LiteGT: Efficient and Lightweight Graph Transformers

Cong Chen
chencong@eee.hku.hk
The University of Hong Kong
Hong Kong SAR, China

Chaofan Tao
cftao@eee.hku.hk
The University of Hong Kong
Hong Kong SAR, China

Ngai Wong
nwong@eee.hku.hk
The University of Hong Kong
Hong Kong SAR, China

ABSTRACT

Transformers have shown great potential for modeling long-term dependencies for natural language processing and computer vision. However, little study has applied transformers to graphs, which is challenging due to the poor scalability of the attention mechanism and the under-exploration of graph inductive bias. To bridge this gap, we propose a Lite Graph Transformer (LiteGT) that learns on arbitrary graphs efficiently. First, a node sampling strategy is proposed to sparsify the considered nodes in self-attention with only $O(N \log N)$ time. Second, we devise two kernelization approaches to form two-branch attention blocks, which not only leverage graph-specific topology information, but also reduce computation further to $O(\frac{1}{2} N \log N)$. Third, the nodes are updated with different attention schemes during training, thus largely mitigating over-smoothing problems when the model layers deepen. Extensive experiments demonstrate that LiteGT achieves competitive performance on both *node classification* and *link prediction* on datasets with millions of nodes. Specifically, *Jaccard + Sampling + Dim. reducing* setting reduces more than 100× computation and halves the model size without performance degradation.

CCS CONCEPTS

• **Computer systems organization** → **Neural networks**; • **Computing methodologies** → *Supervised learning by classification.*

KEYWORDS

graph neural network; transformer; efficient training and inference

ACM Reference Format:
Cong Chen, Chaofan Tao, and Ngai Wong. 2021. LiteGT: Efficient and Lightweight Graph Transformers. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482272

1 INTRODUCTION

Graph Neural Network (GNN) is a type of neural network that processes information on graph data. GNN has been widely employed on various graph-specific tasks. Node-level tasks usually attempt to classify the label of a node or conduct node regression by learning semantically rich representation on nodes. Edge-level tasks output the properties of the target edge, *e.g.* predict the links that may appear in the future or classify the attributes related to edges. Graph-level tasks aggregate node information and edge information to obtain a compact representation for each graph, which is utilized for downstream tasks like graph classification. GNN has been commonly adopted in real-world applications such as traffic prediction [47], molecular fingerprints learning [18] and recommendation systems [44].

On the other hand, transformer [39] is a novel encoder-decoder architecture that mainly consists of self-attention modules and feed-forward networks, which is initially designed for Natural Language Processing (NLP). The success of transformer [16, 37, 38] witnesses the superiority of attention mechanism over recurrent neural networks on various NLP tasks. Transformer demonstrates great potential in attention mechanism which can be effectively leveraged with large neural networks and sufficient data in NLP. For instance, BERT [16] proposed in 2018 has 340 million parameters and obtains state-of-the-art results on 11 NLP tasks. And GPT-3 [5] proposed in 2020 outperforms BERT by a large margin with an overwhelming 175 billion parameters.

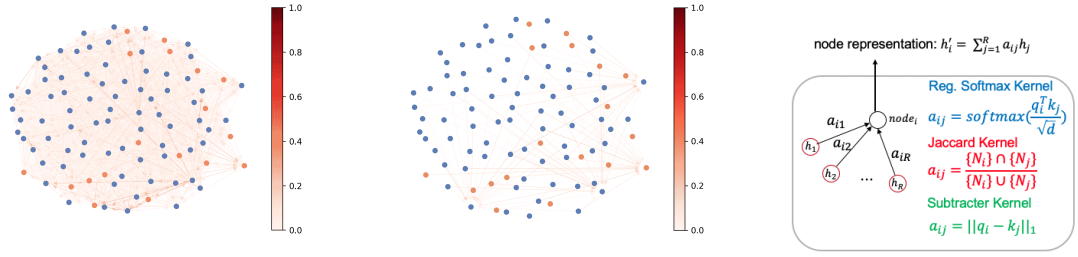
With the development of transformer, there is an increasing interest in applying transformer architectures into computer vision (CV), including image classification [17], object detection [6] and panoptic segmentation [41], video action recognition [22], *etc.* However, few studies have explored transformers architectures on GNNs. Compared with text, image or video that can be measured in the Euclidean space typically, the challenge of learning on graphs lies in the complicated relationships among connected instances (nodes) in the non-Euclidean space.

As for graph data, the attention mechanism in the transformer is a promising tool to investigate the long-term relationships, which are studied in recent works [24, 40, 45, 46]. From the view of message-passing, attention mechanism learns each node's representation by automatically adjusting the weights (contributions) from the representation of its neighbors and then aggregating them together. GAT [40] is a representative approach that employs an attention mechanism to learn the weights between any neighbor nodes, instead of using pre-determined weights like GCN [27] or equal weights like GrahSAGE [23]. GTN [45] utilizes attention mechanism that generates multi-hop connections, *i.e. meta-paths*, for learning high-order relations sequentially. HGT [24] extends the attention to heterogeneous graphs to deal with different types of nodes and edges. Graph-BERT [46] emphasizes the importance of pre-training on GNNs via Graph-based BERT architecture.

However, the aforementioned GNNs neglect the learning efficiency on the attention mechanism. A well-known self-attention module requires $O(N^2)$ time and space complexity to deal with instance-instance relations for each node pair. It introduces heavy

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482272



(a) Vanilla graph transformer (b) Node sampling (c) Attention Kernels

Figure 1: (a) and (b) depict the graph attention weight map of vanilla graph transformer before and after node sampling, respectively. The graph nodes are colored according to their labels, whereas the edges are colored according to the magnitude of the attention weights, corresponding to the color bar on the right. (c) shows the different kernel functions for computing graph attention.

computational needs, thus resulting in poor scalability on large-scale tasks where millions or even billions of nodes are involved. Even worse, approaches [15, 29] that require the global information of the whole graph connectivity are hard for mini-batch training or distributed training. Also, the ignorance of graph inductive bias like graph topology leads to underestimated performance in the previous work. Further, commonly-used compression and speedup techniques in neural networks, like convolution pruning and kernel low-rank decomposition cannot be directly applied on graphs due to different data formats and network components.

Present work. To overcome the shortcomings, we devise a space-time efficient graph transformer dubbed LiteGT for reducing *computation*, *memory* and *model size*, while delivering competitive performance on downstream tasks like node classification and link prediction.

The LiteGT is efficient from three levels. 1) **Instance-level** Instead of considering all the first-hop neighbors or multi-hop neighbors to compute attention, we argue that the attentions are sparsely distributed among the nodes based on the empirical observation on the entropy histograms in the graph attention. We propose a Kullback-Leibler divergence-base measure to evaluate the importance of each node before model training, and then sample only top-$logN$ important nodes for attention value computation, as shown in Figure 1(a) and (b). 2) **Relation-level** Instead of employing dense attention computation for all considered node pairs, we relieve the computation burden of attention itself and learn diverse features. Inspired by the generalized kernelizable attention [11], the conventional attention can be viewed as a special case that employs regularized softmax-kernel function. We then propose two versions of two-branch attention paradigm that one branch considers sampling nodes with regularized softmax kernel, and another branch adopts a lightweight kernel that considers subtraction only, or even fixed graph-specific information, *i.e.* Jaccard patterns (shown as Figure 1(c)). 3) **Vector-level** We further reduce the model size by reducing of hidden features at several block intervals. The pyramid-shape transformer architecture is found to be effective for dimension reduction while upholding performance.

Another important property is that our method effectively alleviates the over-smoothing problem [35] when the layers go deep. Specifically, the nodes are updated with different attention schemes during training, thus avoiding all node features converging to a single representation.

Main contributions. 1) We propose a three-level efficient graph transformer architecture called LiteGT to learn on arbitrary graphs, which saves computation, memory and model size altogether. 2) We propose an efficient node sampling method, and two versions of two-branch attention mechanism are destined to not only learn diverse node relations, but also reduce the computation in attention itself. 3) Extensive experiments on node classification and link prediction verify that LiteGT gives competitive performance vs state-of-the-art approaches.

2 RELATED WORKS

2.1 Graph Neural Networks

GNNs are generally divided into spectral-based models [15, 27, 29] and spatial-based models [19, 40, 45, 47]. Spectral-based models process the graph as a signal filtered by self-defined graph convolutions, which usually remove high-frequency components. Whereas spatial-based models attempt to aggregate the features from each node itself, neighbor nodes and connected edges, to derive a semantically meaningful representation for each node or edge. MPNN [21] formally defines graph convolution as a message-passing process, which runs multiple steps to propagate information from different nodes. Recently, [19] brings transformers into graph learning and shows their effectiveness on graph tasks.

Spectral-based models enjoy strict mathematical guarantees, while they suffer from inflexibility. Since spectral-based model needs the eigenvector of graph Laplacian matrix or the whole graph connectivity, small perturbations on the graph will degrade the performance, as the eigenvectors are changed after perturbation. Alternatively speaking, the generalization of spectral-based models on new graphs is usually not satisfactory. Compared with spectral-based models, spatial-based models are more flexible in handling arbitrary graphs. Transformer is naturally a kind of spatial-based method suitable for arbitrary graphs, and offers great potential in

various fields. To this end, the proposed LiteGT does not rely on global graph information during training, making it easy to train in parallel.

2.2 Transformer Architectures

A typical transformer [5, 6, 16, 17, 37, 38, 41] stacks multiple blocks, each block having a self-attention module and a feed-forward network. Positional embedding is also employed as extra input to indicate the position of each input element. Self-attention module is the key to success that learns long-term relation in the context. However, transformers in both NLP [5] and CV [17] demonstrate that a deep and wide architecture pretrained on large-scale datasets is essential for good performance. The large model size and heavy memory footprint hinder the applications of transformers. To deal with that, different strategies are proposed, including the use of predefined patterns [36], sparsification on attention matrix [10, 48], low-rank mapping [42] and recurrent mechanism on connecting blocks instead of stacking blocks [14]. For example, Linformer [42] projects both key and value to low-dimension space before calculating the attention matrix so as to mitigate computation. Performer [12] recasts attention mechanism as a generalized kernelization to approximate attention matrix. The aforementioned methods generally serve NLP and CV tasks where the input data is easily measured in the Euclidean space, yet cannot directly apply on non-Euclidean graph data having arbitrary structures and millions of nodes.

2.3 Efficiency Improvement on Graphs

Training traditional GNNs like GCNs [27] requires the connectivity information and intermediate features for the whole graph. Alternatively speaking, training on that network needs full batch data. GraphSage [23] provides a mini-batch training solution by sampling from root node recursively with a fixed size. It enables to learn relations from multi-hop neighbors via multiple sampling iterations. Although GraphSage saves memory via batch training, it is time-consuming in extracting multi-hop neighbors. In Fast-GCN [7], the graph convolutions are viewed as integral transforms of embedding functions under probability measures. Monte Carlo is employed to estimate the integral transforms. For each layer, the nodes are sampled independently, which reduces the inter-layer dependencies. Cluster-GCN [9] utilizes graph clustering to sample subgraphs and then performs graph convolutions on the nodes within the subgraph. Graph-BERT [46] proposes to train graphs linklessly based on attention mechanism without feature aggregation. In contrast to the aforementioned approaches that mainly focus on data sampling, the contribution of LiteGT is orthogonal and therefore can be readily plugged in for further improvement. LiteGT not only reduces the time and space complexity of the attention mechanism itself, but also learns diverse relations via a two-branch attention paradigm.

3 PRELIMINARY

In this section, we briefly introduce the idea of the self-attention mechanism in a vanilla transformer. Then, we demonstrate the generalized kernelizable attention. A generalization of transformer structure on graph data is also developed.

3.1 Regular Attention Mechanism

In a vanilla transformer, the attention is defined on receiving the tuple input (query $\boldsymbol{Q}$, key $\boldsymbol{K}$, value $\boldsymbol{V}$) and performing the scaled dot-product as

$$\mathrm{Att}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}})\boldsymbol{V} \tag{1}$$

where $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N\times d}$ (assuming without loss of generality they have the same input dimension d). N and d are respectively the word number and the hidden dimension.

3.2 Generalized Kernelizable Attention

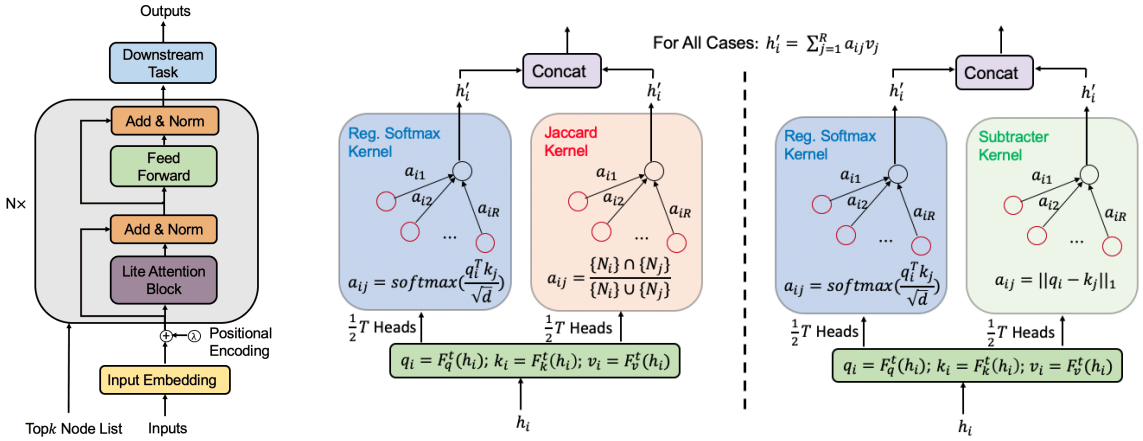
The regular attention mechanism demonstrates high similarity for strong relations between token pairs. Ref [12] shows that the attention mechanism can be extended as a generalized similarity measured via kernel function. Specifically, the query $\boldsymbol{Q}$ and key $\boldsymbol{K}$ are combined by a kernel function. The attention can be defined as

$$\hat{\mathrm{Att}}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \left[g(\boldsymbol{Q}^T)\mathrm{K}(\boldsymbol{Q}^T, \boldsymbol{K}^T)h(\boldsymbol{K}^T)\right]\boldsymbol{V} \tag{2}$$

where K is an arbitrary kernel function. The regular attention can be viewed as a special case that adopts linear kernel function and softmax operation on one's query and another's key, and ignore mapping $g(\cdot)$ and $h(\cdot)$. It inspires us to re-design attention with a lightweight and fast-computing kernel function, or even fixed patterns.

3.3 Transformer on Graph

As we mentioned before, transformer has demonstrated great potential in attention mechanism, and there is a trend to apply attention mechanism on graph data processing in recent years. However, the vanilla transformer is mainly used in NLP tasks, and the data are commonly sequential words. Therefore, when applying a transformer on graph data, two basic settings, namely self-attention mechanism, and positional encoding, need to be modified. Firstly, in the vanilla transformer, each word would attend to all remaining words in a sentence. However, this setting is not suitable for graph tasks. First, the nodes in a graph have a specific connecting structure. If a node attends all the remaining nodes in a graph, the natural graph structure will be undesirably abandoned. Second, real-life graph data, such as social networks and recommender systems often have millions or billions of nodes. It is infeasible to implement the full attention like in the vanilla transformer due to the huge computation and storage costs. Therefore, [19] proposed a graph transformer (GT), wherein a node would only attend to its direct neighbors. As for the positional encoding, the vanilla transformer utilizes the cosine and sinusoidal functions to uniquely record the position of each word. Nevertheless, such a positional encoding strategy would fail when applied to graph data since graphs may exhibit some symmetries in their structures, such as node or edge isomorphism [19, 33]. Therefore, GT proposed to use graph Laplacian eigenvectors [2] as graph node positional encoding, which is also utilized in LiteGT.



(a) The LiteGT model. (b) The two possible formats of lite attention block.

Figure 2: (a) depicts the proposed LiteGT model. (b) shows the two formats of the lite attention block. The two branches of the lite attention block employ different attention kernels. The first branch would always employ regularized softmax-kernel, while the second branch employs either Jaccard or Subtracter kernel.

Let $\boldsymbol{q}_i, \boldsymbol{k}_i, \boldsymbol{v}_i$ be the i-th row of $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ respectively. The update equation of $node_i$ in a graph attention layer of GT reads:

$$\boldsymbol{h}_i^{'} = \sum_{j \in \{\boldsymbol{\mathcal{N}}_i\}} a_{ij} \boldsymbol{v}_j, \tag{3}$$

where $\{\boldsymbol{\mathcal{N}}_i\}$ is the set of the direct neighbors of $node_i$, a_{ij} is the attention score, and can be computed as

$$a_{ij} = softmax_j(\frac{\boldsymbol{q}_i \boldsymbol{k}_j^T}{\sqrt{d}}) = \frac{\exp(\boldsymbol{q}_i \boldsymbol{k}_j^T / \sqrt{d})}{\sum_{r \in \{\boldsymbol{\mathcal{N}}_i\}} \exp(\boldsymbol{q}_i \boldsymbol{k}_r^T / \sqrt{d})}. \tag{4}$$

Therefore, the complexity of computing (4) for the whole graph is $\boldsymbol{O}(NRd)$, where N and R are the graph node number and the average neighbor number of the graph nodes. Moreover, when considering multi-head attention, the computation complexity would be $\boldsymbol{O}(TNRd)$, where T is the head number. This huge computation hinders the deployment of a GT on large graphs.

4 METHODOLOGY

In this section, we first introduce the proposed LiteGT. Then, we analyze the time and space complexity compared with the vanilla graph transformer.

4.1 LiteGT

Inspired the by the generalized kernelizable attention, we extend Equation (4) to a kernel function format, namely $a_{ij} = \mathrm{K}(\boldsymbol{q}_i, \boldsymbol{k}_j) = softmax_j(\frac{\boldsymbol{q}_i \boldsymbol{k}_j^T}{\sqrt{d}})$. And we name it regularized softmax-kernel. Therefore, the computation in the multi-head attention block of vanilla graph transformer can be written as

$$\begin{aligned} \boldsymbol{H}^{'} &= \mathrm{MultiHeadGraphAttention}(\boldsymbol{\mathcal{G}}, \boldsymbol{H}) \\ &= \|_{t=1}^{T} \sum_{j \in \{\boldsymbol{\mathcal{N}}_i\}} \mathrm{K}_{reg.\ softmax}(\boldsymbol{q}_i^t, \boldsymbol{k}_j^t)\, \boldsymbol{v}_j^t, \quad \forall node_i \in \boldsymbol{\mathcal{G}} \end{aligned} \tag{5}$$

where $\|$ denotes concatenation, $\boldsymbol{\mathcal{G}}$ is the input graph, and $\boldsymbol{H} \in \mathbb{R}^{N \times d}$ is the node feature matrix.

To reduce the computation complexity of vanilla graph transformer, we propose LiteGT, which saves computation and model size through three settings. First, instead of applying self-attention mechanism on all nodes, we only consider the top-$logN$ important nodes selected by our node sampling strategy. Second, we propose a two-branch attention paradigm wherein the first branch employs regularized softmax-kernel, while the second branch employs one of the two proposed lightweight computation kernels. Third, we reduce the layer hidden dimension at several block intervals. To compare with vanilla graph transformer, we write the lite attention block of LiteGT as

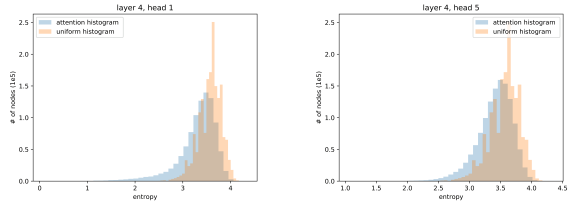
$$\begin{aligned} \boldsymbol{H}^{'} &= \mathrm{TwoBranchAttention}(\mathrm{NodeSampling}(\boldsymbol{\mathcal{G}}), \boldsymbol{H}) \\ &= \mathrm{Concat}(\|_{t=1}^{T/2} \sum_{j \in \{\boldsymbol{\mathcal{N}}_i\}} \mathrm{K}_{branch-1}(\boldsymbol{q}_i^t, \boldsymbol{k}_j^t)\, \boldsymbol{v}_j^t, \\ &\quad \|_{t=\frac{T}{2}+1}^{T} \sum_{j \in \{\boldsymbol{\mathcal{N}}_i\}} \mathrm{K}_{branch-2}(\boldsymbol{q}_i^t, \boldsymbol{k}_j^t)\, \boldsymbol{v}_j^t), \ \forall node_i \in \mathrm{top}\ logN \end{aligned} \tag{6}$$

where $\boldsymbol{H}^{'}$ is followed by a pooling operation for dimension reduction typically. $\mathrm{K}_{branch-1}(\cdot,\cdot)$ is the regularized softmax-kernel, and $\mathrm{K}_{branch-2}(\cdot,\cdot)$ is one of the two proposed lightweight computation kernels, namely Jaccard and Subtracter kernels.

Jaccard kernel measures the similarity of two nodes based on their common direct neighbors. It is purely based on the graph structure and does not relate to the node features.

$$a_{ij} = \mathrm{K}_{Jaccard}(node_i, node_j) = \frac{\{\boldsymbol{\mathcal{N}}_i\} \cap \{\boldsymbol{\mathcal{N}}_j\}}{\{\boldsymbol{\mathcal{N}}_i\} \cup \{\boldsymbol{\mathcal{N}}_j\}}. \tag{7}$$

Compared with regularized softmax-kernel, Equation (7) can be computed before model training. And when the message passing is needed from the neighbors, we can get their weight directly. As for Subtracter kernel, it computes the l_1-norm of the two input vectors,



(a) Example 1 (b) Example 2

Figure 3: The node attention entropy histograms from a 8-layer Graph Transformer trained on the CLUSTER dataset

namely,

$$a_{ij} = \mathrm{K}_{Subtracter}(\boldsymbol{q}_i, \boldsymbol{k}_j) = |\boldsymbol{q}_i - \boldsymbol{k}_j|_1. \tag{8}$$

Subtracter kernel only includes $abs(\cdot)$ and addition and is much faster than regularized softmax-kernel.

The proposed LiteGT is shown in Figure 2(a). In the following, we describe the details of the three key settings in LiteGT.

4.1.1 Instance-level: node sampling. Some previous works [10, 48] have revealed that the attention distribution learned from the vanilla self-attention mechanism has potential sparsity, which motivates the researcher to design some "selective" counting strategies on all attention computation without significantly affecting performance. The attention sparsity is observed in NLP tasks, then the question is if this is also true in graph learning tasks.

To motivate our approach, we run GT model on the CLUSTER dataset (see Section 5.1) to investigate the entropy of the attention distribution of each node. Specifically, for each $node_i$, $\{a_{ij}\}_{j\in\{\boldsymbol{N}_i\}}$ forms a discrete probability distribution over all its neighbors with the entropy given by $H(\{a_{ij}\}_{j\in\{\boldsymbol{N}_i\}}) = -\sum_{j\in\{\boldsymbol{N}_i\}} a_{ij}\log a_{ij}$. A low entropy means a high degree of concentration, and vice versa. An entropy of 0 means all attention is on one neighbor. Attending all neighbors equally has the highest entropy of $\log(|\boldsymbol{N}(i)|)$, where $|\boldsymbol{N}(i)|$ is the number of direct neighbor of $node_i$. The purpose of introducing attention mechanism to GNN models is that we expect the model to learn a distribution of lower entropy (i.e, some neighbors are much more important than the others). Since nodes can have number of neighbors, the maximum entropy of them will also be different. Therefore, we plot the aggregated histogram of entropy values of all nodes in an entire graph. We select the entropy histogram of {Head 1, Head 5} @Layer4 of the trained GT. We also provide the entropy histogram when all nodes have uniform attention weight distribution for comparison. The result is shown in Figure 3. We notice that the two entropy histograms show high coincidence, meaning most nodes attend to their neighbors uniformly and we only need to perform graph attention on a few of them. Based on this observation, we propose to do node sampling in LiteGT such that only the sampled nodes would compute attention with their direct neighbors. For the nodes that are not selected, they attend to their neighbors with the same weight, viz. $a_{ij} = \frac{1}{|\{\boldsymbol{N}_i\}|}$.

For the node sampling strategy, it is desired to select those nodes whose attention distribution (denoted as p) on their direct neighbors is away from the uniform distribution (denoted as q). This 'distance' could be measured by Kullback-Leibler divergence, and we write the distance measurement for $node_i$ as

$$D_i = KL(q||p) = \log \sum_{j\in\{\boldsymbol{N}_i\}} e^{\frac{\boldsymbol{q}_i\boldsymbol{k}_j^T}{\sqrt{d}}} - \frac{1}{|\{\boldsymbol{N}_i\}|}\sum_{j\in\{\boldsymbol{N}_i\}} \frac{\boldsymbol{q}_i\boldsymbol{k}_j^T}{\sqrt{d}} - \log|\{\boldsymbol{N}_i\}|, \tag{9}$$

Apparently, we should select the nodes with higher D_i. However, the computation for D_i includes the attention computation, which means we can not save any computation if we compute D_i for all nodes. Moreover, the Log-Sum-Exp (LSE) operation of the first term has the potential numerical stability issue. Motivated by this, we propose an approximation of the distance measurement as follows

$$\hat{D}_i = \max_{j\in\{\boldsymbol{N}_i\}} \mathrm{K}_{Jaccard}(node_i, node_j) - \frac{1}{|\{\boldsymbol{N}_i\}|}\sum_{j\in\{\boldsymbol{N}_i\}} \mathrm{K}_{Jaccard}(node_i, node_j) - \log|\{\boldsymbol{N}_i\}|. \tag{10}$$

Compared with Equation (9), there are two steps of approximation. First, the LSE operation in (9) is approximated with a max operation since the former is a smooth approximation to the latter [34]. Second, we use the Jaccard kernel value of $node_i$ and $node_j$ to replace the computation of $\frac{\boldsymbol{q}_i\boldsymbol{k}_j^T}{\sqrt{d}}$. The design of $\boldsymbol{q}_i\boldsymbol{k}_j^T$ in the vanilla transformer is to let $node_i$ attends more on $node_j$ if the query vector of $node_i$ is similar (based on inner product value) to the key vector of $node_j$. Jaccard kernel also measures the similarity between two nodes, but based on their common neighbors. Therefore, it is reasonable to make this approximation. We also validate the effectiveness of the proposed node sampling strategy in Section 7.2. Since Jaccard kernel depends only on the graph structure, we can derive the $\hat{D}_i$ for all nodes and therefore determine the sampled nodes before model training.

After deriving the $\hat{D}_i$ for all nodes, we can select the top-k nodes with highest values. Controlled by a constant sampling factor c, we set $k = c \cdot \log N$ which makes LiteGT only need to calculate $\boldsymbol{O}(R\log N)$ dot-product for each attention head instead of $\boldsymbol{O}(RN)$.

4.1.2 Relation-level: two-branch attention. As discussed before, the proposed two kernel functions, namely Jaccard and Subtracter kernels, are computationally much cheaper than regularized softmax-kernel. Moreover, considering different kernel functions will capture different attention patterns, and they might be both useful in graph learning (validated in Section 7.1), we therefore utilize multiple attention kernels in the proposed LiteGT. Specifically, we employ a two-branch setting. In the first branch, the regularized softmax-kernel is utilized, while the second branch employs Jaccard or Subtracter kernel. The results of the two branches are then concatenated together (cf. Figure 2(b)). The effectiveness of the two-branch setting is validated through ablation studies in Section 7.1. We observe that the models which use only one kind of those attention kernels achieve inferior performance compared with the two-branch setting (using two attention kernels). This verifies that different attention patterns learned by the two branches are complementary and both useful in graph learning tasks.

In our experiments, we apply node sampling on the first branch only, while on the second branch, all nodes would compute attention with their neighbors using the two proposed lightweight kernels. The reason that we do not perform node sampling on the

Procedure 1 The forward process of the lite attention block when node sampling strategy is employed in the first branch

Input: The input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; the hidden representation $\boldsymbol{h}_i$ of $node_i$.
Output: The output of lite attention block $\boldsymbol{h}_i^{'}$.

1: Compute the attention sparsity measurement $\hat{D}_i$ by Equation (10) for all nodes in $\mathcal{V}$.
2: Generate the top-k node list, whose nodes have the highest $\hat{D}_i$.
3: Compute $\boldsymbol{q}_i, \boldsymbol{k}_i, \boldsymbol{v}_i$ for $node_i$.
4: **if** $node_i$ is in top-k node list **then**
5: Compute the attention score $a_{ij}^{(1)}$ by Equation (4).
6: **else**
7: $a_{ij}^{(1)} = \frac{1}{|\{\mathcal{N}_i\}|}$.
8: **end if**
9: Compute the output of the first branch: $\boldsymbol{h}_i^{(1)} = \sum_{j \in \{\mathcal{N}_i\}} a_{ij}^{(1)} \boldsymbol{v}_j$.
10: Compute the output of the Second branch: $\boldsymbol{h}_i^{(2)} = \sum_{j \in \{\mathcal{N}_i\}} a_{ij}^{(2)} \boldsymbol{v}_j$, where $a_{ij}^{(2)}$ is computed by Equation (7) or (8).
11: Output $\boldsymbol{h}_i^{'} = \text{Concat}(\boldsymbol{h}_i^{(1)}, \boldsymbol{h}_i^{(2)})$

second branch is twofold. First, the computation on the second branch is much cheaper than the first branch as discussed. Therefore, we can not save much computation when doing node sampling on the second branch. Second, for some graph datasets which do not preserve the attention sparsity, doing node sampling on both branches would achieve a poor result. In Procedure 1, we summarize the entire forward process of the lite attention block when the node sampling strategy is employed in the first branch.

4.1.3 Vector-level: layer dimension reduction. To reduce the network computation further, we decrease the layer dimension by half after stacking several same-size network layers. This procedure will repeat several times and to the end of the network, the layer dimension is thereby decreased exponentially. The overall procedure is depicted in Figure 4.

4.2 Time and Space Complexity

Here we compare the attention complexity between vanilla graph transformer and the proposed LiteGT. Since addition operation is much faster than multiplication, we only consider the number of multiplications in attention computation. Considering the proposed LiteGT has three kinds of settings, we use the setting combinations to represent our models, as listed in Table 1.

According to Equation (4), the time complexity of attention computation for vanilla graph transformer is $\boldsymbol{O}(TNRd + 3TNd^2)$. The first term is to compute the attention between nodes and their neighbors, while the second term is for deriving the $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ matrices from the inputs. As for LiteGT, when only the two-branch setting is employed (namely *Jaccard* or *Subtracter* model), the time complexities are $\boldsymbol{O}(\frac{1}{2}TNRd + 2TNd^2)$ and $\boldsymbol{O}(\frac{1}{2}TNRd + 3TNd^2)$ for *Jaccard* and *Subtracter* respectively, if we neglect the addition operation. When node sampling is further applied on the first branch (namely *Jaccard/Subtracter + Sampling* models), the time complexities are

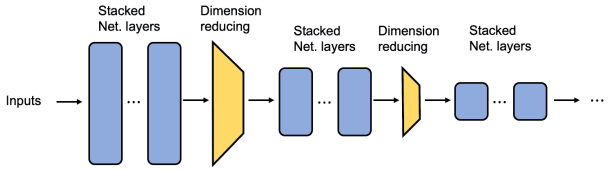


Figure 4: The layer dimension pooling scheme.

$\boldsymbol{O}(\frac{1}{2}TRd\ln N + TNd^2 + Td^2\ln N)$ and $\boldsymbol{O}(\frac{1}{2}TRd\ln N + 2TNd^2 + Td^2\ln N)$ respectively since only $\ln N$ nodes are sampled to compute attention with their neighbors in the first branch. Employing the proposed layer dimension reducing strategy would change the feature dimension d. Moreover, the feature dimension decreases exponentially as we are halving the layer dimension every time. According to the geometric series $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = \sum_{n=1}^{\infty} (\frac{1}{2})^n = 1$, the average feature dimension is $\frac{2d}{n+1}$, where n is the times of implementing the proposed layer dimension reduction strategy. Therefore, the time complexities are $\boldsymbol{O}(\frac{1}{n}TRd\ln N + \frac{4}{n^2}TNd^2 + \frac{4}{n^2}Td^2\ln N)$ and $\boldsymbol{O}(\frac{1}{n}TRd\ln N + \frac{8}{n^2}TNd^2 + \frac{4}{n^2}Td^2\ln N)$ $\boldsymbol{O}(\frac{1}{n}TRd\ln N)$ for our model *Jaccard/Subtracter + Sampling + Dim. reducing*, respectively.

As for space complexity, it is $\boldsymbol{O}(3TNd + 3Td^2)$ for the vanilla graph transformer, where $3Nd$ is the storage for $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ matrices, while $3Td^2$ is the storage of the weight matrices, which transform the input to $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$. As for the proposed LiteGT, *Subtracter* has the same space complexity as in vanilla graph transformer, while the case for *Jaccard* is $\boldsymbol{O}(2TNd + 2Td^2)$. When further considering node sampling, the space complexity is $\boldsymbol{O}(TNd + 2Td\ln N + 3Td^2)$ for *Subtracter + Node Sampling*, while $\boldsymbol{O}(TNd + Td\ln N + 2Td^2)$ for *Jaccard + Node Sampling*. The layer dimension reduction strategy only changes the feature dimension. According to the above analysis, the space complexities for *Jaccard/Subtracter + Sampling + Dim. reducing* are $\boldsymbol{O}(\frac{2}{n}TNd + \frac{2}{n}Td\ln N + \frac{8}{n^2}Td^2)$ and $\boldsymbol{O}(\frac{2}{n}TNd + \frac{4}{n}Td\ln N + \frac{12}{n^2}Td^2)$, respectively. We summarize the time and space complexity of the proposed LiteGT with different settings in Table 2.

5 EXPERIMENTAL SETTING

We first introduce the datasets for performance evaluation and then describe the detail of our model configuration. The baseline methods and evaluation metrics are also briefly discussed.

5.1 Datasets

We evaluate the proposed LiteGT on three medium-scale datasets, namely PATTERN, CLUSTER and TSP.

PATTERN and **CLUSTER** are generated with Stochastic Block Models [1] for node classification tasks, which are widely used to model social networks. There are total 2 and 6 node classes in PATTERN and CLUSTER datasets, respectively. Node feature is provided in PATTERN and CLUSTER to simulate user attributes.

TSP is a link prediction dataset based on the classical Travelling Salesman Problem, which is a binary classification task on edges to identify whether edges belong to the optimal TSP solution. Considering TSP is a NP-Hard combinatorial problem, recently many research works have attempted to utilize GNN for learning better solvers [3, 13, 25].

Table 1: The description of the model names of the proposed LiteGT with different settings

Model name	Description
Jaccard/Subtracter	{Setting 1: The second branch employs Jaccard/Subtracter kernel}
Jaccard/Subtracter + Sampling	{Setting 1} + {Setting 2: The first branch employs node sampling}
Jaccard/Subtracter + Sampling + Dim. reducing	{Setting 1} + {Setting 2} + {Setting 3: Reduce the layer dimension by a half after stacking a few layers}

Table 2: The time and space complexity of LiteGT with different settings.

Model name	Time complexity	Space complexity
Vanilla Graph Transformer	$\boldsymbol{O}(TNRd + 3TNd^2)$	$\boldsymbol{O}(3TNd + 3Td^2)$
Jaccard/Subtracter	$\boldsymbol{O}(\frac{1}{2}TNRd + 2TNd^2)$ / $\boldsymbol{O}(\frac{1}{2}TNRd + 3TNd^2)$	$\boldsymbol{O}(2TNd + 2Td^2)$ / $\boldsymbol{O}(3TNd + 3Td^2)$
Jaccard + Sampling	$\boldsymbol{O}(\frac{1}{2}TRd\ln N + TNd^2 + Td^2\ln N)$	$\boldsymbol{O}(TNd + Td\ln N + 2Td^2)$
Subtracter + Sampling	$\boldsymbol{O}(\frac{1}{2}TRd\ln N + 2TNd^2 + Td^2\ln N)$	$\boldsymbol{O}(TNd + 2Td\ln N + 3Td^2)$
Jaccard + Sampling + Dim. reducing	$\boldsymbol{O}(\frac{1}{n}TRd\ln N + \frac{4}{n^2}TNd^2 + \frac{4}{n^2}Td^2\ln N)$	$\boldsymbol{O}(\frac{2}{n}TNd + \frac{2}{n}Td\ln N + \frac{8}{n^2}Td^2)$
Subtracter + Sampling + Dim. reducing	$\boldsymbol{O}(\frac{1}{n}TRd\ln N + \frac{8}{n^2}TNd^2 + \frac{4}{n^2}Td^2\ln N)$	$\boldsymbol{O}(\frac{2}{n}TNd + \frac{4}{n}Td\ln N + \frac{12}{n^2}Td^2)$

Table 3: Dataset information.

Dataset	#Graphs.	Total #Nodes	Task	#Classes
PATTERN	$12k$	1, 664, 491	Node Classification	2
CLUSTER	$12k$	1, 406, 436	Node Classification	6
TSP	$12k$	3, 309, 140	Link Prediction	2

We summarize the information of those three datasets in Table 3. For more detail about those datasets, including data splitting, we refer the reader to [20].

5.2 Model Configurations

Overall, we follow the benchmarking protocol introduced in Dwivedi et al. (2020) based on PyTorch (Paszke et al. 2019) and DGL (Wang et al. 2019). Specifically, we use the Adam optimizer [26] with the same learning rate decay strategy for all models. The initial learning rate is either 0.001 or 0.0005, which is reduced by a factor of 0.5 if there is no improvement of the validation loss after a fixed number of epochs. The model training is stopped either when it reaches the maximum computational time of 24 hours or the learning rate is smaller than the smallest value of $1e^{-6}$. We set the LiteGT[1] layer number as 12 for CLUSTER and TSP datasets, and 8 for PATTERN dataset. In each LiteGT layer, there are 8 attention heads and 80 hidden dimensions if not otherwise specified. When we do node sampling on the first branch of the LiteGT model, we set the sampling factor to 5 for all experiments. For layer dimension reduction setting, we apply it on the last four layers (denoted as $L1$~$L4$, where $L4$ is the last layer) of the LiteGT model, and the layer dimension is reduced by half at the beginning of $L1$ and $L3$.

5.3 Baselines and Evaluation Metrics

The baseline methods are mainly separated into two groups. The first group is based on message-passing mechanism, and the node representation is updated locally (viz. only dependent on direct neighbors). Typical methods of the first group include vanilla GCN [27], GraphSage [23], MoNet [32], GatedGCN-PE [4], GAT [40] and GT [19]. Meanwhile, GCN and GraphSage treat the node neighbors equally when updating nodes representation, while the other four methods treat them differently. The second group is the recently proposed Weisfeiler-Lehman GNNs based on the WL test [28], and we consider GIN [43], 3WLGNN [31] and RingGNN [8]. For PATTERN and CLUSTER datasets, the model performance is evaluated by weighted accuracy with respect to the node class size. As for TSP, the F_1 score for the positive edges are employed. Both metrics are the higher the better.

Since the proposed LiteGT focuses on reducing the attention computation, we therefore also compare the attention computation time with GatedGCN-PE, GAT and GT considering they also employ attention. Specifically, we count the runtimes of the attention computation on each edge one by one and then sum them up. The reason that we do not calculate them parallelly is that we care more about the actual computer calculation reduction achieved by LiteGT, and parallel computation can not reflect the difference since it would cost a similar time when computing attention on one edge or 1000 edges. We count the attention computation time for each graph of the dataset and report the average time per graph in our experiments.

6 RESULTS

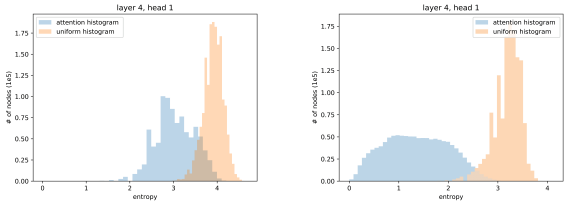
The results of our comparative evaluation experiments are summarized in Table 4. The results of the compared methods are from [20] and [19]. We highlight the best four results in each dataset. Specifically, we color the best performance in red, while the other three good results in blue.

For PATTERN and CLUSTER datasets, LiteGT achieves good results on most of the settings. Moreover, compared with GatedGCN-PE, LiteGT with the setting of *Jaccard + Sampling* and *Jaccard + Sampling + Dim. reducing* achieves more than 10× and 20× speedup w.r.t. attention computation time on PATTERN and CLUSTER datasets, respectively. The acceleration is even significant compared with GT and GAT, where LiteGT achieves more than 100× speedup. As for the classification accuracy, compared with GT, the proposed two-branch setting (no node sampling and layer dimension reducing) achieves around 1% improvement with similar or fewer model parameters. When we further employ the proposed node sampling strategy, the node classification accuracy shows 0.5~1% improvement on CLUSTER dataset. We notice that implementing

[1]The implementation of LiteGT can be found at: https://github.com/ChaofanTao/litegt

Table 4: The performance of different methods on PATTERN, CLUSTER and TSP datasets. The best performance is shown in red, while good performance is shown in blue.

Method	PATTERN			CLUSTER			TSP		
	Test Acc.(%)	Atten. Time per Graph	#Param.	Test Acc.(%)	Atten. Time per Graph	#Param.	Test F_1 score	Atten. Time per Graph	#Param.
GCN	71.892	–	500823	68.498	–	501687	0.630	–	95702
GraphSage	50.492	–	502842	63.844	–	503350	0.665	–	99263
MoNet	85.582	–	511487	66.407	–	511999	0.641	–	99007
GAT	78.271	416.77*s*	526990	70.587	342.64*s*	526990	0.671	192.86*s*	96182
GatedGCN-PE	86.508	55.66*s*	502457	76.082	46.59*s*	504253	0.838	32.96*s*	500770
GIN	85.387	–	508574	64.716	–	517570	0.656	–	99002
RingGNN	86.244	–	504766	22.340	–	524202	0.704	–	507938
3WLGNN	85.341	–	502872	55.489	–	507252	0.694	–	106366
GT	84.808	246.20	522982	73.169	204.85	524026	0.830	491.29*s*	932602
Jaccard	85.752	94.99*s*	368150	74.199	120.21*s*	550970	0.842	298.69*s*	760970
Jaccard + Sampling	84.977	**1.94s**	368150	75.344	2.78*s*	550970	0.758	**2.88s**	760970
Jaccard + Sampling + Dim. reducing	84.809	1.97*s*	**218392**	75.387	**2.13s**	**401156**	0.739	2.94*s*	**547214**
Subtracter	85.648	194.85*s*	419430	73.830	245.42*s*	627866	0.846	610.11*s*	1114378
Subtracter + Sampling	84.504	101.80*s*	419430	74.276	127.99*s*	627866	0.842	314.31*s*	1114378
Subtracter + Sampling + Dim. reducing	84.323	104.23*s*	248424	73.748	131.44*s*	456804	0.828	309.28*s*	807582



(a) PATTERN (b) TSP

Figure 5: The entropy histograms selected from a 8-layer GT trained on PATTERN and TSP datasets, respectively.

node sampling on PATTERN dataset leads to a slightly decrease, from 85.752% to 84.977% and 85.648% to 84.504% on *Jaccard* and *Subtracter* settings, respectively. In Section 4.1.1, we have checked that the entropy histograms of the attention version and the uniform one are highly coincident on the CLUSTER dataset, i.e., most nodes in CLUSTER do not need to compute attention with their neighbors. Therefore, when doing node sampling (only the sampled nodes would do attention with their neighbors) on CLUSTER, the classification accuracy does not decrease. However, this may not be true on PATTERN dataset. To validate this, in Figure 5(a), we plot the entropy histograms on PATTERN the same way we did on CLUSTER. We notice that the two histograms differ more than in the CLUSTER case. Therefore, performing node sampling leads to an accuracy drop as the attention sparsity on PATTERN dataset is not obvious. Apart from the aforementioned two settings, reducing the layer dimension on the last few model layers decreases the model parameters significantly, while at the same time achieving a similar classification accuracy on both datasets.

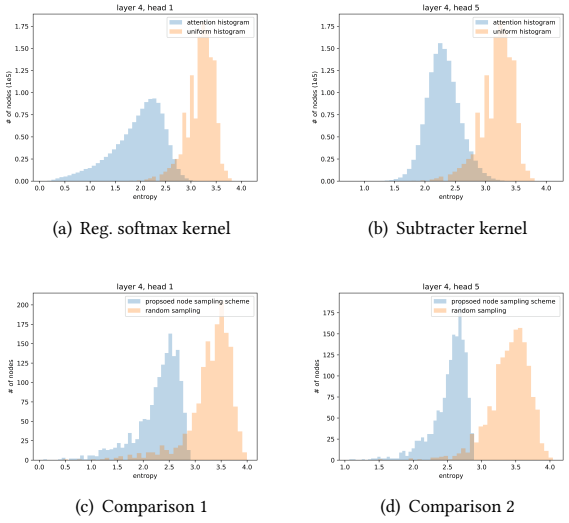
For the TSP dataset, the proposed two-branch setting (*Jaccard* and *Subtracter*) achieves the best classification performance among all methods. Moreover, *Jaccard* consumes only around a half of the attention computation in GT. When further implementing node sampling, the test F_1 scores decrease by a large margin in the *Jaccard* case. To investigate this phenomenon, we plot the entropy histograms on TSP dataset in Figure 5(b). We notice that most of the nodes have a low entropy, which means for those nodes, a few neighbors are much more important than the others, and attention is necessary to distinguish those neighbors. In this case, node sampling does not work well. For the *Subtracter* case, since the second branch still implements full attention (but with $l1$-norm instead of the conventional scaled dot-product), the test F_1 scores are still good when performing node sampling on its first branch. The setting of reducing the layer dimension still reduces the model size significantly and the performance is still acceptable for *Subtracter + Sampling + Dim. reduing*

To summarize, the two-branch setting (*Jaccard* and *Subtracter*) works well on all datasets. For the node sampling strategy, it reduces the attention computation and at the same time improves the model performance if the graph dataset truly preserves the attention sparsity (CLUSTER case). Reducing the layer dimension decreases the model size significantly with negligible performance degradation.

7 DISCUSSION

7.1 Understanding the Advantages of the Two-Branch Setting

To better understand the advantages of the two-branch setting, we plot the attention entropy histograms of the two branches in Figure 6 (a) & (b), where the two branches employ regularized softmax-kernel and Subtracter kernel, respectively. We note that Heads 1~4 are with the first branch, while Heads 5~8 are with the second branch. We observe that the attention learned by the two branches is quite different and they are all well-separated from the



(a) Reg. softmax kernel (b) Subtracter kernel

(c) Comparison 1 (d) Comparison 2

Figure 6: (a) and (b) show the selected entropy histograms of different branches from a 12-layer LiteGT (Subtracter) trained on TSP dataset. (c) and (d) demonstrate the entropy histogram comparison between the nodes sampled by our proposed scheme and the nodes sampled randomly. The results are derived from an 8-layer GT trained on the CLUSTER dataset.

uniform histogram. The first branch tends to learn node attention with a broader entropy range (0 ~3.0), while the second branch learns node attention entropy with a smaller extent (1.5 ~3.0). In Table 5, we also implement ablation study when all heads employ Jaccard or Subtracter kernel. We notice that the performance decreases on all datasets. Considering the performance of GT (all heads employing regularized softmax-kernel) is also inferior to our two-branch setting, we conclude that the attention computation methods on the two branches are complementary and are both useful for node representation learning.

7.2 The Effectiveness of the Node Sampling Strategy

We have already discussed that node sampling works well when only a few nodes in the graph have low attention entropy and most nodes attend equally to their neighbors. Then the question is whether the proposed node sampling strategy truly selects those nodes with low attention entropy. To validate this, we plot the attention entropy histogram of the nodes selected by our proposed node sampling strategy (Group A). To compare, we also prepare Group B, whose nodes are selected randomly. Groups A and B have the same number of nodes. We implement this experiment on CLUSTER, in which only a few nodes have low entropy. We randomly select two heads in Layer 4, and the results are plotted in Figure 6 (c) and (d). We notice that the nodes selected by our strategy have obviously lower entropy than the nodes selected randomly. This validates the effectiveness of the proposed node sampling strategy.

Table 5: The performance of LiteGT (without node sampling and layer dim. reduction) when the node message aggregation way on both branches is just Jaccard kernel or Subtracter kernel or simply average.

Message Aggre. Way	PATTERN	CLUSTER	TSP
Jaccard	79.614	74.403	0.739
Subtracter	74.277	74.277	0.734
Mean	54.362	74.336	0.798

Table 6: The node classification accuracy of GT and LiteGT with different network depths on CLUSTER dataset.

Method	8 layers	10 layers	12 layers	14 layers	16 layers
GT	71.394	73.169	60.455	23.219	26.205
LiteGT	74.283	74.853	75.720	75.915	75.137

7.3 LiteGT for Alleviating Over-smoothing

The over-smoothing phenomenon implies that the node features will converge to a fixed point as the graph network depth increases [30]. In LiteGT, there are several attention kernels to aggregate the node neighbors' message, namely, regularized softmax kernel, Jaccard kernel and subtracter kernel. Moreover, for the nodes that are not sampled in the first branch, their neighbor message aggregation way would be simply average. Therefore, LiteGT employs 4 neighbor message aggregation schemes. We argue that this helps alleviate over-smoothing in graph neural network training since the node updating scheme would not be a single way, which is the case for most GNNs. We validate this by comparing the performance of GT and LiteGT (with Jaccard and node sampling setting) on CLUSTER. GT and LiteGT are both based on graph transformer structure, but GT employs regularized softmax-kernel only for node message aggregation. In this experiment, we increase the network depth from 8 to 16. The result is listed in Table 6. We notice that the best test accuracy of GT appears when the layer number is 10, and when the layer number increases further, its test accuracy drops significantly (to 26.205% with 16 layers). As for the proposed LiteGT, the test accuracy remains around 75% as the network depth increases.

8 CONCLUSIONS

We propose an efficient transformer-based model named LiteGT for arbitrary graph learning. The efficiency of LiteGT stems from 3 levels systemically. At the instance-level, a node sampling strategy is proposed to select important nodes for attention with a theoretical guarantee. At the relation-level, two versions of the two-branch attention paradigm relieve the attention computation and learn diverse relations via generalized kernelization. At the vector-level, a pooling strategy is proposed to reduce the redundancy of hidden features. Experiments have shown the impressive efficacy of LiteGT, thereby shedding light on highly efficient transformer design.

ACKNOWLEDGMENTS

This research was partially supported by ACCESS – AI Chip Center for Emerging Smart Systems, Hong Kong SAR.

REFERENCES

[1] Emmanuel Abbe. 2017. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research* 18, 1 (2017), 6446–6531.

[2] Mikhail Belkin and Partha Niyogi. 2003. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation* 15, 6 (2003), 1373–1396.

[3] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. 2020. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research* (2020).

[4] Xavier Bresson and Thomas Laurent. 2017. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553* (2017).

[5] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems* (2020).

[6] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *European Conference on Computer Vision*. Springer, 213–229.

[7] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. *International Conference on Learning Representations* (2018).

[8] Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. 2019. On the equivalence between graph isomorphism testing and function approximation with gnns. *Neural Information Processing Systems* (2019).

[9] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 257–266.

[10] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509* (2019).

[11] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, David Belanger, Lucy Colwell, et al. 2020. Masked language modeling for proteins via linearly scalable long-context transformers. *arXiv preprint arXiv:2006.03555* (2020).

[12] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. *International Conference on Learning Representations* (2020).

[13] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems* (2017).

[14] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *Annual Meeting of the Association for Computational Linguistics* (2019).

[15] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems* (2016).

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *Annual Conference of the North American Chapter of the Association for Computational Linguistics* (2018).

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. *International Conference on Learning Representations* (2021).

[18] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. *Advances in Neural Information Processing Systems* (2015).

[19] Vijay Prakash Dwivedi and Xavier Bresson. 2021. A Generalization of Transformer Networks to Graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications* (2021).

[20] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2020. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982* (2020).

[21] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*. PMLR, 1263–1272.

[22] Rohit Girdhar, Joao Carreira, Carl Doersch, and Andrew Zisserman. 2019. Video action transformer network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 244–253.

[23] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems* (2017).

[24] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*. 2704–2710.

[25] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. 2019. An efficient graph convolutional network technique for the travelling salesman problem. *INFORMS Annual Meeting* (2019).

[26] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *International Conference on Learning Representations* (2015).

[27] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations* (2017).

[28] AA Leman and B Weisfeiler. 1968. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsiya* 2, 9 (1968), 12–16.

[29] Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. 2018. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing* 67, 1 (2018), 97–109.

[30] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[31] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. 2019. Provably powerful graph networks. *Advances in Neural Information Processing Systems* (2019).

[32] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5115–5124.

[33] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4602–4609.

[34] Frank Nielsen and Ke Sun. 2016. Guaranteed bounds on the Kullback-Leibler divergence of univariate mixtures using piecewise log-sum-exp inequalities. *arXiv preprint arXiv:1606.05850* (2016).

[35] Kenta Oono and Taiji Suzuki. 2020. On asymptotic behaviors of graph cnns from dynamical systems perspective. *International Conference on Learning Representations* (2020).

[36] Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie Tang. 2020. Blockwise self-attention for long document understanding. *Findings of the Association for Computational Linguistics: EMNLP* (2020).

[37] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. (2018).

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

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[40] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *International Conference on Learning Representations* (2017).

[41] Huiyu Wang, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. 2020. MaX-DeepLab: End-to-End Panoptic Segmentation with Mask Transformers. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020).

[42] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768* (2020).

[43] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks? *International Conference on Learning Representations* (2019).

[44] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[45] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *Advances in Neural Information Processing Systems* (2019).

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

[47] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. 2020. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 1234–1241.

[48] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. *The Thirty-Fifth AAAI Conference on Artificial Intelligence* (2021).