Multicoated and Folded Graph Neural Networks with Strong Lottery Tickets

Jiale Yan, Hiroaki Ito, Ángel López García-Arias, Yasuyuki Okoshi, Hikari Otsuka, Kazushi Kawamura, Thiem Van Chu, Masato Motomura*
Tokyo Institute of Technology, Tokyo, Japan
{yan.jiale, ito.hiroaki, lopez, okoshi.yasuyuki, otsuka.hikari, kawamura, thiem, motomura}@artic.iir.titech.ac.jp

Abstract

The Strong Lottery Ticket Hypothesis (SLTH) demonstrates the existence of high-performing subnetworks within a randomly initialized model, discoverable through pruning a convolutional neural network (CNN) without any weight training. A recent study, called Untrained GNNs Tickets (UGT), expanded SLTH from CNNs to shallow graph neural networks (GNNs). However, discrepancies persist when comparing baseline models with learned dense weights. Additionally, there remains an unexplored area in applying SLTH to deeper GNNs, which, despite delivering improved accuracy with additional layers, suffer from excessive memory requirements. To address these challenges, this work utilizes Multicoated Supermasks (M-Sup), a scalar pruning mask method, and implements it in GNNs by proposing a strategy for setting its pruning thresholds adaptively. In the context of deep GNNs, this research uncovers the existence of untrained recurrent networks, which exhibit performance on par with their trained feed-forward counterparts. This paper also introduces the Multi-Stage Folding and Unshared Masks methods to expand the search space in terms of both architecture and parameters. Through the evaluation of various datasets, including the Open Graph Benchmark (OGB), this work establishes a triple-win scenario for SLTH-based GNNs: by achieving high sparsity, competitive performance, and high memory efficiency with up to 98.7% reduction, it demonstrates suitability for energy-efficient graph processing.

1 Introduction

Graph Neural Networks (GNNs) [1–5] have emerged as powerful models for graph-based learning tasks. Within this field, both shallow GNNs [1–3] and deep GNNs [4, 5] hold significant presence, each characterized by their distinct layers and intricate structural complexities. Their success is attributed to the synergistic combination of the neighborhood aggregation scheme and weight sharing. The neighborhood aggregation scheme effectively captures local graph structures and interdependencies, while weight sharing enables GNNs to generalize across nodes, extracting meaningful features from large-scale graphs. Unfortunately, computational overheads increase rapidly when fitting GNNs to large-scale graphs.

To decrease the computational burden, studies [6–9] have utilized the Lottery Ticket Hypothesis (LTH) [10, 11] to identify graph lottery tickets, sparse subnetworks extracted from dense GNNs that can perform comparably to the original models. Among them, [6–8] simultaneously simplify the input graph and prune GNNs' weights. These approaches belong to the Weak Lottery Ticket Hypothesis (WLTH) [12–17], as they require weight re-training. On the other hand, the Untrained GNNs Tickets (UGT) study [9] focuses on pruning GNNs' weights and does not require any subsequent weight training, adhering to the Strong Lottery Ticket Hypothesis (SLTH) [18–22]. This paper primarily focuses on SLTH due to its potential as a hardware-friendly algorithm for GNNs.

*Corresponding author.

J. Yan et al., Multicoated and Folded Graph Neural Networks with Strong Lottery Tickets. *Proceedings of the Second Learning on Graphs Conference (LoG 2023)*, PMLR 231, Virtual Event, November 27–30, 2023.

For one thing, SLTH allows transforming entire GNN computations into Sparse Matrix-Matrix Multiplication (SpMM), potentially paving the way for a paradigm shift that eradicates the need for GNN's hybrid architectures [23, 24], which utilizes General Matrix-Matrix Multiplication (GeMM) and SpMM engines. For another, SLTH can be leveraged for an efficient SLTH inference hardware implementation [25], where random weights are generated on the fly instead of stored in memories. Nevertheless, the application of SLTH in GNNs introduces two challenges:

1. For shallow GNNs such as GCN, GAT, and GIN, UGT [9] provides better accuracy than Edge-Popup [11]. When observing the sparsity value of the weights varies from 0% to nearly 100%, we find discrepancies persist when compared to baseline models trained using dense weights. There is still a question for research: **Is it possible to devise a better SLTH that maintains high accuracy for shallow GNNs?**
2. Concerning deep GNNs, UGT explores the implementation of SLTH on GCN, GAT, and GIN by directly increasing the number of layers. However, a limitation is that these enlarged models do not match the accuracy level of two-layer models. Concurrently, ResGCNs [4, 5] exhibit improved accuracy with increased layer depth but need more memory consumption. Although [7] explored WLTH on ResGCN+ [5], the SLTH has never been explored for them. Here are two questions: **First, is applying SLTH to deep GNNs feasible? Second, can we further enhance model efficiency by reducing the size without compromising the high accuracy?**

Inspired by recent works emerging around the SLTH, including multicoated supermasks (M-Sup) [22] and folding methods [26, 27], this work exploits the SLTH on GNNs by implementing M-Sup on GNNs and analyzes their effects. Furthermore, it adopts the folding method to optimize deep GNNs such as ResGCNs. The contributions of this paper are summarized as follows:

1. This research identifies a significant performance improvement in GNNs using M-Sup, i.e., with solely using randomly initialized weights. By identifying high-performing subnetworks in their random initial states, this work can surpass the performance of single-coated supermask (S-Sup), e.g., UGT, and the accuracy of the dense-weight training baseline. Contrary to the WLTH, this process does not need weight learning.
2. Based on M-Sup, the research analyzes the weight score distribution of GNN models and proposes an adaptive strategy to determine pruning thresholds for different score distributions. Experiments show its advantage in terms of accuracy.
3. This paper is the first time to demonstrate the existence of untrained recurrent graph subnetworks within deep GNNs, exhibiting comparable performance to their trained feed-forward counterparts. Furthermore, this study introduces the Multi-stage folding (MSF) and Unshared Mask methods designed for a broader search space within the domains of network structure and weight scores. By capitalizing on the signed Kaiming Constant (SC) initialization without weight training, ResGCN+ and DyResGEN models [5] achieve memory reductions of 72% and 98.7%, respectively, while maintaining an accuracy comparable to the baseline models.

2 Related Work

2.1 Graph Neural Networks

GNNs have demonstrated state-of-the-art performance on graph-structure tasks [1–3, 28–30]. After GCN [1] proposed a two-layer GNN for node classification, more GNNs such as GAT [2], GIN [3], GraphSage [28], PNA [29], and DiffPool [30] were put forward. Given an undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ is a set of edges, there are $|\mathcal{V}|$ nodes, and each node has a feature vector $x_i \in \mathbb{R}^F$, where F is the number of a node feature. The adjacency matrix is $A \in \mathbb{R}^{|\mathcal{V}\times\mathcal{V}|}$ and node features are $X \in \mathbb{R}^{|\mathcal{V}|\times F}$. A typical 2-layer GNN can be defined as

$$\mathcal{G}^{(2)}\left(\hat{A}, X, W\right) = \mathrm{softmax}\left(\hat{A}\sigma\left(\hat{A}XW^{(0)}\right)W^{(1)}\right), \tag{1}$$

where $W^{(0)}$ and $W^{(1)}$ are learnable weights for two layers, $\hat{A} = \tilde{D}^{-\frac{1}{2}}\left(A + I\right)\tilde{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix, and $\tilde{D}$ is the degree matrix of $\hat{A}$. In addition, $\mathrm{softmax}$ is the softmax function, and $\sigma\left(\cdot\right)$ is an activation function.

Despite GNNs having seen rapid and substantial progress, most prior studies still employ shallow structures. Unlike CNNs, GNNs suffer from vanishing gradient and over-smoothing [31] when going deeper. Inspired by the benefit of training deep CNN-based networks [32, 33], ResGCN variants [4, 5] were developed by adopting residual connections and dilated convolutions to GCNs. They solve the degradation problem and outperform traditional shallow GNNs. Moreover, the development of deep GNNs has also benefited from techniques like dropping DropEdge [34] and DropNode [35], as well as normalization methods [36, 37].

2.2 Weak and Strong Lottery Ticket Hypothesis

A breakthrough paper [10] revealed Lottery Ticket Hypothesis (LTH), which showed that an overparameterized neural network contains a subnetwork that can be trained in isolation to match the original models. Similar studies have quickly developed into two groups: Weak Lottery Ticket Hypothesis (WLTH) and Strong Lottery Ticket Hypothesis (SLTH). For WLTH, the subnetworks need iterative training, pruning, and resetting the remaining weights to their original value. After LTH has shown its potential capability, other studies focused on the early training stage [12] or even before training [13–17]. Not limited to CNNs, GLT [6] was the first study to generalize WLTH to GNNs. Later, DGLT [7] explored transferring a random ticket to a graph lottery ticket. GLT [8] proposed a new auxiliary loss function to guide pruning better when the graph sparsity is high. On the other hand, the SLTH [18] revealed that an overparameterized network has high-performing subnetworks at a randomly initialized state, only needing pruning to be discovered. After that, a series of papers [19–22] added theoretical proof to the theory on CNNs. For the GNN field, distinct from [6–8], UGT [9] achieved competitive accuracy without any model weight training.

2.3 Hidden-Fold Network

The folding method can convert deep feed-forward networks into shallow recurrent models, which is applied in ResNets [32]. A range of studies [26, 38, 39] showed that ResNets can be closely approximated as unrolled shallow recursive neural networks, attributing the benefits of additional layers to recursive iterations. In particular, [39] demonstrated that the accuracy is only marginally impacted when folding ResNet into a 4-layer model. Building on this concept of folded networks, [27] pioneered the exploration of SLTH in folded residual networks. This approach was subsequently enhanced by [26] without causing a significant loss. A blend of SLTH with folded networks is called Hidden-Fold Network (HFN)[26, 27].

3 Multicoated and Folded GNNs

3.1 Single-coated Supermask (S-Sup) in strong GNN tickets

UGT [9] utilizes Edge-Popup [11] method by applying S-Sup to acquire strong tickets from a GNN. Given a $\mathcal{G}\left(\hat{A}, X, W\right)$, UGT formulates the weight matrix of layer l as follows

$$W^{(l)} = W^{(l)}_{\text{rand}} \odot \mathcal{H}(\mathcal{S}^{(l)}), \mathcal{H}(\mathcal{S}^{(l)}) \in \{0, 1\}, \tag{2}$$

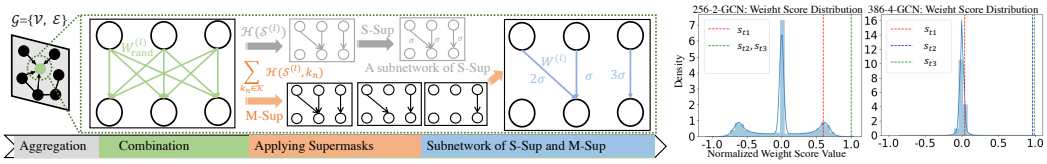
where $\odot$ signifies the Hadamard product operator, $W^{(l)}_{\text{rand}}$ denotes random tensors for weights, $\mathcal{H}$ is the binary supermask generation function, and $\mathcal{S}^{(l)}$ is the underlying score for a specific layer. Each element $h(s_{uv})$ in $\mathcal{H}(\mathcal{S}^{(l)})$ is defined as

$$h(s_{uv}) = \begin{cases} 1 & , |s_{uv}| \geq s_{\text{threshold}} \\ 0 & , |s_{uv}| < s_{\text{threshold}} \end{cases}, \tag{3}$$

where the s_{uv} is the score corresponding to a weight value in $W^{(l)} \in \mathbb{R}^{|U \times V|}$ and $s_{\text{threshold}}$ is a threshold. The $s_{\text{threshold}}$ is based on a predetermined sparsity value $k\%$. This value aids in pruning the smallest $k\%$ of the scores $|s_{uv}| \in \mathcal{S}^{(l)}$.

3.2 M-Sup with adaptive threshold for GNNs

Building on the prior work [22], which applies M-Sup to ResNet, this study expands M-Sup to GNNs without requiring weight training. After applying one supermask with multiple coats, the weight



(a) **(b)**

Figure 1: A GNN with the M-Sup method: (a) shows how to apply M-Sup to a GNN. Weights are randomly initialized to $\pm\delta$, where δ is the standard deviation of the Kaiming normal distribution. (b) illustrates weight score distributions of GNN models. 256-2-GCN has 256 hidden neurons and two layers used on Cora, and 386-4-GCN has 386 hidden neurons and four layers used on OGBN-Arxiv. $s_{t_1}, s_{t_2}, s_{t_3}$ are the thresholds with the Linear method.

matrix of layer l becomes

$$W^{(l)} = W^{(l)}_{\text{rand}} \odot \sum_{k_n \in \mathcal{K}} \mathcal{H}(\mathcal{S}^{(l)}, k_n), \mathcal{H}(\mathcal{S}^{(l)}, k_n) \in \{0, 1\}\,. \tag{4}$$

$\mathcal{K}$ is a set of sparsity values, corresponding to N coats, where $k_1 \leq k_2 \leq ... \leq k_N$. Here, the latter coat is more sparse than the previous one. Each value k_n in the set $\mathcal{K}$ is determined by threshold s_{t_n}, which results in pruning $k_n\%$ of $\mathcal{S}^{(l)}$ based on $|s_{uv}| < s_{t_n}$. This process is illustrated in Figure 1.

Adaptive threshold. This study proposes an adaptive method to define $\mathcal{K}$ for M-Sup. [22] proposed Linear and Uniform methods to set the sparsity $\mathcal{K}$. The Linear method defines $s_{t_n} = s_{t_1} + 3\sigma_s \times \frac{n-1}{N}$, where s_{t_1} is the threshold score corresponding to sparsity k_1, and σ_s refers to the standard deviation of the normalized weight scores. The Uniform method directly defines $k_n = k_1 + (100\% - k_1) \times \frac{n-1}{N}$. In a practical implementation, the Linear method proved less effective. As depicted in Figure 1b, the weight score distribution for a 385-4-GCN model is suitable for the Linear method, with three effective values $s_{t_n} < 1.0$, but for a 256-2-GCN model, the weight score distribution becomes multimodal, presenting a challenge. In this specific case, both s_{t_2} and s_{t_3} equate to 1.0, making the respective coats ineffective. The situation will further deteriorate if UGT's linear decay schedule [9] is applied, where sparsity list are $\mathcal{K}_t = \mathcal{K} \cdot \frac{\beta \cdot t_{\text{current epoch}}}{T_{\text{total epochs}}}$, and β is an empirical hyperparamete. These meaningless coats disrupt training by pruning partial weights since they do not begin at 100% in the training phase, leading to potential problems in the model's performance.

Therefore, for the Linear threshold, an adaptive determination for s_{t_n} corresponding to $k_n \in \mathcal{K}$ is proposed as follows:

$$s_{t_n} = \left\{ s_{t_1} + \frac{3\sigma_s}{N} \times (n-1), \phi \right\}, \text{Linear (w/ pre-training)} \tag{5}$$

The method works as follows: when provided with a pre-trained single-coat model, it utilizes the Linear method to calculate s_{t_n}, but only if s_{t_n} is smaller than α. α is an empirical hyperparameter with a value of 0.9996. Otherwise, it is defined as ϕ, which means the coat is invalid. If no pre-trained models are available, the method uses a uniform approach to directly calculate k_n.

3.3 Folded GNN

Hidden-Folded GNN. Based on typical deep GNNs such as ResGCNs [4, 5], this work folds residual blocks into a single recurrent block. This transformation is represented in Figure 2. This study reuses a general graph convolution operation $\mathcal{F}$ [4] with learnable parameters, including weights. Based on the folding structure, this paper also implements S-Sup and M-Sup. The corresponding formulas are as follows:

$$\begin{cases} \mathcal{G}^{(1)} = \mathcal{F}\left(\mathcal{G}^{(0)}, \mathcal{W}^{(0)}\right) + \mathcal{G}^{(0)} \\ \quad\vdots \\ \mathcal{G}^{(l)} = \mathcal{F}\left(\mathcal{G}^{(l-1)}, \mathcal{W}^{(l-1)}\right) + \mathcal{G}^{(l-1)} \end{cases}, \mathcal{W}^{(l-1)} = \cdots = \mathcal{W}^{(0)} = \mathcal{W}^{(0)}_{\text{rand}} \odot \sum_{k_n \in \mathcal{K}} \mathcal{H}(\mathcal{S}^{(0)}, k_n). \tag{6}$$

Furthermore, the multi-folding stages (MFS) and Unshared Masks methods are proposed to explore optimized hidden-folded GNNs. The former expands multi-folding stages from the model architecture

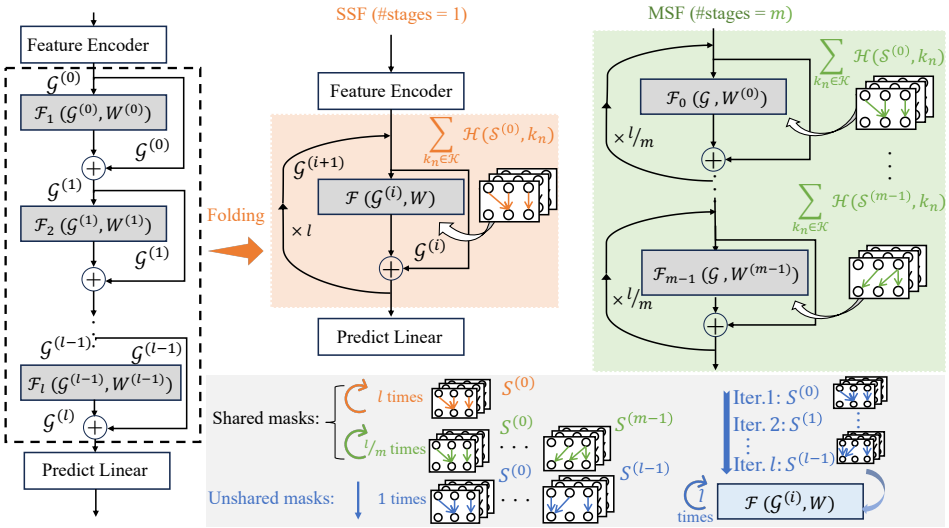


Figure 2: Folded GNN: a ResGNN can be folded into a recursive structure like a 1-stage or an n-stage MSF. Shared masks have a set for each stage and are reused as often as the number of iterations for one stage. Unshared masks are used only once, with individual masks for each iteration.

Algorithm 1 Initialization and Inference about Folded GNN

Input: l-layer , m-stages , iterations $r = \frac{l}{m}$, sparsity list $\mathcal{K} = k_1, ..., k_N$
#If enable SSF, $m = 1$; otherwise, $m > 1$ for MSF.
Output: Features of the last layer: $\mathcal{G}^l$
if Enable Unshared Masks **then**
  Create l sets of scores: $\mathcal{S} = \{\mathcal{S}^{(0)}, \mathcal{S}^{(1)}, \cdots, \mathcal{S}^{(l-1)}\}$
else
  Create m sets of scores: $\mathcal{S} = \{\mathcal{S}^{(0)}, \mathcal{S}^{(1)}, \cdot, \mathcal{S}^{(m-1)}\}$
end if
for $i = 0$ to $l - 1$ **do**
  Calculate the index of a reused layer: $j = i//r$
  if Enable Unshared Masks **then**
    $\mathcal{W}^{(i)} = \mathcal{W}^{(j)}_{\text{rand}} \odot \sum_{k_n \in \mathcal{K}} \mathcal{H}(\mathcal{S}^{(i)}, k_n)$
  else
    $\mathcal{W}^{(i)} = \mathcal{W}^{(j)}_{\text{rand}} \odot \sum_{k_n \in \mathcal{K}} \mathcal{H}(\mathcal{S}^{(j)}, k_n)$
  end if
  Calculate the output of the current layer: $\mathcal{G}^{i+1} = \mathcal{F}_j\left(\mathcal{G}^{(i)}, \mathcal{W}^{(i)}\right)$
end for
Return $\mathcal{G}^l$

perspective and the latter from the weight scores perspective. They are outlined in Algorithm 1 and can be implemented individually or combined - e.g., MSF structures can be used in conjunction with unshared masks. The pseudocode for applying SLTH with the proposed methods is provided in Appendix E.

MSF method. In this approach, continuous stages are folded into multi-stage m, producing m graph mappings denoted as $\{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{\text{m}-1}\}$. Given an l-layer and m-stages, each graph mapping block is utilized $\frac{l}{m}$ times. If l is not a multiple of m, the integral division part of the blocks is reused, and the remainder part of the blocks is left unfolded. When $m = 1$, it is single-stage folding (SSF).

Shared and Unshared Supermasks. In contrast to shared supermasks, unshared supermasks are individually applied to each iteration. SSF structures with shared supermasks, a set of supermasks $\sum_{k_n \in \mathcal{K}} \mathcal{H}(\mathcal{S}^{(0)}, k_n)$, is shared across each iteration. In the unshared approach, l sets of scores are created, denoted as $\sum_{k_n \in \mathcal{K}} \mathcal{H}(\mathcal{S}^{(0)}, k_n) \cdots \sum_{k_n \in \mathcal{K}} \mathcal{H}(\mathcal{S}^{(l-1)}, k_n)$, which are then deployed individually in each iteration.

4 Experiments and Discussions

This section describes experiments using various GNN architectures and datasets to assess the effectiveness of the proposed methods.

GNN Architectures. For shallow GNNs, this study utilizes GCN [1], GAT [2], and GIN [3]. For deep GNNs, ResGCNs [5] are used, including a 7-layer DyResGEN and a 28-layer ResGCN+.

Datasets. The experiments uses three widely used graph datasets, Cora, Citeseer, and PubMed [1], and the large-scale graph dataset OGBN-Arxiv [40] for node-level tasks (metric: Accuracy%). Additionally, the research adopts OGBG-Molhiv and OGBG-Molbace [40] for graph-level tasks (metric: ROC-AUC%). Node-level tasks involve predicting properties or characteristics at the individual node level within a graph. Graph-level tasks focus on predictions about the graph structure.

Experimental settings. During training, this study employs two weight initialization methods as [11]: Signed Kaiming Constant (SC) and Kaiming Normal (KN). Both methods utilize a scaling factor of $\sqrt{1/(1-k_1)}$, defined according to the sparsity parameter k_1. Kaiming Uniform (KU) initialization is applied to weight scores. Through the experiments, SC performs better than KN in the S-Sup method, consistent with results reported in [22]. Detailed findings are in Appendix B. Since the SC is well-suited to a specialized SLTH inference accelerator [25], we prefer using the SC method. For graph-level tasks, the weights of an embedding list, denoted by $\left\{W_1 \in \mathbb{R}^{N_1 \times M}, \cdots, W_n \in \mathbb{R}^{N_n \times M}\right\}$, are also initialized with the SC method. We set the weight scores as a vector in $\mathbb{R}^{1 \times M}$ rather than the same size of weights. It is applied to the look-up table's final output row by row, resulting in a memory-efficient compression. For sparsity scheduling, the experiments adopt a linear decay schedule as UGT [9]. Further implementation details are provided in Appendix A.

Model Compression Scheme. This article discusses the memory sizes of models, taking into account a particular compression method tailored for specialized hardware. Weights and biases are considered to occupy 32 bits each. However, it is not required to store the weights if they are initialized in an SC manner, as they can be generated on the fly from the original seed using a random number generator. Additionally, this seed can be replaced with a hash of other model parameters [25], eliminating the need to store it. As a result, the memory requirement for models employing supermask training consists solely of the supermask's size, equating to one bit per weight for the S-Sup and $(1 + \sum_{n=1}^{N-1} k_n)$ bits per weight averagely for M-Sup with N coats, in addition to the affine normalization parameters, bias and other things that a model may use.

4.1 M-Sup finds better graph lottery tickets than S-Sup

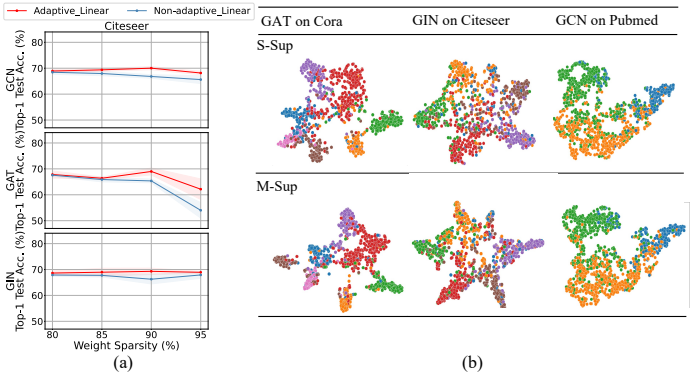
This subsection evaluates shallow and deep GNNs in node- and graph-level tasks. As the weight sparsity varies, the findings show the distinct advantage of M-Sup over S-Sup while achieving comparable accuracy as the baseline models, which use dense-weight learning (DWL).

Shallow GNNs in node- and graph-level tasks. For node-level tasks, we adopt 2-layer GNNs on Cora, Citeseer, and PubMed datasets, and 4-layer GNNs with affine batch normalization (BN) on OGBN-Arxiv. For graph-level tasks, we use a 3-layer GCN with embedding lists on OGBG-Molhiv and OGBG-Molbace.

Figure 3 (a) shows the difference in accuracy between using an adaptive Linear threshold ($\alpha = 0.9996$) and a non-adaptive Linear threshold ($\alpha = 1.0$). Experiments show the adaptive Linear threshold is better when sparsity $\geq 80\%$, because some ineffective coats appear in a non-adaptive Linear threshold. Other GNNs' experiment results are provided in Appendix C. The adaptive Linear threshold is used in the following experiments.

The performance of GCN, GAT, and GIN on node-level tasks is illustrated in Figure 4. When sparsity is near 0%, M-Sup outperforms S-Sup's accuracy. As the sparsity increases from 0% to 20%, although the S-Sup approach sees a sharp accuracy increase, it still trails behind M-Sup. M-Sup maintains consistent accuracy without apparent drops, even with high sparsity ($80 \sim 90\%$). S-Sup models show better accuracy in the 20%-75% sparsity range than other ranges, occasionally even achieving comparable accuracy as DWL models. Based on these GNN models, Figure 3 (b) visualizes the node representations learned by M-Sup and S-Sup. The projection of node representations learned by M-Sup maintains a distinguishable effect. Moreover, the phenomenon persists when fixing the layer and adjusting the number of hidden neurons, especially at lower sparsity values. The results are

detailed in Appendix D. GCN's performance on graph-level tasks is shown in Figure 5 (a) and (b). M-Sup outperforms S-Sup and can achieve accuracy comparable to a dense-weight learning model.



(a) (b)

Figure 3: Advantages of adaptive Linear threshold and M-Sup methods: (a) shows the difference in accuracy between adaptive Linear threshold and no-adaptive Linear threshold for 3 GNN models on Citeseer. (b) shows the T-Distributed Stochastic Neighbor Embedding (TSNE) visualization of node representations learned by S-Sup and M-Sup (sparsity is 15%).

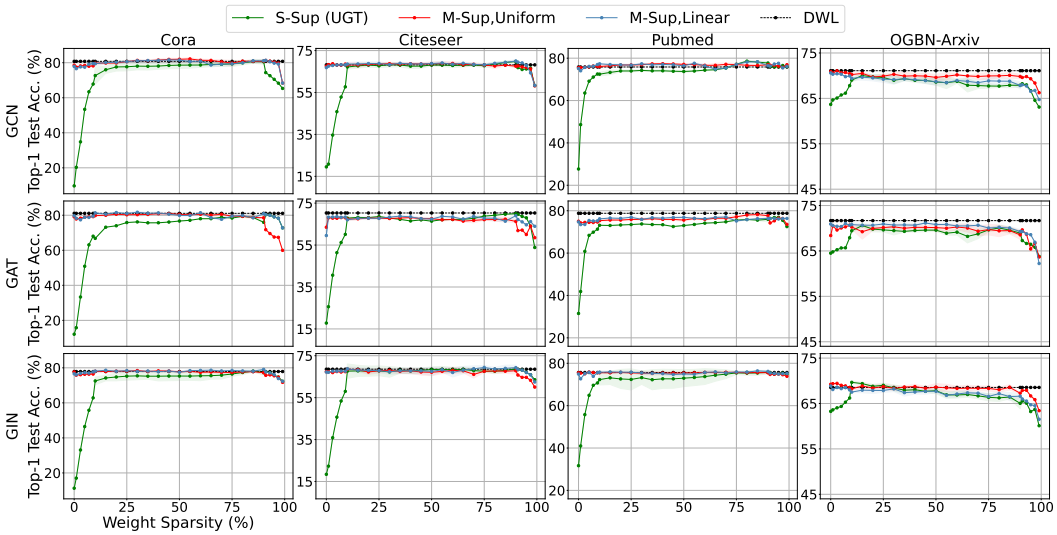


Figure 4: The accuracy of GCN, GAT, and GIN with various sparsity values on four datasets: experiments use 2-layer GNNs of width 256 for the Cora, Citeseer, and Pubmed datasets. For the OGBN-Arxiv datasets, a 4-layer GNN with a width of 386 is used, and affine BN is applied. The head of GAT is 1 for the following experiments.

Deep GNNs in node- and graph-level tasks. We evaluate the method on deep GNNs on both node- and graph-level tasks. Specifically, we employ a 28-layer ResGCN+ for the OGBN-Arxiv dataset and a 7-layer DyResGEN for the OGBG-Molhiv dataset. The numbers of coats $N = \{3, 5\}$ with the Linear threshold. As shown in Figure 5 (c) and (d), M-Sup $N = \{3, 5\}$ surpasses the performance of S-Sup. However, the difference in accuracy between $N = 3$ and $N = 5$ is not apparent.

In addition, comparing the Uniform threshold and Linear threshold, Figure 4 shows no apparent difference on Cora, Citeseer, and Pubmen. On OGBN-Arxiv with GAT, the Linear threshold outperforms, but the Uniform threshold is better with GCN and GIN. The superior approach potentially depends on the neural networks and datasets being used.

4.2 Deep GNNs are folded with SLTH

This subsection explores applying folding methods, including MSF and unshared masks, to deep GNNs. Our results demonstrate that folding methods in combination with SLTH can attain accuracy comparable to the original feed-forward counterparts while enhancing memory efficiency.

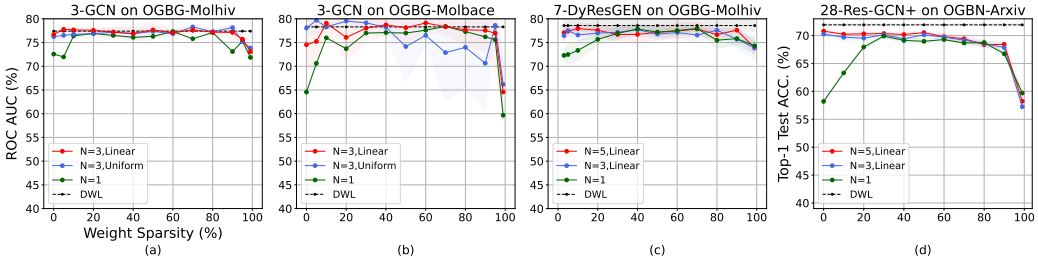


Figure 5: Accuracy of GNN models across various sparsity values with M-Sup ($N = \{3, 5\}$) and S-Sup ($N = 1$, UGT [9]): (a) and (b) depict shallow GNNs utilizing a 3-layer GCN with a width of 448. (c) and (d) are about deep GNNs, where (c) shows a 7-layer DyResGEN on OGBG-Molhiv, and (d) focuses on a 28-layer ResGCN+ applied to OGBN-Arxiv.

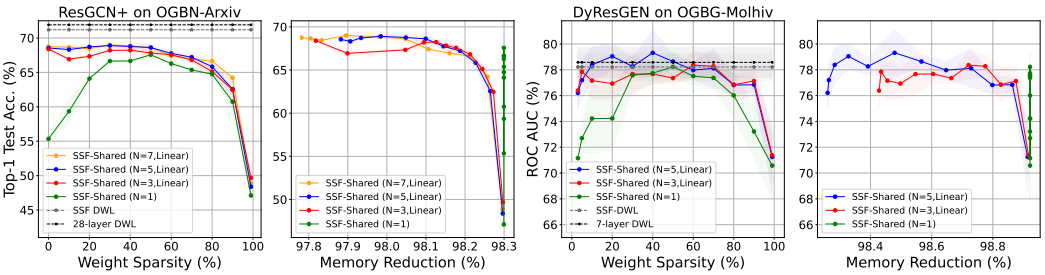


Figure 6: Comparison of accuracy and memory reduction between deep GNNs and DWL models: The figure illustrates using M-Sup ($N = 3, 5, 7$) and S-Sup ($N = 1$) for ResGCN+ and DyResGEN. SSF-DWL represents folding the original model into a single stage with dense-weight learning. The number of hidden neurons are 128 for ResGCN+ and 256 for DyResGEN.

Single-stage folding method with shared masks shows comparable accuracy and high memory efficiency. This study evaluates the single-stage folding (SSF) method with shared masks on deep GNNs and compares their accuracy and memory reduction with the baseline dense-weight leaning (DWL) models. As depicted in Figure 6, M-Sup consistently outperforms S-Sup regarding accuracy across all folded models. In M-Sup, ResGCN+ models employing SSF-Shared achieve competitive accuracy on OGBN-Arxiv, with more than 97% memory reduction. DyResGEN models using SSF-Shared surpass baseline models in accuracy on OGBG-Molhiv, with a 98% memory reduction.

Unshared masks outperform shared masks with both MSF and SSF methods. Likewise, the MSF method outperforms SSF with both shared and unshared masks. Furthermore, to evaluate the effectiveness of the unshared and MSF, this work applies them to deep GNNs. Specifically, unshared masks are utilized in ResGCN+ and DyResGEN, while MSF is applied to the 28-layer ResGCN+, which is suitable to be folded into four stages (S = 4). In the case of the ResGCN+, experiments are segmented into four groups, each characterized by different {folding, sharing} configurations. Within each group, one of the settings remains constant while the other is altered, accommodating both S-Sup and M-Sup ($N = 3$, Linear). For DyResGEN, experiments are segmented into two groups, which have distinct {sharing} configurations. A comparative analysis reveals that the unshared mask method outperforms the shared mask method in SSF and MSF, as illustrated in Figure 7.a, Figure 7.b and Figure 7.e. Likewise, in contrast to the SSF method, the MSF method outperforms in shared and unshared mask configurations, as depicted in Figure 7.c and Figure 7.d.

Although unshared masks and MSF enhance accuracy, they result in an increase in both memory and parameters. This creates a trade-off which will be discussed in the following subsection.

4.3 Trade off between the parameter count, memory size, and accuracy

Optimized GNN models show high memory efficiency with competitive performance. When considering the MSF and unshared mask methods, scores are increased compared to the SSF and shared mask methods. Although the model size increases in MSF and unshared mask methods, the additional parameters are only single-bit, and the accuracy is improved. This subsection explores this trade-off by evaluating optimized GNNs on OGBs.

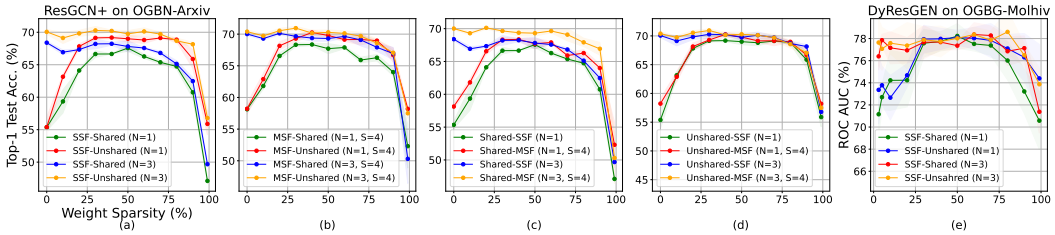


Figure 7: 28-layer ResGCN+ and 7-layer DyResGEN with different folding methods on the OGBN-Arxiv and OGBG-Molhiv datasets: (a)-(d) ResGCN+. and (e) DyResGEN. (a) SSF, Shared and Unshared masks; (b) MSF, Shared and Unshared masks; (c) Shared Mask, SSF and MSF; (d) Unshared Mask, SSF and MSF; (e) SSF, Shared and Unshared masks.

Figure 8 compares the accuracy, memory size, and parameter count of various models across different datasets. For DyResGEN in OGBG-Molhiv, SSF-Shared models are the most parameter-efficient and memory-efficient. Compared to a 7-layer DyResGEN, these models achieve a parameter and memory reduction, with 79.9% fewer parameters and 98.7% less memory, while maintaining comparable accuracy. Conversely, SSF-Unshared models require more parameters and memory but can achieve slightly better accuracy. The optimized 3-layer GCN on OGBG-Molhive also achieves comparable accuracy to the baseline model with a 95.8% memory reduction. For ResGCN+ on OGBG-Arixv, this study investigates two hidden neurons, 128 and 256, and utilizes M-Sup with N-values of $1, 3, 5, 7$ with SSF and MSF (stage = 4) structures. Supermasks are applied in both shared and unshared manners. Among them, the SSF-Shared-128 models with S-Sup achieve a 77.9% reduction in parameters and a 97.9% reduction in memory, albeit with an accuracy 2.8% lower than the 28-Layer ResGCN model. In contrast, SSF-Unshared-256 and MSF-Unshared-256 models with S-Sup attain a 72% memory reduction while maintaining accuracy comparable to the 28-Layer ResGCN model. Exact performance is also shown in Appendix H. Within this space, various choices emerge, offering multiple candidates for balancing trade-offs between memory, parameters, and performance.

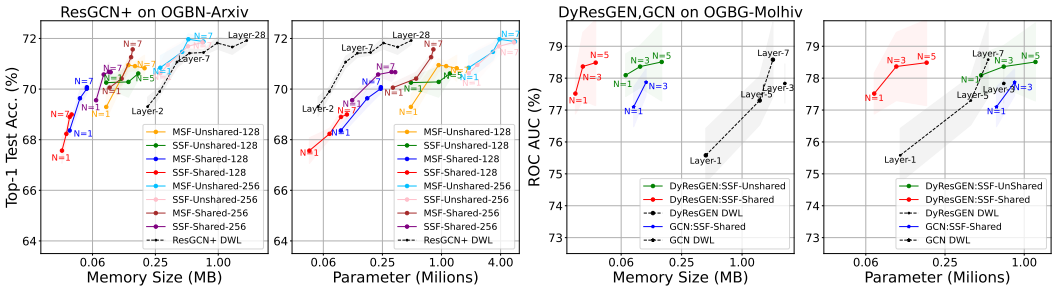


Figure 8: Comparison of the optimized GNN models with different methods on OGBs: optimized ResGCN+ models include $\{128, 256\}$ hidden neurons, S-Sup ($N = 1$, UGT), M-Sup ($N = \{3, 5, 7\}$, Linear threshold), shared and unshared masks. ResGCN+ DWL are baseline models with dense-weight training; their depths contain $\{2, 3, 5, 7, 10, 14, 20, 28\}$. Optimized DyResGEN models contain S-Sup ($N = 1$, UGT), M-Sup ($N = \{3, 5\}$, Linear method), shared and unshared masks. DyResGEN baseline model uses $\{1, 5, 7\}$ layers with dense-weight training. Optimized GCN model with three layers contains S-Sup ($N = 1$, UGT), M-Sup ($N = 3$, Linear method), and shared masks.

5 Conclusion

To enhance the performance of GNNs based on strong lottery tickets, we first applied multicoated supermasks to GNNs with adaptive thresholds. This approach surpasses the previous SLTH method, named UGT. We further reveal the existence of untrained recurrent subnetworks within deep GNNs. By employing Multi-stage Folding and Unshared Mask methods, we explore a larger search space for deep GNNs, achieving significant memory reduction while maintaining an accuracy comparable to the baseline models. These findings are supported by experiments on widely-used GNN models, encompassing diverse datasets, including the Open Graph Benchmark (OGB). This work contributes to energy-efficient graph processing.

Acknowledgements

This work was partially supported by Kakenhi Grant Number 23H05489.

References

[1] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. 1, 2, 6, 13

[2] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. 2, 6

[3] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. 1, 2, 6

[4] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can GCNs go as deep as CNNs? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9267–9276, 2019. 1, 2, 3, 4, 13

[5] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. DeeperGCN: All you need to train deeper GCNs. *arXiv preprint arXiv:2006.07739*, 2020. 1, 2, 3, 4, 6, 13

[6] Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In *International conference on machine learning*, pages 1695–1706. PMLR, 2021. 1, 3, 17

[7] Kun Wang, Yuxuan Liang, Pengkun Wang, Xu Wang, Pengfei Gu, Junfeng Fang, and Yang Wang. Searching lottery tickets in graph neural networks: A dual perspective. In *The Eleventh International Conference on Learning Representations*, 2022. 2, 3, 13, 17

[8] Bo Hui, Da Yan, Xiaolong Ma, and Wei-Shinn Ku. Rethinking graph lottery tickets: Graph sparsity matters. In *The Eleventh International Conference on Learning Representations*, 2023. 1, 3, 13

[9] Tianjin Huang, Tianlong Chen, Meng Fang, Vlado Menkovski, Jiaxu Zhao, Lu Yin, Yulong Pei, Decebal Constantin Mocanu, Zhangyang Wang, Mykola Pechenizkiy, and Shiwei Liu. You can have better graph neural networks by not training weights at all: Finding untrained GNNs tickets. In *The First Learning on Graphs Conference*, 2022. 1, 2, 3, 4, 6, 8, 13, 15, 17

[10] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 1, 3

[11] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11893–11902, 2020. 1, 2, 3, 6

[12] Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Toward more efficient training of deep networks. In *International Conference on Learning Representations*, 2019. 1, 3

[13] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip HS Torr. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations*, 2019. 3

[14] Yulong Wang, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu. Pruning from scratch. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12273–12280, 2020.

[15] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2019.

[16] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *International Conference on Learning Representations*, 2020.

[17] Kartik Sreenivasan, Jy-yong Sohn, Liu Yang, Matthew Grinde, Alliot Nagle, Hongyi Wang, Eric Xing, Kangwook Lee, and Dimitris Papailiopoulos. Rare gems: Finding lottery tickets at initialization. *Advances in Neural Information Processing Systems*, 35:14529–14540, 2022. 1, 3

[18] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in neural information processing systems*, 32, 2019. 1, 3

[19] Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pages 6682–6691. PMLR, 2020. 3

[20] Ankit Pensia, Shashank Rajput, Alliot Nagle, Harit Vishwakarma, and Dimitris Papailiopoulos. Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient. *Advances in neural information processing systems*, 33:2599–2610, 2020.

[21] Laurent Orseau, Marcus Hutter, and Omar Rivasplata. Logarithmic pruning is all you need. *Advances in Neural Information Processing Systems*, 33:2925–2934, 2020.

[22] Yasuyuki Okoshi, Ángel López García-Arias, Kazutoshi Hirose, Kota Ando, Kazushi Kawamura, Thiem Van Chu, Masato Motomura, and Jaehoon Yu. Multicoated supermasks enhance hidden networks. In *Proc. Int. Conf. Mach. Learn.*, pages 17045–17055, 2022. 1, 2, 3, 4, 6

[23] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. Hygcn: A gcn accelerator with hybrid architecture. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 15–29. IEEE, 2020. 2

[24] Mingi Yoo, Jaeyong Song, Jounghoo Lee, Namhyung Kim, Youngsok Kim, and Jinho Lee. Sgcn: Exploiting compressed-sparse features in deep graph convolutional network accelerators. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1–14. IEEE, 2023. 2

[25] Kazutoshi Hirose, Jaehoon Yu, Kota Ando, Yasuyuki Okoshi, Ángel López García-Arias, Junnosuke Suzuki, Thiem Van Chu, Kazushi Kawamura, and Masato Motomura. Hiddenite: 4K-PE hidden network inference 4D-tensor engine exploiting on-chip model construction achieving 34.8-to-16.0 TOPS/W for CIFAR-100 and ImageNet. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022. 2, 6

[26] Ángel López García-Arias, Yasuyuki Okoshi, Masanori Hashimoto, Masato Motomura, and Jaehoon Yu. Recurrent residual networks contain stronger lottery tickets. *IEEE Access*, 11: 16588–16604, 2023. doi: 10.1109/ACCESS.2023.3245808. 2, 3

[27] Ángel López García-Arias, Masanori Hashimoto, Masato Motomura, and Jaehoon Yu. Hidden-fold networks: Random recurrent residuals using sparse supermasks. In *32nd British Machine Vision Conference 2021, BMVC 2021, Online, November 22-25, 2021*, page 205. BMVA Press, 2021. 2, 3

[28] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 2

[29] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020. 2

[30] Zhitao Ying. Jiaxuan you, christopher morris, xiang ren, will hamilton, and jure leskovec. hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31:4800–4810, 2018. 2

[31] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 3

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 3

[33] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 3

[34] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019. 3

[35] Wenbing Huang, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. Tackling over-smoothing for general graph convolutional networks. *arXiv preprint arXiv:2008.09864*, 2020. 3

[36] Kuangqi Zhou, Yanfei Dong, Kaixin Wang, Wee Sun Lee, Bryan Hooi, Huan Xu, and Jiashi Feng. Understanding and resolving performance degradation in deep graph convolutional networks. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2728–2737, 2021. 3

[37] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in GNNs. *arXiv preprint arXiv:1909.12223*, 2019. 3

[38] Klaus Greff, Rupesh K Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. *arXiv preprint arXiv:1612.07771*, 2016. 3

[39] Qianli Liao and Tomaso Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016. 3

[40] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020. 6, 13

[41] Pierre Stock, Angela Fan, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression. In *ICLR 2021-International Conference on Learning Representations*, 2021. 17

[42] Shyam Anil Tailor, Javier Fernandez-Marques, and Nicholas Donald Lane. Degree-quant: Quantization-aware training for graph neural networks. In *International Conference on Learning Representations*, 2020. 17

[43] Haoran You, Tong Geng, Yongan Zhang, Ang Li, and Yingyan Lin. Gcod: Graph convolutional network acceleration via dedicated algorithm and accelerator co-design. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 460–474. IEEE, 2022. 17

[44] Junjie Mu and Bongjin Kim. 29.2 a 21$\times$ 21 dynamic-precision bit-serial computing graph accelerator for solving partial differential equations using finite difference method. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 406–408. IEEE, 2021. 17

A Implementation details

In this paper, experiments about shallow GNNs, including 2 and 4-layer GCN, GAT, and GIN, are conducted on 1 TESLA V100 GPU (32GB). Experiments about deep GNNs, including 28-layer ResGCN+ and 7-layer DyResGEN, are implemented on 1 NVIDIA RTX A6000 (48GB). Five independent repeated runs are implemented in all the results reported in this paper, except 28-layer ResGCN+, which uses three repeated runs. The average value and standard deviation of accuracy are based on these independent runs. We follow [1, 4, 5, 9] to train dense GNN models as baselines. For training GNNs with SLTH, the hyper-parameter configurations are summarized in Table 1 and Table 2.

Table 1: Hyperparameters for Shallow GNNs with SLTH

Dataset	Cora	Citeser	Pubmed	OGBN-Arxiv	OGBG-Molhiv	OGBG-Molbace
GNN	GCN, GAT, GIN				GCN	
Depth	2	2	2	4	3	3
Epochs	400	400	400	400	200	200
Learning Rate	0.01	0.01	0.01	0.01	0.04	0.015
#Hidden neurons	256	256	256	386	448	448
Optimizer	Adam	Adam	Adam	Adam	Adam	Adam
Weight decay	0.0	0.0	0.0	0.0	0.0	0.0
Batch size	-	-	-	-	512	64

Table 2: Hyperparameters for Deep GNNs with SLTH

Dataset	OGBN-Arxiv	OGBG-Molhiv
GNN	ResGCN+	DyResGEN
Depth	SSF, MSF (stage = 4)	SSF
Epochs	500	300
Learning Rate	0.01	0.016
#Hidden neurons	128, 256	256
Optimizer	Adam	Adam
Weight decay	0.001	0.0
Batch size	-	512

For the train-val-test split of the datasets, we follow [9] by using 140 (Cora), 120 (Citeseer), and 60 (PubMed) labeled data for training, 500 nodes for validation and 1,000 nodes for testing. We follow [40] for splitting OGBN-Arxiv dataset. For OGBG-Molhiv, we use 32,901 graphs for training, 4,113 for validation, and 4,113 for testing. For OGBG-Molbace, we use 1,210 graphs for training, 151 for validation, and 152 for testing.

Compared with other pruning methods such as GLT [8] and HGS [7], our proposal does not need to do the fine-tuning or retraining of weights post-pruning, which reduces training time. Specifically, the HGS demands an extra 200 epochs for fine-tuning a shallow GCN on datasets including Cora, Citeseer, and Pubmed after its regular training of 500 epochs.

B Initilization methods for weights

Figure 9 shows the accuracy of GCN, GAT, and GIN on four datasets with different initialization methods. As the figure shows, Singed Kaiming Constant (SC) outperforms Kaiming Normal (KN) when GNNs are applied with S-Sup, and the SC is slightly better than KN when GNNs are used with M-Sup.

C The adaptive Linear threshold and non-adaptive Linear threshold

Experiments evaluate GCN, GAT, and GIN on Cora, Citeseer, and Pubmed datasets with the adaptive Linear threshold ($\alpha = 0.9996$) and the non-adaptive Linear threshold ($\alpha = 1.0$). In Figure 10, results show when the sparsity becomes high, the adaptive Linear threshold outperforms.

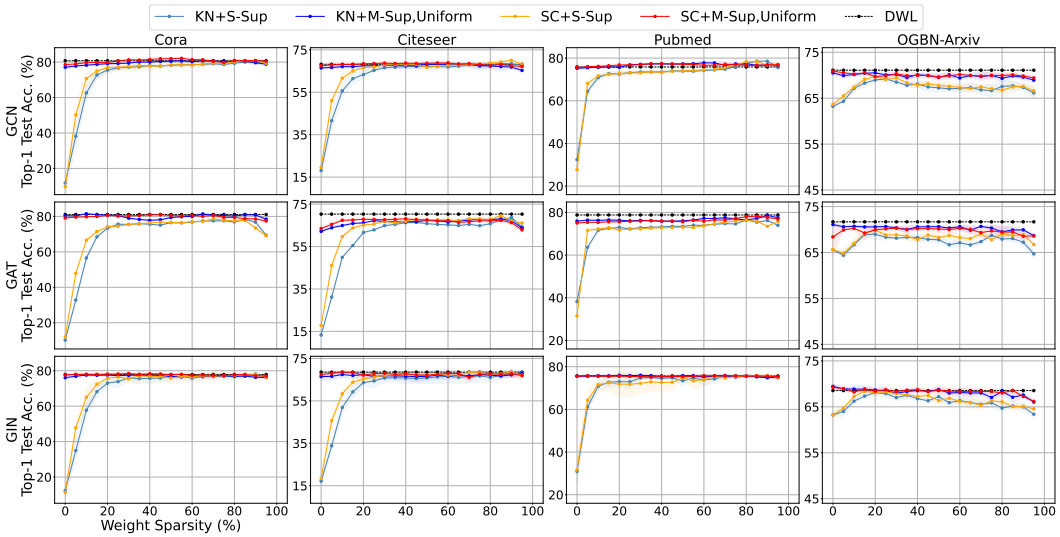


Figure 9: The accuracy of GCN, GAT, and GIN on Cora, Citeseer, Pubmed, and OGBN-Arxiv datasets with SC and KN initialization methods: the models are 256-2-GNN for Cora, Citeseer, and Pubmed, and 386-4-GNN for OGBN-Arxiv with affine batch normalization.

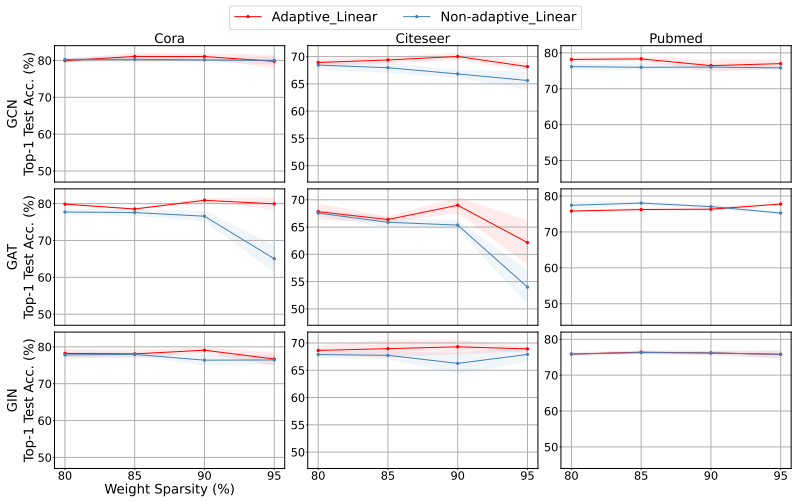


Figure 10: The accuracy of GCN, GAT, and GIN on Cora, Citeseer, Pubmed, and OGBN-Arxiv datasets with different Linear thresholds: the red line is for the adaptive threshold, and the blue line is for the non-adaptive threshold.

D Increasing hidden neurons with S-Sup and M-Sup

Experiments evaluate GCN, GAT, and GIN on Cora, Citeseer, and Pubmed datasets with different hidden neurons. The results are shown in Figure 11. As the number of hidden neurons increases, the accuracy improves. In addition, the M-Sup method still outperforms better than S-Sup.

E Pseudocode for Training

The pseudocode about training weight scores is shown in Algorithm 2.

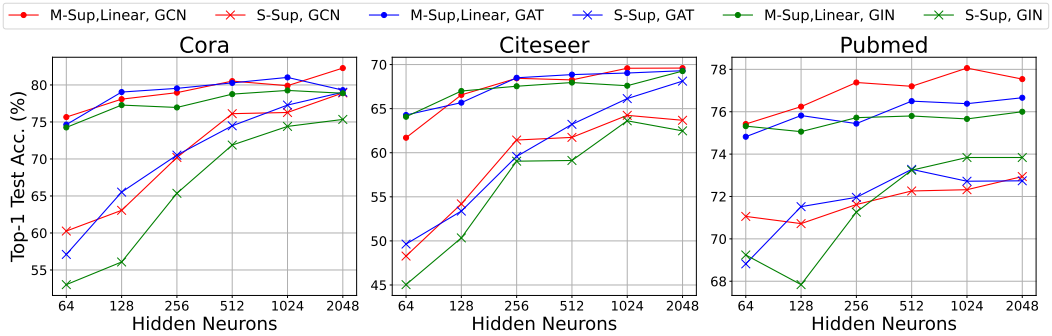


Figure 11: The accuracy of GCN, GAT, and GIN on Cora, Citeseer, and Pubmed datasets with different hidden neurons: experiments set different hidden dimension sizes in 2-layer structures, and the sparsity value is 10%.

Algorithm 2 Multicoated and Folded GNN with SLTH

Input: a GNN $\mathcal{G}\left(\hat{A}, X, W_{\text{rand}}\right)$, learning rate λ, hyperparameters for M-Sup and Folding networks: sparsity list $\mathcal{K} = \{k_1, ..., k_N\}$ with scores $\mathcal{S}$, m-stages, flag_{MSF}, $\text{flag}_{\text{shared}}$.
Output: $\mathcal{G}\left(\hat{A}, X, W_{\text{rand}} \odot \sum_{k \in \mathcal{K}} \mathcal{H}\left(\mathcal{S}, k\right)\right), y$
Randomly initialize weights W_{rand} and Supermasks $\mathcal{S}$ based on m-stages, flag_{MSF}, $\text{flag}_{\text{shared}}$ as Algorithm 1.
for Iter. $t = 1$ to T **do**
  #Get the current sparsity list based on the current epoch, Linear decay schedule as UGT [9]
  if $t < 0.5T$ **then**
    $\mathcal{K}_t = \mathcal{K} \cdot \frac{2t}{T}$
  else
    $\mathcal{K}_t = \mathcal{K}$
  end if
  #Use the global threshold value based on current $\mathcal{K}_t$, and generate the binary masks for weights
  $W = W_{\text{rand}} \odot \sum_{k \in \mathcal{K}_t} \mathcal{H}(\mathcal{S}, k)$
  #Update scores $\mathcal{S}$
  $\mathcal{S} = S - \lambda \bigtriangledown_{\mathcal{S}} \mathcal{L}\left(\mathcal{G}\left(\hat{A}, X, W_{\text{rand}} \odot \sum_{k \in \mathcal{K}_t} \mathcal{H}\left(\mathcal{S}, k\right)\right)\right)$
end for
Return $\mathcal{G}\left(\hat{A}, X, W_{\text{rand}} \odot \sum_{k \in \mathcal{K}} \mathcal{H}\left(\mathcal{S}, k\right)\right), y$

F Small and large sparsity levels for shallow and deep GNNs

To explore the effectiveness in regions of small and large sparsity levels for GNNs, we sampled 1%, 3%, 5%, 7%, and 9% data points at the low sparsity level and 91%, 93%, 95%, 97%, and 99% at the high sparsity level. The Table 3 shows shallow and deep GNNs' performance with M-Sup and S-Sup methods. Moreover, single-stage folding (SSF), multi-stage folding (MSF), Shared supermasks, and Unshared supermasks are also explored. We found that the S-Sup method shows a noticeable accuracy deviation at low sparsity levels compared to the dense weight learning model. For instance, ResGCN+ (SSF, Shared) using the S-Sup method has an accuracy gap of roughly 16%. In contrast, the M-Sup method improves accuracy by 13% when sparsity is close to 1%. Utilizing the MSF method with both S-Sup and M-Sup methods increases accuracy, with improvements ranging from 1% to 4% for ResGCN+. At high sparsity levels (above 90%), the M-Sup and S-Sup methods see marked reductions in accuracy.

G More datasets for DeepGNNs

In addition to OGBN-Arxiv and OGBG-Molhiv, we assessed our proposed methods on three more datasets: Cora, Pubmed, and OGBG-Molbace. Five independent repeated runs are implemented. The

Table 3: Accuracy (%) of models in low (< 10%) and high (> 90%) sparsity levels

Models	Weight Sparsity(%)	1	3	5	7	9	91	93	95	97	99
ResGCN+, OGBN-Arxiv	S-Sup	55.89	56.80	55.85	57.87	58.27	61.87	58.13	57.47	52.91	47.11
SSF, Shared	M-Sup	67.38	67.34	66.64	66.57	66.83	62.32	61.22	57.81	56.65	48.82
ResGCN+, OGBN-Arxiv	S-Sup	58.26	57.61	59.57	59.19	60.81	63.22	61.93	62.17	57.71	51.52
MSF(4), Shared	M-Sup	69.54	68.62	67.73	68.75	67.88	64.30	62.30	61.13	57.64	52.24
DyResGEN, OGBG-Molhiv	S-Sup	74.41	71.15	72.70	73.27	71.90	74.10	75.51	72.39	72.22	70.58
SSF, Shared	M-Sup	78.23	77.19	77.16	76.64	77.22	77.24	75.68	75.19	73.41	70.93
DyResGEN, OGBG-Molhiv	S-Sup	73.10	72.47	71.36	72.57	73.26	77.26	76.51	74.36	75.48	74.05
SSF, Unshared	M-Sup	79.23	78.26	78.71	75.90	77.79	76.77	75.33	76.86	76.49	74.32
3-Layer GCN, OGBG-Molhiv,	S-Sup	71.76	70.47	69.01	69.20	69.64	75.43	75.20	74.90	74.54	70.86
Shared	M-Sup	74.76	76.66	74.05	75.81	74.98	75.69	75.90	75.14	74.34	72.74
3-Layer GCN, OGBG-Molbace,	S-Sup	76.24	75.80	75.28	75.79	74.70	77.40	77.34	75.44	76.08	73.60
Shared	M-Sup	78.33	78.78	77.85	78.51	80.52	79.17	78.15	79.86	73.07	77.76

Note: Here are the accuracy of dense-weight learning models. ResGCN+ on OGBN-Arxiv: 71.92±0.16%, DyResGEN on OGBG-Molhiv: 78.58±1.17%, 3-layer GCN on OGBN-Molhiv: 77.4±1.24%, 3-layer GCN on OGBG-Molbace: 78.3±1.59%.

OGBG-Molbace dataset is about a graph-level task, and the Cora and Pubmed datasets are about node-level tasks. We investigated proposed deepGNN techniques, including M-Sup, Single-Stage Folding (SSF), Multi-Stage Folding (MSF), and both shared and unshared supermasks.

Figures 12 (a) and (b) compare Shared and Unshared supermasks for the Cora and Pubmed datasets, respectively. The results suggest that using Unshared supermasks can enhance accuracy. Figures 12 (c) and (d) contrast SSF with MSF for the same datasets, indicating a performance boost when employing MSF. Lastly, Figure 12 (e) compares S-Sup and M-Sup on the OGBG-Molbace dataset, revealing better performance of M-Sup over S-Sup.

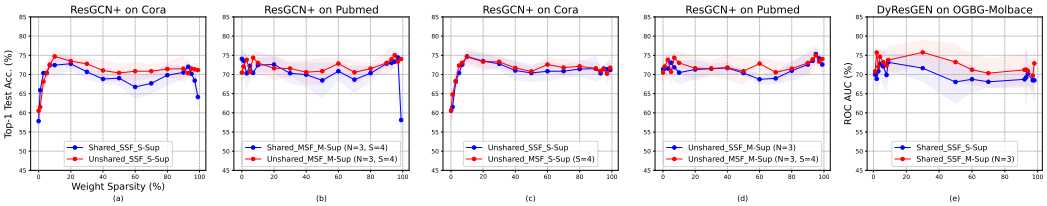


Figure 12: The accuracy of ResGCN+ and DyResGEN on Cora, Pubmed, and OGBG-Molbace datasets: (a) ResGCN+ on Cora with shared and unshared supermasks. (b) ResGCN+ on Pubmed with shared and unshared supermasks. (c) ResGCN+ on Cora dataset with SSF and MSF methods. (d) ResGCN+ on Pubmed with SSF and MSF methods. (e) DyResGEN on OGBG-Molbace with S-Sup and M-Sup methods.

H Exact performance about optimized SLT-GNN models

The Table 4 presents the performance of both shallow and deep GNNs across various datasets. For GNNs employing SLT, M-Sup consistently surpasses S-Sup in accuracy. This is particularly noticeable for shallow GNNs, which achieve accuracy levels similar to DWL models. When integrated with unshared supermasks and the multi-stage folding (MSF) technique, the deep GNNs can also rival the accuracy of DWL models. In terms of memory size, our optimized models boast significant memory reductions. For instance, MSF-Unshared-256 achieves a 72% memory reduction while retaining comparable accuracy. The folding method also contributes to parameter reductions, with the SSF-Shared DyResGEN model achieving a 77.9% reduction. For inference MACs (multiply–accumulate operations), we focus on the linear layer components as they are optimized using SLT. The reduction in MACs corresponds to the sparsity level: for example, GIN can be reduced by 90% and ResGCN+ by 40%.

I Discussion the differences between proposed methods and other quantization methods

Considering we only use un-quantized DWL models and S-Sup models as baselines, we further discuss the differences between our methods and quantization techniques. Quantization techniques can be

Table 4: Exact performance about optimized SLT-GNN models

Models	Configurations	Top-1 Test Acc. / ROC AUC (%)	Memory Size (MB)	Params (Mils)	MACs (Bils)
ResGCN+ on OGBN-Arxiv (for MACs: full batches)	28-layer DWL	71.92±0.16	1.8737	0.4912	717
	Unfolded-Shared-128 (S-Sup, sparsity=60%)	69.28±0.11 (↓ 2.64)	0.0846	0.4874	287
	Unfolded-Shared-128 (M-Sup, N=5, sparsity=50%)	70.52±0.12 (↓ 1.40)	0.1329	0.8929	359
	SSF-Shared-128 (M-Sup, N=5, sparsity=30%)	68.89±0.22 (↓ 3.03)	0.0378	0.0946	502
	MSF-Shared-128 (M-Sup, N=5, sparsity=10%)	69.73±0.25 (↓ 2.19)	0.0546	0.5200	646
	SSF-Unshared-256 (M-Sup, N=5, sparsity=30%)	71.62±0.21 (✓)	0.5678	4.3187	502
	MSF-Unshared-256 (M-Sup, N=5, sparsity=40%)	71.97±0.21 (✓)	0.5174	3.8960	430
DyResGEN on Molhiv (for MACs: batch = 10 graphs)	7-layer DWL	78.58±1.17	2.0293	0.532	3.590
	Unfolded (S-Sup)	77.20±1.04 (↓ 1.38)	0.0693	0.4703	1.795
	SSF-Shared (N=3, sparsity=60%)	78.36±1.06 (✓)	0.0259	0.1066	1.436
	SSF-Unshared (N=3, sparsity=60%)	78.36±0.66 (✓)	0.0964	0.6984	1.436
Shallow GCN on Cora (for MACs: full batches)	3-layer DWL	80.8±0.71	1.4060	0.3686	0.998
	S-Sup (sparsity=55%)	78.5±0.64 (↓ 2.30)	0.0439	0.3686	0.449
	M-Sup (sparsity=55%)	82.1±0.59 (✓)	0.0737	0.6185	0.449
Shallow GAT on Cora (for MACs: full batches)	3-layer DWL	81.1±0.34	1.4080	0.3692	0.999
	S-Sup (sparsity=30%)	75.6±0.41 (↓ 5.50)	0.0440	0.3692	0.699
	M-Sup (sparsity=30%)	81.7±0.55 (✓)	0.0913	0.7654	0.699
Shallow GIN on Cora (for MACs: full batches)	3-layer DWL	77.9±0.72	1.4131	0.3815	1.176
	S-Sup (sparsity=90%)	78.0±0.75 (✓)	0.0455	0.3815	0.118
	M-Sup (sparsity=90%)	79.1±0.62 (✓)	0.0500	0.4197	0.118

Table 5: Comparison of different quantization methods.

Models	**Methods**	**Acc. on Cora (%)**	**Acc. on CiteSeer (%)**	**Acc. on Pubmed (%)**	Memory Reduction
GAT (8 heads, 2 layers)	Vanilla	83.1	72.2	78.8	-
	QAT [41]	81.9(↓ 1.2)	71.2(↓ 1.0)	78.3(↓ 0.5)	75% with 8-bit Activation& Weight
	Degree-Quant [42]	82.7(↓ 0.4)	71.6(↓ 0.6)	78.6(↓ 0.2)	
	GCoD [43]	82.6(↓ 0.5)	71.8(↓ 0.4)	78.8	
GIN (3 layers)	Vanilla	78.6	67.5	78.5	75% with 8-bit Activation& Weight
	QAT [41]	75.6(↓ 3.0)	63.0(↓ 4.5)	77.5(↓ 1.0)	
	Degree-Quant [42]	78.7	67.5	78.1(↓ 0.4)	
	GCoD [43]	78.4(↓ 0.2)	68.7	78.3(↓ 0.2)	
GAT (1 head, 2 layers)	Vanilla	81.1	70.3	78.8	-
	S-Sup	75.6 (↓ 5.50)	67.1 (↓ 3.2)	75.8 (↓ 3.0)	96.8% on Cora, 91.9% on Citeseer, 98.9% on Pubmed
	M-Sup	81.7 (✓)	70.3 (✓)	78.4 (↓ 0.4)	93.5% on Cora, 83.3% on Citeseer, 97.7% on Pubmed
GIN (2 layers)	Vanilla	77.9	68.5	75.6	-
	S-Sup	78.0	68.4(↓ 0.1)	75.7	96.7% on Cora, 91.8% on Citeseer, 98.1% on Pubmed
	M-Sup	79.1 (✓)	69.4 (✓)	76.4(✓)	96.4% on Cora, 91.1% on Citeseer, 98.0% on Pubmed

characterized by the choices between asymmetric and symmetric quantization for clipping range, as well as the decision to use static or dynamic scaling values during iterations. We select typical studies in the field of quantization in GNNs, including QAT [41], Degree-Quant [42], and GCoD [43]. The Table 5 presents two typical shallow GNNs under different quantization methods. Note that for quantization, both weights and activations are quantized into 8-bit fixed points. Compared with these techniques, our method distinguishes them in three perspectives:

1. Our approach integrates both quantization and pruning, with an exclusive emphasis on weights. Rather than designing for hardware's fixed-point processing elements like other quantization methods, we focus on exploring the intrinsic behavior of GNNs, particularly under the strong lottery ticket hypothesis.

2. Our strategy is bit efficient, employing 2 bits for M-Sup (3 supermasks) and 1 bit for S-Sup (a single supermask). This efficiency translates to notable memory savings compared with 8-bit quantization methods. Using bit-serial multipliers [44], our method necessitates only two cycles for multiplication, while an 8-bit approach requires eight cycles. This leads to an empirical improvement in inference time by a factor of four.

3. Regarding accuracy, our method (M-Sup) achieves the comparable accuracy of the FP32 dense-weight learning models in most cases, while for quantization methods, there is still an accuracy gap lower than dense-weight learning models. In the worst-case scenario, QAT on CiteSeer displayed a 4.5% drop.

J Complexity analysis of our proposal and other pruning methods

GNN pruning can be performed on both the adjacency matrix and the weight matrix, such as ULTH-GNN [6] and DGLT [7]. In this work, we focus on the weight matrix as well as UGT [9]. The inference time complexity for these methods is given by $\mathcal{O}(||A_{all}||_0 \times F + ||m_w \odot W||_0 \times |\mathcal{V}| \times F^2)$, where $||A_{all}||_0$ denotes the number of edges, F represents the dimension of the feature, $|\mathcal{V}|$ is the number of nodes, and $||m_w \odot W||_0$ signifies the remaining neurons with supermasks. However, our method could also be extended to pruning adjacency matrices like ULTH-GNN [6] and DGLT [7]. A comparative analysis of our proposal with UGT [9] is presented in Figure 13, where inference MACs

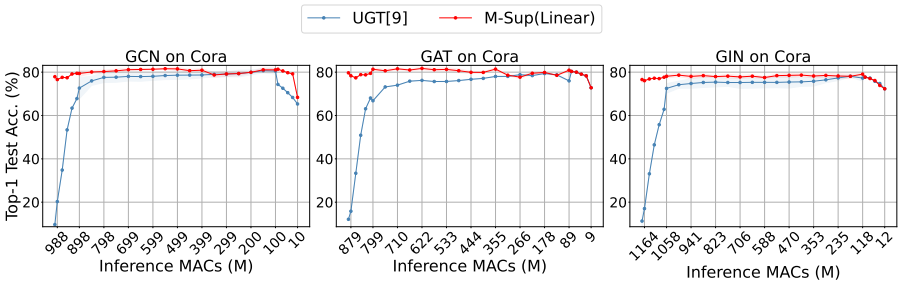


Figure 13: Performance over inference MACs of GCN, GAT, and GIN on Cora.

are about optimized linear layers. Notably, Our method demonstrates better performance under the same inference MACs.