the description as the content-plan. First, for each matched attribute, we store their position (index of the first character of the mentioned attribute value) in the description. Then, we sort the matched attributes based on their positions in ascending order. The sorted sequence of the attribute names forms a content-plan, e.g., for Table 1, the extracted content-plan is ⟨`birth_name`, `birth_date`, `birth_place`, `occupation`, `residence`⟩.

Our proposed model is trained to generate a description from a set of attributes of a target entity, which includes selecting salient attributes to be described. Since we automatically extract the description from Wikipedia, the extracted description may contain information (i.e., entities) not listed in the related extracted attributes, creating noises in the dataset. This problem may be caused by the delayed synchronization of a KG (i.e., the Wikipedia page has been updated, but not the Wikidata records), which often occurs on frequently updated information such as the current club of a football player, the latest movie of an actor, etc. Hence, to obtain high-quality data, we filter descriptions that contain noises. First, we use a Named Entity Recognizer (we use spaCy NER) to detect all entities in a description. Then, we remove records in the dataset whose description contains any entity that is not listed in the related extracted attributes.

The collected dataset contains $152,231$ triples of attributes, content-plan, and description (we call it the **WIKIALL** dataset). The dataset contains 53 entity types with an average of 15 attributes per entity, and an average of 20 tokens per description. For benchmarking, we also use the **WIKIBIO** dataset [2], which contains 728,321 biographies from Wikipedia. The average number of attributes per entity of WIKIBIO dataset is 19, and the average number of tokens per description is 26. We split each dataset into train set (80%), dev set (10%), and test set (10%).

We evaluate our model on the datasets above. The attributes in WIKIALL are disordered since they are the result of a query to Wikidata. The attributes in WIKIBIO are practically ordered, i.e., the salient attributes are ordered by their appearances in the target entity description but may have noises (non-salient attributes) between them. To test on disordered attributes of WIKIBIO dataset, we randomly shuffle the attributes.

5.2 Experiment Settings

We implement our model [1] in TensorFlow and train it on NVIDIA Tesla K40c. We use grid search to tune the hyper-parameters. We select the embedding size from $[8, 16, 32, 64, 128]$, the hidden units for the networks from $[128, 256, 512]$, the dropout rate from $[0.1, 0.3, 0.5]$, and the learning rate from $[1e^{-2}, 1e^{-4}]$. The best hyper-parameter settings are as follows. We use 512 hidden units for the networks. We use 128, 64, and 8 dimensions of attribute value token embeddings, attribute key embeddings, and position embeddings, respectively. We use a 0.1 dropout rate. We use Adam [44] with a learning rate of $1e^{-4}$.

5.3 Content-planner Evaluation

In this experiment, we aim to evaluate the effectiveness of the neural content-planner. The evaluation protocol is as follows. Given a set of attributes as input, the content-planner aims to generate a content-plan. We take the generated content-plan and compute the **precision**, **recall**, and *Damerau-Levenshtein Distance* (**DLD**). The precision indicates the portion of the generated content-plan that is correct, i.e., the percentage of the attributes that belong to the actual content-plan (ground truth) over all the attributes listed in the generated content-plan. The recall indicates the portion of the ground truth that is included in the generated content-plan. Both the precision and the recall show how well the content-planner selects salient attributes for the content-plan. The DLD measures how well the content-planner orders the selected attributes compared to the order of attributes in the ground truth.

5.3.1 *Tested Content-planners*

There are three representative neural data-to-text generation models that include content-planning (i.e., with a content-planning stage). For this experiment, we take the content-planner of these models for comparison with our proposed content-planner. The first is **GCP** [21], which uses entity order-aware topological traversal algorithms for content-planning. The second is CovATT (the model proposed in our previous conference paper) [27], which use an LSTM-based Pointer Network to generate a content-plan. We refer to this model as **LSTM-PN**. The last one is Tree-PLAN, which uses Transformer-based Pointer Network [25]. We refer to this model as **Transformer-PN**.

We further perform ablation tests to show the effectiveness of the importance masking mechanism. For all the

1. The source code and data are available at: https://bitbucket.org/bayudt/ent_desc/src/master/TransCP/

TABLE 2: Content-planner Comparisons (over three runs with random seeds).

Model	WIKIALL			WIKIBIO (disordered)			WIKIBIO (ordered)		
	Precision	Recall	DLD	Precision	Recall	DLD	Precision	Recall	DLD
GCP	42.24 ±.00	**100.00** ±.00	26.15 ±.00	31.25 ±.00	**100.00** ±.00	19.48 ±.00	30.47 ±.00	**100.00** ±.00	19.54 ±.00
LSTM-PN	84.07 ±.32	71.45 ±.26	73.88 ±.43	72.45 ±.40	56.34 ±.32	54.59 ±.35	74.45 ±.26	60.35 ±.29	57.55 ±.24
+ masking	86.15 ±.24	74.35 ±.20	74.85 ±.28	74.25 ±.26	59.74 ±.25	56.77 ±.31	75.54 ±.21	63.25 ±.15	58.95 ±.26
Transformer-PN	87.14 ±.36	80.54 ±.23	78.14 ±.22	76.22 ±.25	64.55 ±.24	60.54 ±.18	76.35 ±.21	64.47 ±.15	60.58 ±.22
+ masking	88.76 ±.23	83.01 ±.21	80.33 ±.16	78.77 ±.15	67.57 ±.23	61.35 ±.28	79.05 ±.25	67.53 ±.24	61.02 ±.21
TransCP (Transformer only)	87.01 ±.42	80.23 ±.46	79.02 ±.26	76.74 ±.34	64.15 ±.42	60.12 ±.23	76.02 ±.32	64.13 ±.45	60.65 ±.33
+ gating	90.21 ±.32	84.51 ±.27	82.22 ±.24	81.24 ±.20	67.31 ±.25	63.32 ±.22	81.32 ±.32	67.54 ±.27	63.22 ±.15
+ masking	91.32 ±.24	85.45 ±.27	84.45 ±.29	82.37 ±.24	69.87 ±.24	65.24 ±.24	82.45 ±.26	69.77 ±.24	65.35 ±.28
+ dilation (proposed)	**92.10** ±.15	85.61 ±.06	**85.12** ±.12	**83.15** ±.08	70.54 ±.14	**66.35** ±.12	**83.05** ±.10	70.33 ±.12	**66.21** ±.06

content-planners based on Pointer Network (i.e., LSTM-PN, Transformer-PN, and our proposed TransCP content-planner), we test them with and without importance masking. For our proposed **TransCP**, we also test the gated residual attention and the head-dilation techniques presented in Section 4.2.2.

5.3.2 Results

The results of content-planner comparisons are listed in Table 2. GCP achieves 100% recall on all datasets but very low precision and DLD scores. This is because it only orders the attributes (with a topological traversal) without filtering them. The non-salient attributes are still listed in the resultant content-plan. Furthermore, the input of the task is a set of attributes. This input forms a star-shaped graph. The traversal algorithm used in GCP may not properly capture the relationships between attributes in this kind of structure.

Table 2 also shows that the Transformer-based Pointer Network models (i.e., the Transformer-PN and our proposed model TransCP) outperform LSTM-PN. Moreover, the Transformer-based Pointer Network models are order-agnostic, as evidenced by their stable performance on both the WIKIBIO ordered and WIKIBIO disordered datasets. These results confirm that the Transformer model captures the relationships between attributes in an input set better than LSTM, especially when the input is disordered. Our gated residual attention further improves the capability of the Transformer-based Pointer Network models by aggregating the attentions from multiple layers (and heads). Overall, our TransCP model achieves the best DLD scores: 85.12, 66.35, 66.21 on WIKIALL, WIKIBIO disordered, and WIKIBIO ordered datasets, respectively.

The ablation test results confirm the effectiveness of our gating mechanism, importance masking and dilation strategies. The gated residual attention substantially improves the content-planner as it considers all attentions. The importance masking strategy applied to the Pointer-Network-based content-planner (i.e., LSTM-PN, Transformer-PN, and our TransCP) gives one to two points of improvement in the DLD score. Meanwhile, the dilation strategy provides stable performance for TransCP, which is shown by a smaller standard deviation of the performance scores of the model's multiple runs. The best dilation configuration is dividing the attention head in half until only one attention head is left (e.g., 8,4,2,1).

5.4 Description Generator Evaluation

Next, we compare the overall description generation performance of our full **TransCP** model with five representative data-to-text generation models. We use three evaluation metrics, including BLEU [45], METEOR [46], and TER [47].

5.4.1 Models

We compare our TransCP with six representative data-to-text-generation models:

- **FGDA**, which uses a field gating and dual attention mechanism [14]. This model does not explicitly learn content-planning.
- **GraphWriter**, which uses Graph Attention Networks and Transformer to encode an input graph [18]. This model does not explicitly learn content-planning.
- **GCP**, which uses an entity order-aware topological traversal algorithm to encode an input graph [21]
- **NCP**, which is two-stage data-to-text generation model [22]. The content-planner and the description generation are learned separately.
- **Tree-PLAN**, which uses Hierarchical Attention Pointer Network as the content-planner and Transformer as the description generator.
- **CovATT**, which is the model proposed in our previous paper [27]. It uses coverage mechanism to integrate the content-plan and the description generator.

It is worth noting that all of these baseline models (including our model TransCP) use copy mechanism [43], [48] to handle the out of vocabulary as described in their papers. We use the same vocabulary taken from the most frequent 20,000 words in the training set for fair comparisons. We rerun all the models using the code provided by the authors, except for Tree-PLAN. For this method, we try our best to re-implement the model based on the paper's description.

5.4.2 Results

Table 3 shows the description generation performance. The two models without explicit content-planning, FGDA and GraphWriter, have lower performance than the other methods. FGDA has a substantial performance drop when running on the WIKIBIO disordered dataset. In comparison, GraphWriter performs closer on both the ordered and disordered datasets. These results confirm that the Transformer model is more suitable to encode an input set than LSTM. GCP performs content-planning via its entity order-aware topological traversal algorithm. However, as discussed in Section 5.3.2, it may fail to capture the relationships between

TABLE 3: Description Generation Comparisons (over three runs with random seeds).

Model	WIKIALL			WIKIBIO (disordered)			WIKIBIO (ordered)		
	BLEU↑	METEOR↑	TER↓	BLEU↑	METEOR↑	TER↓	BLEU↑	METEOR↑	TER↓
FGDA	61.38 ±.23	42.78 ±.35	31.00 ±.22	40.00 ±.31	30.99 ±.28	54.76 ±.27	42.26 ±.33	31.76 ±.39	53.98 ±.15
GraphWriter	62.09 ±.28	43.99 ±.23	29.96 ±.22	41.91 ±.21	32.36 ±.38	54.35 ±.26	42.29 ±.25	32.56 ±.32	54.02 ±.24
GCP	62.38 ±.23	43.97 ±.40	29.44 ±.13	41.28 ±.19	32.37 ±.28	54.26 ±.19	41.25 ±.35	32.04 ±.20	54.67 ±.16
NCP	63.03 ±.18	44.09 ±.27	28.43 ±.11	42.89 ±.18	33.79 ±.14	52.97 ±.30	43.34 ±.18	33.76 ±.19	53.14 ±.17
CovATT	64.55 ±.17	45.02 ±.23	27.84 ±.15	44.43 ±.18	34.09 ±.18	51.81 ±.13	44.94 ±.21	34.25 ±.15	52.97 ±.17
Tree-PLAN	64.02 ±.18	45.82 ±.12	27.25 ±.30	45.01 ±.24	34.30 ±.17	51.29 ±.26	45.35 ±.11	34.54 ±.22	51.14 ±.23
TransCP (Transformer only)	63.20 ±.35	44.21 ±.32	28.05 ±.21	43.12 ±.31	33.69 ±.32	52.63 ±.32	43.54 ±.26	33.21 ±.24	52.87 ±.26
+ Copy mechanism	64.43 ±.23	45.24 ±.23	27.87 ±.25	44.42 ±.24	34.28 ±.27	52.17 ±.25	44.76 ±.24	34.21 ±.26	52.03 ±.28
+ Coverage mechanism	64.95 ±.24	45.98 ±.15	27.11 ±.12	45.02 ±.15	34.78 ±.13	51.85 ±.11	45.86 ±.13	34.95 ±.13	51.27 ±.11
+ CP tracking (proposed)	**65.95** ±.10	**46.56** ±.19	**26.85** ±.15	**45.92** ±.20	**35.19** ±.14	**51.09** ±.17	**46.14** ±.16	**35.40** ±.18	**50.93** ±.18

TABLE 4: Human evaluation results.

Model	Correctness	Grammaticality	Fluency
FGDA	2.51	2.58	2.54
GraphWriter	2.45	2.52	2.48
GCP	2.31	2.40	2.36
NCP	2.54	2.68	2.51
Tree-PLAN	2.60	2.57	2.46
CovATT	2.68	**2.76**	2.57
TransCP	**2.72**	2.74	**2.65**

attributes in a star-shaped graph, which affects the overall model performance. It is slightly outperformed by GraphWriter on WIKIBIO ordered dataset.

Among the models with content-planning, NCP is the one with a two-stage framework that separates the content-planner from the description generator. It performs worse than CovATT, Tree-Plan, and TransCP. This confirms that the two-stage model is prone to error propagation between the content-planner and the description generator.

Overall, TransCP achieves a consistent improvement over the baselines, and the improvement is statistically significant, with $p < 0.01$ based on the t-test of the BLEU scores. We use MultEval to compute the p-value based on an approximate randomization [49]. Our model achieves higher BLEU scores than the baselines, which indicates that our model generates descriptions with a better order of attribute mentions. Moreover, the better (lower) TER scores indicate that our model generates a concise description (i.e., following the content-plan). Compared with the two end-to-end models CovATT and Tree-PLAN, TransCP achieves better performance for two reasons. First, it has a better content-planner than those of the other two models (cf. Section 5.3.2). Second, it effectively integrates the content-plan into the description generator with a tracking mechanism that records the used content-plan in the previous time-steps. In comparison, Tree-PLAN appends the attributes that are not selected in the content-plan into the generated content-plan as the input for the description generator, which is sub-optimal.

Table 3 also shows the results for the ablation test of our proposed TransCP decoder module. The copy and content-plan tracking mechanisms further help our models produce a better output. Specifically, the copy mechanism helps in generating the (rare) entity names which are not in the vocabulary. Meanwhile, the content-plan tracking mechanism helps in producing a concise description, as shown by the lower TER scores. In the ablation test, we also compare the coverage mechanism proposed in our previous conference paper [27] and the content-plan tracking mechanism proposed in this extension. The result shows that the new content-plan tracking mechanism helps improve the model's overall performance by roughly 1 BLEU score.

5.4.3 Human Evaluation.

Following Trisedya et al. [21], we conduct manual evaluations on the generated descriptions using three metrics, including *correctness*, *grammaticality*, and *fluency*. Correctness measures the semantics of the generated description (i.e., containing wrong order of attribute mentions or not, e.g., `"born in USA, New York"`); grammaticality measures grammatical and spelling errors; and fluency measures the fluency of the output (e.g., containing repetitions or not). For each metric, a score of 3 is given to an output that contains no errors; a score of 2 is given to an output that contains one error; and a score of 1 is given to an output that contains more than one error. We randomly choose 300 records of the WIKIALL dataset along with the output of each model. We recruited six annotators who have studied English for at least ten years and completed education in an English environment for at least two years. The total time spent on these evaluations is around 250 hours. Table 4 shows the results of the human evaluation. The results are consistent with those of the automatic evaluations, i.e., our proposed model again achieves the best scores.

5.4.4 Discussion

We further perform experiments to study the impact of different content-plans on our proposed description generator model. Table 5 shows some samples of this experiment. Sample-1 shows the generated description using a gold standard content-plan. In Sample-2, we remove a common attribute `birth_date` from the content-plan to replicate an incomplete content-plan. In this sample, the attribute `birth_date` is included in the generated output, which shows that our model can handle the attribute-missing error in the generated content-plan.

Next, we examine whether the generated description follows the content-plan by removing a less common attribute `residence` and replacing the attribute `birth_name` with `name` in Sample-3 and Sample-4, respectively. We can see that the generated output follows the content-plan. This result shows that, in our model, the content-plan can be used to intervene the description generation phase and control what to be generated in the final output.

TABLE 5: Descriptions generated with different content-plans. Output formatting (e.g., capitalization) are manually done for readability.

Attribute set input:		`⟨name; Keanu⟩, ⟨name; Reeves⟩, ⟨birth_place; Beirut⟩, ⟨birth_place, Lebanon⟩, ⟨occupation; actor⟩, ⟨occupation; musician⟩, ⟨birth_date; September⟩, ⟨birth_date;2⟩, ⟨birth_date; 1964⟩, ⟨residence; California⟩, ⟨residence; U.S.⟩, ⟨birth_name; Keanu⟩, ⟨birth_name; Charles⟩, ⟨birth_name; Reeves⟩, ⟨citizenship; Canada⟩, ⟨citizenship; United⟩, ⟨citizenship; States⟩`
1	**Content-plan:**	`⟨birth_name; Keanu⟩, ⟨birth_name; Charles⟩, ⟨birth_name; Reeves⟩, ⟨birth_date; September⟩, ⟨birth_date;2⟩, ⟨birth_date; 1964⟩, ⟨birth_place; Beirut⟩, ⟨birth_place, ⟨occupation; actor⟩, ⟨residence; California⟩, ⟨residence; U.S.⟩`
	Output:	**"Keanu Charles Reeves (born September 2, 1964 in Beirut, Lebanon) is an actor who lives in California, U.S."**
2	**Content-plan:**	`⟨birth_name; Keanu⟩, ⟨birth_name; Charles⟩, ⟨birth_name; Reeves⟩, ⟨birth_place; Beirut⟩, ⟨birth_place, ⟨occupation; actor⟩, ⟨residence; California⟩, ⟨residence; U.S.⟩`
	Output:	**"Keanu Charles Reeves (September 2, 1964), was born in Beirut, Lebanon, is an actor lives in California, U.S."**
3	**Content-plan:**	`⟨birth_name; Keanu⟩, ⟨birth_name; Charles⟩, ⟨birth_name; Reeves⟩, ⟨birth_date; September⟩, ⟨birth_date;2⟩, ⟨birth_date; 1964⟩, ⟨birth_place; Beirut⟩, ⟨birth_place, ⟨occupation; actor⟩`
	Output:	**"Keanu Charles Reeves, who was born in September 2, 1964 in Beirut, Lebanon, is an actor"**
4	**Content-plan:**	`⟨name; Keanu⟩, ⟨name; Reeves⟩, ⟨birth_date; September⟩, ⟨birth_date;2⟩, ⟨birth_date; 1964⟩, ⟨birth_place; Beirut⟩, ⟨birth_place, ⟨occupation; actor⟩, ⟨residence; California⟩, ⟨residence; U.S.⟩`
	Output:	**"Keanu Reeves (September 2, 1964) born in Beirut, Lebanon is an actor from California, U.S."**

6 Conclusions and Future Work

We proposed an end-to-end data-to-text generation model on top of a Transformer-based encoder-decoder framework. Our model addresses two common problems in encoder-decoder based models for data-to-text generation: input encoding and output ordering. Our model employs joint learning of content-planning and description generation to reduce error propagation between the two components for generating a description of an entity from its attributes. To learn a content-plan explicitly, we propose a Transformer-based Pointer Network with *gated residual attention* and *importance masking*. To integrate the content-plan into the description generator, we propose a content-plan tracking mechanism that effectively captures salient attributes in an order suitable for output generation. Experimental results on content-plan generation and description generation show that our model outperforms the competitors and achieves the best scores in all metrics on the WIKIALL and WIKIBIO test datasets. Moreover, our model retains its high effectiveness on disordered input and achieves a consistent improvement over the competitors by up to 3%.

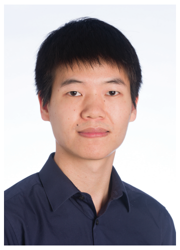
The proposed model requires training data in the form of triples of attributes, content-plan, and description. Extracting a content-plan from a description manually could be time-consuming. We bypass this issue by using string matching to find the order of attributes in the description as a content-plan. String matching may not capture the semantic similarity between attributes and text. For example, in Table 1, the extracted content-plan does not include attribute `citizenship` since the string matching cannot capture the similarity between `United States` and `American`. We plan to address this issue using semantic similarity search in future work.

References

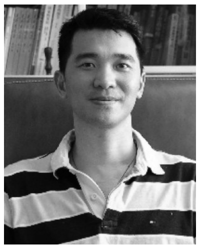
[1] A. Bordes, S. Chopra, and J. Weston, "Question answering with subgraph embeddings," in *EMNLP*, 2014, pp. 615–620.
[2] R. Lebret, D. Grangier, and M. Auli, "Neural text generation from structured data with application to the biography domain," in *EMNLP*, 2016, pp. 1203–1213.
[3] R. Zhang, B. D. Trisedya, M. Li, Y. Jiang, and J. Qi, "A benchmark and comprehensive survey on knowledge graph entity alignment via representation learning," *The VLDB Journal*, pp. 1–26, 2022.
[4] B. D. Trisedya, J. Qi, and R. Zhang, "Entity alignment between knowledge graphs using attribute embeddings," in *AAAI*, 2019.
[5] J. Bao, D. Tang, N. Duan, Z. Yan, Y. Lv, M. Zhou, and T. Zhao, "Table-to-text: Describing table region with natural language," in *AAAI*, 2018, pp. 5020–5027.
[6] B. D. Trisedya, J. Qi, R. Zhang, and W. Wang, "GTR-LSTM: A triple encoder for sentence generation from rdf data," in *ACL*, 2018, pp. 1627–1637.
[7] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini, "Creating training corpora for nlg micro-planners," in *ACL*, 2017, pp. 179–188.
[8] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *EMNLP*, 2014, pp. 1724–1734.
[9] J. Li, A. Sun, and S. R. Joty, "Segbot: A generic neural text segmentation model with pointer network." in *IJCAI*, 2018, pp. 4166–4172.
[10] J. Li, P. Han, X. Ren, J. Hu, L. Chen, and S. Shang, "Sequence labeling with meta-learning," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021.
[11] B. D. Trisedya, G. Weikum, J. Qi, and R. Zhang, "Neural relation extraction for knowledge base enrichment," in *ACL*, 2019, pp. 229–240.
[12] J. Li, B. Chiu, S. Feng, and H. Wang, "Few-shot named entity recognition via meta-learning," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2020.
[13] Z. Fu, W. Lam, A. M.-C. So, and B. Shi, "A theoretical analysis of the repetition problem in text generation," in *AAAI*, 2021, pp. 12 848–12 856.
[14] T. Liu, K. Wang, L. Sha, Z. Sui, and B. Chang, "Table-to-text generation by structure-aware seq2seq learning," in *AAAI*, 2018, pp. 4881–4888.
[15] T. Liu, S. Ma, Q. Xia, F. Luo, B. Chang, and Z. Sui, "Hierarchical encoder with auxiliary supervision for table-to-text generation:

Learning better representation for tables," in *AAAI*, 2019, pp. 6786–6793.
[16] L. Sha, L. Mou, T. Liu, P. Poupart, S. Li, B. Chang, and Z. Sui, "Order-planning neural text generation from structured data," in *AAAI*, 2018, pp. 5414–5421.
[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[18] R. Koncel-Kedziorski, D. Bekal, Y. Luan, M. Lapata, and H. Hajishirzi, "Text generation from knowledge graphs with graph transformers," in *NAACL-HLT*, 2019, pp. 2284–2293.
[19] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.
[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017, pp. 5998–6008.
[21] B. D. Trisedya, J. Qi, W. Wang, and R. Zhang, "GCP: Graph encoder with content-planning for sentence generation from knowledge base," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.
[22] R. Puduppully, L. Dong, and M. Lapata, "Data-to-text generation with content selection and planning," in *AAAI*, 2019, pp. 6908–6915.
[23] K. Chen, F. Li, B. Hu, W. Peng, Q. Chen, H. Yu, and Y. Xiang, "Neural data-to-text generation with dynamic content planning," *Knowledge-Based Systems*, vol. 215, p. 106610, 2021.
[24] Y. Bai, Z. Li, N. Ding, Y. Shen, and H.-T. Zheng, "Infobox-to-text generation with tree-like planning based attention network," in *IJCAI*, 2020, pp. 3773–3779.
[25] T. Wang and X. Wan, "Hierarchical attention networks for sentence ordering," in *AAAI*, 2019, pp. 7184–7191.
[26] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *NeurIPS*, 2015, pp. 2692–2700.
[27] B. D. Trisedya, J. Qi, and R. Zhang, "Sentence generation for entity description with content-plan attention." in *AAAI*, 2020, pp. 9057–9064.
[28] K. McKeown, *Text Generation*. Cambridge University Press, 1992.
[29] P. A. Duboue and K. R. McKeown, "Statistical acquisition of content selection rules for natural language generation," in *EMNLP*, 2003, pp. 121–128.
[30] R. Barzilay and M. Lapata, "Collective content selection for concept-to-text generation," in *EMNLP*, 2005, pp. 331–338.
[31] K. R. McKeown, D. A. Jordan, S. Pan, J. Shaw, and B. A. Allen, "Language generation for multimedia healthcare briefings," in *ANLP*, 1997, pp. 277–282.
[32] E. Reiter and R. Dale, *Building natural language generation systems*. Cambridge University Press, 2000.
[33] W. Lu and H. T. Ng, "A probabilistic forest-to-string model for language generation from typed lambda calculus expressions," in *EMNLP*, 2011, pp. 1611–1622.
[34] P. A. Duboue and K. R. Mckeown, "Empirically estimating order constraints for content planning in generation," in *ACL*, 2001, pp. 172–179.
[35] J. Kim and R. J. Mooney, "Generative alignment and semantic parsing for learning from ambiguous supervision," in *COLING*, 2010, pp. 543–551.
[36] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.
[37] I. V. Serban, A. García-Durán, C. Gulcehre, S. Ahn, S. Chandar, A. Courville, and Y. Bengio, "Generating factoid questions with recurrent neural networks: The 30M factoid question-answer corpus," in *ACL*, 2016, pp. 588–598.
[38] S. J. Wiseman, S. M. Shieber, and A. S. M. Rush, "Challenges in data-to-document generation," in *EMNLP*, 2017, pp. 2253–2263.
[39] H. Mei, M. Bansal, and M. R. Walter, "What to talk about and how? selective generation using lstms with coarse-to-fine alignment," in *NAACL-HLT*, 2016, pp. 720–730.
[40] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," in *ICLR*, 2016.
[41] D. Marcheggiani and L. Perez-Beltrachini, "Deep graph convolutional encoders for structured data to text generation," in *INLG*, 2018, pp. 1–9.
[42] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
[43] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *EMNLP*, 2017, pp. 1073–1083.
[44] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.
[45] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *ACL*, 2002, pp. 311–318.
[46] M. J. Denkowski and A. Lavie, "Meteor 1.3: Automatic metric for reliable optimization and evaluation of machine translation systems," in *Workshop on Statistical Machine Translation*, 2011, pp. 85–91.
[47] M. Snover, B. Dorr, R. Schwartz, L. Micciulla, and J. Makhoul, "A study of translation edit rate with targeted human annotation," in *Workshop on Machine Translation in the Americas*, 2006, pp. 223–231.
[48] R. Nallapati, B. Zhou, C. dos Santos, Ç. GuÌ‡lçehre, and B. Xiang, "Abstractive text summarization using sequence-to-sequence RNNs and beyond," in *CoNLL*, 2016, pp. 280–290.
[49] J. H. Clark, C. Dyer, A. Lavie, and N. A. Smith, "Better hypothesis testing for statistical machine translation: Controlling for optimizer instability," in *EMNLP*, 2011, pp. 176–181.

Bayu Distiawan Trisedya is a Lecturer in the Faculty of Computer Science Universitas Indonesia, who is currently an Honorary Research Fellow at The University of Melbourne. He received the bachelor's and master's degrees from Universitas Indonesia in 2009 and 2011, respectively and the Ph.D. degree in computer science from The University of Melbourne in 2021. His research interests include information extraction and NLP.



Jianzhong Qi is a Senior Lecturer in the School of Computing and Information Systems at The University of Melbourne. He received his Ph.D. degree from The University of Melbourne in 2014. His research interests include machine learning and data management and analytics, with a focus on spatial, temporal, and textual data.



Haitao Zheng received the bachelor's and master's degrees from the Department of Computer Science, Sun Yat-Sen University, in 2001 and 2004, respectively and the Ph.D. degree in medical informatics, Seoul National University. He is currently an Associate Professor with the Graduate School, Shenzhen, Tsinghua University, China. His research interests include artificial intelligence, semantic web, information retrieval, machine learning, and medical informatics.

Professor Flora Salim is the CISCO Chair of Digital Transport, School of Computer Science and Engineering, UNSW Sydney. Her research, on behaviour modelling, AI and machine learning on time-series and spatio-temporal sensor data, has been funded by the ARC, Humboldt Foundation, Bayer Foundation, Microsoft Research, Qatar National Research Fund, and many local and international industry partners. She won the Women in AI Awards 2022 ANZ - Defence and Intelligence category.

Rui Zhang is a visiting Professor at Tsinghua University and was a Professor at University of Melbourne. His research interests include machine learning and big data, especially recommender systems, search, knowledge graph and chatbot. Professor Zhang has won several awards, including Future Fellowship by the Australian Research Council in 2012, Chris Wallace Award for Outstanding Research by the Computing Research and Education Association of Australasia in 2015, and Google Faculty Research Award in 2017.