

Exploring Sparsity in Graph Transformers

Chuang Liu,¹ Yibing Zhan,² Xueqi Ma,³ Liang Ding,² Dapeng Tao,⁴
Jia Wu,⁵ Wenbin Hu,^{1*} Bo Du,¹

¹ School of Computer Science, Wuhan University, Wuhan, China

² JD Explore Academy, JD.com, China

³ School of Computing and Information Systems, The University of Melbourne, Melbourne, Australia

⁴ School of Computer Science, Yunnan University, Kunming, China

⁵ School of Computing, Macquarie University, Sydney, Australia

{chuangliu, hwb, dubo}@whu.edu.cn, zhanyibing@jd.com, xueqim@student.unimelb.edu.au, liangding.liam@gmail.com, dptao@ynu.edu.cn, jia.wu@mq.edu.au

Abstract

Graph Transformers (GTs) have achieved impressive results on various graph-related tasks. However, the huge computational cost of GTs hinders their deployment and application, especially in resource-constrained environments. Therefore, in this paper, we explore the feasibility of sparsifying GTs, a significant yet under-explored topic. We first discuss the redundancy of GTs based on the characteristics of existing GT models, and then propose a comprehensive Graph Transformer SParsification (GTSP) framework that helps to reduce the computational complexity of GTs from four dimensions: the input graph data, attention heads, model layers, and model weights. Specifically, GTSP designs differentiable masks for each individual compressible component, enabling effective end-to-end pruning. We examine our GTSP through extensive experiments on prominent GTs, including GraphTrans, Graphormer, and GraphGPS. The experimental results substantiate that GTSP effectively cuts computational costs, accompanied by only marginal decreases in accuracy or, in some cases, even improvements. For instance, GTSP yields a reduction of 30% in Floating Point Operations while contributing to a 1.8% increase in Area Under the Curve accuracy on OGBG-HIV dataset. Furthermore, we provide several insights on the characteristics of attention heads and the behavior of attention mechanisms, all of which have immense potential to inspire future research endeavors in this domain. Code is available at <https://anonymous.4open.science/tr/gtsp>.

1 Introduction

Recently, Graph Transformer (GT) (Dwivedi and Bresson 2021) and its variants (Wu et al. 2021; Ying et al. 2021; Rampásek et al. 2022) have achieved performance comparable or superior to state-of-the-art Graph Neural Networks (GNNs) on a series of graph-related tasks, particularly on graph-level tasks such as graph classification. Nevertheless, GTs are more resource-intensive than GNNs due to their stack of multi-head self-attention modules (MHA), which suffer from quadratic complexity that renders their deployment impractical under resource-limited scenarios. Therefore, reducing the computational costs of GTs while maintaining their performance has become highly significant.

Graph model compression has experienced a surge of interest, with pruning emerging as a prominent technique (Liu

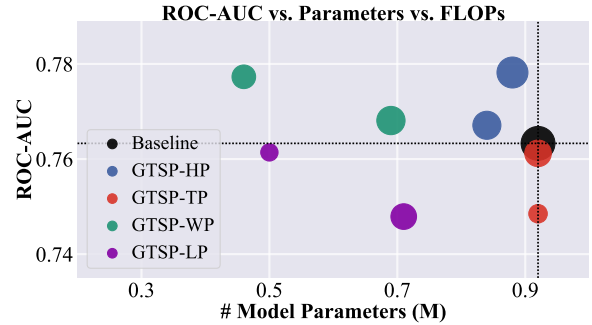


Figure 1: Performance (y-axis) analysis of GraphTrans baseline and our GTSP with varying model sizes (x-axis) and inference FLOPs (the size of markers) on OGBG-HIV. HP, TP, WP, and LP correspond to head pruning, token pruning, weight pruning, and layer pruning, respectively. Notably, GTSP achieves comparable or sometimes even slightly better performance than the baseline model with far fewer parameters and FLOPs.

et al. 2022). Currently, several works (Chen et al. 2021b; You et al. 2022; Hui et al. 2023; Liu et al. 2023) have been proposed to prune the GNN models from the perspectives of the model weights, the graph adjacency matrix, and the feature channels. These pruning works can accelerate GNNs' training and inference for node classification tasks on large-scale graphs. However, it remains unclear whether these methods are still effective for GTs due to the following differences: **1)** GNN pruning methods mainly target node classification tasks and prune edges from graphs. In contrast, GT mainly focuses on graph classification tasks, meaning that the efficiency is influenced by the number of input nodes to a greater extent. **2)** The architectural designs of GNNs and GTs differ considerably. Therefore, new pruning methods specifically designed for GTs are required. However, no study has yet been conducted to explore sparsification techniques explicitly tailored for GTs.

Therefore, in this paper, we aim to investigate the feasibility of sparsification techniques for GTs. To this end, we propose the Graph Transformer SParsification framework (GTSP), which is the first framework designed to re-

*Corresponding Author

duce the computational resources of GTs. Our exploration begins by examining the redundancy present in GTs. In this context, redundancy refers to extraneous information that can be eliminated without significantly impacting the performance—a phenomenon commonly observed in transformer models (Dalvi et al. 2020; Bian et al. 2021). We conduct a comprehensive analysis of GT redundancy across various dimensions, such as input nodes, attention heads, model layers, and model weights. Such an analysis provides valuable insights into the nature of GTs and guides our approach to effectively compress them. Subsequently, we propose a range of sparsification methods within the GTSP framework to discard the aforementioned redundancy. Specifically, GTSP incorporates a learnable token selector to dynamically prune input nodes, customizes an importance score to guide the attention head pruning process, employs skip connectivity patterns across different GT layers, and dynamically extracts sparse sub-networks. Benefiting from the above design features, GTSP effectively reduces the model redundancy, leading to reduced computational resource requirements without compromising on performance. Please note that this paper focuses on exploring the feasibility of pruning four compressible components in GTs, and analyzing the advantages and insights of pruning each individual component. A joint pruning of all compressible components in GTs in consideration of the complicated interactions between them is an area worth exploring in the future.

To evaluate the effectiveness of our GTSP, we conduct extensive experiments on various commonly-used datasets, including the large-scale Open Graph Benchmark (Hu et al. 2020), with three prominent GTs: GraphTrans (Wu et al. 2021), Graphormer (Ying et al. 2021), and GraphGPS (Rampášek et al. 2022). The experimental results demonstrate that our GTSP reliably improves the efficiency of GT models while maintaining their performance. For example, in Figure 1, our GTSP-WP (●) slightly outperforms the baseline full models (GraphTrans) with only 50% of the parameters and 69.8% of FLOPs.

Insights: Our thorough analysis of GTSP has yielded several noteworthy insights: **1)** Removing a large percentage of the attention heads does not significantly affect performance. The attention heads serve distinct roles in capturing various types of information, such as long-range and neighbor information within the graph. **2)** In general, up to 50% of the model’s neurons are redundant and can be pruned, which prevents over-fitting during training and can potentially improve accuracy. **3)** Redundancy is evident among adjacent layers within the network, with deeper layers displaying even greater redundancy in relation to their neighboring layers. Selectively trimming certain layers not only accelerates training but also alleviates the over-smoothing issue on the graph. **4)** Finally, we observe that GTs tend to prioritize important nodes while disregarding redundant nodes that can be safely removed from the graph.

Contributions: Our main contributions can be summarized as follows: **1)** We present a premier investigation into the redundancy of GT models, which enhances the understanding of these models and provides valuable guidance on the design of sparsification methods. **2)** For the first time,

we propose a comprehensive framework for sparsifying GT models, known as GTSP. This framework aims to improve the efficiency of GT models by sparsifying their components across four dimensions: input nodes, attention heads, model layers, and model weights. **3)** Experimental results on large-scale datasets with three popular GTs consistently validate the effectiveness and versatility of GTSP in offering enormous computation savings without compromising on accuracy. Additionally, we provide several valuable insights into the characteristics of existing GT models, which have the potential to inspire further research in this field.

2 Background

Notations A graph \mathcal{G} can be represented by an adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$ and a node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, where n is the number of nodes, d is the dimension of node features, and $\mathbf{A}[i, j] = 1$ if there exists an edge between node v_i and node v_j (otherwise, $\mathbf{A}[i, j] = 0$).

Transformer The vanilla Transformer (Vaswani et al. 2017) contains two key components: a multi-head self-attention (MHA) module and a position-wise feed-forward network (FFN). Given the input matrix of node embeddings $\mathbf{H} \in \mathbb{R}^{n \times d}$, where d represents the hidden dimension, a MHA module at layer l is computed as follows:

$$\text{MHA}(\mathbf{H}^{(l)}) = \text{Att}^{(l)} \left(\mathbf{W}_Q^{(l)}, \mathbf{W}_K^{(l)}, \mathbf{W}_V^{(l)}, \mathbf{H}^{(l)} \right), \quad (1)$$

where $\mathbf{W}_Q^{(l)}, \mathbf{W}_K^{(l)}, \mathbf{W}_V^{(l)} \in \mathbb{R}^{d \times d'}$ denote the query, key, and value projection matrices, respectively. $\text{Att}^{(l)}$ denotes the self-attention function and d' denotes the output dimension. Note that Eq. (1) denotes the single-head self-attention module, which can straightforwardly generalize to MHA. To construct a deeper model, each MHA layer and FFN layer is accompanied by a residual connection and subsequently normalized by means of layer normalization (LN).

Graph Transformer Many transformer variants, inspired by transformer models, have been successfully applied to graph modeling. These variants often outperform or match GNNs across various tasks. Unlike images and texts, graphs possess inherent structural characteristics; hence, the graph structure is crucial in graph-related tasks. Consequently, the most straightforward way to incorporate graph structure information is to combine GNNs with the transformer architecture (Rong et al. 2020; Wu et al. 2021; Rampášek et al. 2022). This integration can be represented as follows:

$$\mathbf{H}_G^{(l+1)} = \text{GNN}^{(l)} \left(\mathbf{H}^{(l)}, \mathbf{A} \right), \mathbf{H}^{(l+1)} = \text{MHA}^{(l)} \left(\mathbf{H}_G^{(l+1)} \right),$$

where $\text{GNN}^{(l)}$ denotes a GNN layer. Additionally, several existing works have attempted to compress the graph structure into positional embedding (PE) vectors that are then incorporated into the input features (Dwivedi and Bresson 2021; Kreuzer et al. 2021; Hussain, Zaki, and Subramanian 2022). Alternatively, the graph structure can be injected into the attention computation through bias terms (Ying et al. 2021). For further information on these topics, refer to recent reviews of GTs (Min et al. 2022). However, most of these

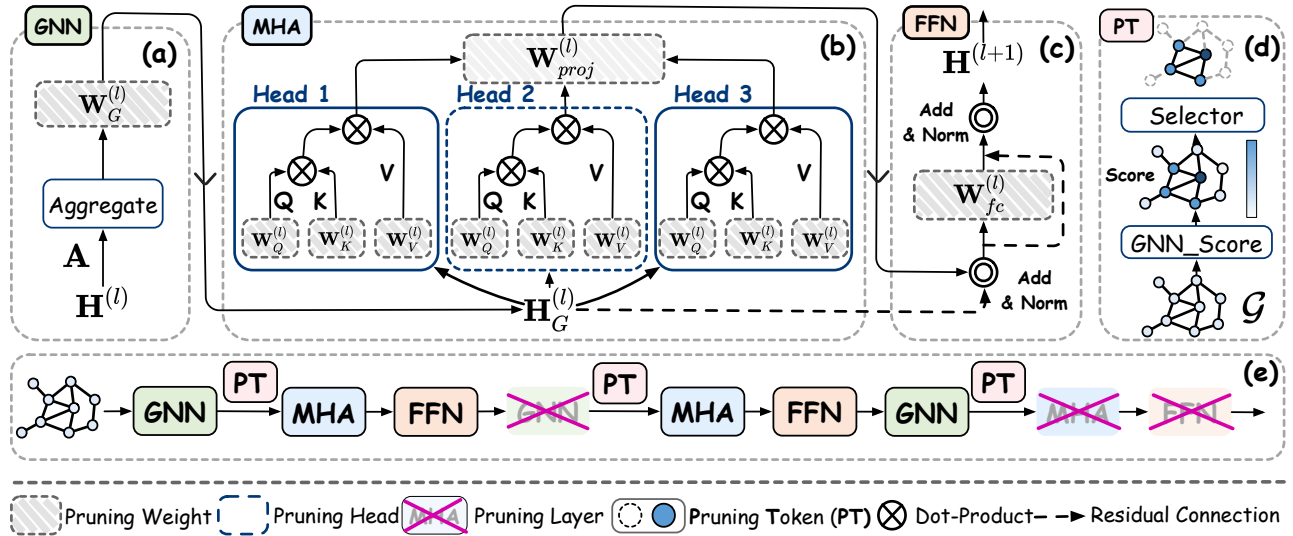


Figure 2: Overview of our proposed graph transformer sparsification framework (GTSP). Note that each compressible component is pruned separately in this paper.

methods encounter challenges due to the time and memory constraints imposed by their complex frameworks, particularly the high computational complexity of MHA (Zhang et al. 2022; Chen et al. 2023). It is, therefore, crucial to identify approaches for reducing the computational resources required by GTs while preserving their performance.

Model Pruning Pruning is a promising method for reducing the memory footprint and computational cost by removing unimportant elements based on pre-defined scores (Hoeffler et al. 2021). Various pruning methods, such as the magnitude-based, first-order and second-order based, and lottery ticket hypothesis-based methods (Liu et al. 2021; Mocanu et al. 2018; Frankle and Carbin 2019), have been used to remove redundant weights. Additionally, some studies have explored the pruning of input tokens (Kim et al. 2022; Liang et al. 2022), attention heads (Voita et al. 2019; Michel, Levy, and Neubig 2019), and even entire layers within the model architecture (Fan, Grave, and Joulin 2020; Yu et al. 2022). In the field of graphs, attempts have been made to co-prune model weights, graph adjacency matrices, and feature channels in order to speed up the training and inference of GNNs on large-scale graphs (You et al. 2022; Chen et al. 2021b; Hui et al. 2023; Liu et al. 2023). However, to the best of our knowledge, no investigation has yet been conducted into the pruning of GTs.

3 Methodology

This section explores redundancy within GT architectures and presents a comprehensive framework, called GTSP, which aims to enhance the efficiency of GTs through pruning. Specifically, GTSP is a mask-based pruning method, which contains three crucial steps in the overall pruning procedure: ①initializing masks for compressible components; ②determining the values of these masks; and ③pruning based on the masks. Figure 2 illustrates the application of

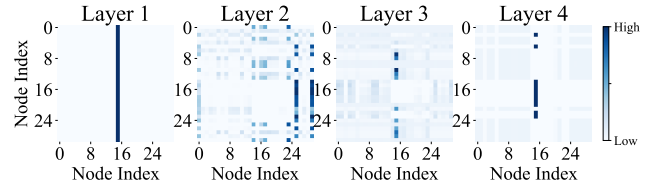


Figure 3: **Token Redundancy.** Attention probability between query (y -axis) and key (x -axis) vectors obtained from GraphGPS. The probabilities in each row all sum to 1.

GTSP in compressing GTs across four dimensions: input data (§3.1), attention heads (§3.2), model layers (§3.3), and model weights (§3.4). Please note that this paper primarily focuses on the individual pruning of each component; the joint pruning of all components, considering their intricate interactions, is left as a topic for future research.

3.1 Pruning Tokens

Self-attention is capable of modeling long-term dependencies. However, the computational complexity of computing the attention matrix increases quadratically with the length of the input tokens ($\mathcal{O}(n^2)$). Consequently, when working with large graphs and limited resources, the attention operation becomes a bottleneck. To achieve efficiency improvements, we aim to eliminate less important or relevant tokens before they pass through the transformer layers. By reducing the number of tokens n for subsequent blocks, we can reduce the complexity of both the MHA and FFN layers.

Analyzing Token Redundancy Figure 3 illustrates the attention probability, which measures how much all other tokens attend to a specific token. It is worth noting that only a limited number of tokens (*e.g.*, node 15) receive high atten-

tion scores, while others are considered less important and may be pruned. Furthermore, we offer supplementary validation in §4.4 to support our findings.

Discarding Token Redundancy In general, we use a trainable mask to discard tokens with low importance scores for token sparsification in an end-to-end optimization. To accomplish this, ❶ we first initialize a binary decision mask $\mathbf{M}_T^{(l)} \in \{0, 1\}^n$ that indicates whether a token should be dropped or kept. We then design a learnable prediction module that generates important scores for input nodes, ❷ which helps determine the values in the mask $\mathbf{M}_T^{(l)}$. Specifically, we project the tokens using a GCN (Kipf and Welling 2017) to capture both their feature and structure information:

$$\mathbf{S}_T^{(l)} = \text{GCN}(\mathbf{A}, \mathbf{H}^{(l)}) \in \mathbb{R}^{n \times c}, \quad (2)$$

where c is the dimension of the scores. To preserve the significant tokens and remove the useless ones, we first rank tokens in order of their scores and then prune them using a top- k selection strategy. If the score $\mathbf{S}_T^{(l,i)}$ of token i is smaller than the k largest values among all tokens, $\mathbf{M}_T^{(l,i)} = 0$, which indicates that the token i is pruned at layer l .

However, performing the above operation is not straightforward in practice because using the top- k operation to generate a mask is not differentiable, which hinders end-to-end training. To address this issue, we introduce the Gumbel-Softmax and straight-through tricks, which facilitate gradient back-propagation through the top- k selection. Another obstacle arises when using GCN to generate scores. GNNs tend to share similar information among directly connected nodes. As a result, models may assign similar scores to nearby nodes with similar keys; this causes models to get stuck in significant local structures and select redundant nodes, while ignoring important nodes from other substructures and losing structure information. To address this issue, we propose applying perturbations to the significance scores. Therefore, the decision mask is calculated as follows:

$$\mathbf{M}_T^{(l)} = \text{Gumbel-Softmax}(\mathbf{I}_{\lceil p_s \times n \rceil} \mathbf{S}_T^{(l)}) \in \{0, 1\}^n, \quad (3)$$

where $\mathbf{I}_{\lceil p_s \times n \rceil} \in \mathbb{R}^{n \times n}$ is a matrix generated by randomly dropping $\lceil p_s \times n \rceil$ non-zero elements of a unit matrix with n dimensions, $\lceil \cdot \rceil$ is the rounding up operation, and p_s is the score dropping rate. Finally, ❸ we prune tokens by applying the mask $\mathbf{M}_T^{(l)}$ to the node embedding matrix $\mathbf{H}^{(l)}$, and the mask is updated at each epoch.

3.2 Pruning Attention Heads

The attention mechanism employed by current GTs incorporates multiple attention heads, commonly utilizing either four or eight self-attention heads. However, there is a dearth of analysis and discussion on the necessity of using such a substantial number of heads, as well as the specific information focused on by each head during the process of representation learning and its relevance to downstream tasks.

Analyzing Head Redundancy To determine if the attention mechanism involves redundant computations, we calculate the similarity of head distributions. We define the attention redundancy matrix as the pairwise distance matrix of

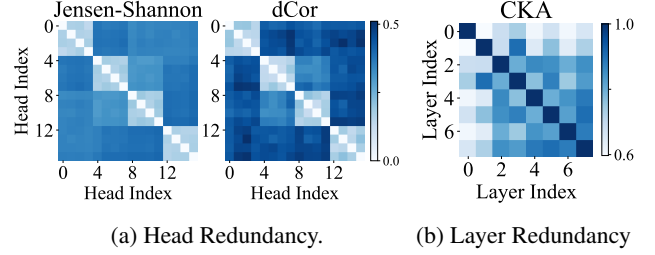


Figure 4: **(a) Head Redundancy.** Distance between different attention heads in GraphTrans (4-layer-4-head self-attention) is assessed using both the Jensen–Shannon and dCor metrics. Smaller distance values reflect more redundancy. **(b) Layer Redundancy.** Pairwise similarity between the layers in GraphGPS is measured using the CKA metric. Darker colors represent higher levels of similarity.

the 4×4 attention matrices from the GT model, using both node-level (Jensen–Shannon distance (Clark et al. 2019)) and graph-level (Distance Correlation, or dCor (Székely, Rizzo, and Bakirov 2007)) measures. Taking the GraphTrans model as an example, Figure 4a illustrates the redundancy (distance) among 16×16 pairs of attention matrices in a 4-layer-4-head configuration. It is evident that redundancy exists in the attention layers, particularly in the earlier ones. Additionally, the redundancy patterns remain consistent between two different distance measures used in our analysis. Based on these findings, we can conclude that certain attention heads can be safely removed.

Discarding Head Redundancy In the initial step, ❶ we proceed with the initialization of a decision mask, denoted as $\mathbf{M}_H^{(l)} \in \{0, 1\}^{N_h}$, where N_h represents the number of attention heads. Subsequently, ❷ we introduce a criterion for use in estimating the significance of each head. More precisely, the significance of a given head is determined based on the model’s sensitivity to the masked head (Michel, Levy, and Neubig 2019; Chen et al. 2021a). To calculate this significance value for each head, we use the following expression after applying the chain rule:

$$\mathbf{S}_H^{(l,h)} = \left| (\mathbf{Z}^{(l,h)})^T \cdot \frac{\partial \mathcal{L}(\mathbf{H}^{(l)})}{\partial \mathbf{Z}^{(l,h)}} \right|, \quad (4)$$

where $\mathbf{Z}^{(l,h)}$ denotes the embeddings of the l -th layer and h -th head computed by Eq. (1). Additionally, $\mathcal{L}(\cdot)$ refers to the cross-entropy loss used in GTs. Once we have obtained importance scores for attention heads, we remove those with the smallest $\mathbf{S}_H^{(l,h)}$. ❸ To prune these heads, we modify the formula for MHA in Eq. (1):

$$\text{MHA}(\mathbf{H}^{(l)}) = \sum_{h=1}^{N_h} \mathbf{M}_H^{(l,h)} \text{Att}^{(l)} \left(\mathbf{W}_Q^{(l,h)}, \mathbf{W}_K^{(l)}, \mathbf{W}_V^{(l,h)}, \mathbf{H}^{(l)} \right).$$

In addition, to prevent the premature removal of valuable attention heads, we propose a mechanism inspired by the concept of regrowth in weight pruning (Liu

et al. 2023). This mechanism utilizes magnitude gradients, $\|\partial\mathcal{L}(\mathbf{H}^{(l)})/\partial\mathbf{Z}^{(l,h)}\|_{\ell_1}$, as a criterion for determining which (if any) pruned heads should be regenerated. The attention heads with the highest gradients will be reactivated.

3.3 Pruning Layers

GTs often combine GNN layers with transformer layers. For example, in GraphTrans, the initial layers are GNN layers followed by multiple transformer layers. In GraphGPS, there is one GNN layer followed by one transformer layer. Although different studies have proposed various combinations, none has investigated whether it is necessary to stack multiple layers or if redundancy exists between them.

Analyzing Layer Redundancy To evaluate layer redundancy, we compare representations from different layers using linear Center Kernel Alignment (CKA) (Kornblith et al. 2019), which is a widely used method for identifying relationships between layers across architectures. In our analysis, we compute pairwise similarity between representations obtained from all eight intermediate layers (four GNN layers and four transformer layers) of GraphGPS and present the corresponding heatmaps in Figure 4b. From the results, we can make several observations: **1)** Adjacent layers show notable similarity, indicating some redundancy among them. **2)** Deeper layers exhibit greater similarity with each other, suggesting increased redundancy at deeper levels. This implies that these deeper layers may contribute minimally to the discriminative power of the model. These observations justify the motivation to drop certain layers. Furthermore, GT models have uniform internal structures with identical input/output sizes for MHA, FFN, and GNN models. This uniformity allows us to remove any of these components while directly concatenating the remaining ones without causing any feature dimension compatibility issues.

Discarding Layer Redundancy Building upon the previous study (Huang et al. 2016), we simultaneously prune redundant GNN and transformer layers in GTs using a random drop-layer approach. For instance, let us consider pruning a MHA layer. The original formulation of the residual connection in GTs is as follows:

$$\mathbf{H}^{(l+1)} = \text{LN}\left(\text{MHA}\left(\mathbf{H}^{(l)}\right)\right) + \mathbf{H}^{(l)}. \quad (5)$$

To determine whether to keep or drop a layer, **1)** we introduce a mask $\mathbf{M}_L^{(l)} \in \{0, 1\}^L$, where L is the number of layers, into the above equation. Consequently, Eq. (5) is reformulated as follows:

$$\mathbf{H}^{(l+1)} = \text{LN}\left(\mathbf{M}_L^{(l)} \text{MHA}\left(\mathbf{H}^{(l)}\right)\right) + \mathbf{H}^{(l)}, \quad (6)$$

2) where the value of masks is sampled from a Bernoulli distribution. Specifically, if $\mathbf{M}_L^{(l)} = 0$, **3)** it means that the layer is pruned and the node’s $(l + 1)$ -th layer representation remains consistent with its l -th layer representation. On the other hand, if $\mathbf{M}_L^{(l)} = 1$, then the node’s $(l + 1)$ -th layer representation is updated.

3.4 Pruning Weights

Over-parameterization is a common issue in neural networks that causes further information redundancy. To tackle this issue in GTs, we draw inspiration from sparse training techniques that dynamically extract and train sparse sub-networks instead of the entire model (Hoefler et al. 2021).

Discarding Weight Redundancy Generally speaking, we mask small-magnitude model weights in our approach. Specifically, during the model initialization stage, **1)** we create non-differentiable binary masks \mathbf{M}_W that match the size of the model weights in different layers, \mathbf{W} . Initially, all elements in the mask are set to 1. At regular intervals, **2)** our pruning strategy updates the mask matrix by setting parameters below a threshold to 0. **3)** The weight matrix is then multiplied by this updated mask to determine which weights participate in the subsequent forward execution of the graph. This procedure can be described as follows:

$$\begin{aligned} \text{idx} &= \text{TopK}(-|\mathbf{W}|, \lceil p_w \|\mathbf{W}\|_0 \rceil); \\ \mathbf{M}'_W &= \text{Zero}(\mathbf{M}_W, \text{idx}); \mathbf{W}' = \mathbf{M}'_W \odot \mathbf{W}, \end{aligned} \quad (7)$$

where TopK is the function that returns the indices of the top $\lceil p_w \|\mathbf{W}\|_0 \rceil$ values in $|\mathbf{W}|$, Zero is the function that sets the values in \mathbf{M}_W with indices idx to 0, \mathbf{W}' is the pruned weight matrix, p_w is the sparsity of the model weights, $\|\mathbf{W}\|_0$ is the number of model weights, and \odot is the element-wise product. We utilize gradual magnitude pruning, as demonstrated in previous studies (Zhu and Gupta 2017; Liu et al. 2021) to gradually prune the weights over m iterations until the desired sparsity is reached. If we perform sparsification of all elements every Δt steps, we can determine the pruning rate at each iteration t as follows:

$$p_{(w,t)} = p_f + (p_i - p_f) \left(1 - \frac{t - t_0}{m\Delta t}\right)^3, \quad (8)$$

where p_i/p_f is the initial/target sparsity degree, and t_0 is the epoch at which gradual pruning begins. The pruning scheme described above involves initially pruning a large number of redundant connections, followed by gradually reducing the number of connections pruned as fewer remain.

In addition, premature pruning may occur during the pruning process, particularly in early iterations, resulting in the loss of important information. To address this issue, we propose incorporating the gradient-based regrowth schemes (Evci et al. 2020) into the gradual pruning schedule. The regrowth operation, which is performed at regular intervals (Δt) throughout training, serves to reactivate weights with high-magnitude gradients. It is worth noting that previous studies have demonstrated the efficiency of this regrowth scheme for training (Liu et al. 2021, 2023).

3.5 Complexity Analysis

The computational costs of the GNN, MHA, and FFN modules in GTs are $\mathcal{O}(\|\mathbf{A}\|_0 d + nd^2)$, $\mathcal{O}(n^2 d)$, and $\mathcal{O}(nd^2)$, respectively. Our GTSP aims to reduce the computational complexity of these modules. The additional computation introduced by GTSP primarily arises from the element-wise product between weights and masks, which is acceptable. For a detailed discussion on the complexity and FLOPs, please refer to the Appendix.

Table 1: Performance measured by the number of **Parameters / FLOPs Saving / Accuracy** (\uparrow) or **ROC-AUC** (\uparrow). The results with higher accuracy than the baseline have been highlighted in **bold**. “GraphTrans-S” refers to the reduced-scale GraphTrans model, featuring a decreased number of layers. “GNN-Dense” denotes the models solely composed of GNN layers.

Models	Spar.	NCI1			OGBG-HIV			OGBG-Molpcba			
		# Para.	FS	Acc. (%)	# Para.	FS	ROC-AUC	# Para.	FS	ROC-AUC(V)	ROC-AUC(T)
GraphTrans	0 %	0.82M	0%	83.11 \pm 1.78	0.92M	0%	0.7633 \pm 0.0111	2.41M	0%	0.2893 \pm 0.0050	0.2756 \pm 0.0039
• GTSP-HP	25%	0.75M	6.1%	82.62 \pm 1.41	0.88M	8.52%	0.7782 \pm 0.0064	2.34M	4.4%	0.2871 \pm 0.0061	0.2756 \pm 0.0075
• GTSP-TP	25%	0.82M	22.8%	82.59 \pm 1.47	0.92M	21.5%	0.7612 \pm 0.0116	2.41M	20.7%	0.2604 \pm 0.0068	0.2451 \pm 0.0060
• GTSP-WP	25%	0.61M	18.7%	82.24 \pm 1.60	0.69M	16.5%	0.7681 \pm 0.0225	1.80M	20.3%	0.2893 \pm 0.0012	0.2794 \pm 0.0025
• GTSP-LP	25%	0.64M	23.5%	82.53 \pm 1.71	0.71M	24.3%	0.7479 \pm 0.0360	1.90M	22.1%	0.2863 \pm 0.0019	0.2749 \pm 0.0015
• GTSP-HP	50%	0.69M	12.2%	82.23 \pm 1.67	0.84M	17.1%	0.7671 \pm 0.0158	2.27M	8.9%	0.2712 \pm 0.0081	0.2615 \pm 0.0064
• GTSP-TP	50%	0.82M	47.4%	82.77 \pm 1.79	0.92M	45.3%	0.7485 \pm 0.0103	2.41M	39.2%	0.2573 \pm 0.0073	0.2436 \pm 0.0074
• GTSP-WP	50%	0.41M	37.4%	81.91 \pm 2.06	0.46M	30.2%	0.7773 \pm 0.0073	1.20M	40.6%	0.2868 \pm 0.0010	0.2774 \pm 0.0033
• GTSP-LP	50%	0.44M	47.1%	82.46 \pm 1.93	0.50M	48.8%	0.7614 \pm 0.0176	1.41M	46.6%	0.2748 \pm 0.0013	0.2664 \pm 0.0034
GraphTrans-S	0%	0.56M	47.1%	81.97 \pm 1.98	0.51M	48.8%	0.7617 \pm 0.0176	2.00M	46.6%	0.2830 \pm 0.0034	0.2752 \pm 0.0043
GNN-Dense	0%	0.58M	88.3%	80.00 \pm 1.40	0.18M	95.4%	0.7575 \pm 0.0104	1.60M	43.1%	0.2305 \pm 0.0027	0.2266 \pm 0.0028

4 Experiments

In this section, we validate the effectiveness of our proposed GTSP on three benchmark graph datasets, including OGB datasets. We demonstrate that GTSP is efficient and accurate (§4.2) and can generalize well to various baseline GT models (§4.3). Furthermore, we present insightful findings from our GTSP research (§4.4), such as a deeper understanding of attention heads and the advantage of using GTSP to mitigate over-fitting and over-smoothing issues.

4.1 Experimental Settings

Datasets We choose three graph classification benchmarks: one small dataset (NCI1) and two large-scale datasets from Open Graph Benchmark (OGBG-HIV and OGBG-Molpcba) (Hu et al. 2020). These datasets consist of approximately 4,000, 41,127, and 437,929 graphs, respectively. We strictly follow the original settings of these datasets, including their splitting methods.

Implementation Details We evaluate the effectiveness of our GTSP on three commonly-used GT models: GraphTrans (Wu et al. 2021), GraphGPS (Rampášek et al. 2022), and Graphormer (Ying et al. 2021) by parameters, FLOPs, and a graph classification metric (accuracy or ROC-AUC). We aim to maintain the original settings of these models as much as possible. Our main results are based on 10 runs, except for OGBG-Molpcba, which is based on five runs. All models are trained using NVIDIA A100 GPUs (40G). The detailed parameter settings can be found in the Appendix.

4.2 Accuracy w.r.t Efficiency

We evaluate the effectiveness of GTSP based on parameters, FLOPs, and accuracy. The comparison between GTSP and baseline models (including reduced-scale GraphTrans and GNN-Dense) is presented in Table 1. Based on these results, we can make several inspiring observations: **1)** Our

Table 2: Performance of other Graph Transformer models on the NCI1 dataset measured by the number of **Parameters / FLOPs Saving / Accuracy** (\uparrow). The results with higher accuracy than the baseline have been highlighted in **bold**.

Models	Spar.	GraphGPS			Graphormer		
		# Para.	FS	Acc. (%)	# Para.	FS	Acc. (%)
Base	0 %	0.18M	0%	83.71 \pm 1.77	0.17M	0%	83.36 \pm 1.18
• HP	25%	0.16M	12.0%	84.46 \pm 1.58	0.16M	12.0%	82.07 \pm 1.98
• TP	25%	0.18M	22.9%	83.24 \pm 1.20	0.17M	22.9%	82.47 \pm 1.96
• WP	25%	0.13M	20.9%	83.72 \pm 1.81	0.13M	20.9%	83.57 \pm 1.11
• LP	25%	0.15M	22.4%	83.52 \pm 1.00	0.15M	22.4%	83.52 \pm 1.91
• HP	50%	0.14M	23.6%	83.02 \pm 2.19	0.14M	23.6%	82.46 \pm 2.42
• TP	50%	0.18M	40.0%	82.92 \pm 1.60	0.17M	40.0%	82.77 \pm 1.86
• WP	50%	0.09M	45.7%	83.82 \pm 1.58	0.09M	45.7%	83.48 \pm 1.20
• LP	50%	0.11M	47.1%	82.80 \pm 1.22	0.11M	47.1%	83.26 \pm 1.34

GTSP achieves better accuracy and computation trade-offs than baseline models. **2)** As for pruning heads, GTSP can reduce the number of heads by 50% while still achieving a 0.5% improvement in accuracy, which indicates that GTSP can successfully reduce the head redundancy. **3)** Token pruning does tend to significantly reduce FLOPs (*e.g.*, 47.4% FLOPs reduction on NCI1), but it also leads to a relatively larger accuracy degradation probably because of the relatively small number of nodes in these datasets (around 20–30 nodes in a graph). **4)** Pruning weights has relatively minimal impact on accuracy and may even yield improvements, particularly on large-scale datasets (*e.g.*, 1.8% ROC-AUC increase on OGBG-HIV with 50% sparsity). **5)** After halving the number of network layers on three datasets, the performance only drops by 0.2%–3.3%, with the number of parameters decreasing by 32.9%–45.6% and FLOPs de-

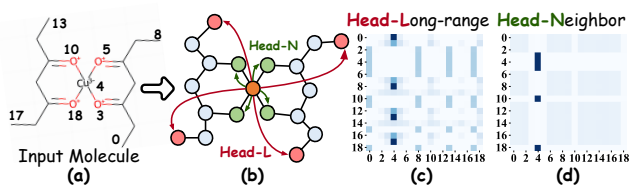


Figure 5: Characterizing the roles of different attention heads. Attention heatmaps (subfigures c and d) are shown for the molecule from OGBG-HIV.

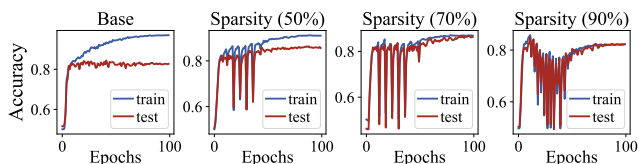


Figure 6: Training dynamics *w.r.t.* epochs of dense model (Base) and our GTSP-WP with varying sparsity levels.

creasing by 46.6%–48.8%. **6)** Finally, the extent of accuracy drop depends on the original network size: at the same scale of FLOPs reduction, smaller networks (*e.g.*, GraphTrans on NCI1) exhibit a bigger accuracy drop than large networks (*e.g.*, GraphTrans on OGBG-HIV).

4.3 Generalization to Other GT Models

To validate the generalizability of our GTSP, we apply it to another two representative GT baseline models: GraphGPS and Graphormer. The results in Table 2 demonstrate that GTSP effectively compresses these models. Despite a relatively low sparsity ratio, GTSP can match or even exceed the performance of the baseline models. These findings confirm that our GTSP is architecture-independent and can be easily integrated into other GT models.

4.4 Broader Evaluation of GTSP

Unique roles of attention heads We investigate whether heads in the model have interpretable roles. In Figure 5, we visualize two attention score matrices from a model trained on OGBG-HIV. It is observed that each head concentrates on distinct nodes: **Head-L** captures long-range information by attending to distant nodes, while **Head-N** focuses on neighboring nodes. These findings indicate that the heads have identifiable functions and are highly interpretable.

Pruning weights help alleviate over-fitting In training, over-fitting frequently occurs in GT models due to limited graph data and the large number of parameters. In this section, we will investigate whether our weight pruning strategy effectively alleviates this common problem. Figure 6 illustrates the training progress with respect to epochs for the dense model (baseline) and various sparse models with different degrees of sparsity. It is evident that as the sparsity increases, the training curve comes to more closely approximate the test curve. This confirms that our weight pruning strategies do indeed help to alleviate over-fitting in GTs.

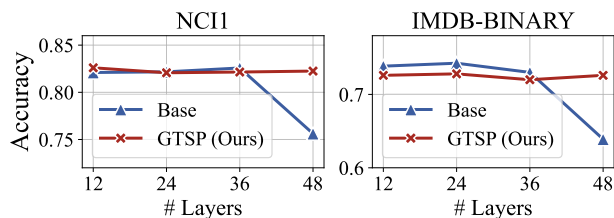


Figure 7: Accuracy *w.r.t.* the number of layers on two datasets. Base refers to the GraphTrans model.

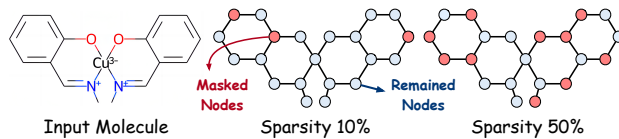


Figure 8: Token selection visualization on OGBG-HIV.

Pruning layers help alleviate over-smoothing Our previous experiments have confirmed both that the GT model layer contains redundancy and that our pruning strategy effectively reduces it. However, in our previous network, we assumed a network depth of only four layers. Recent research suggests that when the network reaches a certain depth (*e.g.*, 48 layers), it not only leads to redundancy but also causes over-smoothing (Zhao et al. 2023). Therefore, we are conducting further research to investigate whether our pruning strategy can mitigate the over-smoothing problem in excessively deep models. Figure 7 illustrates that while the accuracy of the baseline model drops suddenly at 48 layers, our GTSP maintains performance and significantly outperforms baselines. These experiments demonstrate that our GTSP helps to alleviate over-smoothing and has the potential for breaking the depth limitation of GTs.

Graph Transformer prioritizes informative nodes To further investigate the behavior of GTSP, we present a visualization of the token selection process during testing in Figure 8. The results demonstrate that our GTSP primarily focuses on chemical atoms or motifs, rather than other common motifs such as benzene rings. This indicates that our GTSP can effectively distinguish informative nodes from less-informative ones. Moreover, this phenomenon suggests that GTSP enhances the interpretability of GTs by identifying the key nodes in the graph that contribute significantly to graph property identification.

5 Conclusion

In this paper, we propose GTSP, a framework that compresses GT models by reducing the redundancy in input data, attention heads, model layers, and model weights. Extensive experiments on large-scale datasets demonstrate the effectiveness and generalizability of GTSP. Furthermore, the experimental results offer several valuable insights into existing GTs, which can potentially inspire further research. However, there remain several challenges. For instance, adjusting the pruning ratio for different graphs and jointly

pruning all components while considering their complicated interactions merit further exploration.

References

- Bian, Y.; Huang, J.; Cai, X.; Yuan, J.; and Church, K. 2021. On Attention Redundancy: A Comprehensive Study. In *ACL*.
- Chen, J.; Gao, K.; Li, G.; and He, K. 2023. NAGphormer: A Tokenized Graph Transformer for Node Classification in Large Graphs. In *ICLR*.
- Chen, T.; Cheng, Y.; Gan, Z.; Yuan, L.; Zhang, L.; and Wang, Z. 2021a. Chasing sparsity in vision transformers: An end-to-end exploration. *NeurIPS*, 34: 19974–19988.
- Chen, T.; Sui, Y.; Chen, X.; Zhang, A.; and Wang, Z. 2021b. A unified lottery ticket hypothesis for graph neural networks. In *ICML*.
- Clark, K.; Khandelwal, U.; Levy, O.; and Manning, C. D. 2019. What Does BERT Look at? An Analysis of BERT’s Attention. In *ACL Workshop*.
- Dalvi, F.; Sajjad, H.; Durrani, N.; and Belinkov, Y. 2020. Analyzing Redundancy in Pretrained Transformer Models. In *EMNLP*.
- Dwivedi, V. P.; and Bresson, X. 2021. A Generalization of Transformer Networks to Graphs. *AAAI Workshop*.
- Evcı, U.; Gale, T.; Menick, J.; Castro, P. S.; and Elsen, E. 2020. Rigging the lottery: Making all tickets winners. In *ICML*.
- Fan, A.; Grave, E.; and Joulin, A. 2020. Reducing Transformer Depth on Demand with Structured Dropout. In *ICLR*.
- Frankle, J.; and Carbin, M. 2019. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *ICLR*.
- Hoefler, T.; Alistarh, D.; Ben-Nun, T.; Dryden, N.; and Peste, A. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *JMLR*, 22(1): 10882–11005.
- Hu, W.; Fey, M.; Zitnik, M.; et al. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv:2005.00687*.
- Huang, G.; Sun, Y.; Liu, Z.; Sedra, D.; and Weinberger, K. Q. 2016. Deep networks with stochastic depth. In *ECCV*.
- Hui, B.; Yan, D.; Ma, X.; and Ku, W.-S. 2023. Rethinking Graph Lottery Tickets: Graph Sparsity Matters. In *ICLR*.
- Hussain, M. S.; Zaki, M. J.; and Subramanian, D. 2022. Global Self-Attention as a Replacement for Graph Convolution. In *SIGKDD*.
- Kim, S.; Shen, S.; Thorsley, D.; Gholami, A.; Kwon, W.; Hassoun, J.; and Keutzer, K. 2022. Learned Token Pruning for Transformers. In *SIGKDD*.
- Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.
- Kornblith, S.; Norouzi, M.; Lee, H.; and Hinton, G. 2019. Similarity of Neural Network Representations Revisited. In *ICML*.
- Kreuzer, D.; Beaini, D.; Hamilton, W. L.; Létourneau, V.; and Tossou, P. 2021. Rethinking Graph Transformers with Spectral Attention. In *NeurIPS*.
- Liang, Y.; GE, C.; Tong, Z.; Song, Y.; Wang, J.; and Xie, P. 2022. EViT: Expediting Vision Transformers via Token Reorganizations. In *ICLR*.
- Liu, C.; Ma, X.; Zhan, Y.; et al. 2023. Comprehensive Graph Gradual Pruning for Sparse Training in Graph Neural Networks. *IEEE TNNLS*.
- Liu, S.; Chen, T.; Chen, X.; et al. 2021. Sparse training via boosting pruning plasticity with neuroregeneration. In *NeurIPS*.
- Liu, X.; Yan, M.; Deng, L.; et al. 2022. Survey on Graph Neural Network Acceleration: An Algorithmic Perspective. In *IJCAI*.
- Michel, P.; Levy, O.; and Neubig, G. 2019. Are sixteen heads really better than one? In *NeurIPS*.
- Min, E.; Chen, R.; Bian, Y.; et al. 2022. Transformer for Graphs: An Overview from Architecture Perspective. *arXiv:2202.08455*.
- Mocanu, D. C.; Mocanu, E.; Stone, P.; et al. 2018. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*.
- Rampášek, L.; Galkin, M.; Dwivedi, V. P.; et al. 2022. Recipe for a General, Powerful, Scalable Graph Transformer. In *NeurIPS*.
- Rong, Y.; Bian, Y.; Xu, T.; et al. 2020. Self-Supervised Graph Transformer on Large-Scale Molecular Data. In *NeurIPS*.
- Székely, G. J.; Rizzo, M. L.; and Bakirov, N. K. 2007. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*.
- Vaswani, A.; Shazeer, N.; Parmar, N.; et al. 2017. Attention is All you Need. In *NeurIPS*.
- Voita, E.; Talbot, D.; Moiseev, F.; Sennrich, R.; and Titov, I. 2019. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. In *ACL*.
- Wu, Z.; Jain, P.; Wright, M.; et al. 2021. Representing Long-Range Context for Graph Neural Networks with Global Attention. In *NeurIPS*.
- Ying, C.; Cai, T.; Luo, S.; et al. 2021. Do Transformers Really Perform Badly for Graph Representation? In *NeurIPS*.
- You, H.; Lu, Z.; Zhou, Z.; Fu, Y.; and Lin, Y. 2022. Early-Bird GCNs: Graph-Network Co-Optimization Towards More Efficient GCN Training and Inference via Drawing Early-Bird Lottery Tickets. In *AAAI*.
- Yu, F.; Huang, K.; Wang, M.; et al. 2022. Width & depth pruning for vision transformers. In *AAAI*.
- Zhang, Z.; Liu, Q.; Hu, Q.; and Lee, C.-K. 2022. Hierarchical graph transformer with adaptive node sampling. In *NeurIPS*.
- Zhao, H.; Ma, S.; Zhang, D.; Deng, Z.-H.; and Wei, F. 2023. Are More Layers Beneficial to Graph Transformers? In *ICLR*.

Zhu, M.; and Gupta, S. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv:1710.01878*.