Transformer for Graphs: An Overview from Architecture Perspective

Erxue Min[1*], **Runfa Chen**[3], **Yatao Bian**[2], **Tingyang Xu**[2], **Kangfei Zhao**[2], **Wenbing Huang**[4], **Peilin Zhao**[2], **Junzhou Huang**[5], **Sophia Ananiadou**[1], **Yu Rong**[2†]

[1]National Centre for Text Mining, Department of Computer Science, The University of Manchester
[2]Tencent AI lab
[3]Department of Computer Science and Technology, Tsinghua University
[4]Institute for AI Industry Research (AIR), Tsinghua University
[5]Department of Computer Science and Engineering, University of Texas at Arlington

Abstract

Recently, Transformer model, which has achieved great success in many artificial intelligence fields, has demonstrated its great potential in modeling graph-structured data. Till now, a great variety of Transformers has been proposed to adapt to the graph-structured data. However, a comprehensive literature review and systematical evaluation of these Transformer variants for graphs are still unavailable. It's imperative to sort out the existing Transformer models for graphs and systematically investigate their effectiveness on various graph tasks. In this survey, we provide a comprehensive review of various Graph Transformer models from the architectural design perspective. We first disassemble the existing models and conclude three typical ways to incorporate the graph information into the vanilla Transformer: 1) GNNs as Auxiliary Modules, 2) Improved Positional Embedding from Graphs, and 3) Improved Attention Matrix from Graphs. Furthermore, we implement the representative components in three groups and conduct a comprehensive comparison on various kinds of famous graph data benchmarks to investigate the real performance gain of each component. Our experiments confirm the benefits of current graph-specific modules on Transformer and reveal their advantages on different kinds of graph tasks.

1 Introduction

Graph is a kind of data structure that structurally depicts a set of objects (nodes) with their relationships (edges). As a unique non-Euclidean data structure, graph analysis focuses on tasks such as node classification [Yang *et al.*, 2016], link prediction [Zhang and Chen, 2018], and clustering [Aggarwal, 2010]. Recent research on analyzing graphs using deep learning has attracted more and more attention due to the rich expressive power of deep learning models. Graph Neural Networks (GNNs) [Kipf and Welling, 2016], as a kind of deep learning-based method, are recently becoming a widely-used graph analysis tools due to their convincing performance. Most of the current GNNs are based on the Message Passing paradigm [Gilmer *et al.*, 2017], the expressive power of which is bounded by the Weisfeiler-Lehamn isomorphism hierarchy [Maron *et al.*, 2019; Xu *et al.*, 2018a]. Worse still, as pointed out by [Kreuzer *et al.*, 2021], GNNs suffer from the *over-smoothing* problem due to repeated local aggregation, and the *over-squashing* problem due to the exponential computation cost with the increase of model depth. Several studies [Zhao and Akoglu, 2019; Rong *et al.*, 2019; Xu *et al.*, 2018b] try to address such problems. Nevertheless, none of them seems to be able to eliminate these problems from the Message Passing paradigm.

On the other hand, Transformers and its variants, as a powerful class of models, are playing an important role in various areas including Natural Language Processing (NLP) [Vaswani *et al.*, 2017], Computer Vision (CV) [Forsyth and Ponce, 2011], Time-series Analysis [Hamilton, 2020], Audio Processing [Purwins *et al.*, 2019], etc. Moreover, recent years have witnessed many successful Transformer variants in modeling graphs. These models have achieved competitive or even superior performance against GNNs in many applications, such as Quantum Property Prediction [Hu *et al.*, 2021], Catalysts Discovery [Chanussot* *et al.*, 2021] and Recommendation Systems [Min *et al.*, 2022]. The literature reviews and systematic evaluations for these Transformer variants are, however, lacking.

This survey gives an overview of the current research progress on incorporating Transformers in graph-structured data. Concretely, we provide a comprehensive review of over 20 Graph Transformer models from the architectural design perspective. We first dismantle the existing models and conclude three typical ways to incorporate the graph information into the vanilla Transformer: **1) GNNs as Auxiliary Modules**, *i.e.*, directly inject GNNs into Transformer architecture. Specifically, according to the relative position between GNN layers and Transformer layers, existing GNN-Transformer architectures can be categorized into three types: (a) build Transformer blocks on top of GNN blocks, (b) stack GNN blocks and Transformer blocks in each layer, (c) parallel GNN blocks and Transformer blocks in each layer. **2) Improved Positional Embedding from Graphs**, *i.e.*, compress the graph structure into positional embedding vectors and add

*Email: erxue.min@manchester.ac.uk
†Corresponding Author: yu.rong@hotmail.com

them to the input before it is fed to the vanilla Transformer model. This graph positional embedding can be derived from the structural information of graphs, such as degree and centrality. **3) Improved Attention Matrices from Graphs**, *i.e.*, inject graph priors into the attention computation via graph bias terms, or restrict a node only attending to local neighbours in the graph, which can be computationally formulated as an attention masking mechanism.

Additionally, in order to investigate the effectiveness of existing models in various kinds of graph tasks, we implement the representative components in three groups and conduct comprehensive ablation studies on six popular graph-based benchmarks to uniformly test the real performance gain of each component. Our experiments indicate that: 1). Current models incorporating the graph information can improve the performance of Transformer on both graph-level and node-level tasks. 2). Utilizing GNN as auxiliary modules and improving attention matrix from graphs generally contributes more performance gains than encoding graphs into positional embeddings. 3). The performance gain on graph-level tasks is more significant than that on node-level tasks. 4) Different kinds of graph tasks enjoy different group of models.

The rest of this survey is organized as follows. We first review the general vanilla Transformer Architecture in Section 2. Section 3 summarizes existing works about the Transformer variant on Graphs and systematically categorizes these methods into three groups. In Section 4, comprehensive ablation studies are conducted to verify the effectiveness and compatibility of these proposed models. In Section 5, we conclude this survey and discuss several future research directions.

2 Transformer Architecture

Transformer [Vaswani *et al.*, 2017] architecture was first applied to machine translation. In the paper, Transformer is introduced as a novel encoder-decoder architecture built with multiple blocks of self-attention. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ to be the input of each Transformer layer, where n is number of tokens, d is the dimension of each token, then one block layer can be a function $f_\theta : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$ with $f_\theta(\mathbf{X}) =: \mathbf{Z}$ defined by:

$$\mathbf{A} = \frac{1}{\sqrt{d}} \mathbf{X}\mathbf{Q}(\mathbf{X}\mathbf{K})^\top, \tag{1}$$

$$\widetilde{\mathbf{X}} = \mathrm{SoftMax}(\mathbf{A})(\mathbf{X}\mathbf{V}), \tag{2}$$

$$\mathbf{M} = \mathrm{LayerNorm}_1(\widetilde{\mathbf{X}}\mathbf{O} + \mathbf{X}), \tag{3}$$

$$\mathbf{F} = \sigma(\mathbf{M}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2, \tag{4}$$

$$\mathbf{Z} = \mathrm{LayerNorm}_2(\mathbf{M} + \mathbf{F}), \tag{5}$$

where Equation 1, Equation 2, and Equation 3 are the attention computation; while Equation 4 and Equation 5 are the position-wise feed-forward network (FFN) layers. Here, Softmax$(\cdot)$ refers to the row-wise softmax function, LayerNorm$(\cdot)$ refers to layer normalization function [Ba *et al.*, 2016], and σ refers to the activation function. $\mathbf{Q}, \mathbf{K}, \mathbf{V}, \mathbf{O} \in \mathbb{R}^{d \times d}, \mathbf{W}_1 \in \mathbb{R}^{d \times d_f}, \mathbf{b}_1 \in \mathbb{R}^{d_f}, \mathbf{W}_2 \in \mathbb{R}^{d_f \times d}, \mathbf{b}_2 \in \mathbb{R}^d$ are trainable parameters in the layer. Furthermore, it is common to consider multiple attention heads to extend the self-attention to Multi-head Self-attention (MHSA). Specifically, $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are decomposed into H heads with $\mathbf{Q}^{(h)}, \mathbf{K}^{(h)}, \mathbf{V}^{(h)} \in \mathbb{R}^{d \times d_h}$ with $d = \sum_{h=1}^{H} d_h$, and the matrices $\widetilde{\mathbf{X}}^{(h)} \in \mathbb{R}^{n \times d_h}$ from attention heads are concatenated to obtain $\widetilde{\mathbf{X}}$. In this case, Equation 1 and Equation 2 respectively become:

$$\mathbf{A}^{(h)} = \frac{1}{\sqrt{d}} \mathbf{X}\mathbf{Q}^{(h)}(\mathbf{X}\mathbf{K}^{(h)})^\top, \tag{6}$$

$$\widetilde{\mathbf{X}} = \|_{h=1}^{H}(\mathrm{SoftMax}(\mathbf{A}^{(h)})\mathbf{X}\mathbf{V}^{(h)}). \tag{7}$$

The multi-head mechanism enables the model to implicitly learn representation from different aspects. Apart from the attention mechanism, the paper uses sine and cosine functions with different frequencies as positional embedding to distinguish the position of each token in the sequence.

Table 1: A summary of papers that applied Transformers on graph-structured data. **GA**: GNNs as Auxiliary Modules; **PE**: Improved Positional Embedding from Graphs; **AT**: Improved Attention Matrix from Graphs.

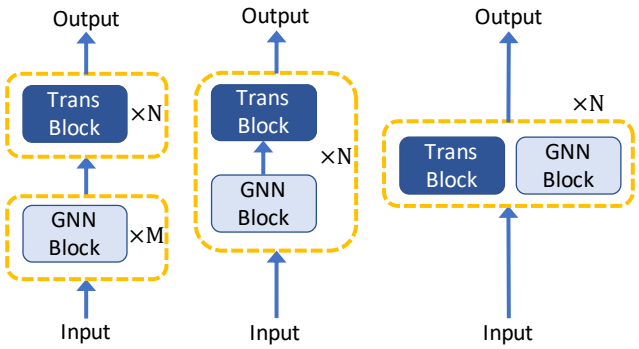
Method	GA	PE	AT	Code
[Zhu *et al.*, 2019]			✓	✓
[Shiv and Quirk, 2019]		✓		
[Wang *et al.*, 2019]		✓	✓	
U2GNN [Nguyen *et al.*, 2019]			✓	✓
HeGT [Yao *et al.*, 2020]			✓	✓
Graformer [Schmitt *et al.*, 2020]			✓	✓
PLAN [Khoo *et al.*, 2020]			✓	✓
UniMP [Shi *et al.*, 2020]			✓	
GTOS [Cai and Lam, 2020]		✓	✓	✓
Graph Trans [Dwivedi and Bresson, 2020]		✓	✓	✓
Grover [Rong *et al.*, 2020]	✓			✓
Graph-BERT [Zhang *et al.*, 2020]	✓	✓		✓
SE(3)-Transformer [Fuchs *et al.*, 2020]			✓	✓
Mesh Graphormer [Lin *et al.*, 2021]	✓	✓		✓
Gophormer [Zhao *et al.*, 2021]			✓	
EGT [Hussain *et al.*, 2021]		✓	✓	✓
SAN [Kreuzer *et al.*, 2021]		✓	✓	✓
GraphiT [Mialon *et al.*, 2021]	✓	✓	✓	✓
Graphormer [Ying *et al.*, 2021]	✓	✓	✓	✓
Mask-transformer [Min *et al.*, 2022]			✓	✓
TorchMD-NET [Thölke and de Fabritiis, 2022]			✓	✓

3 Transformer Architecture for Graphs

The self-attention mechanism in the standard Transformer actually considers the input tokens as a fully-connected graph, which is agnostic to the intrinsic graph structure among the data. Existing methods that enable Transformer to be aware of topological structures are generally categorized into three groups: 1) GNNs as auxiliary modules in Transformer (**GA**), 2) Improved positional embedding from graphs (**PE**), 3) Improved attention matrices from graphs (**AT**). We summarize relevant literature in terms of these three dimensions in Table 1.

3.1 GNNs as Auxiliary Modules in Transformer

The most direct solution of involving structural knowledge to benefit from global relation modeling of self-attention is



(a) Before Trans (b) Alternatively (c) Parallel

Figure 1: Three types of GNN-as-Auxiliary-Modules architecture.

to combine graph neural networks with Transformer architecture. Generally, according to the relative postion between GNN layers and Transformer layers, existing Transformer architectures with GNNs are categorized into three types as illustrated in Figure 1: (1) building Transformer blocks on top of GNN blocks, (2) alternately stacking GNN blocks and Transformer blocks, (3) parallelizing GNN blocks and Transformer blocks.

The first architecture is most-frequently adopted among the three options. For example, GraphTrans [Jain *et al.*, 2021] adds a Transformer subnetwork on top of a standard GNN layer. The GNN layer performs as a specialized architecture to learn local representations of the structure of a node's immediate neighbourhood, while the Transformer subnetwork computes all pairwise node interactions in a position-agnostic fashion, empowering the model global reasoning capability. GraphTrans is evaluated on graph classification task from biology, computer programming and chemistry, and achieves consistent improvement over benchmarks.

Grover [Rong *et al.*, 2020] consists of two GTransformer modules to represent node-level features and edge-level features respectively. In each GTransformer, the inputs are first fed into a tailored GNNs named dyMPN to extract vectors as queries, keys and values from nodes of the graph, followed by standard multi-head attention blocks. This bi-level information extraction framework enables the model to capture the structural information in molecular data and make it possible to extract global relations between nodes, enhancing the representational power of Grover.

GraphiT [Mialon *et al.*, 2021] also falls in the first architecture, which adopts one Graph Convolutional Kernel Network (GCKN) [Chen *et al.*, 2020] layer to produce a structure-aware representation from original features, and concatenate them as the input of Transformer architecture. Here, GCKNs is a multi-layer model that produces a sequence of graph feature maps similar to a GNN. Different from GNNs, each layer of GCKNs enumerates local sub-structures at each node, encodes them using a kernel embedding, and aggregates the sub-structure representations as outputs. These representations in a feature map carry more structural information than traditional GNNs based on neighborhood aggregation.

Mesh Graphormer [Lin *et al.*, 2021] follows the second architecture by stacking a Graph Residual Block (GRB) on a multi-head self-attention layer as a Transformer block to model both local and global interactions among 3D mesh vertices and body joints. Specifically, given the contextualized features $\mathbf{M}$ generated by multi-head self-attentions (MHSA) in Equation 3, Mesh Graphormer improves the local interactions using a graph convolution in each Transformer block as:

$$\mathbf{M}' = \text{GraphConv}(\mathbf{A}^G, \mathbf{M}; \mathbf{W}^G) = \sigma(\mathbf{A}^G \mathbf{X} \mathbf{W}^G). \quad (8)$$

where $\mathbf{A}^G \in \mathbb{R}^{n \times n}$ denotes the adjacency matrix of a graph and $\mathbf{W}_G$ denotes the trainable parameters. $\sigma(\cdot)$ implies the non-linear activation function. Mesh Graphormer also implements another two variants: building GRB before MHSA and building those two blocks in parallel, but they perform worse than the proposed one.

Graph-BERT [Zhang *et al.*, 2020] adopts the third architecture by utilizing a graph residual term in each attention layer as follows:

$$\mathbf{M}' = \mathbf{M} + \text{G-Res}(\mathbf{X}, \mathbf{X}_r, \mathbf{A}^G), \quad (9)$$

where the notation G-Res $(\mathbf{X}, \mathbf{X}_r, \mathbf{A}^G)$ represents the graph residual term introduced in [Zhang and Meng, 2019] and $\mathbf{X}_r$ is the raw features of all nodes in the graph.

3.2 Improved Positional Embeddings from Graphs

Although combining graph neural networks and Transformer has shown effectiveness in modeling graph-structured data, the best architecture to incorporate them remains an issue and requires heavy hype-parameter searching. Therefore, it is meaningful to explore a graph-encoding strategy without adjustment of the Transformer architecture. Similar to the positional encoding in Transformer for sequential data such as sentences, it is also possible to compress the graph structure into positional embedding (PE) vectors and add them to the input before it is fed to the actual Transformer model:

$$\widetilde{\mathbf{X}} = \mathbf{X} + f_{\text{map}}(\mathbf{P}), \quad (10)$$

where $\mathbf{X} \in \mathrm{R}^{n \times d}$ is the matrix of input embeddings, $\mathbf{P} \in \mathrm{R}^{n \times d_p}$ represents the graph embedding vectors, and $f_{\text{map}} : \mathrm{R}^{d_p} \rightarrow \mathrm{R}^{d}$ is a transformation network to align the dimension of both vectors. The graph positional embedding $\mathbf{P}$ is normally generated from the adjacent matrix $\mathbf{A}^G \in \mathrm{R}^{n \times n}$.

[Dwivedi and Bresson, 2020] adopt Laplacian eigenvectors as $\mathbf{P}$ in Graph Transformer. For each graph in the dataset, they pre-compute the Laplacian eigenvectors , which are defined by the factorization of the graph Laplacian matrix:

$$\mathbf{U}^T \Lambda \mathbf{U} = \mathbf{I} - \mathbf{D}^{-1/2} \mathbf{A}^G \mathbf{D}^{-1/2}, \quad (11)$$

where $\mathbf{D}$ is the degree matrix, and $\Lambda, \mathbf{U}$ correspond to the eigenvalues and eigenvectors respectively. They use eigenvectors of the k smallest non-trivial eigenvalues as the positional embedding, with $\mathbf{P} \in \mathbb{R}^{n \times k}$ in this case. Since these eigenvectors have multiplicity occurring due to the arbitrary sign of eigenvectors, they randomly flip the sign of the eigenvectors during training.

[Hussain *et al.*, 2021] employs pre-computed SVD vectors of the adjacent matrix as the positional embeddings $\mathbf{P}$. They

use the largest r singular values and corresponding left and right singular vectors to represent the positional encoding.

$$\mathbf{A}^G \overset{\text{SVD}}{\approx} \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = (\mathbf{U}\sqrt{\mathbf{\Sigma}}) \cdot (\mathbf{V}\sqrt{\mathbf{\Sigma}})^T = \hat{\mathbf{U}}\hat{\mathbf{V}}^T, \quad (12)$$

$$\mathbf{P} = \hat{\mathbf{U}} \| \hat{\mathbf{V}}, \quad (13)$$

where $\mathbf{U}, \mathbf{V} \in \mathrm{R}^{n \times r}$ contain the r left and right singular vectors corresponding to the top r singular values in the diagonal matrix $\Sigma \in \mathbb{R}^{r \times r}$, $\|$ denotes concatenation operator along columns. They also randomly flip the signs during training as a form of data augmentation to avoid over-fitting, since similar to Laplacian eigenvectors, the signs of corresponding pairs of left and right singular vectors can be arbitrarily flipped.

Different from Eigen PE and SVD PE, which attempt to compress the adjacent matrix into dense positional embeddings, there exist some heuristic methods that encode specific structural information from the extraction of the adjacent matrix. For example, Graphormer [Ying *et al.*, 2021] uses the degree centrality as an additional signal to the neural network. To be specific, Graphformer assigns each node two real-valued embedding vectors according to its indegree and outdegree. For the i-th node, the degree-aware representation is denoted as:

$$\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{z}^-_{\deg^-(v_i)} + \mathbf{z}^+_{\deg^+(v_i)}, \quad (14)$$

where $\mathbf{z}^-, \mathbf{z}^+ \in \mathbb{R}^d$ are learnable embedding vectors specified by the indegree $\deg^-(v_i)$ and outdegree $\deg^+(v_i)$ respectively. For undirected graphs, $\deg^-(v_i)$ and $\deg^+(v_i)$ could be unified to be $\deg(v_i)$. By using the centrality encoding in the input, the softmax attention will catch the node importance signal in queries and keys of the Transformer.

Graph-BERT [Zhang *et al.*, 2020] introduces three types of PE to embed the node position information, *i.e.*, an absolute WL-PE which represents different codes labeled by the Weisfeiler-Lehman algorithm, an intimacy based PE and a hop based PE which are both variant to the sampled subgraphs. [Cai and Lam, 2020] applies graph transformer to tree-structured abstract meaning representation (AMR) graph. It adopts a distance embedding for each node by encoding the minimum distance from the root node as a flag of the importance of the corresponding concept in the whole-sentence semantics. [Kreuzer *et al.*, 2021] proposes a learned positional encoding that can utilize the full Laplacian spectrum to learn the position of each node in a given graph.

3.3 Improved Attention Matrices from Graphs

Although node positional embedding is a convenient practice to inject graph priors into Transformer architectures, the progress of compressing graph structure into fixed-sized vectors suffers from information loss, which might limit their effectiveness. For this sake, another group of works attempts to improve the attention matrix computation based on graph information:

$$\begin{aligned} \mathbf{A} &= f_{\text{G\_att}}(\mathbf{X}, \mathbf{A}^G, \mathbf{E}; \mathbf{Q}, \mathbf{K}, \mathbf{W}_1), \\ \mathbf{M} &= f_{\text{M}}(\mathbf{X}, \mathbf{A}, \mathbf{A}^G, \mathbf{E}; \mathbf{V}, \mathbf{W}_2), \end{aligned} \quad (15)$$

where $\mathbf{X}$ is the input features, $\mathbf{A}^G$ is the adjacent matrix of the graph, $\mathbf{E} \in \mathbb{R}^{n \times n \times d_e}$ is the edge features if available, $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are the attention parameters, $\mathbf{W}_1, \mathbf{W}_2$ are extra graph encoding parameters.

One line of models adapts self-attention mechanism to GNN-like architectures by restricting a node only attending to local node neighbours in the graph, which can be computationally formulated as an attention masking mechanism:

$$\mathbf{A} = (\frac{1}{\sqrt{d}}\mathbf{X}\mathbf{Q}(\mathbf{X}\mathbf{K})^\top) \odot \mathbf{A}^G, \quad (16)$$

where $\mathbf{A}^G \in \mathbb{R}^{n \times n}$ is the adjacent matrix of the graph. In [Dwivedi and Bresson, 2020], $\mathbf{A}^G_{ij} = 1$ if there is an edge between i-th node and j-th node. Given the simple and generic nature of this architecture, they obtain competitive performance against standard GNNs on graph datasets. In order to encode edge features, they also extend Equation 16 as:

$$\mathbf{A} = (\frac{1}{\sqrt{d}}\mathbf{X}\mathbf{Q}(\mathbf{X}\mathbf{K})^\top) \odot \mathbf{A}^G \odot \mathbf{E}'\mathbf{W}_E, \quad (17)$$

where $\mathbf{W}_E \in \mathbb{R}^{d_e \times d}$ is the parameter matrix, $\mathbf{E}'$ is the edge embedding matrix from the previous layer which is updated based on $\mathbf{A}$, we do not elaborate these details for simplicity.

One possible extension of this practice is masking the attention matrices of different heads with different graph priors. In the original multi-head self-attention blocks, different attention heads implicitly attend to information from different representation subspaces of different nodes. While in this case, using the graph-masking mechanism to enforce the heads explicitly attend to different subspaces with graph priors further improves the model representative capability for graph data. For example, [Yao *et al.*, 2020] computes the attention based on the extended Levi graph. Since Levi graph is a heterogeneous graph that contains different types of edges. They first group all edge types into a single one to get a homogeneous subgraph referred to as connected subgraph. The connected subgraph is actually an undirected graph that contains the complete connected information in the original graph. Then they split the input graph into multiple subgraphs according to the edge types. Besides learning the directly connected relations, they introduce a fully-connected subgraph to learn the implicit relationships between indirectly connected nodes. Multiple adjacent matrices are assigned to different attention heads to learn a better representation for AMR task. [Min *et al.*, 2022] adopts a similar practice, which carefully designs four types of interaction graphs for modeling neighbourhood relations in CTR prediction task: induced subgraph, similarity graph, cross-neighbourhood graph, and complete graph. And they use the masking mechanism to encode these graph priors to improve neighbourhood representation.

GraphiT [Mialon *et al.*, 2021] extends the adjacent matrix to a kernel matrix, which is more flexible to encode various graph kernels. Besides, they use the same matrix for keys and queries following the recommendation of [Tsai *et al.*, 2019] to reduce parameters without hurting the performance in practice, and adopt a degree matrix to reduce the overwhelming influence of highly connected graph components. The update equation can be formulated as:

$$\begin{aligned} \mathbf{A} &= (\frac{1}{\sqrt{d}}\mathbf{X}\mathbf{Q}(\mathbf{X}\mathbf{Q})^\top) \odot \mathbf{K}_r, \\ \widetilde{\mathbf{X}} &= \text{SoftMax}(\mathbf{A})(\mathbf{X}\mathbf{V}), \\ \mathbf{M} &= \text{LayerNorm}(\mathbf{D}^{-\frac{1}{2}}\widetilde{\mathbf{X}} + \mathbf{X}), \end{aligned} \quad (18)$$

where $\mathbf{D} \in \mathbb{R}^{n\times n}$ is the diagonal matrix of node degrees, $\mathbf{K}_r \in \mathbb{R}^{n\times n}$ is the kernel matrix on the graph, which is used as diffusion kernel and p-step random walk kernel.

Another line of models attempts to add soft graph bias to attention scores. Graphormer [Ying *et al.*, 2021] proposes a novel Spatial Encoding mechanism. Concretely, they consider a distance function $\phi(v_i, v_j)$, which measures the spatial relation between nodes v_i and v_j in the graph. They select $\phi(v_i, v_j)$ as the shortest path distance (SPD) between v_i and v_j. If they are not connected, the output of ϕ is set as a special value, *i.e.*, -1. They assign each feasible value of ϕ a learnable scale parameter as a graph bias term. So the update rule is:

$$\mathbf{A} = (\frac{1}{\sqrt{d}}\mathbf{XQ}(\mathbf{XK})^\top) + \mathbf{B}^s. \tag{19}$$

$\mathbf{B}^s$ is the bias matrix, where $\mathbf{B}^s_{ij} = b_{\phi(v_i,v_k)}$ is the learnable scalar indexed by $\phi(v_i, v_k)$, and shared across all layers. In order to handle graph structure with edge features, they also design an edge feature bias term. Specifically, for each ordered node pair (v_i, v_j), they search (one of) the shortest path $\mathrm{SP}_{ij} = (e_1, e_2, \ldots, e_N)$ from v_i to v_j, and then compute an average of dot-products of the edge features and a learnable embedding along the path. Combined with the above spatial bias, the unnormalized attention score can be modified as:

$$\mathbf{A} = (\frac{1}{\sqrt{d}}\mathbf{XQ}(\mathbf{XK})^\top) + \mathbf{B}^s + \mathbf{B}^c \tag{20}$$

where $\mathbf{B}^c$ is the edge feature bias matrix. $\mathbf{B}^c_{ij} = \frac{1}{N}\sum_{n=1}^{N} x_{e_n}(w^E_n)^\top$, where x_{e_n} is the feature of the n-th edge e_n in SP_{ij}, $w^E_n \in \mathbb{R}^{d_e}$ is the n-th weight embedding, and d_e is the dimensionality of edge feature.

Gophormer [Zhao *et al.*, 2021] proposes proximity-enhanced multi-head attention (PE-MHA) to encode multi-hop graph information. Specifically, for a node pair $\langle v_i, v_j\rangle$, M views of structural information is encoded as a proximity encoding vector, denoted as $\phi_{ij} \in \mathbb{R}^M$, to enhance the attention mechanism. The proximity-enhanced attention score $\mathbf{A}_{ij}$ is defined as:

$$\mathbf{A}_{ij} = (\frac{1}{\sqrt{d}}\mathbf{x}_i\mathbf{Q}(\mathbf{x}_j\mathbf{K})^\top) + \boldsymbol{\phi}^{ij}\mathbf{b}^\top, \tag{21}$$

where $\mathbf{b} \in \mathbb{R}^M$ is the learnable parameters that compute the bias of structural information. The proximity encoding is calculated by M structural encoding functions defined as:

$$\boldsymbol{\phi}_{ij} = \mathrm{Concat}(\Phi_m(v_i, v_j)|m \in 0, 1, \ldots, M-1), \tag{22}$$

where each structural encoding function $\Phi_m(\cdot)$ encodes a view of structural information.

PLAN [Khoo *et al.*, 2020] also proposes a structure aware self-attention to model the tree structure of rumour propagation in social media. The modified attention calculation can be defined as:

$$\mathbf{A}_{ij} = \frac{1}{\sqrt{d}}(\mathbf{x}_i\mathbf{Q}(\mathbf{x}_j\mathbf{K})^\top) + a^K_{ij} \tag{23}$$

$$\mathbf{M}_i = \sum_{j=1}^{n} \mathrm{SoftMax}(\mathbf{A}_{ij})(\mathbf{x}_j\mathbf{V} + a^V_{ij}) \tag{24}$$

Both a^V_{ij} and a^K_{ij} are learnable parameter vectors that represent one of the five possible structural relationships between the pair of the tweets (i.e. parent, child, before, after and self).

Table 2: Details of the datasets.

Dataset Type	Name	#Graph	#Nodes(Avg.)	#Edges(Avg.)	Task type	Metric
Graph-level	ZINC	12,000	23.2	49.8	Regression	MAE
	ogbg-molhiv	41,127	25.5	27.5	Binary cla.	ROC-AUC
	ogbg-molpcba	437,929	26.0	28.1	Binary cla.	AP
Node-level	Flickr	1	89,250	899,756	Multi-class cla.	Accuracy
	ogbg-arxiv	1	169,343	1,166,243	Multi-class cla.	Accuracy
	ogbg-product	1	2,449,029	61,859,140	Multi-class cla.	Accuracy

Table 3: Hyper-parameters of various Transformer sizes.

	Transformer Size		
	Small	Middle	Large
#Layers	6	12	12
Hidden Dimension	80	80	512
FFN Inner Hidden Dimension	80	80	512
#Attention Heads	8	8	32
Hidden Dimension of Each Head	10	10	16

4 Experimental Evaluations

We conduct an extensive evaluation to study the effectiveness of different methods. On the basis of standard Transformer, methods in the three groups, auxiliary GNN modules (GA), positional embeddings (PE), and improved attention (AT) are compared. Since most methods are composed of more than one graph-specific module and are trained with various tricks, it is difficult to evaluate the effectiveness of each module in a fair and independent way. In our experiments, we extract the representative modules of existing models and evaluate their performance individually. For GA methods, we compare the three architectures described in Section 3.1. In both *alternately* and *parallel* settings, the GNNs and Transformer layers are combined before the FFN layers. PE methods include degree embedding [Ying *et al.*, 2021], Laplacian Eigenvectors [Dwivedi and Bresson, 2020] and SVD vectors [Hussain *et al.*, 2021]. AT methods contain spatial bias (SPB) [Ying *et al.*, 2021], proximity-enhanced multi-head attention (PMA) [Zhao *et al.*, 2021], attention masked with 1-hop adjacent matrix (Mask-1) [Dwivedi and Bresson, 2020]. We also mask attention with different hops of the adjacent matrix, denoted as (Mask-n), inspired by the multi-head masking mechanisms in [Yao *et al.*, 2020; Min *et al.*, 2022]. We evaluate the methods on both graph-level tasks on small graphs and node-level tasks on a single large graph. The details of the tasks and datasets are listed in Table 2.

4.1 Settings and Implementation Details

To ensure a consistent and fair evaluation, we fix the scale of the Transformer architecture in three levels: small, middle, and large, whose configurations are listed in Table 3. The remaining hyper-parameters are fixed to their empirical value as shown in Table 5. For node-level tasks in large-scale graphs, we adopt the shadow k-hop sampling [Zeng *et al.*, 2020] to generate a subgraph for a target node, to which the graph-aware modules are applied. We calculate the required inputs of all modules based on the subgraph instead of the original graph because most of them are not computationally scalable to large graphs. We set the maximum sampling hop to be 2 and the maximum neighbourhood per node to be 10. For PE methods, we select the embedding size from $\{3, 4, 5\}$. For GA methods, we select the GNN type from GCN [Kipf and Welling, 2016], GAT [Velivckovic *et al.*, 2017] and GIN [Xu

Table 4: Performance comparison on graph-level tasks. For each column, the values with underline are the best results in each group. The values in bold are the best results across the groups.

		graph-level tasks									node-level tasks								
		ZINC(MAE↓)			molhiv(ROC-AUC↑)			molpcba(AP↑)			Flickr(Acc↑)			arixv(Acc↑)			product(Acc↑)		
		Small	Middle	Large	Small	Middle	Large	Small	Middle	Large	Small	Middle	Large	Small	Middle	Large	Small	Middle	Large
TF	vanilla	0.6689	0.6700	0.6699	0.7466	0.7230	0.7269	0.1624	0.1673	0.1676	0.5270	0.5279	0.5214	0.5539	0.5571	0.5598	0.7887	0.7887	0.7956
GA	before	0.4700	0.4809	0.5169	0.6758	0.7339	0.7182	0.2105	0.1989	0.2269	0.5352	0.5369	0.5272	0.5608	0.5590	0.5614	**0.7953**	0.7888	0.8012
	alter	0.3771	0.3412	0.2956	0.7200	0.7086	0.7433	0.2474	0.2417	0.2244	**0.5374**	0.5357	0.5162	0.5599	0.5555	0.5592	0.7905	0.7915	**0.8057**
	parallel	0.3803	**0.2749**	0.2984	0.7138	0.7750	0.7603	0.2345	0.2444	0.2205	0.5370	**0.5379**	0.5209	**0.5647**	0.5600	0.5529	0.7878	0.7896	0.7999
PE	degree	0.5364	0.5364	0.5435	0.7506	0.6818	0.7357	0.1672	0.1646	0.1650	0.5291	0.5250	0.5133	0.5551	0.5618	0.5502	0.7920	0.7913	0.7947
	eig	0.6031	0.6074	0.6166	0.7407	0.7279	0.7351	0.2194	0.2087	0.2131	0.5253	0.5278	0.5257	0.5575	0.5637	0.5658	0.7893	0.7887	0.8017
	svd	0.5811	0.5462	0.5400	0.7350	0.7155	0.7275	0.1648	0.1663	0.1767	0.5281	0.5317	0.5203	0.5614	**0.5663**	**0.5706**	0.7856	0.7893	0.8007
AT	SPB	0.5122	0.4878	0.6100	0.7065	0.7589	0.7255	0.2409	0.2524	**0.2621**	0.5368	0.5364	0.5234	0.5503	0.5605	0.5576	0.7921	**0.7918**	0.7984
	PMA	0.6194	0.6057	0.5963	0.6902	0.7054	0.7314	0.2115	0.1956	0.2518	0.5240	0.5288	0.5204	0.5567	0.5571	0.5492	0.7877	0.7853	0.7945
	Mask-1	**0.2861**	0.2772	**0.2894**	**0.7610**	**0.7753**	**0.7960**	0.2573	**0.2662**	0.1594	0.5295	0.5300	0.5236	0.5598	0.5559	0.5583	0.7923	0.7917	0.7963
	Mask-n	0.3906	0.3596	0.4765	0.7286	0.7423	0.7128	**0.2619**	0.2577	0.2380	0.5359	0.5349	**0.5348**	0.5593	0.5603	0.5576	0.7935	0.7917	0.8016

Table 5: The training hyper-parameters.

attention dropout	0.1	FFN dropout	0.1
maximum steps	1e+6	warm-up steps	4e+4
peak learning rate	2e-4	batch size	256
weight decay value	1e-3	lr decay	Linear
gradient clip norm	5.0	Adam $\epsilon, \beta_1, \beta_2$	1e-8, 0.9, 0.99

et al., 2018a]. Our learning framework is built on PyTorch 1.8, and PyTorch Geometric 2.0, and the code is public at https://github.com/qwerfdsaplking/Graph-Trans.

4.2 Experimental Results

Table 4 reports the performance of ten graph-specific modules from three categories on six datasets of graph-level and node-level tasks, respectively. We summarize the main observations as follows:

- Not surprisingly, in most cases, the evaluated graph-specific modules of Transformer lead to to better performance. For example, on molpcba, we observe at most a 56% performance improvement compared with the vanilla Transformer. This observation confirms the effectiveness of graph-specific modules on the various kinds of graph tasks.
- The improvement on graph-level tasks is more significant than that on node-level tasks. This may be due to the graph sampling process when dealing with a single large graph. The sampled induced subgraphs fail to keep the original graph intact, incurring variance and information loss in the learning process.
- GA and AT methods bring more benefits than PE methods. In conclusion, PE's weaknesses are twofold. First, as introduced in Section 3.2, PE does not contain intact graph information. Second, since PE is only fed into the input layer of the network, the graph-structural information would decay layer by layer across the model, leading to a degeneration of the performance.
- It seems that different kinds of graph tasks enjoy different group of models. GA methods achieve the best performance in more than half of the cases (5 out of 9 cases) of node-level tasks. More significantly, AT methods achieve the best performance in almost all the cases (8 out of 9 cases) of graph-level tasks. We conjecture that GA methods are able to better encode the local information of the sampled induced subgraphs in node-level tasks, while AT methods are suitable for modeling the global information of the single graphs in graph-level tasks.

In summary, our experiments confirm the value of the existing graph-specific module on Transformer and reveal their advantages for a variety of graph tasks.

5 Conclusion and Future Directions

In this survey, we present a comprehensive review of the current progress of applying Transformer model on graphs. Based on the injection part of graph data and graph model in Transformer, we classify the existing graph-specific modules into three typical groups: GNNs as auxiliary modules, improved positional embeddings from graphs, and improved attention matrices from graphs. To investigate the real performance gains under a fair setup, we implement the representative modules from three groups and compare them on six benchmarks with different tasks. We hope our systematic review and comparison will foster understanding and stimulate new ideas in this area.

In spite of the current successes, we believe several directions would be promising to investigate further and to start from in the future, including:

- **New paradigm of incorporating the graph and the Transformer.** Most studies treat the graphs as strong prior to modifying the Transformer model. There is a great interest to develop the new paradigm that not just takes graphs as a prior, but also better reflects the properties of graphs.
- **Extending to other kinds of graphs.** Existing Graph Transformer models mostly focus on homogeneous graphs, which consider the node type and edge type as the same. It is also important to explore their potential on other forms of graphs, such as heterogeneous graphs and hypergraphs.
- **Extending to large-scale graphs.** Most existing methods are designed for small graphs, which might be computationally infeasible for large graphs. As illustrated in our experiments, directly applying them to the sampled subgraphs would impair performance. Therefore, designing salable Graph-Transformer architecture is essential.

References

[Aggarwal, 2010] Charu C. Aggarwal. *Graph Clustering*, pages 459–467. Springer US, Boston, MA, 2010.

[Ba *et al.*, 2016] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[Cai and Lam, 2020] Deng Cai and Wai Lam. Graph transformer for graph-to-sequence learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7464–7471, 2020.

[Chanussot* *et al.*, 2021] Lowik Chanussot*, Abhishek Das*, Siddharth Goyal*, Thibaut Lavril*, Muhammed Shuaibi*, Morgane Riviere, Kevin Tran, Javier Heras-Domingo, Caleb Ho, Weihua Hu, Aini Palizhati, Anuroop Sriram, Brandon Wood, Junwoong Yoon, Devi Parikh, C. Lawrence Zitnick, and Zachary Ulissi. Open catalyst 2020 (oc20) dataset and community challenges. *ACS Catalysis*, 2021.

[Chen *et al.*, 2020] Dexiong Chen, Laurent Jacob, and Julien Mairal. Convolutional kernel networks for graph-structured data. In *International Conference on Machine Learning*, pages 1576–1586. PMLR, 2020.

[Dwivedi and Bresson, 2020] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[Forsyth and Ponce, 2011] David Forsyth and Jean Ponce. *Computer vision: A modern approach.* Prentice hall, 2011.

[Fuchs *et al.*, 2020] Fabian Fuchs, Daniel Worrall, Volker Fischer, and Max Welling. Se (3)-transformers: 3d roto-translation equivariant attention networks. *Advances in Neural Information Processing Systems*, 33:1970–1981, 2020.

[Gilmer *et al.*, 2017] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[Hamilton, 2020] James Douglas Hamilton. *Time series analysis*. Princeton university press, 2020.

[Hu *et al.*, 2021] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

[Hussain *et al.*, 2021] Md Shamim Hussain, Mohammed J Zaki, and Dharmashankar Subramanian. Edge-augmented graph transformers: Global self-attention is enough for graphs. *arXiv preprint arXiv:2108.03348*, 2021.

[Jain *et al.*, 2021] Paras Jain, Zhanghao Wu, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems*, 34, 2021.

[Khoo *et al.*, 2020] Ling Min Serena Khoo, Hai Leong Chieu, Zhong Qian, and Jing Jiang. Interpretable rumor detection in microblogs by attending to user interactions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8783–8790, 2020.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Kreuzer *et al.*, 2021] Devin Kreuzer, Dominique Beaini, William L Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *arXiv preprint arXiv:2106.03893*, 2021.

[Lin *et al.*, 2021] Kevin Lin, Lijuan Wang, and Zicheng Liu. Mesh graphormer. *arXiv preprint arXiv:2104.00272*, 2021.

[Maron *et al.*, 2019] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019.

[Mialon *et al.*, 2021] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

[Min *et al.*, 2022] Erxue Min, Yu Rong, Tingyang Xu, Yatao Bian, Peilin Zhao, Junzhou Huang, Da Luo, Kangyi Lin, and Sophia Ananiadou. Masked transformer for neighhourhood-aware click-through rate prediction. *arXiv preprint arXiv:2201.13311*, 2022.

[Nguyen *et al.*, 2019] Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. Universal graph transformer self-attention networks. *arXiv preprint arXiv:1909.11855*, 2019.

[Purwins *et al.*, 2019] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

[Rong *et al.*, 2019] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[Rong *et al.*, 2020] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. *arXiv preprint arXiv:2007.02835*, 2020.

[Schmitt *et al.*, 2020] Martin Schmitt, Leonardo FR Ribeiro, Philipp Dufter, Iryna Gurevych, and Hinrich Schütze. Modeling graph structure via relative position for text generation from knowledge graphs. *arXiv preprint arXiv:2006.09242*, 2020.

[Shi *et al.*, 2020] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[Shiv and Quirk, 2019] Vighnesh Shiv and Chris Quirk. Novel positional encodings to enable tree-based transformers. *Advances in Neural Information Processing Systems*, 32, 2019.

[Thölke and de Fabritiis, 2022] Philipp Thölke and Gianni de Fabritiis. Equivariant transformers for neural network based molecular potentials. 2022.

[Tsai *et al.*, 2019] Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: A unified understanding of transformer's attention via the lens of kernel. *arXiv preprint arXiv:1908.11775*, 2019.

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

[Velivckovic *et al.*, 2017] Petar Velivckovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[Wang *et al.*, 2019] Xing Wang, Zhaopeng Tu, Longyue Wang, and Shuming Shi. Self-attention with structural position representations. *arXiv preprint arXiv:1909.00383*, 2019.

[Xu *et al.*, 2018a] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[Xu *et al.*, 2018b] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR, 2018.

[Yang *et al.*, 2016] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[Yao *et al.*, 2020] Shaowei Yao, Tianming Wang, and Xiaojun Wan. Heterogeneous graph transformer for graph-to-sequence learning. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7145–7154, 2020.

[Ying *et al.*, 2021] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation? *arXiv preprint arXiv:2106.05234*, 2021.

[Zeng *et al.*, 2020] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. Deep graph neural networks with shallow subgraph samplers. *arXiv preprint arXiv:2012.01380*, 2020.

[Zhang and Chen, 2018] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in neural information processing systems*, 31, 2018.

[Zhang and Meng, 2019] Jiawei Zhang and Lin Meng. Gresnet: Graph residual network for reviving deep gnns from suspended animation. *arXiv preprint arXiv:1909.05729*, 2019.

[Zhang *et al.*, 2020] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.

[Zhao and Akoglu, 2019] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

[Zhao *et al.*, 2021] Jianan Zhao, Chaozhuo Li, Qianlong Wen, Yiqi Wang, Yuming Liu, Hao Sun, Xing Xie, and Yanfang Ye. Gophormer: Ego-graph transformer for node classification. *arXiv preprint arXiv:2110.13094*, 2021.

[Zhu *et al.*, 2019] Jie Zhu, Junhui Li, Muhua Zhu, Longhua Qian, Min Zhang, and Guodong Zhou. Modeling graph structure in transformer for better amr-to-text generation. *arXiv preprint arXiv:1909.00136*, 2019.