國立臺灣大學電機資訊學院電信工程學研究所

碩士論文

Graduate Institute of Communication Engineering

College of Electrical Engineering & Computer Science

National Taiwan University

Master Thesis

基於結構標記生成帶文字標籤的圖結構

TextGraphBART: Unifying Graph and Text Generation with Structure Token

鄭景文

Ching-Wen Cheng

指導教授: 葉丙成 博士, 李宏毅 博士

Advisor: Ping-Cheng Yeh Ph.D., Hung-yi Lee Ph.D.

中華民國 112 年 8 月

August, 2023

doi:10.6342/NTU202302806

Acknowledgements

I would like to express my deepest gratitude to my thesis advisors, Prof. Ping-Cheng Yeh and Prof. Hung-yi Lee, for their unwavering support, guidance, and invaluable insights throughout this research. Their combined expertise and encouragement have been the cornerstone of my academic journey, and I am deeply indebted to both of them for their dedication.

Special thanks are extended to RelationalAI for generously providing the A100 GPU that was vital for conducting the experiments in this research. Their collaboration and support have been instrumental in achieving the results presented in this thesis.

I am also grateful to my friends, my family, and anyone who contributed directly or indirectly to this thesis, for their understanding, help, and moral support throughout my studies. Especially, Che-Kang Chang's unwavering support and encouragement have been a constant source of strength throughout my research journey. The discussions with Chong-Heng Weng and Yueh-Hua Tu also help me polish my ideas. I am truly grateful for the countless hours and immeasurable effort they have dedicated to helping me succeed.

doi:10.6342/NTU202302806

doi:10.6342/NTU202302806

摘要

近年來生成式模型越來越受到重視，尤其是基於 Transformer 或是 Attention 的模型在各個領域都有不少的成果，像是文章、音樂、圖片、影片等等。與此同時，在生成帶文字標籤的圖結構（如知識圖譜、心智圖等）上並沒有太多發展，由於該問題同時牽扯到圖結構的生成與文字標籤的生成，以往的方法大致上會分成兩種，一種是將文字與圖結構分別用兩個不同的模型，另一種則是將圖拆解成一段段的文字序列並使用序列模型來處理。然而，使用兩個模型的方法容易缺少圖結構與文字之間交互的資訊，而將圖拆解成序列的方法則是會損失部分的圖結構資訊並且將低生成效率。本論文提出了一種結構標記，能夠將圖結構與文字共同轉成單一的表示法。透過這種表示法，模型可以更有效率的學習以及生成圖結構與文字，在此之上我們也提出了一種預訓練的方法。為了證明方法的有效性，我們在兩個公開的資料集上做測試，並且結果顯示我們的方法可以用更少的參數量達到跟過去模型可比的分數。

關鍵字：圖結構生成、知識圖譜、深度學習

doi:10.6342/NTU202302806

doi:10.6342/NTU202302806

Abstract

Transformer layer has been proved to work well in several domains beyond text, like audio, image, and even multi-modal. The idea behind these models is that we can treat different kind of input as a series of tokens. Recent research also shown that with carefully designed input token, a pure transformer encoder can also be a powerful graph encoder. Taking steps further in this direction, we propose a new kind of input representation called ”Structure Token”. With structure token, we can represent graph with text label as a sequence of tokens. By converting both graph and text into structure token, we train a pure transformer encoder-decoder that learn a unified representation and generate both graph and text with the same model. We also propose a new pretrain method similar to mBART pre-training but with the structure token. In this paper, we show that with the proposed method, we are able to train a smaller model that has performance comparable to the T5 variants on text-to-graph and graph-to-text tasks.

Keywords: Graph Generation, Knowledge Graph, Deep Learning

doi:10.6342/NTU202302806

doi:10.6342/NTU202302806

Contents

doi:10.6342/NTU202302806

doi:10.6342/NTU202302806

doi:10.6342/NTU202302806

doi:10.6342/NTU202302806

List of Figures

doi:10.6342/NTU202302806

doi:10.6342/NTU202302806

List of Tables

doi:10.6342/NTU202302806

doi:10.6342/NTU202302806

Chapter 1 Introduction

Natural Language Processing (NLP) is a field of artificial intelligence that involves the use of computational methods to analyze and generate human language. NLP has become increasingly popular in recent years, as the amount of natural language data available on the internet has grown exponentially and the adoption of deep learning methods has gain significant advancements. One important area of research in NLP is to extract structured information from unstructured text since structured information can be processed more efficiently and easily by both human and machine. For example, Automatic Knowledge Graph Construction (AKGC) is a task that aims to automatically capture the entities and relationships in the text as graphs [55]. Our work presents a new method that can be used to generate knowledge graph from text.

In this chapter, we first introduce the background in Section 1.1 and the common method used for graph generation in Section 1.2. Then we describe the motivation of this work in Section 1.3 and list our contributions in Section 1.4. Last, the organization of this thesis is presented in Section 1.5.

doi:10.6342/NTU202302806

1.1 Graph Structure in Natural Language Processing

Graph is a mathematical representation of a set of objects and the connections between them. In the Computer Science domain, a graph usually refers to a data structure consisting of something that can be regarded as nodes and edges. There are many variants of graph and each has its own attributes. Some of them only care about the sparsity and locality of the edges (i.e. the structure), while some require extra information like node labels or edge labels. In NLP, nodes are often used to represent concepts or entities, and edges are used to represent the relationships. Since the labels are all texts, we will refer to this kind of graph as "text graph" in this thesis.

Text graph is pretty common in linguistics. For example, we can use graph to express linguistic relations between words, phrases, and sentences. The relations can be either syntactic or semantic. These kind of representations have been widely used in NLP, such as constituency parsing or dependency parsing. Besides, graph have many other applications in NLP. In the dialog system, we can use graph to capture the dialogue states [38]. Alternatively, graph can be used to represent the entities and attributes in a knowledge base of a question answering system [52, 54].

Despite the usefulness of graph, obtaining the graph can be challenging. For instance, in the question answering system, we want to store the information or knowledge in a structured format. However, most of the language data available are in the form of unstructured text. Manually extracting important information from these texts is resource consuming. Therefore, automatic methods for constructing the graph have become an important task.

doi:10.6342/NTU202302806

1.2 Common Methods for Generating Text Graph

The key aspect of automatic graph construction is the way that the graph is generated. Unlike those graph without labels, graph generation in NLP is a complex problem since we need to find out not only the structure but also the labels. As a result, we either need a model that can directly handle both structure and labels at the same time, or we use different models for each parts.

Several methods have been proposed for graph generation in NLP, each with its own trade-offs. In the following section, we talk about two common methods for generating graph, the multi-stage approach and graph linearization approach.

1.2.1 Multi-stage Approach

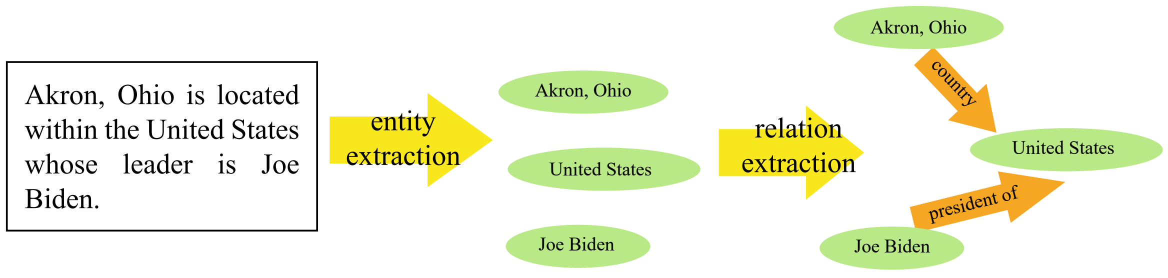


Figure 1.1: Example of multi-stage approach

The multi-stage approach involves generating a graph in multiple stages [19, 35]. The idea is that we can divide the graph generation problem into an entity recognition from text problem and a relation extraction from entities problem. In the first stage, a node model is used to extract the entities from the text into a set of nodes. Second, an edge model is used to find the relationships among all the nodes. This would require the edge model to take all possible pairs of nodes and predict whether they have any relationship between them or not. The most common implementation would be treating no relationship as a "no relation" label and perform the inference assuming that the graph is a complete graph [35].

Although the multi-stage approach simplified the graph generation problem, it also introduce a label imbalance issue [35]. Because the prediction of edge model need to include a "no relation" label, that label would be the majority in the dataset. The issue make it more difficult to train the edge model [35]. Moreover, since we are using two model in a chain, it is also possible to suffered from error propagation.

1.2.2 Graph Linearization

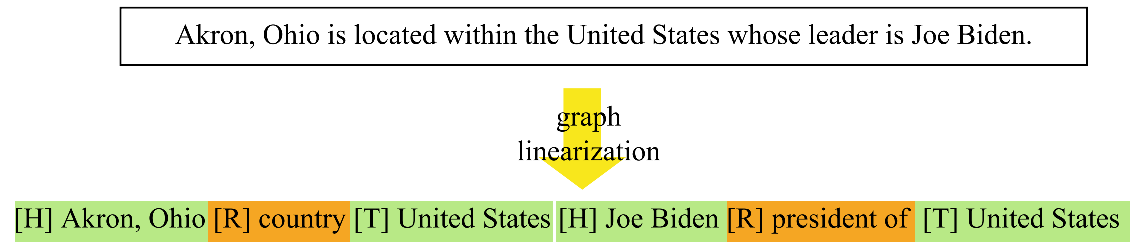


Figure 1.2: Example of graph linearization approach. The "[H]", "[R]", and "[T]" is a special token indicating the following text is a "head", "relation", or "tail", respectively. A group of head, relation, and tail is a triple and the order of the triples can be randomly permuted.

An alternative to the multi-stage approach is graph linearization, which refers to a process that convert a graph into a sequence of object that can represent the origin graph. It treats the graph generation task as a sequence generation problem [2]. There are many different ways to represent graph with sequence. In NLP, we usually represent the graph with sequence of triple (a tuple with 3 elements) containing the edge and the two nodes on each side of that edge [2, 11, 13]. With this approach, we can directly transfer the technique and models used in sequence modeling to the graph modeling problem. Meanwhile, it do not need the "no relation" label as the one in multi-stage approach.

However, if any node have more than one node connecting to it, the same node would

doi:10.6342/NTU202302806

appear in two different triple in the sequence. In other word, the length of the sequence of triple will be longer than the sum of all the label in that graph. As a result, the model would require more computation resources for larger capacity.

1.3 Motivation

As mentioned in the previous section, the multi-stage approach suffers from a label imbalance issue and possible error propagation due to its two-stage nature. On the other hand, the graph linearization approach can be inefficient due to the possible duplication of nodes in the sequence of triples, which could increase the computation resources required. Therefore, we aim to design a new method similar to the graph linearization approach but do not need to duplicate any element in the graph.

Additionally, we want to unified the representation for text graph and text. On one hand, the text in a text graph usually follows the same language rules as normal text. On the other hand, some researches have shown that language model trained on unstructured text also implicitly learn some structured features [21, 42]. Correspondingly, we should be able to share the model for both kind of data to some extent. Moreover, one challenge for AKGC is the lack of paired data. One solution is using the semi-supervised learning technique [19, 48], that we iteratively train two model to generate the data pair as new training data. If we can unified the representation for both text graph and text, then we can reduce the number of models in semi-supervised learning to only one model.

doi:10.6342/NTU202302806

1.4 Contribution

In this thesis, we propose a new method addressing the desired characteristic above.

1. We present a new representation, called "Structure Token", which served as a data format for both graph and text. This approach can be viewed as a different graph linearization without duplication. With this representation, we can train a single model that is capable of generating both text graph and text.
2. We designed a pretrain method above our structure token. We show that with the proposed method, we can effectively and efficiently train models comparable to other approaches.

1.5 Thesis Organization

The remainder of this thesis is organized as follows:

In Chapter 2, we provide the essential prerequisites related to this thesis. We start from the basics of generative model and the main neural network architecture we are using. Then we talk about some training technique such as pretraining and cycle training. Last, we introduce some common metrics for evaluate text generation and graph generation.

In Chapter 3, we describe our proposed method in detail. We will talk about how the data is represented with the structure token and how to convert structure tokens into numeric features for the model to process. Then the training method is also presented in this chapter.

In Chapter 4, we present our experimental setup, including the datasets used, evalu-

doi:10.6342/NTU202302806

ation metrics, and the implementation details of our method. We will discuss the results of our experiments, comparing the performance of our method with existing methods.

Finally, in Chapter 5, we conclude the thesis with a summary of our contributions and a discussion of potential future work.

doi:10.6342/NTU202302806

doi:10.6342/NTU202302806

Chapter 2 Preliminaries

This chapter is dedicated to discussing the essential background information required to understand the problem we tackled and the solution we proposed in this thesis. We start with a fundamental understanding of generative models, particularly in the context of text and graph generation, and then move onto the primary model for our method, the Transformer model [46]. We explain the attention mechanism and position embedding, key components of the Transformer model. We also discuss the significance and benefits of pre-training and fine-tuning in improving model performance. Lastly, we outline the metrics for evaluating text and text graph generation used in this thesis.

2.1 Introduction to Generative Model

Generative models are a class of models that aim to learn the true data distribution of the training set in order to generate new data points from the same distribution. In the context of NLP and graph generation, generative models can be designed to generate coherent sequences of text or connected graphs.

doi:10.6342/NTU202302806

2.1.1 Basics of Text Generation

In order to generate texts of natural language, we want to train models that learn the distribution of natural texts. However, it is almost impossible to find the true distribution of natural languages. Instead, we use the joint probability of the tokens in the sequence, like the characters or words in the sentences. Moreover, to make it suitable for generation, the joint probability is rewritten in an autoregressive manner. That is to say, the probability of the k-th token only depends on the previous $k - 1$ tokens. Autoregressive models generate sequences token by token, with each token being generated based on all previously generated tokens or manually inputted tokens. This property makes autoregressive models particularly suitable for sequential data, where the context, i.e., preceding tokens, is crucial in deciding the next token.

One common category of model in the field is the Sequence-to-Sequence (Seq2Seq) model [45], which is used in tasks such as machine translation that accepting sequence as input and generating corresponding sequence as output. The Seq2Seq model usually contains two part, a encoder model and a decoder model. The encoder model is responsible for converting the input sequence to feature representation. On the other hand, the decoder model will take the previously generated tokens and the feature representation from encoder model to generate a new token. The decoder model is often designed as an autoregressive model, while the encoder model can be any kind of model.

Originally the design of Seq2Seq model is to convert the sequence of the encoder domain to the decoder domain [45]. People found that it is possible to generate sequences from multiple domains [25]. Since the autoregressive decoder model is conditioned on the encoder feature and some preceding tokens, we can switch the domain of the decoder by

doi:10.6342/NTU202302806

preceding different tokens. For example, in the machine translation scenario, instead of training the decoder on one language, we can use a special language token as the preceding token to switch the language we want to translate to [25, 33].

2.1.2 Basics of Graph Generation

The generation of graph is more complicated than generating text. Unlike the sequential nature of text, graphs inherently embody more intricate structures. In contrast to text generation, it is more challenging to model the distribution of graph and express the probability in a generation-friendly way. Several types of methods have been proposed for generating different graphs [56]. For example, GraphRNN is an autoregressive model that generate graph node by node and depending on the previously generated subgraphs [51].

However, the task is even more complicated in the context of NLP because text generation becomes part of the text graph generation. Since the task include two kind of generation problem, the intuitive solution would be either cooperating multiple models or adapting methods that generate graph and text in a similar manner such as autoregressive models. As mentioned and explained in Section 1.2, the multi-stage approach and graph linearization approach are two prevalent methods.

2.2 Introduction to Transformer Model

The Transformer model, introduced by Vaswani et al. [46], is a powerful model architecture designed to handle sequence data. They employ a mechanism called "attention" to capture the dependencies between tokens, no matter how far apart they are in the se-

doi:10.6342/NTU202302806

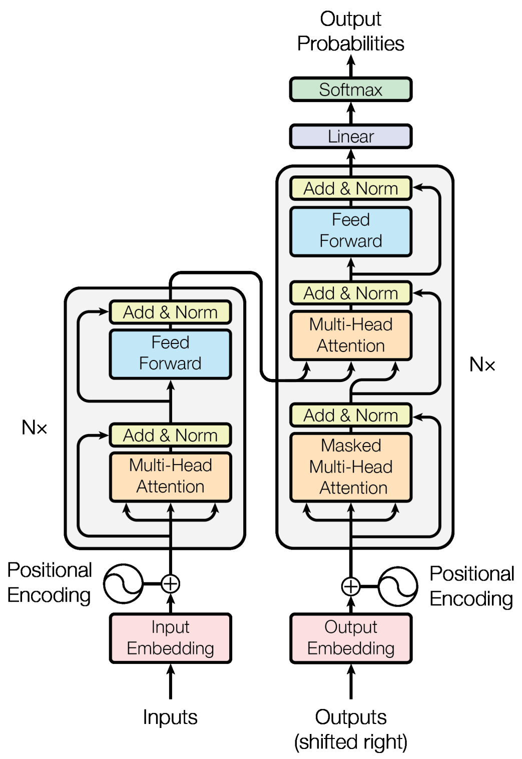


Figure 2.1: Illustration of Transformer model from Vaswani et al. [46]

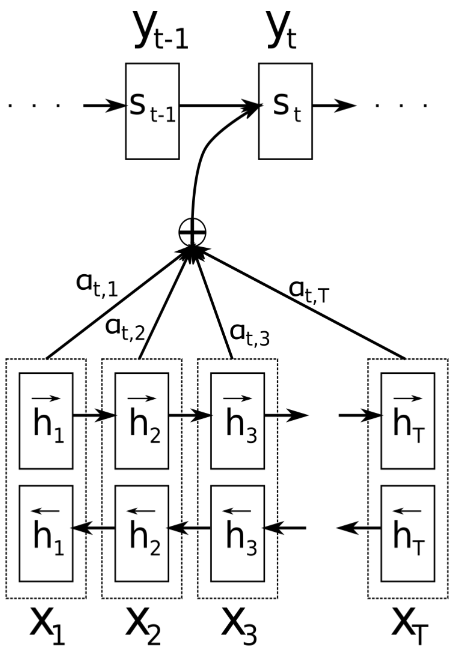
quence. This capability enables Transformers to model complex language structures and generate high-quality text. It has revolutionized the field of natural language processing and is the foundation for state-of-the-art pretrain models like BERT, GPT, and many others [12, 30, 40]. There are many variant of Transformers and they have also achieved considerable success in many tasks other than NLP [31].

2.2.1 The Core Module: Attention Mechanism

The concept of attention was first introduced in a paper for machine translation [4]. As indicated by the name, the attention mechanism allows the model to "focus" on different parts of the input when producing an output. In the paper, the attention is used to enhance the decoder model so that it can learn to align the translated word to the origin words, as showed in Figure 2.2. The attention operation computes a weighted sum of all input features, where the weights are determined by a similarity function taking an input

doi:10.6342/NTU202302806

token and an output token.



y_{t-1}
y_t
s_{t-1}
s_t
$a_{t,1}$
$a_{t,2}$
$a_{t,3}$
$a_{t,T}$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
X_1 X_2 X_3 X_T

Figure 2.2: Illustration of the origin attention from Bahdanau et al. [4]. The weight $\alpha_{t,i}$ is computed by the similarity function applying on s_t and h_i

In the Transformer paper, they made two modifications to the original attention operation [46]. First, the attention operation is generalized. Instead of computing the weights and weighted sum on the same sequence, they use two sequence so that the weight can be decoupled from the weighted features. This is often referred as query-key-value attention (or QKV attention for short), where “query” is the output token, “key” are the input tokens, and “value” are the weighted features. Second, they propose the idea of multi-head attention, which simply concatenate the result of multiple attention. With multi-head, the model can learn to focus on multiple part at the same time. Beside the modifications, the Transformer paper also introduce the idea of self attention, which compute the attention on the input sequence itself instead of on the output sequence and input sequence. This enable the model to ignore irrelevant tokens and get better features from the interaction between tokens.

doi:10.6342/NTU202302806

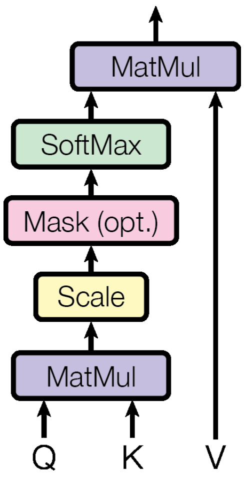


Figure 2.3: Illustraction of the QKV attention operation from Vaswani et al. [46]. The expression is in matrices form. The result of "SoftMax" is the attention weights.

2.2.2 Handling Sequential Data: Position Embedding

As we can see in the design of attention mechanism, only the choice of similarity function and the input features would affect the result of the attention operation. The order of tokens in the sequence will be ignored due to the weighted sum operation. Different from the original attention paper, which use recurrent neural network (RNN) to get the order-aware sequence features, Transformer only use attention and feed-forward layer in the model [4, 46]. Therefore, the Transformer model cannot handle the order in the sequence. To overcome this problem, they propose the idea of position embedding, which assign a vector to each position of the sequence. The input feature become the addition of word embedding and position embedding. The position embedding is designed to have higher similarity with closer positions, hence affect the result of attention.

There are many variant of position embedding [14]. Generally position embedding is a function converting positional information such as absolute indices or relative distance to vector representation. The vector representation can either be carefully and manually designed or joint trained with the other part of model. For example, in the Transformer

doi:10.6342/NTU202302806

paper, they use periodic functions like sin and cos with several frequencies depending on the index [46]. On the other hand, in the BERT model, the position embedding is a set of vectors joint trained with the model, hence the allowing context length being fixed [12].

2.3 Transformer for Graph Data

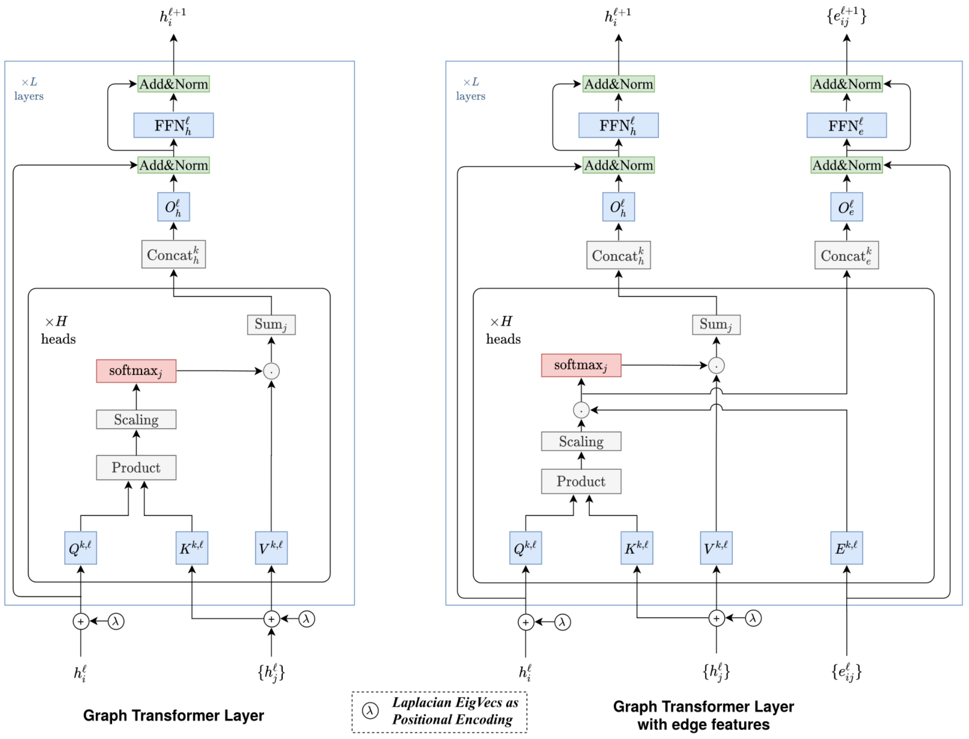

Graph Transformer Layer
Laplacian EigVecs as Positional Encoding
Graph Transformer Layer with edge features

Figure 2.4: Illustration of Graph Transformer from Dwivedi et al. [15]. Comparing to Figure 2.1 and Figure 2.3, the architecture is almost the same as Transformer for sequence.

In the previous section, we mentioned that the attention mechanism is actually order-agnostic without the position embedding. This makes it suitable for operating on set input. We can even view the input features as nodes and the attention weights as adjacency matrix. Then the attention become an aggregation of node features depending on the adjacency matrix, similar to the graph convolution [28]. Therefore, Transformer model can also be regarded as graph neural network (GNN). For instance, the Graph Transformer paper [15] shows how to adapt Transformer model to graph with minimum modifications,

doi:10.6342/NTU202302806

as illustrated in Figure 2.4.

Multiple attempts have been made to apply Transformer model on graph data [15, 26, 36, 50]. Despite the resemblance of Transformer and GNN, non-Transformer models remain dominant in graph related tasks [16]. Correspondingly, some of the works focus on enhancing Transformer with some GNN module [36]. On the other hand, some works have been made to prove that the Transformer itself can be as powerful as other GNN. In the paper of Graphormer [50], some graph-specific enhancements of attention, as showed in Figure 2.5, are proposed. They show that some of GNN models can be mathematically expressed in Graphormer.

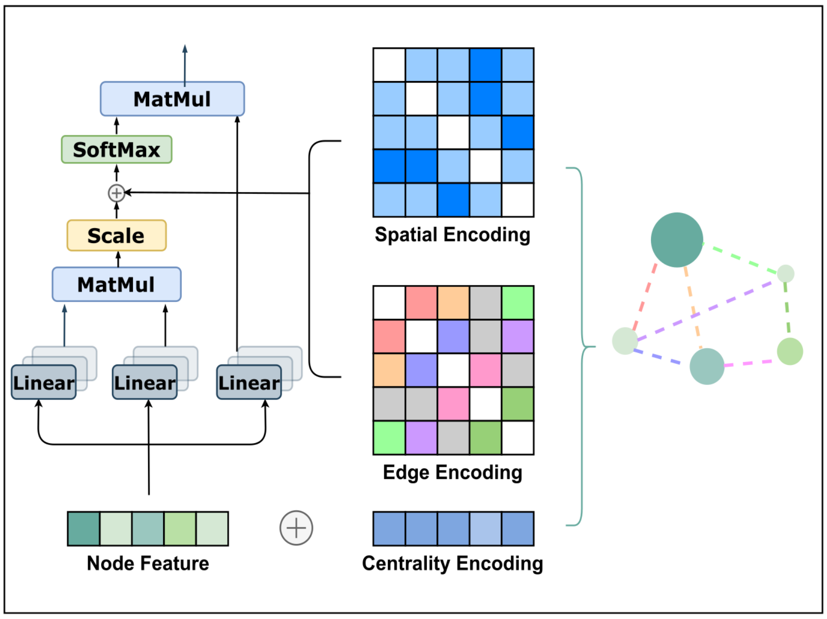


Figure 2.5: Illustration of Graphormer from Ying et al. [50]. The structured information is further encoded and applied to the attention weight.

TokenGT is another work focus on proving the ability of Transformer upon graph data [26]. Instead of modifying Transformer or attention operation, TokenGT convert graph into tokens, as showed in Figure 2.6. This approach share some similarities with the position embedding. As mentioned in Section 2.2.2, the idea of position embedding is to indicate the position of the token in the sequence and affect the calculation of attention

doi:10.6342/NTU202302806

weights. Similarly, TokenGT design some identifiers that can indicate the position in the graph and affect the attention. With this approach, we can use the same Transformer model for sequence without graph-specific modifications.

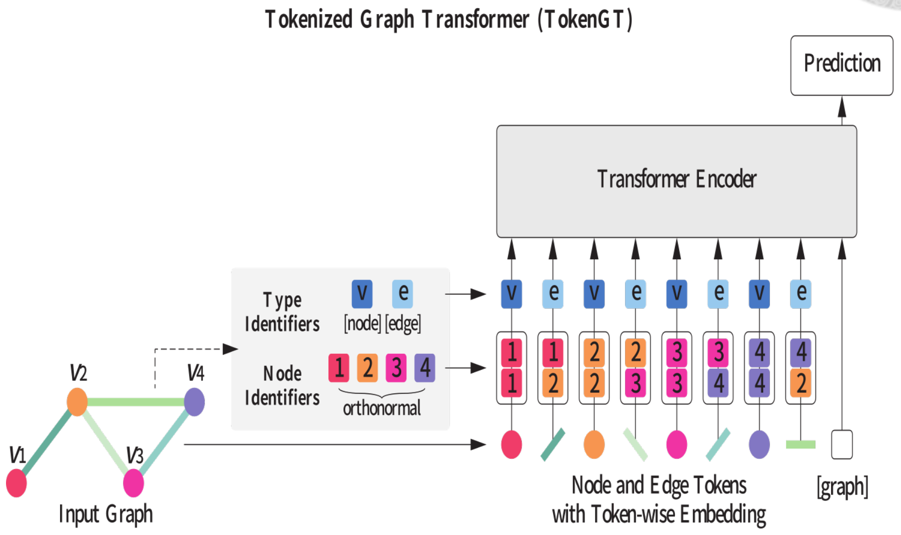


Figure 2.6: Illustration of TokenGT from Kim et al.[26].

On the other hand, the development of Transformer for text graph takes additional directions [49]. In contrast to explicitly using the graph structure in the model, the graph linearization based methods, as introduced in Section 1.2, also gain significant popularity due to the imbalanced development of Transformer for sequence and for graph.

2.4 Improve Model Performance: Pre-Training and Fine-Tuning

Pre-Training and Fine-Tuning is a two-stage training strategy arose in recent years [12, 33, 40, 41]. It shows a significant improvement of performance over a large variety of NLP tasks. The idea behind is a combination of transfer learning and self-supervised learning. For example, the BERT model is pre-trained on a cloze-like objective that learn the dependencies among words, and then fine-tuned for the downstream tasks such as

doi:10.6342/NTU202302806

question answering, as illustrated in Figure 2.7 [12].

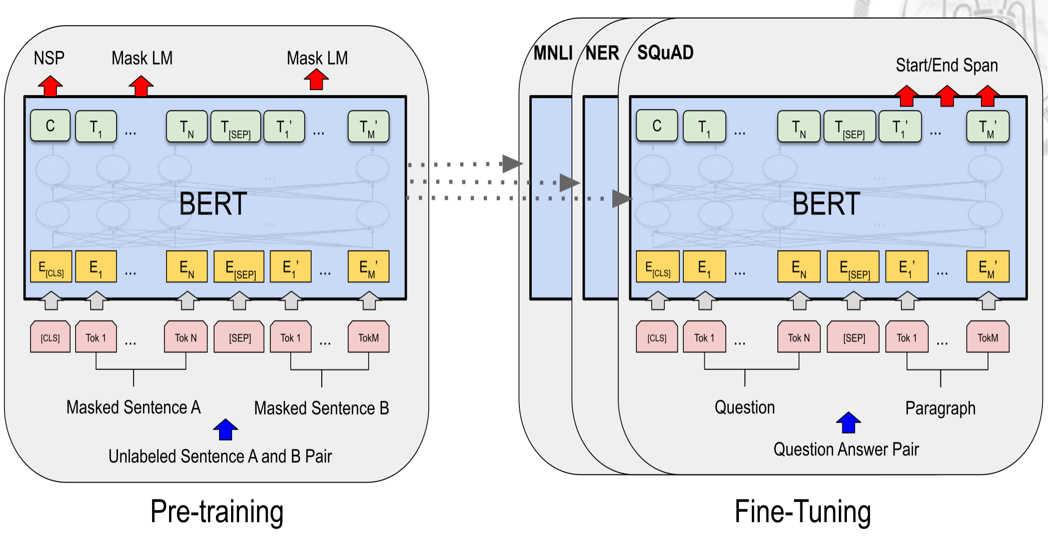


Figure 2.7: Illustraction of BERT training from Devlin et al.[12].

In the pre-training stage, the model is trained on a large amount of unlabeled data with a task-agnostic training objective. This help the model learn general features of the data. The first pre-trained Transformer model, named GPT, was introduced by Radford et al. [40], which is pre-trained on generating sentence based on some prefixes. Although the pre-training can be done with other objectives, the self-supervised scenario suit NLP well due to the amount of language data on the internet. Then in the fine-tuning stage, the model is further trained on the labeled dataset with task-specific objective. This stage allows the model to adapt its learned features to the specific task. Since the model has already learned generic language features in the pre-training stage, it can achieve better performance on the specific task with less training data.

2.5 Evaluate Generation Result

Evaluating the quality of generated texts and text graphs is a critical part of the model development process. It allows us to quantify the performance of our models and to com-

doi:10.6342/NTU202302806

pare different models or different versions of the same model.

2.5.1 Metrics for Text Generation

Metrics for text generation are basically a functions that compare the generated text (or candidate) to reference text(s). There are several metrics designed to evaluate the quality of generated text and each measures different properties [8]. Text generation models are often evaluated using multiple metrics so that we can show how "similar" is the candidate to reference text. Here are the metrics for text generation we used in this thesis:

1. **BLEU (BiLingual Evaluation Understudy) [39]:** BLEU score is a widely used metric that compares n-grams of the generated text to those of the reference text. In general, the value is the number of n-grams from the generated text appearing in the reference text dividing by the total number of n-grams from generated text. It can be viewed as a ratio of overlapping text between candidate and reference text.

2. **METEOR (Metric for Evaluation of Translation with Explicit ORdering) [5]:** Comparing to BLEU, METEOR is a more advanced evaluation. Instead of simply counting the appearance of n-grams, it match the words, synonyms, word stems, and other linguistic phenomena between candidate and reference text to form an alignment. Then the aligned texts is used to calculate a weighted F-score, which is the major part of METEOR.

3. **BERTScore [53]:** BERTScore is a relatively new metric that leverages the BERT language model to embed the generated and reference texts. Unlike BLEU and METEOR that rely primarily on surface-level string matching, BERTScore computes token-level similarity based on the contextualized word embeddings of BERT

doi:10.6342/NTU202302806

model. Therefore, we could get a more accurate matching based on the semantic similarity. Then we use the matches to calculate the F1-score.

2.5.2 Metrics for Text Graph Generation

For text graph generation, it need to compare the generated text graph to the reference text graph. In this thesis, we use the metric from WebNLG2020 [7], which is a triple F1-score. To evaluate the generated text graph, the graph is first converted into a set of triple. Then the generated triples and reference triples is aligned with a triple matching criterion. There are three possible triple matching criteria proposed:

1. **Strict matching:** A generated triple need to be exactly the same as a reference triple to be a match.
2. **Exact matching:** A slightly loosen criterion of strict matching. It treat the triple as a set of three element, so it only require the text to be the same.
3. **Partial matching:** The loosest matching criterion. Beside treating triple as set, it also allow the text to be partially equal.

By the three matching criteria, we get three triple F1-score for evaluating the generated text graph.

doi:10.6342/NTU202302806

Chapter 3 Method

In this chapter, we elucidate the foundation and design of our approach for structured data representation and generation. An overview of our method is given in Section 3.1. Then in Section 3.2, we detailed explain the proposed method. Last, we introduce the training strategy corresponding to our model in Section 3.3.

3.1 Model Overview

As mentioned in Section 1.3, the goal of this thesis is to develop a new method suitable for representing and generating both texts and text graphs without redundant computation. Our method arise from the TokenGT approach, illustrated in Figure 2.6 [26]. TokenGT, introduced in Section 2.3, converts graph into tokens containing labels and graph-level identifiers. Since the main model used in TokenGT is primarily the unmodified Transformer encoder, it shows a possibility of multi-modality Transformer trained on both graph data and text data [26]. However, the idea does not directly fit in our scenario of text graph for two reasons. First, a single graph element (node or edge) in TokenGT need to be representable by a single token. On the contrary, it would require multiple tokens for an element of text graph because the label is a multi-token text. Second, TokenGT only focus on representing the graph, while we are interested in graph generation

doi:10.6342/NTU202302806

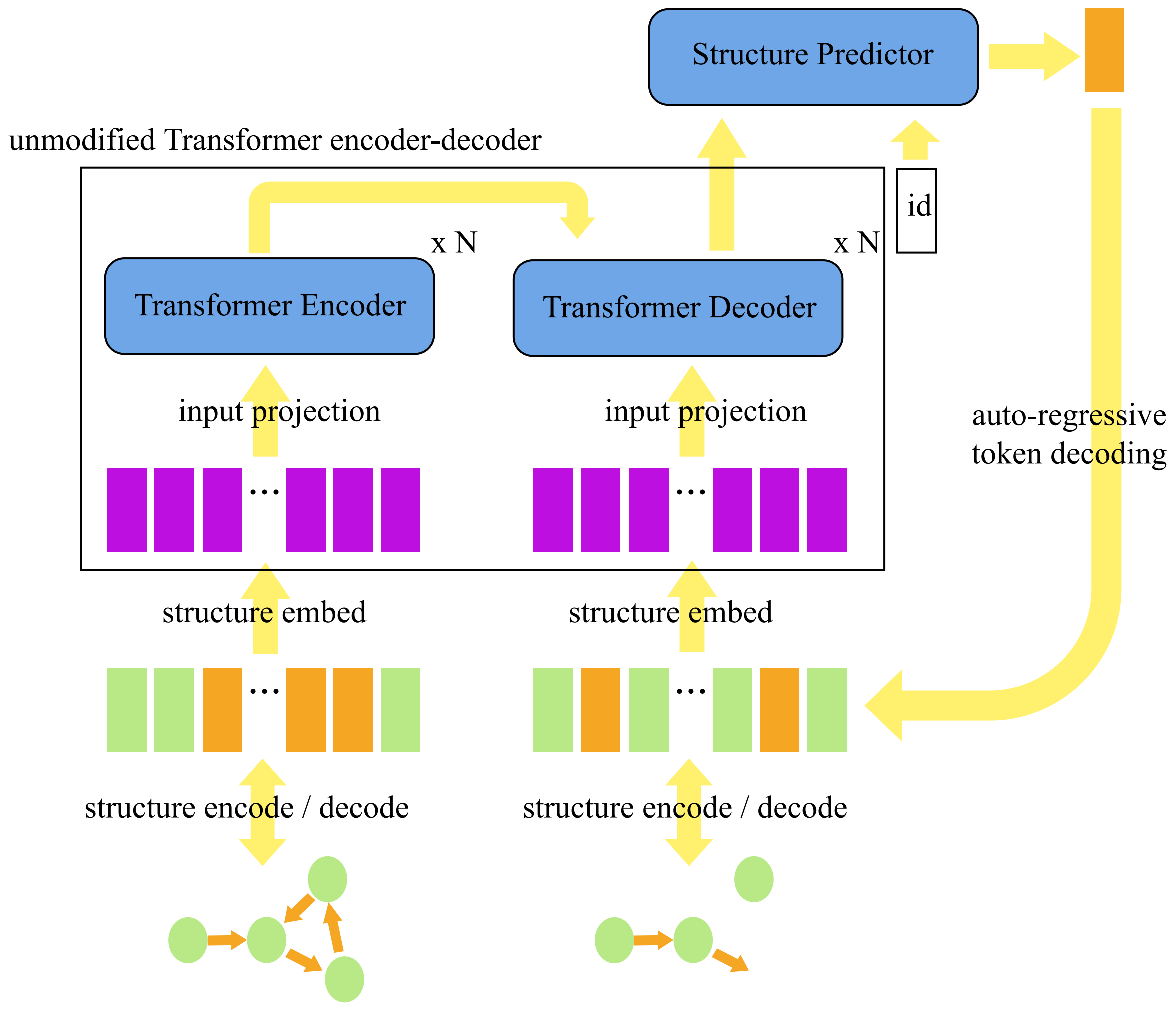


Figure 3.1: Overview of the proposed structure token approach. The graph on the left is the input text graph. The model take the input text graph and autoregressively decode a new structure token to form the generated subgraph (the graph on the right). The detail of the structure predictor is expanded in Section 3.2.5.

doi:10.6342/NTU202302806

as well. Based on the notion and challenge, we develop a new method that address the desired properties.

Our method employs a concept we called “Structure Token” which losslessly encode the text graph. The overview of our model is illustrated in Figure 3.1. The idea behind is straightforward. Normal text generation model autoregressively generate a new token. The generated tokens are combined into a subsentence that would potentially be fed back to the model for another generation step. Likewise, our model generate a new structure token autoregressively and the generated tokens form a subgraph, as illustrated in Figure 3.2. A structure token is a small piece of a text graph containing a text token of a graph element and a few identifiers for the precise location of the text token in the graph. We encode the text graph into structure tokens and then embed the tokens into vector representations. Meanwhile, we treat text as a special graph without any edge, so text can also be encoded into structure tokens. Our model incorporates an unmodified Transformer Seq2Seq model and a structure predictor that take the vector representations and predicting a new structure token. Once the generation stop, we can decode the sequence of structure tokens into the target text graph.

Since our method convert text graph into sequence of structure tokens, our method can be viewed as graph linearization approach. However, previous linearization method mainly focus on aligning graph to text. That is to say, the graph or subgraph is regarded as a special sentence. Even though the special sentence can be converted back to the graph, the model need to learn the implicit “graph grammar” of the special sentence for handling the structure information, which introduce an extra complexity to the model. On the contrary, our method aligns text to graph. The structure information is explicitly encoded in the tokens. Moreover, our method leverages the design of TokenGT, which provide a better

doi:10.6342/NTU202302806

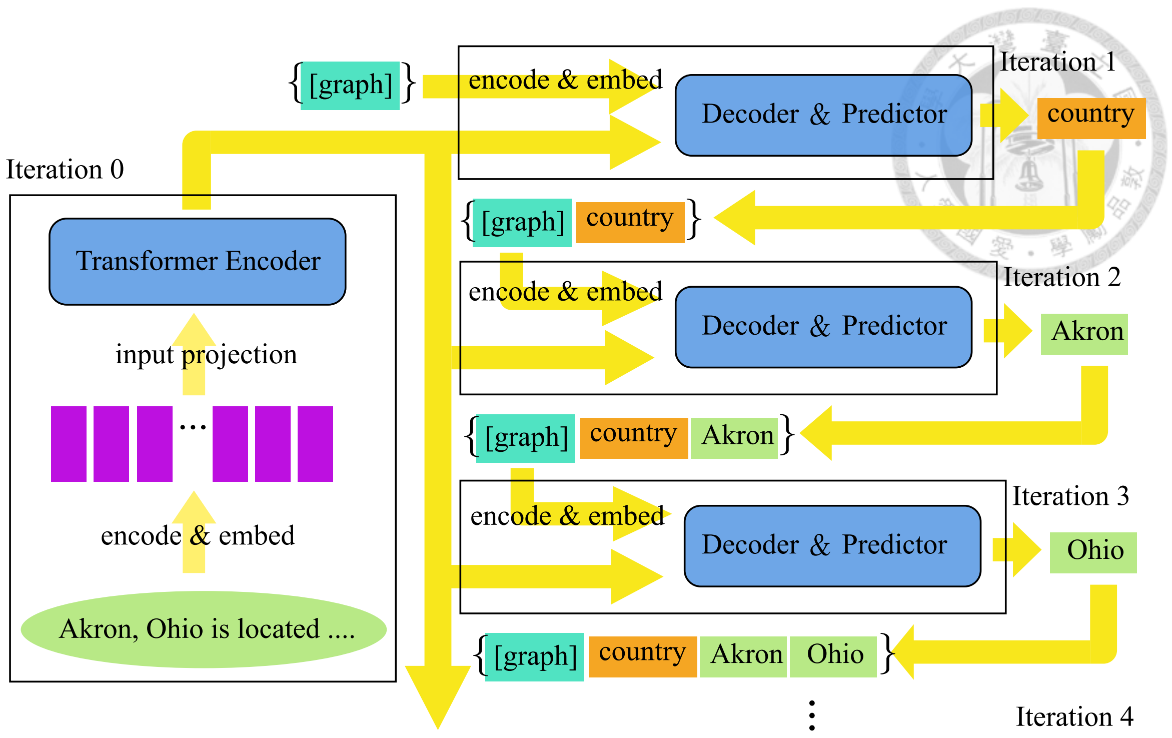


Figure 3.2: Example of autoregressive generation with our structure tokens using the same text-to-graph generation example as Figure 1.1. "[graph]" is a domain token explained in Section 3.2.2.

theoretical foundation for learning graph features [26].

3.2 The Core Design: Representing Graph via Structure Token

In this section, we present the design of our method. We rigorously define the structure token representation of text graph in Section 3.2.1 and Section 3.2.2. Based on the definition, we describe the embedding method for structure token in Section 3.2.3. Then the generation method is presented in Section 3.2.4 and Section 3.2.5. Lastly, we show our method is more efficient than graph linearization in Section 3.2.6.

doi:10.6342/NTU202302806

3.2.1 Problem Setup

To begin with, we elucidate some of the terminologies and setups we used in this thesis. When generating texts, the model perform the prediction on a fixed set of possible text tokens. The text token is a small piece of text that constitute the text label, while the text label is a sequence of text tokens. For example, words or characters could be the text tokens that constitute a sentence, which could be the text label. We pair the text tokens and their positions in a text label so that we can express the text label as a set instead of a sequence. As an instance, the text label "I like bird" can be express as $\{(\text{"I"}, 1), (\text{"like"}, 2), (\text{"bird"}, 3)\}$ where "bird" is a text token and 3 is the position. The text graph is a structure containing nodes and edges. The nodes and edges contain text labels, as illustrated in Figure 3.3. We define an arc as a triple of head node, edge, and tail node. Then the text graph can be defined as a pair of node set and arc set. In order to distinguish between nodes (or edges) with same text label, we assign each nodes (or edges) with an unique id. As a consequence, we define a node (or edge) as a pair of text label and its unique id. Consider the text graph in Figure 3.3 as an example. There are five text labels: $s_1 = \{(\text{"Akron"}, 1), (\text{","}, 2), (\text{"Ohio"}, 3)\}$, $s_2 = \{(\text{"country"}, 1), \}$, $s_3 = \{(\text{"United"}, 1), (\text{"States"}, 2)\}$, $s_4 = \{(\text{"president"}, 1), (\text{"of"}, 2)\}$, $s_5 = \{(\text{"Joe"}, 1), (\text{"Biden"}, 2)\}$. The node set of the text graph would be $\{(s_1, 1), (s_3, 2), (s_5, 3)\}$, and the arc set become $\{((s_1, 1), (s_2, 4), (s_3, 2)), ((s_5, 3), (s_4, 5), (s_3, 2))\}$.

Having the idea in mind, we provide the mathematical setup of our method and the formal definition of text and text graph. Let $\mathcal{T}$ be the set of all possible text tokens. As mentioned above, a text label is defined as a set of pairs containing the text token and positions. We generalize this definition by replacing the position with a contiguous

doi:10.6342/NTU202302806

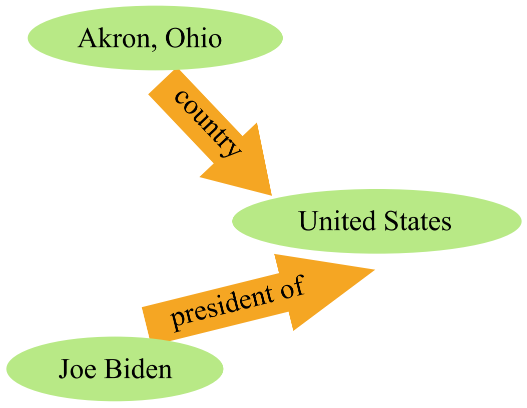


Figure 3.3: Example of a text graph.

sequence of unique id. Given an infinite unique id sequence:

$$\mathcal{I} = (\text{id}_i)_{i=0}^{\infty} \quad \text{where} \quad \text{id}_i \in \mathbb{Z}^+ \wedge \text{id}_i \neq \text{id}_j \text{ if } i \neq j \tag{3.1}$$

Each id_i is a positive integer. We also define $\text{id}(i) = \text{id}_i$ for simplicity. By picking a corresponding id sequence, we can use any positive integer sequence as the positions. Then a text label **S** of length l is a set of token-id pairs defined as:

$$\mathbf{S} = \{(t_i, \text{id}_{j+i}) \mid 1 \leq i \leq l, t_i \in \mathcal{T}\} \subseteq \mathcal{T} \times \mathbb{Z}^+ \tag{3.2}$$

We can conditionally specify the start point $j \in \mathbb{N}$ with $\mathbf{S}^j$. Let $\mathcal{S}$ be the set of all possible text labels and $\mathcal{S}^j$ for specific start point. With the id sequence $\mathcal{I}$, the positions $(p)_{p=1}^{l}$ can be replaced with $(\text{id}_{j+k})_{k=1}^{l}$. Then we can union text labels without missing information by picking non-overlapped id sequences, which is a desired property for the attention operation. For example, the text labels of Figure 3.3 mentioned above is $\{s_1, s_2, s_3, s_4, s_5\} \subset \mathcal{S}^0$. If we directly union s_3 and s_5, we cannot know whether $(\text{“United”}, 1)$ is followed by $(\text{“Biden”}, 2)$ or $(\text{“States”}, 2)$.

A text graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ is composed of a node set $\mathcal{N}$ with q nodes and an arc set $\mathcal{A}$ with r arcs. The node set $\mathcal{N}$ is a set of node labels paired with unique ids of the nodes,

doi:10.6342/NTU202302806

defined as:

$$\mathcal{N} = \{(N_i, n_i) \mid 1 \leq i \leq q, n_i \in \mathbb{Z}^+, N_i \in \mathcal{S}\} \subseteq \mathcal{S} \times \mathbb{Z}^+ \tag{3.3}$$

where N_i is the node label and n_i is the corresponding node id. Similarly, a edge set $\mathcal{E}$ is a set of edge labels paired with unique ids, defined as:

$$\mathcal{E} = \{(E_i, e_i) \mid 1 \leq i \leq r, e_i \in \mathbb{Z}^+, E_i \in \mathcal{S}\} \subseteq \mathcal{S} \times \mathbb{Z}^+ \tag{3.4}$$

where E_i is the edge label and e_i is the edge id. Notably, the id used in $\mathcal{N}$ and $\mathcal{E}$ are disjoint. For example, the node set of Figure 3.3 is $\{(s_1, 1), (s_3, 2), (s_5, 3)\}$ and the edge set is $\{(s_2, 4), (s_4, 5)\}$.

Then the arc set is defined as:

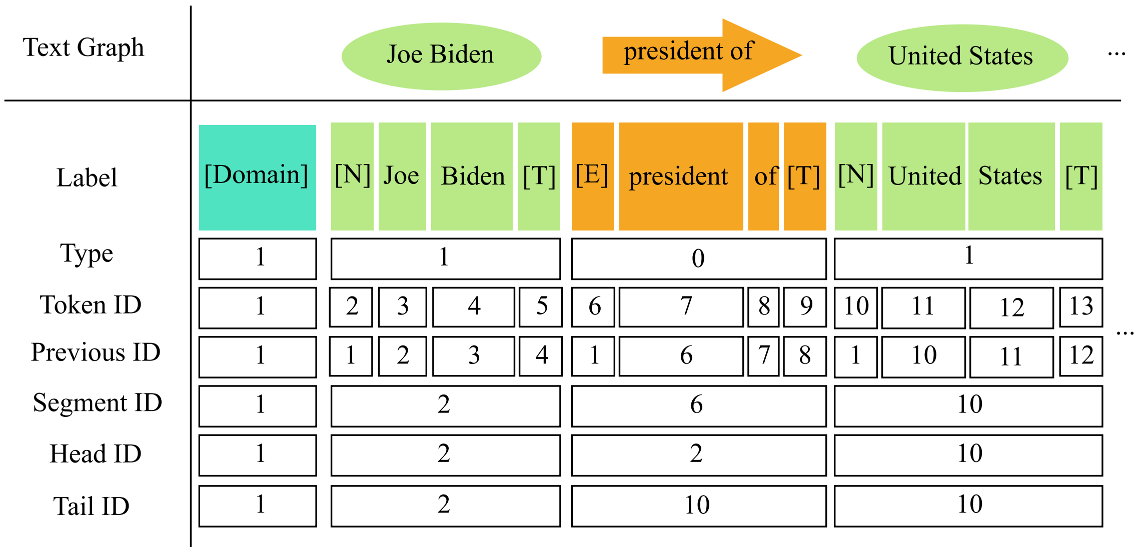
$$\mathcal{A} = \{(\mathbf{N}_i^h, \mathbf{E}_i, \mathbf{N}_i^t) \mid 1 \leq i \leq r, \mathbf{N}_i^h, \mathbf{N}_i^t \in \mathcal{N}, \mathbf{E}_i \in \mathcal{E}\} \subseteq \mathcal{N} \times \mathcal{E} \times \mathcal{N} \tag{3.5}$$

where $\mathbf{E}_i$ is the edge and $\mathbf{N}_i^h$, $\mathbf{N}_i^t$ is the head node and tail node, respectively. For example, the arc set of Figure 3.3 is $\{((s_1, 1), (s_2, 4), (s_3, 2)), ((s_5, 3), (s_4, 5), (s_3, 2))\}$ and $((s_5, 3), (s_4, 5), (s_3, 2))$ is an arc containing the head node $(s_5, 3)$, the edge $(s_4, 5)$, and the tail node $(s_3, 2)$. With these definitions, we will show how the conversion is done in the following sections.

3.2.2 Convert Graph/Text to Structure Token

The proposed structure token is a data representation that can losslessly encode all data in a text graph as a sequence of tokens. As defined above, a text graph is a pair of two sets. In order to convert text graph to structure tokens, we express the two sets of different components into one unified structure of graph elements. Each graph element will be represented by multiple structure tokens. A structure token consist of seven parts:

1. **Label:** The text token t_i of a graph element.
2. **Type:** A binary indicator specifying whether this graph element is a node or an edge.
3. **Token ID:** An unique id for this token.
4. **Previous ID:** The unique id for the previous token. The correct order of the text tokens can be reconstruct using the previous id.
5. **Segment ID:** An unique id for the graph element.
6. **Head ID:** If this token is part of an edge, head id is the segment id of the head node. Otherwise, the head id is the segment id of itself.
7. **Tail ID:** If this token is part of an edge, tail id is the segment id of the tail node. Otherwise, the tail id is the segment id of itself.



Label	[Domain]	[N]	Joe	Biden	[T]	[E]	president	of	[T]	[N]	United	States	[T]	
Type	1	1				0				1				
Token ID	1	2	3	4	5	6	7	8	9	10	11	12	13	
Previous ID	1	1	2	3	4	1	6	7	8	1	10	11	12	...
Segment ID	1	2				6				10				
Head ID	1	2				2				10				
Tail ID	1	2				10				10				

Figure 3.4: Example of structure token. Each column is a structure token.

With these information, we are able to differentiate between structure tokens if they are from different part of the graph. A real example of text graph and corresponding structure tokens can be found in Figure 3.4. The idea of type, head id, and tail id are inherited from TokenGT, which use identifiers to indicate the connection [26]. We modify their definition and introduce extra identifiers for our text components. The segment id, head id, and tail id are graph-level identifiers. For tokens of a specific graph element, the graph-level identifiers of each token will be the same. On the other hand, the token id and previous id are text-level identifier. The text order is determined by the token id and previous id for reconstructing the text label.

Furthermore, We add an extra "domain token" to the structure tokens of a text graph to indicate the domain of the graph, like the special language token mentioned in Section 2.1.1 [25, 33]. With the domain, we can specify what kind of data the text graph is holding. For example, since text is treated as a text graph without any edge, we use a "text" domain token indicating that this text graph represent a text. Besides, we use the domain token as the first token of every text label, so the first previous id of all text labels are point to the domain token.

With the setup in mind, the formal definition of the structure token representation is defined as a set of septuples (tuple with 7 elements). Given a text graph $\mathcal{G}$ and its

doi:10.6342/NTU202302806

components:

$$
\begin{aligned}
&\mathcal{G} = (\mathcal{N}, \mathcal{A}) \\
&\mathbf{N}_i = (N_i, n_i) \in \mathcal{N}, \quad 1 \leq i \leq |\mathcal{N}| \\
&\mathbf{A}_j = ((N_j^h, n_j^h), (E_j, e_j), (N_j^t, n_j^t)) \in \mathcal{A}, \quad 1 \leq j \leq |\mathcal{A}| \\
&\mathbf{E}_j \in \mathcal{E} = \{(E_j, e_j) \mid 1 \leq j \leq |\mathcal{A}|\} \\
&l_i^N = |N_i|, \quad l_j^E = |E_j|
\end{aligned}
\tag{3.6}
$$

where l_i^N, l_j^E are the length of the text labels. We define two sequences:

$$
\begin{aligned}
&\mathcal{L}^N = (\mathbf{L}_k^N)_{k=1}^{|\mathcal{N}|} \quad \text{where} \quad \mathbf{L}_k^N = \begin{cases} 0, & \text{if } k = 1 \\ l_{k-1}^N + \mathbf{L}_{k-1}^N, & \text{otherwise.} \end{cases} \\
&\mathcal{L}^E = (\mathbf{L}_k^E)_{k=1}^{|\mathcal{A}|} \quad \text{where} \quad \mathbf{L}_k^E = \begin{cases} \sum_{k=1}^{|\mathcal{N}|} l_k^N, & \text{if } k = 1 \\ l_{k-1}^E + \mathbf{L}_{k-1}^E, & \text{otherwise.} \end{cases}
\end{aligned}
\tag{3.7}
$$

Without loss of generality, we assign $n_i = \text{id}(\mathbf{L}_i^N + 1)$, $e_j = \text{id}(\mathbf{L}_j^E + 1)$ and specify the start point of each text label such that $N_i \in \mathcal{S}^{\mathbf{L}_i^N}$, $E_j \in \mathcal{S}^{\mathbf{L}_j^E}$. By doing so, we can get the node (or edge) id from the positional ids of its text label. For each node $\mathbf{N}_i$, we define the corresponding structure token representation X_i^N as:

$$
X_i^N = \{(t_k, 1, uid_k, uid_{k-1}, n_i, n_i, n_i) \mid 1 \leq k \leq l_i^N, (t_k, uid_k) \in N_i, uid_0 = \text{id}_0\} \tag{3.8}
$$

The septuple $(t_k, 1, uid_k, uid_{k-1}, n_i, n_i, n_i)$ contains the label t_k, the type 1, the token id uid_k, the previous id uid_{k-1}, the segment id n_i, the head id n_i, and the tail id n_i. For example, $(\text{“States”}, 1, 12, 11, 10, 10, 10)$ is a structure token of the node in Figure 3.4. On

doi:10.6342/NTU202302806

the other hand, for each edge $\mathbf{E}_j$, we define the structure token representation X_j^E as:

$$X_j^E = \{(t_k, 0, uid_k, uid_{k-1}, e_j, n_j^h, n_j^t) \mid 1 \leq k \leq l_j^E, (t_k, uid_k) \in E_j, uid_0 = \text{id}_0\} \quad (3.9)$$

The septuple $(t_k, 0, uid_k, uid_{k-1}, e_j, n_j^h, n_j^t)$ contains label t_k, the type 0, the token id uid_k, the previous id uid_{k-1}, the segment id e_j, the head id n_j^h, and the tail id n_j^t. For example, ("of", $0, 8, 7, 6, 2, 10)$ is a structure token of the edge in Figure 3.4. Then the corresponding structure token representation $\mathcal{G}'$ of the text graph $\mathcal{G}$ is defined as:

$$\begin{aligned} \mathcal{G}' &= \bigcup_{k=1}^{|\mathcal{N}|} X_k^N \cup \bigcup_{k=1}^{|\mathcal{A}|} X_k^E \cup X_D \\ &\subseteq \mathcal{T} \times \{0,1\} \times \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+ \end{aligned} \quad (3.10)$$

where $t_D \in \mathcal{T}$, $X_D = \{(t_D, 1, \text{id}_0, \text{id}_0, \text{id}_0, \text{id}_0, \text{id}_0)\}$ is the domain token. Each septuple $X \in \mathcal{G}'$ is a structure token containing the label, type, token id, previous id, segment id, head id, and tail id.

With this definition, we can represent every possible token id assignment by specifying the unique id sequence $\mathcal{I}$.For instance, Figure 3.4 use a id sequence of $(k)_{k=1}^{\infty}$. On the other hand, since the graph element can be randomly permuted, every possible ordering is also representable with our set $\mathcal{G}'$ by picking the corresponding unique id sequence.

3.2.3 Convert Structure Token to Vector Representation

Once the input data is represented as structure tokens, these tokens are then transformed into fixed-size high-dimensional vector representations for feeding to the model. Since the structure token contain multiple parts, we follow the simple but common method that convert each of the part into vector and concatenate them together to form a larger

doi:10.6342/NTU202302806

vector. Then the vectorized result will be fed to a trainable projection layer for getting the token embedding, as illustrated in Figure 3.5. For the label and type, we simply use the one-hot encoding to convert them to vectors. We denote the one-hot encoding function as $OneHot_n \colon \mathbb{A} \to \mathbb{E}_n$ where $\mathbb{A}$ is a set of n elements and $\mathbb{E}_n$ is the standard basis of $\mathbb{R}^n$. On the other hand, the ids need to be handled differently.

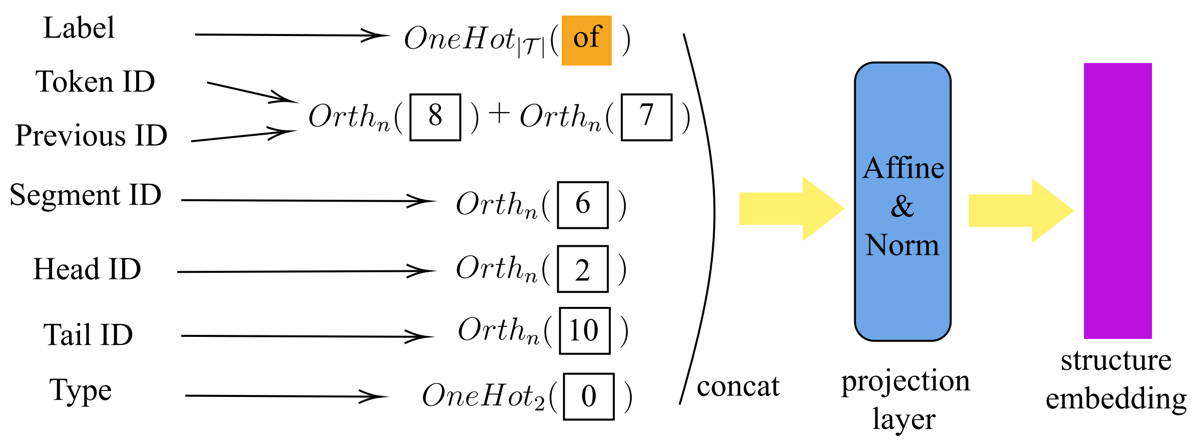


Figure 3.5: Our method for converting structure token to embedding. The function argument is the ”of” token from Figure 3.4.

In order to preserve the graph structure in the tokens with the Transformer model, each id need to be converted into orthonormal vectors as stated in TokenGT paper [26]. Two possible methods for orthonormal vectors are proposed in TokenGT: the eigendecomposition of graph Laplacian matrix or QR decomposition of random Gaussian matrix [26]. However, with eigendecomposition of graph Laplacian matrix, the dimensionality of the generated vectors varied depending on the number of nodes. Although the QR decomposition of random Gaussian matrix can be used to generate fixed-size orthonormal vectors, the dimensionality is proportional to the size of the matrix. To overcome these issues, we loose the requirement of orthonormality. We use a set of orthonormal-like vectors that the dot product of every two vectors is close to zero or less than some thresholds. To get this kind of orthonormal-like vectors, we modify and normalize the sinusoidal position encoding by Vaswani et al. with different frequencies [46].

doi:10.6342/NTU202302806

In the original Transformer paper by Vaswani et al. [46], the sinusoidal position encoding PE at position i is defined as:

$$\begin{aligned} \text{PE}(i) &= \mathop{\Big\|}_{k=1}^{d/2} pe(i,k) \quad \in \mathbb{R}^d \\ pe(i,k) &= \sin\left(\frac{i}{10000^{k/d}}\right) \| \cos\left(\frac{i}{10000^{k/d}}\right) \quad \in \mathbb{R}^2 \end{aligned} \tag{3.11}$$

where d is a predefined dimensionality and $\|$ denote the vector concatenation. We generalize the definition of PE with a frequency function f:

$$\begin{aligned} \text{PE}^*\{f\}(i) &= \mathop{\Big\|}_{k=1}^{d/2} pe^*\{f\}(i,k) \quad \in \mathbb{R}^d \\ pe^*\{f\}(i,k) &= \sin(i * f(k)) \| \cos(i * f(k)) \quad \in \mathbb{R}^2 \end{aligned} \tag{3.12}$$

Then by taking $f'(k) = 10000^{-k/d}$, the original PE can be defined with PE $=$ PE$^*\{f'\}$. We use this generalized position encoding to define a transform function of the ids for our orthonormal-like vectors. The transform function $Orth\colon \mathbb{Z}^+ \to \mathbb{R}^d$ is defined as:

$$Orth_d = norm_2 \circ \text{PE}^*\{-\log(k)\} \tag{3.13}$$

where $norm_2$ is the L2 normalization and $\circ$ denote the function composition. We find that by picking the frequency function $-\log(k)$, the generated vectors satisfied the desired properties. In Figure 3.6, we generate 1024 vectors by applying $Orth$ on $1 \leq i \leq 1024$ and compute the cosine similarity of every possible pair. We can see that the similarity values are mostly close to zero. With this transform function, we are able to convert the ids into vectors and preserve the graph-level features.

Although the design works with graph-level identifiers, it does not work directly with the text-level identifiers. As mentioned in Section 2.2.2, we want the tokens from

doi:10.6342/NTU202302806

nearby positions to have higher attention weights. As a result, instead of concatenate the orthonormal-like vectors, we add the vectors of token id and previous id together, as illustrated in Figure 3.5. By adding the vectors, the dot product value of token a and b become:

$$\begin{aligned}
&(Orth(\text{tid}_a) + Orth(\text{pid}_a)) \cdot (Orth(\text{tid}_b) + Orth(\text{pid}_b)) \\
&= Orth(\text{tid}_a) \cdot Orth(\text{tid}_b) + Orth(\text{tid}_a) \cdot Orth(\text{pid}_b) + Orth(\text{pid}_a) \cdot Orth(\text{tid}_b) \\
&\qquad + Orth(\text{pid}_a) \cdot Orth(\text{pid}_b) \\
&= Orth(\text{tid}_a) \cdot Orth(\text{pid}_b) + Orth(\text{pid}_a) \cdot Orth(\text{tid}_b) + \varepsilon
\end{aligned} \tag{3.14}$$

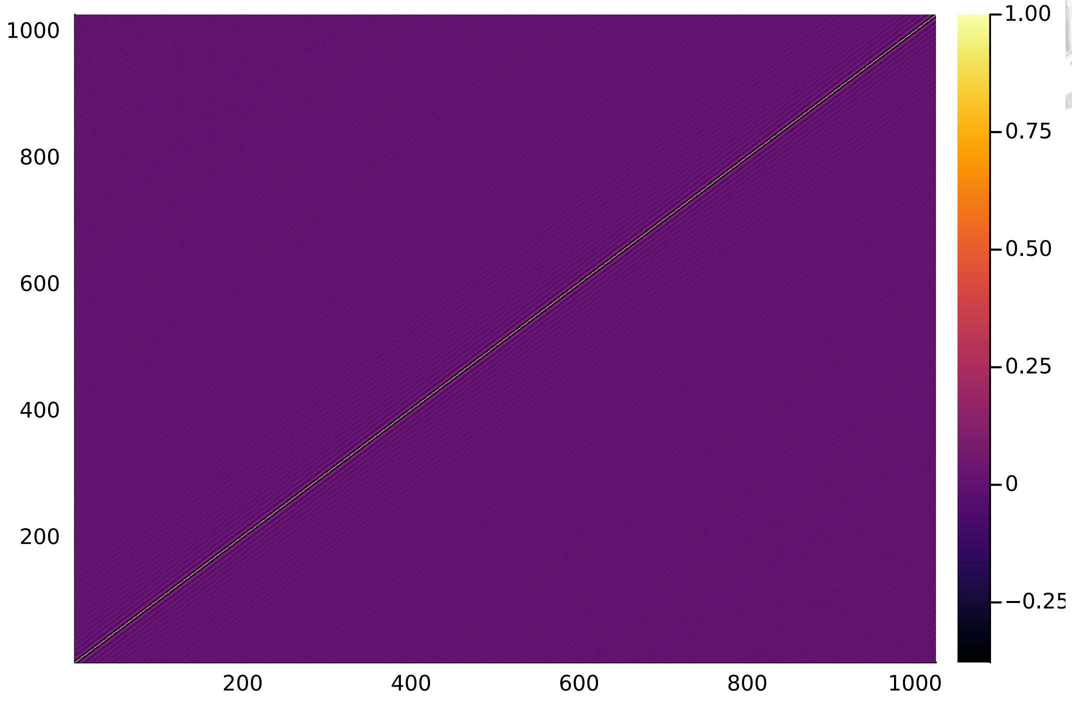
where tid and pid is the token id and previous id, respectively. The value will only be meaningful if one token is the previous token of the other, which satisfied the desired properties of text-level position embedding.

With these designs, we define our structure token vectorize function Vec as:

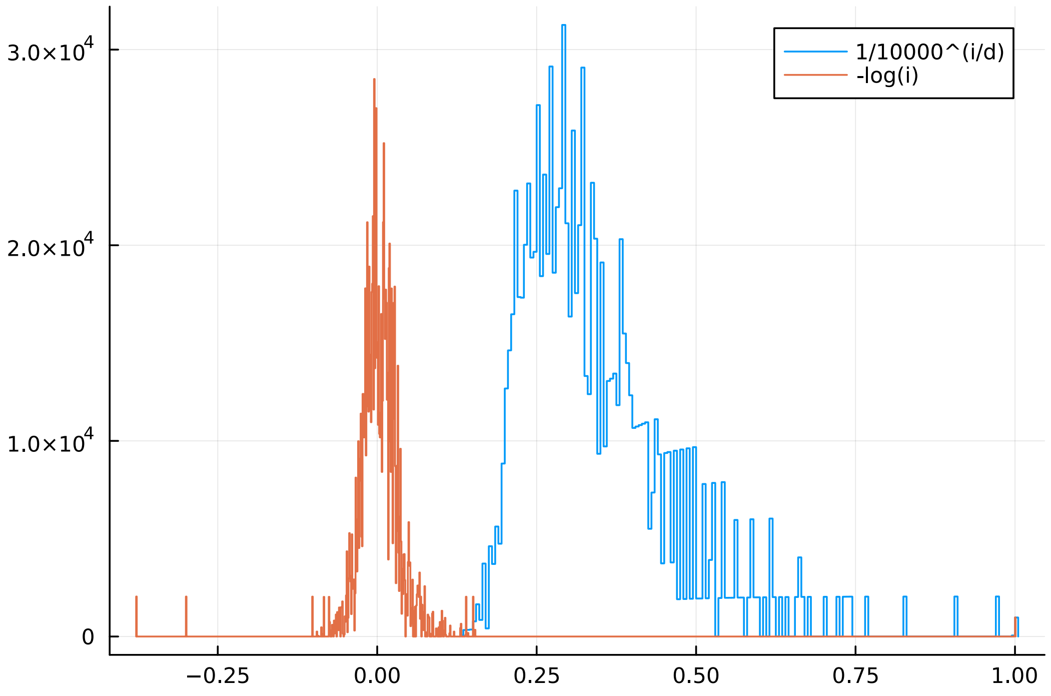
$$\begin{aligned}
Vec(X) = &\, OneHot_{|\mathcal{T}|}(\text{proj}_1(X)) \,\|\, OneHot_2(\text{proj}_2(X)) \\
&\| \, (Orth_n(\text{proj}_3(X)) + Orth_n\text{proj}_4(X))) \,\|\, Orth_n(\text{proj}_5(X)) \\
&\| \, Orth_n(\text{proj}_6(X)) \,\|\, Orth_n(\text{proj}_7(X)) \in \mathbb{R}^{4n+|\mathcal{T}|+2}
\end{aligned} \tag{3.15}$$

where $\text{proj}_i(X)$ denote the i-th element of the septuple X. The $OneHot_k$ is a function convert the input to a one-hot vector with k dimensions and $Orth_n$ convert the input to a n-dimensional orthonormal-like vector. The vectorized result will be fed into a trainable projection layer $Emb\colon \mathbb{R}^{4n+|\mathcal{T}|+2} \to \mathbb{R}^d$ to get the structure embedding with d dimensions.

doi:10.6342/NTU202302806



(a) Heatmap of cos-similarity compute from $Orth(1)$ to $Orth(1024)$



(b) Histogram of cos-similarities compute on PE* with different frequency functions.

Figure 3.6: Properties of our orthonormal-like vector. A dimensionality of 512 is used in this figure.

doi:10.6342/NTU202302806

3.2.4 Text Generation through Structure Token

After converting the structure tokens into embeddings, those embeddings are fed into the unmodified Transformer Seq2Seq model. Conceptually, our model generate a structure token at each step which contain seven objects. However, we do not really need to generate seven objects at every steps. The token id is unique for every token and we can randomly pick any id sequence $\mathcal{I}$ beforehand. Notably, the structure token representation is a set, while the autoregressive generation manner makes the generated tokens resemble a sequence. Although the design of structure tokens enable the possibility of non-monotonic order of text generation, we slightly restrict the generation order of the structure tokens from the same graph element to be ordered and contiguous. With this restriction, we do not need to predict the token id and previous id. We can use the same generation scheme of other text generation Transformer model that simply generate the next text token until we are done with this element. Meanwhile, since the graph-level identifiers are the same within a graph element, we do not need to predict the graph-level identifiers during the process of text generation for this graph element.

The generation of a text can be further simplified. As we mentioned above, a text data is being treated as a special text graph with single node without any arcs. There is no need to predict the graph-level identifiers since those ids are always just the token id of the first text token. Thus, the generation process of text with our structure token design is actually the same as other Transformer based text generation models.

doi:10.6342/NTU202302806

3.2.5 Graph Generation through Structure Token

For graph generation, the same methodology applies. We use a structure predictor for predicting the graph structure. As mentioned in Section 3.2.4, the graph-level identifiers are the same within a graph element. The prediction of graph-level identifiers can be done only one time per graph element. Moreover, the type and segment id can also be omitted because we can tell the values once we get the head id and tail id. As a result, our structure predictor only need to predict the head id and tail id.

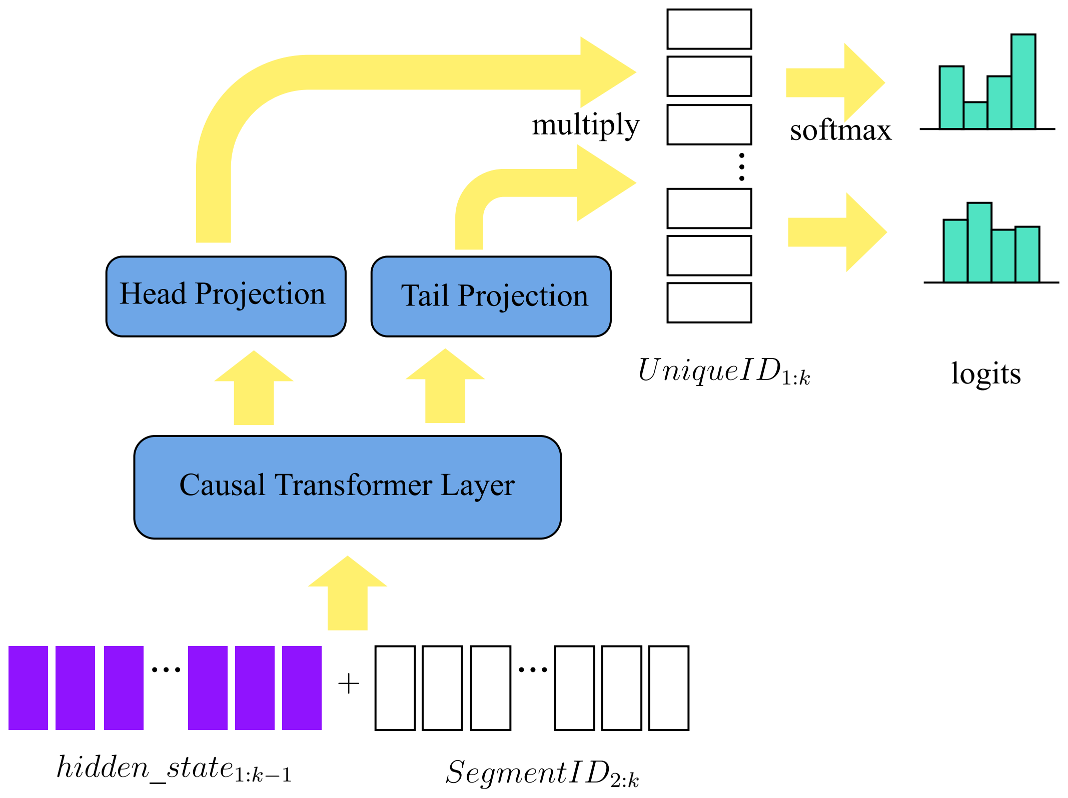


Figure 3.7: The Structure Predictor for predicting the graph-level identifiers. The $hidden\_state$ is the output of the Transformer Seq2Seq model. $SegmentID$ is the corresponding vector representations of segment ids. $UniqueID$ is the vector representation of all possible token ids.

For predicting the graph-level identifiers, we employ a single causal Transformer layer (a layer of the Transformer decoder), as illustated in Figure 3.7. The causal Transformer layer take the output of the Transformer Seq2Seq model plus the transformed seg-

doi:10.6342/NTU202302806

ment id to produce a hidden vector. The hidden vector will be fed into two projections layer to get a prediction of the head id and tail id. To get the id, we multiply the final head hidden vector and tail hidden vector with a list of our orthonormal-like vectors, and perform softmax on the multiplication result to get the logits. Then we obtain the corresponding id predictions by calling argmax on the logits. With this setup, we can apply the same teacher forcing technique used in the original Transformer paper by Vaswani et al.[46], so the training process is also parallelized.

3.2.6 Efficiency of Structure Token

As stated in Section 3.1, we hypothesize the proposed method is more efficient than the graph linearization approach because we can avoid redundant computations. Consider a text graph $\mathcal{G}$ with the same setup as Equation 3.6. With our structure token approach, the number of tokens the model need to process is:

$$c_1 = \sum_{k=1}^{|\mathcal{A}|} |E_k| + \sum_{k=1}^{|\mathcal{N}|} |N_k| \tag{3.16}$$

On the other hand, the same number with graph linearization approach will be:

$$\begin{aligned} c_2 &= \sum_{k=1}^{|\mathcal{A}|} (|N_k^h| + |E_k| + |N_k^t|) \\ &= \sum_{k=1}^{|\mathcal{A}|} |E_k| + \sum_{k=1}^{|\mathcal{A}|} (|N_k^h| + |N_k^t|) \end{aligned} \tag{3.17}$$

We define $C_1 = c_1^2$, $C_2 = c_2^2$. Since the model is based on the Transformer model and attention, the complexity is proportional to the square of the number of tokens C_1 and C_2. We can observe that the number c_2 is generally larger than or equal to c_1 because a node can be counted more than twice if it appears in two or more arcs. For example, if a

doi:10.6342/NTU202302806

node $\mathbf{N}_c = \mathbf{N}_a^h = \mathbf{N}_b^h$, which means the node $\mathbf{N}_c$ is the head node in two arcs $\mathbf{A}_a$ and $\mathbf{A}_b$, then $\sum_{k=1}^{|\mathcal{A}|}(|N_k^h| + |N_k^t|) \geq \sum_{k=1}^{|\mathcal{N}|} |N_k|$ hence $c_2 \geq c_1$. To validate this observation, we measure the fraction of C_1 and C_2 in Table 4.4.

3.3 Transformer Model Training

Beside the design of structure token and the generation method, we also introduce a pre-training method for our model. We call our pre-trained model TextGraphBART based on the mBART pre-trained model for multilingual text generation [33]. The pre-training method contain two types of training objectives, as illustrated in Figure 3.8. The first one is the self-supervised objective that force the model to reconstruct the incomplete input text graph. This objective allow the model to learn the "rule" of the language or the structure. The second one is the translation-like objective that force the model to decode the graph in another domain. The model can learn to depend on the domain token to generate different kind of text graph. With these objectives, we can utilize many different datasets to improve our model. Meanwhile, our pre-training method provide strong training signal so that the model can reach some level of performance with less training data.

doi:10.6342/NTU202302806

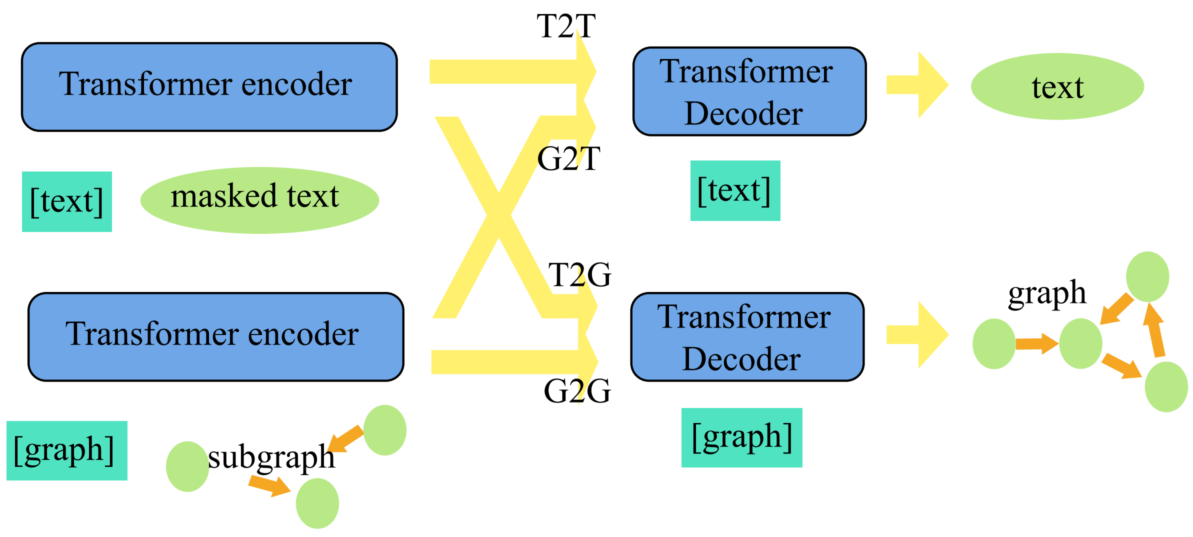


Figure 3.8: Illustration of our pre-training method. T2T, G2T, T2G, G2G stands for text-to-text, graph-to-text, text-to-graph, and graph-to-graph, respectively.

doi:10.6342/NTU202302806

Chapter 4 Experiments

This chapter aims to explore the practical implementation and evaluate the effectiveness of our method. The detail of the experiment setup such as model parameters and chosen datasets are presented in Section 4.1. We evaluate the performance of our method on graph-to-text and text-to-graph generation tasks in Section 4.2 and Section 4.3, respectively. Lastly, an ablation study of our structure embedding is presented in Section 4.4.

4.1 Experiment Setup

In this section, we describe the setup of our experiments. The general training setup is presented in Section 4.1.1, followed by the model hyperparameters in Section 4.1.2. Then we introduce the dataset for our experiments and the data processing method in Section 4.1.3 and Section 4.1.4, respectively.

4.1.1 Training Setup

Our model is trained in two phases: the pre-training and fine-tuning. We initialize our model from scratch and perform the pre-training method mentioned in Section 3.3. For pre-training, We use the RAdam optimizer with a learning rate of 0.0001 [32]. The model is updated with an effective batch size of 256 and being trained for 5 epochs on

doi:10.6342/NTU202302806

a single A100 40GB GPU. The pre-trained model is called TextGraphBART and will be used for all the fine-tuning experiments. For fine-tuning, the model is trained on a single RTX 3090 24GB GPU. The setup of fine-tuning depends on the task and dataset and will be described in the relevant sections.

4.1.2 Model Parameters

We use an overall hidden size of 512 for our model. The unmodified Transformer encoder and decoder both have 6 layers. Each attention has 16 heads, and we use a hidden size of 32 for self attentions and 64 for cross attentions. The feed-forward layer in Transformer has an input and output hidden size of 512, and the intermediate hidden size is 2048. We use these numbers for the structure predictor as well. For the activation functions, we use the GELU activation function [20] for Transformers and hyperbolic tangent activation function for the projection layers of structure predictor. During pre-training, we apply the dropout on the attention weights and the residual connections with a dropout rate of 0.1 [44]. The model weights are randomly initialized with a mean of 0 and a standard deviation of 0.02.

4.1.3 Datasets

We use four paired datasets for our experiments, as presented in Table 4.1. The paired datasets contain both text and text graph. However, most of them are created by distant supervision [37]. That is to say, they are generated by aligning text with existing database. With distant supervision, we can get a large but noisy paired training dataset. We use this kind of data for our pre-training. Specifically, we use TEKGEN [1] and GenWiki [24] for

doi:10.6342/NTU202302806

pre-training, and we fine-tune the pre-trained model on EventNarrative [11] and WebNLG (2020) [7] for evaluating our model on graph-to-text and text-to-graph generation, respectively.

Dataset	Size (# samples / disk space)		Usage	Creation
TEKGEN [1]	6.3 M	1.5 GB	pre-training	distant supervision
GenWiki [24]	750 K	1.1 GB	pre-training	distant supervision
EventNarrative [11]	180 K	135 MB	fine-tune	distant supervision
WebNLG (2020) [7]	13 K	16 MB	fine-tune	crowd source

Table 4.1: Datasets.

TEKGEN TEKGEN is a large paired dataset built with distant supervision [1]. It align a subset of the Wikipedia text with Wikidata knowledge database [47]. We use the training set of TEKGEN, which contain 6.3 million samples, for our pre-training. Some samples from the TEKGEN training set can be found in Table 4.2 and Table 4.3.

GenWiki GenWiki is another large paired dataset built on Wikipedia text [24]. The text graphs are collected from DBpedia [3]. Despite being large and paired, it is possible that some node labels in text graph do not appear in the text. We use the training set of GenWiki, which contain 750K samples, for our pre-training. Some samples from the GenWiki training set can be found in Table 4.2 and Table 4.3.

EventNarrative EventNarrative is an event-centric dataset that contains text graph from the EventKG [18] and Wikidata [47]. The text is also a subset of Wikipedia text. We use EventNarrative to evaluate our method on graph-to-text generation task, because the text graph is more aligned with the text in EventNarrative than TEKGEN and GenWiki. In other words, the node labels are more likely to appear in the corresponding text. We fine-tune our pre-trained model on the training set of EventNarrative which contain 180K

doi:10.6342/NTU202302806

Dataset	Text
TEKGEN	“ The framing story is set in the 21st century and follows Desmond Miles as Assassin ’s Creed II relives the genetic memories of his ancestor Ezio Auditore da Firenze.”
	“ The series featured appearances from famous pioneering space scientists and explorers, and was narrated by Samuel West in the original 1999 edition, and Mark Halliley in the 2004 remastered edition. ”
	“ Mikhail Alekseyevich Belyaev (Russian: ; December 23, 1863Â - 1918) was a Russian general of the Infantry, statesman, Chief of Staff of the Imperial Russian Army from August 1, 1914 to August 10, 1916, and was the last Minister of War of the Russian Empire from January 3, 1917 to February 28, 1917. ”
GenWiki	“ The Roses Theatre is an arts centre located in the centre of Tewkesbury , Gloucestershire , England . ”
	“ This cabin , known as the “Warden ’ s Camp ” was built in 1960 by the chief pilot for the Maine Warden Service and is rented out along with the six cabins at the main camp on the north shore . ”
	“ With respect to the album title , Alder indicated that it was inspired , although accidentally , by Matheos ’ lyrics for the song “ And Yet It Moves ” . ”
EventNarrative	“ This is a record of Jamaica national football team’s results at the Jamaica at the FIFA World Cup. The Jamaica at the FIFA World Cup, sometimes called the Football World Cup or the Soccer World Cup, but usually referred to simply as the World Cup, is an international association football competition contested by the men’s national teams of the members of Fédération Internationale de Football Association (FIFA), the sport’s global governing body. Jamaica national football team has qualified for the finals of the Jamaica at the FIFA World Cup once with it happening in 1998 after they finished third in the final round of CONCACAF qualifying. Nine players have been fielded in all three of Jamaica national football team’s Jamaica at the FIFA World Cup matches, making them record World Cup players for their country: The two goals scored by Theodore Whitmore during Jamaica national football team’s only World Cup win, their 2-1 over Japan, make him Jamaica national football team’s record scorer at World Cup tournaments. ”
	“ The 1974 Mongolia Premier League was the eleventh recorded edition of the Niislel League for association football, with the first tournament taking place in 1955 and no tournament held in 1965. ”
	“ The athletics at the 2012 Summer Olympics – women’s javelin throw competition at the 2012 Summer Olympics in London, United Kingdom. The event was held at the London Stadium on 07 August 2012–09 August 2012. ”
WebNLG (2020)	“ Aarhus Airport has a runway length of 2776.0. ”
	“ Allen Forrest first starting performing hip hop music in 2005. ”
	“ The School of Business and Social Sciences at the Aarhus University is located in Aarhus , Denmark . It was established in 1928 and it has 16,000 students . Its dean is Thomas Pallesen . It is affiliated to the European University Association in Brussels. ”

Table 4.2: Samples of text from each dataset. We show three sample per datasets. Corresponding text graphs are shown in Table 4.3.

doi:10.6342/NTU202302806

Dataset	Graph as Triples
TEKGEN	(“Assassin ’s Creed II”, “character role: Desmond Miles”, “Nolan North”), (“Assassin ’s Creed II”, “character role: Ezio Auditore da Firenze”, “Roger Craig Smith”)
	(“The Planets (1999 TV series)”, “publication date”, “01 January 1999”), (“The Planets (1999 TV series)”, “narrator”, “Samuel West”)
	(“Mikhail Belyaev”, “position held”, “minister of war”), (“Mikhail Belyaev”, “allegiance”, “Russian Empire”), (“Mikhail Belyaev”, “date of death”, “01 January 1918”), (“Mikhail Belyaev”, “date of birth”, “23 December 1863”), (“Mikhail Belyaev”, “country of citizenship”, “Russian Empire”)
GenWiki	(“Roses Theatre”, “location”, “Tewkesbury”), (“Roses Theatre”, “location”, “England”), (“Roses Theatre”, “foundingYear”, “1973”)
	(“Bulldog Camps”, “isPartOf”, “Maine”), (“Bulldog Camps”, “pushpinMap”, “Maine”), (“Bulldog Camps”, “location”, “Maine”)
	(“Darkness in a Different Light”, “rev”, “Blabbermouth”), (“Darkness in a Different Light”, “rev”, “About.com”), (“Darkness in a Different Light”, “title”, “O Chloroform”), (“Darkness in a Different Light”, “rev”, “Loudwire”), ...
EventNarrative	(“Jamaica at the FIFA World Cup”, “sport”, “association football”), (“Jamaica at the FIFA World Cup”, “subclass of”, “Jamaica national football team”)
	(“1974 Mongolia Premier League”, “sports season of league or competition”, “Niislel League”), (“1974 Mongolia Premier League”, “sport”, “association football”)
	(“athletics at the 2012 Summer Olympics −women’s javelin throw”, “start time”, “07 August 2012”), (“athletics at the 2012 Summer Olympics −women’s javelin throw”, “end time”, “09 August 2012”), (“athletics at the 2012 Summer Olympics −women’s javelin throw”, “point in time”, “2012”), ...
WebNLG (2020)	(“Aarhus Airport”, “runwayLength”, “2776.0”)
	(“Allen Forrest”, “genre”, “Hip hop music”), (“Allen Forrest”, “activeYearsStartYear”, “2005”)
	(“School of Business and Social Sciences at the Aarhus University”, “dean”, “Thomas Pallesen”), (“School of Business and Social Sciences at the Aarhus University”, “city”, “Aarhus”), (“School of Business and Social Sciences at the Aarhus University”, “numberOfStudents”, “16000”), (“European University Association”, “headquarter”, “Brussels”), ...

Table 4.3: Samples of text graph from each dataset. We show three sample per datasets. The “...” denote that the graph is too large for displaying. Corresponding texts are shown in Table 4.2.

doi:10.6342/NTU202302806

samples and evaluate the model on the EventNarrative test set. Some samples from the EventNarrative training set can be found in Table 4.2 and Table 4.3.

WebNLG (2020) WebNLG (2020) is crowd-sourced dataset crafted by human annotators. The text graphs are collected from DBpedia [3], while the text are manually written by the human annotators. As a result, the text and text graph in WebNLG are highly correlated, while the dataset size is the smallest. The dataset contain 16 categories in training set and 19 categories (3 extra categories) in the test set. We use WebNLG (2020) to evaluate our method on text-to-graph generation task, because WebNLG is the only dataset that the text and text graph contain the same amount of information. We fine-tune our pre-trained model on the training set of WebNLG (2020) which contain 13K samples and evaluate the model on WebNLG (2020) test set. Some samples from the WebNLG (2020) training set can be found in Table 4.2 and Table 4.3.

4.1.4 Data Processing

Dataset	# tokens in texts	# tokens in graphs	average C_1/C_2
TEKGEN [1]	218,476,541	98,978,382	0.784
GenWiki [24]	27,287,572	11,091,592	0.741
EventNarrative [11]	11,705,598	3,807,326	0.574
WebNLG (2020) [7]	398,576	300,632	0.652

Table 4.4: Statistics of the datasets. The average $\frac{C_1}{C_2}$ measures the efficiency of our method comparing to graph linearization approach. Smaller number of $\frac{C_1}{C_2}$ means our method would be more efficient on that dataset, as mentioned in Section 3.2.6.

For data processing, we use the same subword tokenizer as T5 which use the Unigram tokenization method [29, 41]. The tokenizer has a vocabulary of 32100 text tokens, which contain 32000 subword text tokens and 100 reserved special tokens. We use the reserved special tokens for our domain tokens. Each dataset is assigned with a corresponding do-

doi:10.6342/NTU202302806

main token for the graph data, while all text data from different dataset share the same text domain token. The samples in each dataset is truncated with a maximum length of 128 or 256 text tokens depending on the training stage. A random unique id sequence is determined for each sample at every epoch. As mentioned in Section 3.2.6, our structure token approach can result in less amount of tokens than graph linearization approach. We use our tokenizer to process all the datasets and compute the ratios in Table 4.4. During the pre-training, we randomly assign an unique id sequence with a maximum value of 512. For the encoder input, we randomly drop 15% of graph elements or tokens depending on the domain.

4.2 Effectiveness of Structure Token on Graph-to-Text Generation

Metric / Model	# Params	BLEU	METEOR	BERTScore
T5-Base	220 M	12.80	22.77	89.59
T5-Large	770 M	34.31	26.84	93.02
BART-Base	140 M	31.38	26.68	93.12
GAP	153 M	**35.08**	**27.50**	93.38
TextGraphBART	**75 M**	33.06	27.17	**94.23**

Table 4.5: Performance of Graph-to-Text on EventNarrative. Higher value is better, except # Params. We directly report the numbers from corresponding papers.

In this section, we show that our structure tokens capture sufficient graph-level features for the decoder by comparing the text generation performance.

Setup To evaluate our method on graph-to-text generation, we fine-tuned our TextGraph-BART model on the EventNarrative dataset [11]. During fine-tuning, we used the Lion optimizer with a learning rate of 0.00001 [9]. The model was updated with an effective

doi:10.6342/NTU202302806

batch size of 128 and trained for 20 epochs. As for the metrics, we use the BLEU [39], METEOR [5], and BERTScore [53] mentioned in Section 2.5.1 to evaluate the performance.

Baseline We compared our model with T5 [41], BART [30], and GAP [10]. Both T5 and BART are Transformer Seq2Seq model pre-trained on text data and fine-tuned with graph linearization [11], while GAP modify the encoder of Transformer Seq2Seq model with graph-aware modules for extracting graph features [10]. It is noteworthy that all these models use a similar Transformer decoder. The main difference among TextGraphBART and these models is the way we represent and handle the text graph input.

Result The result is shown in Table 4.5. In comparison to T5 and BART, our structure token method achieve better score with less parameters than graph linearization approach. Meanwhile, our model is comparable with GAP without modifying the Transformer model. As a conclusion, our structure token representations enabled the Transformer model to capture better features from the text graph than the graph linearization approach.

4.3 Effectiveness of Structure Token on Text-to-Graph Generation

In this section, we show that our generation method with structure tokens successfully generate text graph in autoregressive manner by comparing the graph generation performance.

doi:10.6342/NTU202302806

Model \ Metric	# Params	F1 (Strict)	F1 (Exact)	F1 (Partial)
CycleGT	N/A	0.309	0.342	0.360
BT5	770 M	0.675	0.682	**0.713**
Grapher (Query)	770+ M	0.289	0.395	0.325
Grapher (Text)	809 M	**0.681**	**0.683**	**0.713**
Grapher-small* (Text)	95 M	0.561	0.569	0.592
TextGraphBART	**75 M**	0.555	0.570	0.602

Table 4.6: Performance of Text-to-Graph on WebNLG (2020). Higher value is better, except # Params. The Grapher-small* is trained by us with the officially released source code of Grapher, otherwise we directly report the numbers from corresponding papers. The # Params of CycleGT is not disclosed [7].

Setup To evaluate our method on text-to-graph generation, we fine-tuned our TextGraphBART model on the WebNLG (2020) dataset [7]. During fine-tuning, we used the Adam optimizer with a learning rate of 0.0001 [27]. The model was updated with an effective batch size of 128 and trained for 100 epochs. As for the metrics, we use the official evaluation script of WebNLG (2020) mentioned in Section 2.5.2 to evaluate the performance.

Baseline We compare our model with CycleGT [7, 19], BT5 [2], and Grapher [35]. BT5 is just T5 pre-trained and fine-tuned with graph linearization. On the other hand, both CycleGT and Grapher adopt the multi-stage approach. They use T5 for generating nodes and use another model for generating the edges [7, 35]. The CycleGT is a well-known multi-stage approach for text-to-graph generation using cycle training [19], while Grapher perform supervised learning with a special loss function [35]. Grapher has two variants: Grapher (Query) and Grapher (Text). Grapher (Text) use T5 to generate the node labels as one long sequence. On the other hand, Grapher (Query) use T5 to generate a sequence of node features and use another text generation model to generate the node labels from the node features. Meanwhile, we use the officially released source code of Grapher to train a Grapher-small (Text) which has a similar model size (95M) with our model (75M). Both Grapher-small and our TextGraphBART are trained for 100 epochs

Category	# Samples (train / test)		TextGraphBART			Grapher-small (Text)		
			F1 (Strict)	F1 (Exact)	F1 (Partial)	F1 (Strict)	F1 (Exact)	F1 (Partial)
Total	13211	2155	0.555	**0.570**	**0.602**	**0.561**	0.569	0.592
Airport	1085	111	0.798	0.799	0.801	**0.831**	**0.831**	**0.833**
Artist	1222	129	0.636	0.650	0.666	**0.696**	**0.709**	**0.727**
Astronaut	529	102	0.797	0.805	0.809	**0.847**	**0.847**	**0.848**
Athlete	903	68	0.632	0.635	0.641	**0.732**	**0.732**	**0.734**
Building	972	53	0.811	0.812	0.817	**0.889**	**0.889**	**0.890**
CelestialBody	634	63	**0.713**	**0.716**	**0.716**	0.664	0.664	0.669
City	1110	104	**0.565**	**0.580**	**0.588**	0.387	0.390	0.395
ComicsCharacter	285	35	0.781	0.781	0.787	**0.917**	**0.917**	**0.919**
Company	351	93	0.878	0.878	0.880	**0.919**	**0.919**	**0.919**
Film	**0**	333	**0.338**	**0.391**	**0.439**	0.260	0.284	0.323
Food	1406	51	0.756	0.761	0.761	**0.908**	**0.908**	**0.908**
MeanOfTransportation	1132	65	**0.623**	**0.628**	**0.647**	0.585	0.587	0.590
Monument	263	53	0.848	0.848	0.848	**0.915**	**0.915**	**0.916**
MusicalWork	**0**	355	**0.244**	**0.257**	**0.354**	0.163	0.174	0.247
Politician	1194	34	0.805	0.805	0.807	**0.810**	**0.810**	**0.811**
Scientist	**0**	302	**0.499**	**0.510**	**0.538**	0.483	0.490	0.516
SportsTeam	782	51	0.689	0.691	0.696	**0.856**	**0.856**	**0.862**
University	406	107	**0.636**	**0.640**	**0.674**	0.627	0.629	0.659
WrittenWork	937	46	**0.400**	**0.405**	**0.526**	0.297	0.308	0.384

Table 4.7: Performance of our model on each category of WebNLG (2020) test set comparing to Grapher-small. The dotted lines denote the unseen categories.

doi:10.6342/NTU202302806

with the same learning rate and effective batch size.

Result The result is shown in Table 4.6. In comparison to CycleGT and Grapher (Query), our simple generation method with structure tokens outperform models with special training methods. Although our model does not directly match the performance of the large models like BT5 or Grapher (Text), our model is comparable with Grapher-small that has similar model size. By comparing the performance among BT5, Grapher (Text), and Grapher-small, we argue that the model size might have larger impact on the performance of WebNLG (2020) test set. In conclusion, our structure token approach can achieve comparable performance on text-to-graph generation under similar model size without using special training methods or loss functions.

Furthermore, we analyze the result by measuring the performance on each categories of the WebNLG (2020) test set comparing to Grapher-small. The result is shown in Table 4.7. Even though Grapher-small is based on the T5-small pre-trained model, which is trained on an extremely large dataset of 750 GB: the Colossal Clean Crawled Corpus (C4) [41], we can see that our model perform slightly better than Grapher-small on unseen categories (0 samples in training set). Some of the generation results are shown in Table 4.8.

doi:10.6342/NTU202302806

	Category	Input Text	Generated Graph as Triples
Seen	Company	“The location of Trane is Swords , Dublin.”	(“Trane”, “location”, “Swords , Dublin”)
	Food	“Bionico , a food found in Mexico , is a dish served for dessert.”	(“Bionico”, “course”, “Dessert”), (“Bionico”, “country”, “Mexico”)
	Airport	“The Adolfo Suarez Madrid - Barajas airport is operated by ENAIRE and is located in Alcobendas. Its runway name is 14L/32R, and its length is 3,500. ENAIRE is in the city of Madrid.”	(“Adolfo Suarez MadridBarajas Airport”, “runwayLength”, “3500.0”), (“Adolfo Suarez MadridBarajas Airport”, “location”, “Alcobendas”), (“Adolfo Suarez MadridBarajas Airport”, “operatingOrganisation”, “ENAIRE”), (“Adolfo Suarez MadridBarajas Airport”, “country”, “Spain”), (“Adolfo Suarez MadridBarajas Airport”, “elevationAboveTheSeaLevel”, “610.0”)
Unseen	MusicalWork	“Turn Me On is a 35.1 minute long album produced by Wharton Tiers that was followed by the album entitled Take it Off.”	(“Turn Me On”, “followed”, “Take It Off”), (“Turn Me On”, “recordLabel”, “Wharton Tiers”), (“Turn Me On”, “numberOfPages”, “35.1 (minute)”)
	Scientist	“Nurhan Atasoy is a Turkish national born in Turkey . Turkey has its largest city as Istanbul and its currency is Turkish lira.”	(“Turkey”, “leader”, “Nurhan Atasoy”), (“Turkey”, “largestCity”, “Istanbul”), (“Turkey”, “birthPlace”, “Istanbul”), (“Turkey”, “currency”, “Turkish lira”)
	File	“Jamie Lawrence wrote music for the 83 minute film ’ Death on a Factory Farm ’ which was produced by Sarah Teale.”	(“Death on a Factory Farm”, “author”, “Sarah Teale”), (“Jamie Lawrence”, “musicFusionGenre”, “Death on a Factory Farm”), (“Death on a Factory Farm”, “mediaType”, “83 minute”), (“The Amazing Area (film)”, “musicAuthor”, “Jamie Lawrence”)

Table 4.8: Examples of graph generation with TextGraphBART on WebNLG (2020) test set. The graph is expressed as triples.

doi:10.6342/NTU202302806

4.4 Ablation Study

	F1 (Strict)	F1 (Exact)	F1 (Partial)
TextGraphBART	0.555	0.570	0.602
w/o segment id	0.547 (-0.008)	0.562 (-0.008)	0.595 (-0.04)
w/o type	0.544 (-0.011)	0.561 (-0.009)	0.594 (-0.008)
w/o head id & tail id	0.489 (-0.066)	0.507 (-0.063)	0.539 (-0.063)
w/o token id & previous id	0.365 (-0.190)	0.378 (-0.192)	0.404 (-0.198)

Table 4.9: Ablation results of our structure embedding on WebNLG (2020) test set. The number in parenthesis denote the difference from TextGraphBART.

To investigate the performance contribution of the components of structure tokens, we conducted the ablation study on our structure embedding by fine-tuning our model with the removal of some parts of the embeddings. The model is trained on the WebNLG (2020) with the same setup in Section 4.3. The results are shown in Table 4.9. In all ablations, the model performance was attenuated as expected. First, the ablation of the token id and previous id removes the text order information in the text labels hence the degeneration of performance. Similarly, the head id and tail id provide the connectivities of the graph. Removal of this embedding decrease the performance, indicating the importance of the connectivities. On the other hand, the ablation of type and segment id are not as detrimental as others because the type and segment id may be inferred from other ids. Therefore, our model is still able to perform albeit less performant. In conclusion, the ablation study showed that all of our structure embedding are important for good model performance.

doi:10.6342/NTU202302806

doi:10.6342/NTU202302806

Chapter 5 Conclusions and Future Work

This chapter serves to wrap up our research. We first summarize our works in Section 5.1. Then we outline the limitation and potential future research in Section 5.2. Lastly, we cap off our thesis with Section 5.3.

5.1 Summary

This thesis presents a novel approach to the problem of text graph generation leveraging the strength of Transformer models. Our exploration has led to an effective method for structured data representation and generation via "Structure Tokens". In the structure token, we use several identifiers for indicating the connectivities of the graphs and the order of the texts. Then an embedding method for structure token is proposed, allowing the Transformer model to utilize the structural information. We show that the structure token approach can be use to represent and generate both texts and text graphs. The experiment results demonstrated the effectiveness of our method with less data and parameters. Meanwhile, the ablation study further confirmed the importance of various elements of the structure tokens.

doi:10.6342/NTU202302806

5.2 Discussion and Future Work

Although we have designed an effective method for representing and generating text graphs and examined the design with experiments, there are a few subjects left unexplored. In this section, we discuss the application and limitations of our research and delineate potential directions for future research.

Application In this thesis, we show that the proposed structure token approach enabled the Transformer model to capture better features from text graph. The structure tokens can be used to represent a wide variety of text graphs such as the dialogue states or the commonsense knowledge graph [23, 38]. We hypothesize that the structure token representation might improve the performance of tasks beyond graph-to-text generation. For example, some fake news detection model process on factual knowledge graph [22, 34]. Despite the effectiveness of structure tokens on graph-to-text generation, the impact of structure tokens on other tasks require further investigations.

Besides, the proposed model can be used in a few programming tools. For example, we can encode the Abstract Syntax Tree (AST) of a programming language with our structure token and use the extracted features to perform similarity search. This would improve the string matching based code search functionality, such as code plagiarism checker or the search functionality of large codebase like GitHub's search engine. Moreover, it might also be used in the compiler optimization pipelines. For example, we can view the graph intermediate representation as a text graph, and then the graph optimization passes become a text-graph-to-text-graph translation problem, which is aligned with the proposed model.

Furthermore, the proposed method could also be used for detecting malicious binaries

doi:10.6342/NTU202302806

(i.e. Malware). For example, one method for detecting malicious binaries is to convert the assembly into Control-Flow Graph (CFG) and then comparing the CFG with the malicious samples. We can treat the CFG as text graph and then extract the features with our model.

The above ideas can be apply to many other kind of graph databases, such as the academic knowledge graph [43] or semantic web [6], for improving either the constructing or querying pipelines. However, these concepts necessitates a more thorough examination that goes beyond the scope of the present thesis, thereby presenting a compelling area for future investigations

Model Architecture In this thesis, we frequently mentioned that we use an unmodified Transformer model. The intent of this repetition is to underscore that the observed performance stem from the design of the token itself, rather than modifications to the model. Although we can get comparable result with our structure token approach, whether the unmodified Transformer is the most appropriate choice remains questionable. For example, in another point of view, our structure embedding can be viewed as an variant of position embedding that directly added to the input embeddings at the first layer only. Under this perspective, there are several modifications we can do. As an instance, we can add our embeddings to the input of every layers. Furthermore, we can even adapt the similar modification of relative or rotary position embedding that alter the attention operation [14]. On the other hand, we use the generalized sinusoidal position encoding with a specific negative log frequency function to generate the orthonormal-like vectors and empirically show that the chosen function satisfy the desired properties. Whether negative log is the most appropriate choice of frequency function or whether there exist a better method for generating orthonormal-like vectors are yet to be determined.

doi:10.6342/NTU202302806

Self-Supervised and Semi-Supervised Training In this thesis, a pre-training method is presented. The pre-training method containg two types of training objectives: the translation-like objective and the self-supervised objective. The translation-like objective is aligned with the fine-tuning tasks and pre-training with this objective works as transfer learning for our fine-tuning tasks. On the other hand, the self-supervised objective would provide extra training signal that allow the model to learn better features for the input. Moreover, paired training data is not required for self-supervised objective, which could benefit from using the large amount of unpaired text dataset or graph database. In the scope of our thesis, we only pre-trained our model with paired datasets, as mentioned in Section 4.1. The effectiveness of self supervised training require further investigations.

Cycle Training could further improve the model performance. As mentioned in Section 1.3, lacking paired data is a challenge that might be solvable with semi-supervised learning [19, 48]. In this thesis, we show that our model is able to perform comparably on both text-to-graph generation and graph-to-text generation task. Therefore, our model could generate new paired data, enabling cycle training with single model. It offers a promising prospect for additional research in the future.

Generation Order Our structure token method is free from the problem of linearization order since the whole structure token representation is a set. In our implementation, the order of the graph elements are completely random. It is unclear whether the generation order would affect the performance. It might be possible to find an optimal order of nodes and edges that can train the model more efficiently. Moreover, in Section 4.2 and Section 4.3, we make a few assumptions about the generation order to simplify the generation and reduce the number of predictions required. If we remove the assumptions

doi:10.6342/NTU202302806

of the generation order, despite requiring more computation resources, the possibility of non-monotonic generation order emerged. While this issue is undoubtedly worth examination, it ventures beyond the reach of this thesis, thus laying the groundwork for further inquiry.

Generation Algorithm In this thesis, we simply use the greedy decoding algorithm for both text and text graph generation. Despite the effectiveness of such a simple algorithm, we can also apply more complicated decoding algorithms that widely used in machine translation techniques, which is not possible with the multi-stage approach. For example, we could apply the beam search that explore multiple generation candidates at the same time to produce better generation results [17, 45]. An exploration of this particular aspect falls outside the present study and suggests an interesting avenue for subsequent research.

Graph Generation Metric Throughout this thesis, we use the evaluation script of WebNLG to evaluate our method for graph generation. The WebNLG evaluation method is currently the de facto standard for evaluating those text graph generation models. Although being widely used, there are certain limitations because the metric is based on string matching. That is to say, the metric does not consider the possibility of paraphrases or synonyms. For example, the ground truth triple (“Nord (Year of No Light album)”, “genre”, “Sludge metal”) from the WebNLG test set is not the same as the generated triple (“Nord by Year of No Light”, “genre”, “Sludge metal”) under the evaluation metric. Similarly, the ground truth triple (“Death on a Factory Farm”, “author”, “Sarah Teale”) is not the same as the generated triple (“Death on a Factory Farm”, “producer”, “Sarah Teale”) under the metric. Therefore, current metric could potentially underestimate or overestimate the model performance. To facilitate a better quantification and comparison of model performance, we

doi:10.6342/NTU202302806

need a more comprehensive metric. For example, we might be able to avoid these issues by replacing string matching algorithm with text generation metrics such as METEOR or BERTScore.

Ablation Study In this thesis, we perform ablations on the structure embedding to measure the effectiveness and necessity of those components in our structure tokens. We use the same pre-trained model and fine-tine it with removal. The ablation result shows the importance of our structure token to the pre-trained model. As expected, the removal of the component damage the model performance, especially the removal of text order and graph structure. However, it is arguable that the same pre-trained model used in this ablation study does not provide optimal condition for the ablations. Therefore, A more comprehensive ablation study would require training of a new pre-trained models with the removal of the components of structure token and then fine-tuning them.

doi:10.6342/NTU202302806

5.3 Conclusions

The results obtained not only validate our method but also present valuable insights into the broader field of structured data representation and generation. The success of the structure tokens illustrates the potential benefits of a more informative token design. Our work only target on the specific graph structure. It might be possible to design a corresponding token for other structures or graphs with different modalities of contents. On the other hand, the embedding of token id and previous id is similar to the position embedding. It enables not only the potential representation of non-linear texts but also the possibility of non-monotonic text generation.

In this thesis, we show that the structure token is effective with an unmodified Transformer model. However, there remains substantial room for further investigation and refinement of our proposed method, such as using different Transformer variants, different ways to generate orthonormal-like vectors, or different generation algorithm. Future work could include the extension of this approach to larger and more diverse datasets, refining the structure token design to capture more intricate graph-level features, or adopting other training methods like cycle training to further enhance performance. The development of more sophisticated evaluation metrics could also aid in better quantifying the quality of the generated text graphs, driving further improvements in model performance. The insights and findings presented in this work are expected to serve as a foundation for these future endeavours.

doi:10.6342/NTU202302806

doi:10.6342/NTU202302806

References

[1] O. Agarwal, H. Ge, S. Shakeri, and R. Al-Rfou. Knowledge graph based synthetic corpus generation for knowledge-enhanced language model pre-training. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 3554–3565, Online, June 2021. Association for Computational Linguistics.

[2] O. Agarwal, M. Kale, H. Ge, S. Shakeri, and R. Al-Rfou. Machine translation aided bilingual data-to-text generation and semantic parsing. In Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), pages 125–130, Dublin, Ireland (Virtual), 12 2020. Association for Computational Linguistics.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference, ISWC'07/ASWC'07, page 722–735, Berlin, Heidelberg, 2007. Springer-Verlag.

[4] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014.

[5] S. Banerjee and A. Lavie. METEOR: An automatic metric for MT evaluation

doi:10.6342/NTU202302806

with improved correlation with human judgments. In Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[6] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. Scientific American, 284(5):34–43, May 2001.

[7] T. Castro Ferreira, C. Gardent, N. Ilinykh, C. van der Lee, S. Mille, D. Moussallem, and A. Shimorina. The 2020 bilingual, bi-directional WebNLG+ shared task: Overview and evaluation results (WebNLG+ 2020). In Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), pages 55–76, Dublin, Ireland (Virtual), 12 2020. Association for Computational Linguistics.

[8] A. Celikyilmaz, E. Clark, and J. Gao. Evaluation of text generation: A survey. ArXiv, abs/2006.14799, 2020.

[9] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, Y. Liu, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le. Symbolic discovery of optimization algorithms. ArXiv, abs/2302.06675, 2023.

[10] A. Colas, M. Alvandipour, and D. Z. Wang. Gap: A graph-aware language model framework for knowledge graph-to-text generation. In International Conference on Computational Linguistics, 2022.

[11] A. Colas, A. Sadeghian, Y. Wang, and D. Z. Wang. Eventnarrative: A large-scale event-centric dataset for knowledge graph-to-text generation, 2022.

doi:10.6342/NTU202302806

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[13] P. Dognin, I. Padhi, I. Melnyk, and P. Das. ReGen: Reinforcement learning for text and knowledge base generation using pretrained language models. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 1084–1099, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics.

[14] P. Dufter, M. Schmitt, and H. Schütze. Position information in transformers: An overview. Computational Linguistics, 48(3):733–763, Sept. 2022.

[15] V. P. Dwivedi and X. Bresson. A generalization of transformer networks to graphs. ArXiv, abs/2012.09699, 2020.

[16] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. ArXiv, abs/2003.00982, 2020.

[17] M. Freitag and Y. Al-Onaizan. Beam search strategies for neural machine translation. In Proceedings of the First Workshop on Neural Machine Translation, pages 56–60, Vancouver, Aug. 2017. Association for Computational Linguistics.

[18] S. Gottschalk and E. Demidova. Eventkg: A multilingual event-centric temporal knowledge graph. In Extended Semantic Web Conference, 2018.

doi:10.6342/NTU202302806

[19] Q. Guo, Z. Jin, X. Qiu, W. Zhang, D. Wipf, and Z. Zhang. CycleGT: Unsupervised graph-to-text and text-to-graph generation via cycle training. In Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+), pages 77–88, Dublin, Ireland (Virtual), 12 2020. Association for Computational Linguistics.

[20] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). arXiv: Learning, 2016.

[21] J. Hewitt and C. D. Manning. A structural probe for finding syntax in word representations. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4129–4138, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[22] L. Hu, T. Yang, L. Zhang, W. Zhong, D. Tang, C. Shi, N. Duan, and M. Zhou. Compare to the knowledge: Graph neural fake news detection with external knowledge. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 754–763, Online, Aug. 2021. Association for Computational Linguistics.

[23] F. Ilievski, P. Szekely, and B. Zhang. Cskg: The commonsense knowledge graph. Extended Semantic Web Conference (ESWC), 2021.

[24] Z. Jin, Q. Guo, X. Qiu, and Z. Zhang. GenWiki: A dataset of 1.3 million content-sharing text and graphs for unsupervised graph-to-text generation. In Proceedings of the 28th International Conference on Computational Linguistics, pages 2398–2409,

doi:10.6342/NTU202302806

Barcelona, Spain (Online), Dec. 2020. International Committee on Computational Linguistics.

[25] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. B. Viégas, M. Wattenberg, G. S. Corrado, M. Hughes, and J. Dean. Google’s multilingual neural machine translation system: Enabling zero-shot translation. Transactions of the Association for Computational Linguistics, 5:339–351, 2016.

[26] J. Kim, D. Nguyen, S. Min, S. Cho, M. Lee, H. Lee, and S. Hong. Pure transformers are powerful graph learners. Advances in Neural Information Processing Systems, 35:14582–14595, 2022.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. CoRR, abs/1412.6980, 2014.

[28] T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. ArXiv, abs/1609.02907, 2016.

[29] T. Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[30] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

doi:10.6342/NTU202302806

[31] T. Lin, Y. Wang, X. Liu, and X. Qiu. A survey of transformers. AI Open, 3:111–132, 2021.

[32] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. On the variance of the adaptive learning rate and beyond. ArXiv, abs/1908.03265, 2019.

[33] Y. Liu, J. Gu, N. Goyal, X. Li, S. Edunov, M. Ghazvininejad, M. Lewis, and L. Zettlemoyer. Multilingual denoising pre-training for neural machine translation. Transactions of the Association for Computational Linguistics, 8:726–742, 2020.

[34] M. Mayank, S. Sharma, and R. Sharma. Deap-faked: Knowledge graph based approach for fake news detection. 2022 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pages 47–51, 2021.

[35] I. Melnyk, P. Dognin, and P. Das. Knowledge graph generation from text. In Findings of the Association for Computational Linguistics: EMNLP 2022, pages 1610–1622, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics.

[36] E. Min, R. Chen, Y. Bian, T. Xu, K. Zhao, W. bing Huang, P. Zhao, J. Huang, S. Ananiadou, and Y. Rong. Transformer for graphs: An overview from architecture perspective. ArXiv, abs/2202.08455, 2022.

[37] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP, pages 1003–1011, Suntec, Singapore, Aug. 2009. Association for Computational Linguistics.

doi:10.6342/NTU202302806

[38] S. Ouyang, Z. Zhang, and H. Zhao. Dialogue graph modeling for conversational machine reading. In Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 3158–3169, Online, Aug. 2021. Association for Computational Linguistics.

[39] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[40] A. Radford and K. Narasimhan. Improving language understanding by generative pre-training. 2018.

[41] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research, 21(140):1–67, 2020.

[42] A. Rogers, O. Kovaleva, and A. Rumshisky. A primer in BERTology: What we know about how BERT works. Transactions of the Association for Computational Linguistics, 8:842–866, 2020.

[43] A. Sinha, Z. Shen, Y. Song, H. Ma, D. Eide, B.-J. P. Hsu, and K. Wang. An overview of microsoft academic service (mas) and applications. In International World Wide Web Conferences. Microsoft, May 2015.

[44] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(56):1929–1958, 2014.

doi:10.6342/NTU202302806

[45] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, Advances in Neural Information Processing Systems, volume 27. Curran Associates, Inc., 2014.

[46] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In NIPS, 2017.

[47] D. Vrandečić and M. Krötzsch. Wikidata: A free collaborative knowledgebase. Commun. ACM, 57(10):78–85, sep 2014.

[48] Y. Xu, L. Fu, Z. Lin, J. Qi, and X. Wang. Infinity: A simple yet effective unsupervised framework for graph-text mutual conversion. ArXiv, abs/2209.10754, 2022.

[49] J. Yang, G. Xiao, Y. Shen, W. Jiang, X. Hu, Y. Zhang, and J. Peng. A survey of knowledge enhanced pre-trained models. ArXiv, abs/2110.00269, 2021.

[50] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu. Do transformers really perform bad for graph representation? In Neural Information Processing Systems, 2021.

[51] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In International Conference on Machine Learning, 2018.

[52] L. Zhang and R. Li. KE-GCL: Knowledge enhanced graph contrastive learning for commonsense question answering. In Findings of the Association for Computational Linguistics: EMNLP 2022, pages 76–87, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics.

doi:10.6342/NTU202302806

[53] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. Bertscore: Evaluating text generation with bert. ArXiv, abs/1904.09675, 2019.

[54] Y. Zhang, H. Dai, K. Toraman, and L. Song. Kg^2: Learning to reason science exam questions with contextual knowledge graph embeddings, 2018.

[55] L. Zhong, J. Wu, Q. Li, H. Peng, and X. Wu. A comprehensive survey on automatic knowledge graph construction, 2023.

[56] Y. Zhu, Y. Du, Y. Wang, Y. Xu, J. Zhang, Q. Liu, and S. Wu. A survey on deep graph generation: Methods and applications. In LOG IN, 2022.

doi:10.6342/NTU202302806