

SGQuant: Squeezing the Last Bit on Graph Neural Networks with Specialized Quantization

Boyuan Feng*, Yuke Wang*, Xu Li, Shu Yang, Xueqiao Peng, and Yufei Ding

University of California, Santa Barbara, USA

{boyuan,yuke_wang,shuyang1995,yufeid@cs.ucsb.edu}

{lixu9906,ameliaapxq0131}@gmail.com

Abstract—With the increasing popularity of graph-based learning, Graph Neural Networks (GNNs) win lots of attention from research and industry field because of their high accuracy. However, existing GNNs suffer from high memory footprints (*e.g.*, node embedding features). This high memory footprint hurdles the potential applications towards memory-constrained devices, such as the widely-deployed IoT devices. To this end, we propose a specialized GNN quantization scheme, SGQuant, to systematically reduce the GNN memory consumption. Specifically, we first propose a GNN-tailored quantization algorithm design and a GNN quantization fine-tuning scheme to reduce memory consumption while maintaining accuracy. Then, we investigate the multi-granularity quantization strategy that operates at different levels (components, graph topology, and layers) of GNN computation. Moreover, we offer an automatic bit-selecting (ABS) to pinpoint the most appropriate quantization bits for the above multi-granularity quantizations. Intensive experiments show that SGQuant can effectively reduce the memory footprint from $4.25 \times$ to $31.9 \times$ compared with the original full-precision GNNs while limiting the accuracy drop to 0.4% on average.

I. INTRODUCTION

Recently, Graph Neural Networks (GNNs) emerge as a new tool to manage various graph-based deep learning tasks (*e.g.*, node classification [1]–[3] and link prediction [4]–[6]). In the comparison with standard methods for graph analytics, such as random walk [7], [8] and graph laplacians [9]–[11], GNNs highlight themselves with significantly higher accuracy [12]–[14] and better generality [15]. In addition, the well-learned GNNs [12]–[15] can be easily applied towards different types of graph structures or dynamic graphs without much re-computing overhead.

However, the GNNs featured with high memory footprint prevent them from being effectively applied towards the vast majority of resource-constrained settings, such as embedded systems and IoT devices, which are essential for many domains. There are two major reasons behind such an awkward situation. First, the input of GNNs consists of two types of inputs, *graph structures* (edge list) and *node features* (embeddings), which would easily lead to a dramatic increase in their storage sizes when the graph becomes large. This will stress the very limited memory budgets of those small devices. Second, the larger size of graphs demands more data operations (*e.g.*, addition and multiplication) and data movements (*e.g.*, memory transactions), which will consume lots of energy and drain the limited power budget on those tiny devices. To tackle these challenges, data quantization can emerge as an “one-stone-two-bird” solution for resource-constrained devices that can 1) effectively reduce the memory

size of both the graph structure and node embeddings, leading to less memory usage; 2) effectively minimize the size of manipulated data, leading to less power consumption.

Nevertheless, an efficient approach for GNN quantization is still missing. Existing approaches may 1) choose a simple yet aggressive uniform quantization to all data to minimize memory and power cost, which leads to high accuracy loss; 2) choose a very conservative quantization to maintain accuracy, which leads to sub-optimal memory and energy-saving performance. While numerous works have been explored for quantization on CNNs [16]–[19], directly applying these existing techniques without considering GNN-specific properties, would easily result in unsatisfactory quantization performance. To address these problems, we believe that three critical questions are noteworthy: 1) what types of data (weight or features) should be quantized? 2) what is the efficient quantization scheme suitable for GNNs? 3) How to determine the quantization bits?

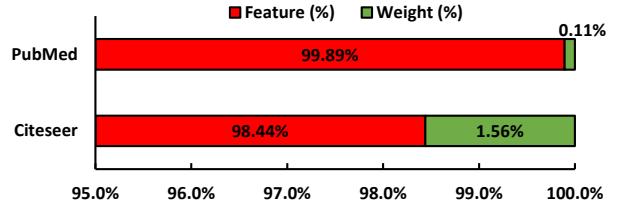


Fig. 1: GAT Feature/Weight Memory Size Ratio.

To answer these questions, we make the following observations: a) quantization on node embedding features is more effective. As shown in Figure 1, the features take up to 99.89% of the overall memory size, which demonstrates their significant memory impact; b) GNNs computing paradigms are different across different layers, different graphs nodes, different components. And these differences could be leveraged as the major “guideline” for enforcing more efficient character-driven quantization.

Based on these observations, we make the following contributions in this paper to systematically quantize GNNs, as illustrated in Figure 2.

- We propose a GNN-tailored quantization algorithm design to reduce memory consumption and a GNN quantization finetuning to maintain the accuracy.
- We propose a multi-granularity quantization featured with *component-wise*, *topology-aware*, and *layer-wise* quantization to meet the diverse data precision demands.
- We propose end-to-end bits selecting in an automatic manner that makes the most appropriate choice for the aforementioned different quantization granularities.

*The first two authors contribute equally.

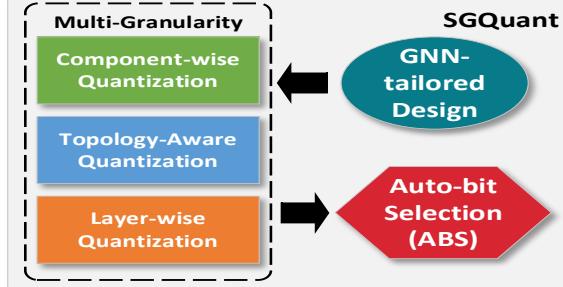


Fig. 2: Overview of SGQuant.

- Rigorous experiments show SGQuant can reduce the memory up to $31.9\times$ (from $4.25\times$) compared with the original full-precision model meanwhile limiting the accuracy to 0.4% on average.

II. BACKGROUNDS AND RELATED WORK

In this section, we will first introduce the basics of Graph Neural Networks (GNNs), and then give some background knowledge of applying data quantization on GNNs.

A. Graph Neural Network

Graph Neural Networks (GNNs) are now becoming the major way of gaining insights from the graph structures. It generally includes several graph convolutional layers, each of which consists of two components: an *Attention Component* and a *Combination Component*, as illustrated in Figure 3. Formally, given two neighboring nodes u and v (*i.e.*, $u \in \mathcal{N}(v)$), and their node embedding $h_u^k \in \mathcal{R}^D$ and $h_v^k \in \mathcal{R}^D$ at layer k , GNNs first use the attention component to measure the relationship between these two nodes:

$$\alpha_{u,v}^k = \text{Attention}(h_u^k, h_v^k, \mathcal{W}_{att}^k | u \in \mathcal{N}(v)) \quad (1)$$

One instantiation of the attention component is a single-layer neural network that concatenates the node embedding h_u^k and h_v^k , and multiplies with the attention weight matrix \mathcal{W}_{att}^k , as the case in GAT [14]. Note that GCN [12] is a special case that has the attention weight matrix with all elements equaling to one. Overall, $\alpha^k \in \mathcal{R}^{N \times N}$ is an *attention matrix* measuring the pairwise relationship between nodes, whose memory consumption increases quadratically as the number of nodes N increases.

Then, GNN computes the node embedding h_v^{k+1} for node v at layer $k+1$ with the combination component:

$$h_v^{k+1} = \text{Combination}(h_v^k, h_u^k, \alpha_{u,v}^k, \mathcal{W}_{com}^{k+1} | u \in \mathcal{N}(v)) \quad (2)$$

One popular instantiation of the combination component is to 1) average over embeddings from neighboring nodes weighted by the attention matrix *alpha*; and 2) multiply the averaged embedding with a combination weight \mathcal{W}_{com}^{k+1} :

$$h_v^{k+1} = \mathcal{W}_{com}^{k+1} \cdot \sum_{u \in \mathcal{N}(v)} \alpha_{u,v}^k h_u^k \quad (3)$$

For each layer k , GNNs have N_k -dimension embedding vector for each node and an *embedding matrix* $h^k \in \mathcal{R}^{N \times N_k}$ when storing all embeddings for N nodes. This embedding matrix will increase linearly with the number of nodes N and introduce heavy memory overhead for large graphs (*e.g.*, Reddit [20] with 232,965 nodes).

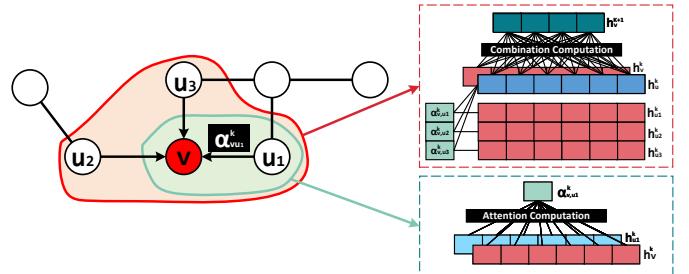


Fig. 3: Example illustrating the standard GNNs. v , u_1 , u_2 , and u_3 stands for four graph vertices and h_v^k refers to the feature vector for node v at the k^{th} layer.

Besides the concepts of *Layer* and *Component*, GNNs also consider the third concept – *Topology*. The topology of GNNs characterizes the graph structure based on the properties of nodes and the edge connections among them.

B. Quantization

Numerous works of quantization mostly focus on data compression around Convolutional Neural Networks (CNNs). Song et al. [16] reduces the size of CNNs without accuracy loss by network pruning, weight quantization, post-quantization network fine-tuning. Ron et al. [17] offers a post-training quantization targeting at weights and activations and minimizes memory consumption at the tensor level. Z. Chen et al. [18] proposes ternary-value based weight quantization to reduce the size of neural networks with minimal accuracy loss. Darryl [19] introduces a layerwise quantizer for fixed-point implementation of DNNs.

Despite such great success in CNNs, efficient GNN quantization is yet to come. And we believe that GNNs have great potential for quantization. There are several reasons, 1) GNN architectures display different levels of computation hierarchy, which allow for more specialized quantization based on their properties (*e.g.*, different quantization strategies for nodes with different degrees, or for layers with different hidden dimensions), which facilitates a fine-grained quantization scheme based on the types of operations, whereas CNNs have fixed shape of all feature maps that pass through the same set of NN layers; 2) GNNs are more diverse in their inputs, such as graph topologies (*e.g.*, edge connections) and node features (embeddings), which enable quantization based on the categories of the data. Overall, we are the first to systematically and comprehensively explore the quantization on GNNs by exploiting graph properties and GNN architecture.

III. GNN-TAILORED QUANTIZATION

In this section, we introduce our GNN-tailored quantization to convert a full-precision GNN to a quantized GNN with reduced memory consumption.

A. Quantization Algorithm Design

Two key designs differentiate our GNN quantization from existing work on CNN quantizations. First, SGQuant quantizes both the attention matrix $\alpha^k \in \mathcal{R}^{N \times N}$ and the embedding matrix $h^k \in \mathcal{R}^{N \times D_k}$, while the CNN quantization generally only considers feature quantization due to the intrinsic model difference between GNNs and CNNs. Second, when assigning different quantization bits for the attention matrix

and the embedding matrix, SGQuant contains a “rematching” mechanism that matches their quantization bits and enables the computation in Equation 3.

Formally, given a quantization bit q and the 32-bit attention matrix $\alpha^k \in \mathcal{R}^{N \times N}$ computed from Equation 1, we quantize it as a q-bit attention matrix

$$\alpha^{k,(q)} = \left\lfloor \frac{\alpha^k - \alpha_{min}}{scale} \right\rfloor. \quad (4)$$

where α_{min} is an empirical lower bound of the attention matrix values, $scale$ is the ratio between the attention matrix range and the q-bit representation range, and $\lfloor \cdot \rfloor$ is the floor function. Specifically, we evaluate the GNN on large graph benchmarks and collect the statistics on its attention matrices, including the minimal value α_{min} , the maximum value α_{max} . Then, we can compute a 32-bit $scale$ parameter on the ratio as the feature range $\alpha_{max} - \alpha_{min}$ over the q-bit representation range 2^q . While Equation 1 generates a 32-bit attention matrix $\alpha^k \in \mathcal{R}^{N \times N}$ requiring $32 \times N \times N$ -bit memory space, our quantized q-bit attention matrix requires only $q \times N \times N$ -bit memory space. In this way, our quantization on the attention matrix reduces the memory consumption to $q/32$ of its full-precision version. In particular, once we have computed the 32-bit attention value $\alpha_{u,v}^k$, we can immediately quantize it into a q-bit value and store it in the memory. Similarly, given a quantization bit p and the 32-bit embedding matrix $h^k \in \mathcal{R}^{N \times D_k}$ from Equation 3, we can generate a p -bit embedding matrix $h^{k,(p)}$ that reduces the memory consumption to $p/32$ of its full-precision version.

Suppose we assign different quantization bits p and q to the attention matrix and the embedding matrix, there would be an “unmatching bit” problem in Equation 3 that the attention value $\alpha_{u,v}^{k,(q)}$ and the embedding $h_u^{k,(p)}$ have unmatching bits. To solve this problem, we propose a “rematching” mechanism that recovers the quantized value to 32-bit value before these values enter the combination component. Specifically, we compute the recovered 32-bit attention as $\alpha_{u,v}^{k,(q)'} = scale \cdot \alpha_{u,v}^{k,(q)} + \alpha_{min}$. Similarly, we can compute the recovered 32-bit embedding $h_v^{k,(p)'}$. Feeding these recovered values into the combination component, we can compute the Equation 3 as

$$h_v^{k+1} = \mathcal{W}_{com}^{k+1} \cdot \sum_{u \in \mathcal{N}(v)} \alpha_{u,v}^{k,(q)'} h_u^{k,(p)'} \quad (5)$$

Note that the “rematching” mechanism introduces negligible memory overhead since, when we compute the node embedding h_v^{k+1} , we only recover a small set of nodes u that have edge connections with the node v . In addition, the \mathcal{W}_{com}^{k+1} here is a 32-bit value since SGQuant only quantizes the GNN features as discussed in Figure 1. Similarly, we can compute the attention component at layer k as

$$\alpha_{u,v}^k = \text{Attention}(h_u^{k,(p)'}, h_v^{k,(p)'}, \mathcal{W}_{att}^k | u \in \mathcal{N}(v)) \quad (6)$$

where the $\alpha_{u,v}^k$ is a 32-bit value and can be quantized into q -bit values with Equation 4. Due to this similarity, we will only discuss quantizing the combination component in the following sections.

B. GNN Quantization Finetuning

One challenge in GNN quantization is that directly applying the quantization to GNNs during inference usually leads to

high accuracy loss up to 10%. This accuracy loss can be largely recovered to less than 0.5% when we finetune the quantized GNNs. Note that this finetuning procedure only needs to be conducted once for a quantized GNN model. Overall, SGQuant uses the same loss as the original GNN model (e.g., negative log-likelihood (NLL) for semi-supervised node classification task). On the backpropagation related to GNN quantization, we derive the gradient as follows

$$\begin{aligned} \frac{\partial L}{\partial \alpha_{u,v}^{k,(q)'}} &= \mathcal{W}_{com}^{k+1} \cdot \left(\frac{\partial L}{\partial h_v^{k+1}} \cdot h_u^{k,(p)'} + \frac{\partial L}{\partial h_u^{k+1}} \cdot h_v^{k,(p)'} \right) \\ \frac{\partial L}{\partial \alpha_{u,v}^k} &= \frac{\partial L}{\partial \alpha_{u,v}^{k,(q)'}} \cdot scale \cdot \frac{\partial \alpha_{u,v}^{k,(q)}}{\partial \alpha_{u,v}^k} \end{aligned} \quad (7)$$

Note that the computation of $\alpha^{k,(q)}$ uses a floor function $\lfloor \cdot \rfloor$, whose gradient is zero almost-everywhere and hinders the backpropagation. Our SGQuant uses the straight-through estimator that assigns the gradient $\frac{\partial \alpha_{u,v}^{k,(q)}}{\partial \alpha_{u,v}^k}$ to be $1/scale$. To this end, we can rewrite the gradient $\frac{\partial L}{\partial \alpha_{u,v}^k}$ in Equation 7 as

$$\begin{aligned} \frac{\partial L}{\partial \alpha_{u,v}^k} &= \frac{\partial L}{\partial \alpha_{u,v}^{k,(q)'}} \cdot scale \cdot \frac{\partial \alpha_{u,v}^{k,(q)}}{\partial \alpha_{u,v}^k} \\ &= \frac{\partial L}{\partial \alpha_{u,v}^{k,(q)'}} \end{aligned} \quad (8)$$

We implement a tailored GNN quantization layer in PyTorch-Geometric [21] that enables both the quantized inference and the backpropagation, such that SGQuant can easily conduct end-to-end finetuning.

IV. MULTI-GRANULARITY QUANTIZATION

When designing our specialized graph quantization (SGQuant) method, the quantization granularity is an important aspect to be considered. In this section, we propose four different types of granularity: *component-wise*, *topology-aware*, *layer-wise*, and *uniform*, as illustrated in Figure 4. The simplest granularity is the uniform quantization, which applies the same quantization bits to all layers and components in the GNN. It helps reduce the memory consumption by replacing the 32-bit values with the corresponding q -bit quantized data representation. However, when applying the same quantization bit to all layers, nodes, and components, we ignore their different sensitivity to quantization bits and the introduced numerical error, leading to degraded accuracy. To this end, we need the quantization at finer granularity to cater the different sensitivity.

A. Component-wise Quantization

Component-wise Quantization (CWQ) considers the quantization sensitivity at each GNN component and applies different quantization bit to different components, as illustrated in Figure 4(a). In each layer, modern GNNs usually contain the attention component for measuring the relationship for each pair of nodes, and the combination component for computing the embedding h_v^{k+1} for the next layer. While the combination component is critical for providing fine-grained features for the next GNN layer, the attention component usually only provides a coarse-grained hint on the importance of one node u to another node v . Our key insight is that *the attention component is more robust to the numerical error in the GNN quantization compared to the combination component*. Thus,

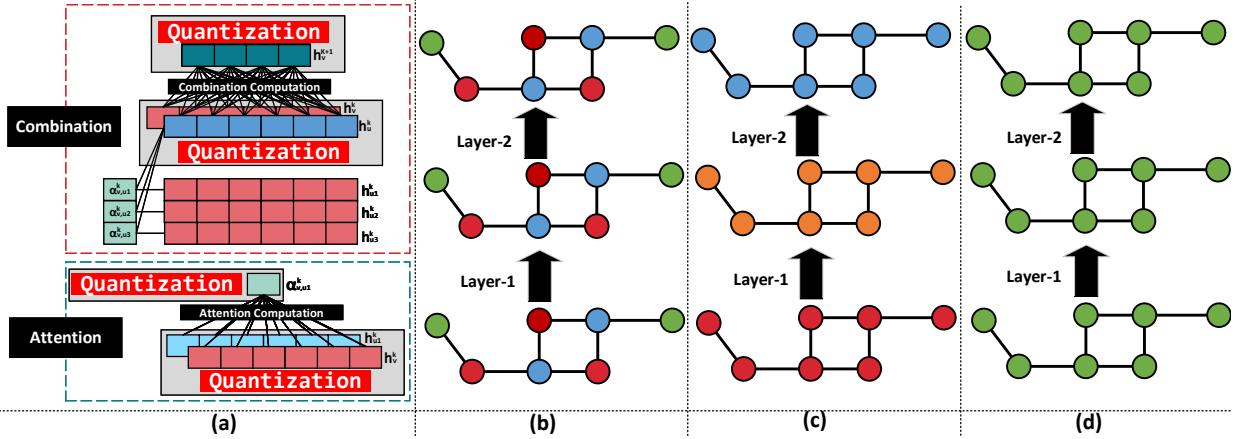


Fig. 4: Multi-Granularity Quantization: (a) Component-wise, (b) Topology-aware, (c) Layer-wise, and (d) Uniform Quantization.
NOTE: the same color represents the same quantization bit.

we can usually apply a lower quantization bit on the attention component than the combination component.

Formally, CWQ maintains a quantization configuration

$$\{att : q_{att}, com : q_{com}\} \quad (9)$$

for the quantization bits of each GNN component (*i.e.*, att for the aggregation component and com for the combination component), where q_{att} and q_{com} are the quantization bits for the attention and the combination component, respectively. During the quantization, we will check the quantization bits for each component and conduct quantization correspondingly. Formally, we compute the quantized attention matrix $\alpha^{k,(q_{att})}$ and the quantized embedding $h^{k,(q_{com})}$, as described in Equation 4. Note that the *scale* parameter in Equation 4 varies for components according to the assigned quantization bit q_{att} and q_{com} . While CWQ may lead to multiplying two values with “unmatching” bits during the combination component, we can resolve with the “rematching” mechanism in Equation 5. In particular, during combination, we first recover the quantized component values $\alpha_{u,v}^{k,(q_{att})}$ and $h_u^{k,(q_{com})}$ to their corresponding 32-bit representation $\alpha_{u,v}^{k,(q_{att})'}$ and $h_u^{k,(q_{com})'}$, then compute with 32-bit values

$$h_v^{k+1} = W_{com}^{k+1} \cdot \sum_{u \in \mathcal{N}(v)} \alpha_{u,v}^{k,(q_{att})'} h_u^{k,(q_{com})'} \quad (10)$$

B. Topology-aware Quantization

Topology-aware Quantization (TAQ) exploits the graph topology information and applies different quantization bits for different nodes based on their most essential topology property – degree, as illustrated in Figure 4(b). In GNN computation, nodes with higher degrees usually have more abundant information from their neighboring nodes, which makes them more robust to low quantization bits since the random error from quantization can usually be averaged to 0 with a large number of aggregation operation. In particular, given a quantization bit q , the quantization error $Error_u$ of each node u is a random variable and follows the uniform distribution $Error_u \sim \mathcal{U}(-\frac{\text{range}}{2^q}, \frac{\text{range}}{2^q})$ where range represents the difference between the maximum embedding value and the minimum embedding value. For a node with a large degree, we will aggregate a large number of $Error_u$ and

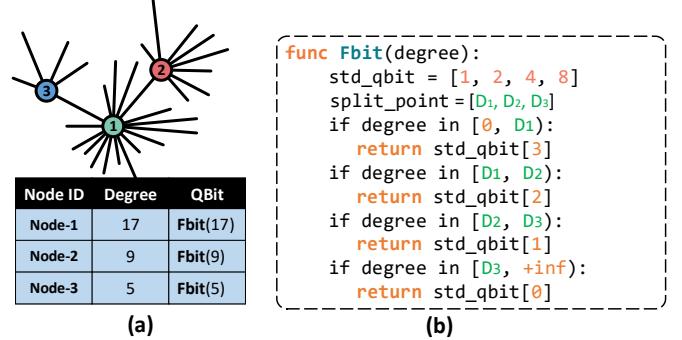


Fig. 5: Topology-aware Quantization.

$Error_u$ from the node u and its neighboring nodes v and the average results will converge to 0 following the *law of large numbers* [22]. To this end, nodes with a large degree are more robust to the quantization error and we can use smaller quantization bits to these high-degree nodes.

Formally, TAQ maintains a quantization configuration that are selected according to the node degrees

$$\{[D_j, D_{j+1}) : q_{D_j} | j \in [0, 1, 2, 3]\} \quad (11)$$

where the features of a node are assigned the quantization bit q_{D_j} if the node degree falls into $[D_j, D_{j+1})$. Here, we set $D_0 = 0$ and $D_4 = +\infty$, as illustrated in Figure 5. Suppose there are three nodes: *node-1*, *node-2* and *node-3*, which have the node degree 17, 9, and 5 respectively. Our TAQ determines the quantization bits of each node based on their degrees. To get the appropriate quantization bit for different nodes, we propose and implement a *Fbit* function, as illustrated in Figure 5(b). We first create the most commonly used quantization bits as a template list (std_qbit), and pre-define the degree *split_point* list $[D_1, D_2, D_3]$. *Fbit* function maps the nodes with corresponding quantization bits based on their degrees. The strategy behind such a mapping is to maintain higher quantization bits for low-degree nodes, while penalizing high-degree nodes with low bit quantization.

Once we have assigned different quantization bits q_u to different nodes u , there are still “unmatching” bits problem across nodes, similar to the “unmatching” problem across components. We can use the “rematching” technique on node

embeddings and compute the combination component as

$$h_v^{k+1} = \mathcal{W}_{com}^{k+1} \cdot \sum_{u \in \mathcal{N}(v)} \alpha_{u,v}^k h_u^{k,(q_u)'} \quad (12)$$

where $\alpha_{u,v}^k$ is a 32-bit value. TAQ does not quantize the attention matrix since we consider only the first-order topology information and skip second-order topology information that, for an edge uv , two nodes u and v have different degrees.

C. Layer-wise Quantization

The Layer-wise Quantization (LWQ) exploits the diverse quantization sensitivity in individual GNN layers and provides different quantization bits to each layer. Our key motivation is that leading layers usually take the detailed data and capture the low-level features while the succeeding layers usually abstract these low-level details into high-level features. To this end, leading layers require large quantization bits to represent the low-level details while the succeeding layers need only small quantization bits for storing the high-level features. Our evaluation empirically confirms that, under the same memory consumption, assigning higher bits to the leading layers generally leads to higher accuracy, compared to assigning higher bits to the succeeding layers.

Formally, LWQ maintains a quantization configuration

$$\{k : q_k \mid k \in [1, 2, \dots, n]\} \quad (13)$$

where q_k is the quantization bit at the layer k and n is the number of GNN layers. In particular, GNN quantized with LWQ has the same quantization bits q_k for both the attention matrix α^k and the embedding matrix h^k at the layer k and computes the combination component h_v^{k+1} at $k + 1$ as

$$h_v^{k+1} = \mathcal{W}_{com}^{k+1} \cdot \sum_{u \in \mathcal{N}(v)} \alpha_{u,v}^{k,(q_k)'} h_u^{k,(q_k)'} \quad (14)$$

D. Combine Multiple Granularities

Besides applying the above granularities stand-alone, SGQuant can effectively combine them in collaborative ways. And we detail two major types of combinations as follows.

a) **LWQ+CWQ**: Note that LWQ and CWQ are complementary and can be easily combined to provide more fine-grained quantization configuration

$$\{(k, att):q_k, att, (k, com):q_k, com \mid k \in [1, 2, \dots, n]\} \quad (15)$$

Our evaluation shows that $LWQ+CWQ$ can provide lower quantization bits at the same accuracy, compared to only applying LWQ or CWQ alone. The main insight is that $LWQ+CWQ$ provides more fine-grained granularities and could potentially generate models with higher accuracy under the same memory budget. Formally, GNN quantized with $LWQ+CWQ$ computes the combination component as

$$h_v^{k+1} = \mathcal{W}_{com}^{k+1} \cdot \sum_{u \in \mathcal{N}(v)} \alpha_{u,v}^{k,(q_k, att)'} h_u^{k,(q_k, com)'} \quad (16)$$

We can similarly use $LWQ+TAQ$ and $CWQ+TAQ$. We omit the details of these two combinations here due to page limits.

b) **LWQ+TAQ+CWQ**: We can also combine TAQ, LWQ, and CWQ to generate quantization configuration

$$\{(k, att):q_k, att, (k, com, [D_j, D_{j+1}]):q_k, com, D_j \mid k \in [1, 2, \dots, n], j \in [1, \dots, 4]\} \quad (17)$$

Note that the quantization bits $q_{k,att}$ on the attention matrix does not depend on the topology information, as the case in TAQ. Formally, GNN with $LWQ+TAQ+CWQ$ computes as

$$h_v^{k+1} = \mathcal{W}_{com}^{k+1} \cdot \sum_{u \in \mathcal{N}(v)} \alpha_{u,v}^{k,(q_k, att)'} h_u^{k,(q_k, com, D_j)'} \quad (18)$$

where D_j is decided by the degree of node u .

V. AUTO-BIT SELECTION

Given the rich set of quantization granularities, one natural question arises: *How can we assign quantization bits for different granularities to achieve the sweet point between accuracy and memory saving?* Essentially, we need to solve a combinatorial optimization problem that minimizes the end-to-end loss by selecting a group of discrete quantization bits. Suppose we are considering $LWQ+TAQ+CWQ$, we can formalize the combinatorial optimization problem as

$$\min_{\substack{q_k, att, \\ q_k, com, D_j}} Loss(\alpha_{u,v}^{k,(q_k, att)}, h_u^{k,(q_k, com, D_j)'}, \mathcal{W}_{com}^k, \mathcal{W}_{att}^k) \quad (19)$$

where $Loss(\cdot, \cdot, \cdot)$ is typically cross-entropy for classification tasks and L_2 norm for regression tasks, q_k, att is the quantization bits for attention matrix at layer k , q_k, com, D_j is the quantization bits for the node embedding at layer k for node u . We also include the weight matrices \mathcal{W}_{com}^k and \mathcal{W}_{att}^k in the loss function, since we conduct end-to-end finetuning for the quantized GNN, as discussed in Section III-B.

There are three challenges in solving this combinatorial optimization problem. First, there is a large design space due to the abundant quantization granularity. When we apply LWQ, CWQ, and TAQ simultaneously, the number of possible quantization configurations increases exponentially, leading to huge manual efforts in exploration. Second, large diversity exists in the GNN model design in terms of the attention generation in the aggregation components and the neural network design in the combination components. This diversity makes it hard to analytically compute the end-to-end quantization error and the impact of quantization bits towards the GNN predictions. Third, graph topology usually varies in terms of the number of nodes and the degree distribution, making the measurement of quantization intractable. As we have discussed in Section IV-B, this topology information usually has a high impact on the quantization error, requiring the consideration of both the graph topology and the GNN design during the selection of quantization bits.

To address these challenges, we build an auto-bit selection (ABS) with two main components: a *machine learning cost model* that predicts the accuracy of the quantized GNN under a given quantization configuration, and an *exploration scheme* to select the promising configurations.

A. Machine Learning Cost Model

Before we dive into our machine learning (ML) cost model, we will first discuss two baseline approaches. The first one is the random search with trial-and-error that randomly samples a large number of quantization configurations and examines all samples to find the best one. However, this approach usually requires a large number of samples to find a good configuration. The second is to build a pre-defined cost model to analyze the impact of quantization bits over the predictions

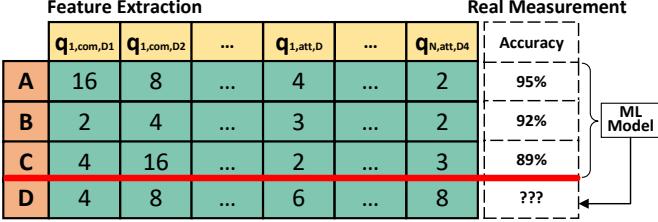


Fig. 6: Machine Learning Cost Model.

for a particular GNN model and a graph topology. However, this approach usually fails to generalize well to various GNN models and graph inputs.

To this end, we build a ML cost model that learns on the fly the interaction among *quantization bits*, *GNN models*, and the *graph topology*. Figure 6 illustrates our ML cost model design. Given the quantization granularity and the bits to select, we can randomly generate a set of configurations and extract the quantization bits as the features. Then, we will train and evaluate these configurations, and measure the accuracy as true labels. Finally, we will use the collected features and labels to train our ML cost model and use it to predict the remaining configurations. We treat this task as a regression problem and use a traditional ML model — regression tree [23], as our ML cost model. We prefer the regression tree over neural networks since the former one has faster inference speed and does not require a large amount of training data.

B. Exploration Scheme

Given the ML cost model, a simple exploration scheme would evaluate all remaining quantization configurations and select the one with the highest predicted accuracy and the lowest memory size. However, this approach may fail in two cases. First, it is time-consuming to evaluate all remaining quantization configurations, especially when we use LWQ, CWQ, and TAQ simultaneously for a large GNN. Second, we may select a small number of quantization configurations for training the ML cost models to reduce the overhead from auto-bit selection, such that the trained ML model cannot predict precise accuracies for all remaining quantization configurations.

To this end, we propose an exploration scheme that iteratively trains the ML cost model and selects promising configurations. In this way, we can balance the low overhead in training the ML cost model and the precise prediction of configuration accuracies. In particular, there are five steps in our exploration scheme.

- **Step1:** Randomly select a small number N_{mea} of configurations, extract features, and measure their accuracies.
- **Step2:** Train the ML cost model based on the collected features and labels.
- **Step3:** Sample a large number N_{sample} of configurations, use the ML cost model to predict their accuracy, and find the ones with the top- N_{mea} accuracy.
- **Step4:** Extract features of the selected configurations and measure their accuracies.
- **Step5:** Repeat **Step2 - Step4** until reaching N_{iter} iterations.

During this procedure, only configurations with negligible accuracy drop ($< 0.5\%$) will be kept. Among the remaining configurations, we select the one with the lowest memory

TABLE I: GNN Architectures.

Arch	Specification
GCN	hidden=32, #layers=2
AGNN	hidden=16, #layers=4
GAT	hidden=256, #layers=2

TABLE II: Datasets for Evaluation.

Dataset	#Vertex	#Edge	#Dim	#Class
Citeseer	3,327	9,464	3,703	6
Cora	2,708	10,858	1,433	7
Pubmed	19,717	88,676	500	3
Amazon-computer	13,381	245,778	767	10
Reddit	232,965	114,615,892	602	41

consumption. Here, N_{mea} , N_{iter} , and N_{sample} are hyper-parameters in ABS that balance the selection overhead and the ML cost model accuracy. Smaller N_{measre} and N_{iter} lead to lower selection overhead by reducing the number of quantization configurations that are trained and evaluated. We have experimented with diverse N_{mea} and N_{iter} on an extensive collection of GNNs and datasets, and find that a small $N_{measure} = 40$ and a small $N_{iter} = 5$ hit the balance between selection overhead and the ML cost model accuracy. The reason is that our cost model is a traditional regression tree model that can be trained with a small amount of data. Using a larger N_{sample} , we can generally select configurations with lower memory consumption and higher accuracy, since more configurations are evaluated by our ML cost model. By default, we set $N_{sample} = 2000$ in our evaluation. This leads to negligible latency (< 0.1 seconds) at each iteration due to the fast inference speed from the regression tree.

VI. EVALUATION

In this section, we show the strength of our proposed quantization method through intensive experiments over various GNN models and datasets.

A. Experiment Setup

1) **GNN Architectures: Graph Convolutional Network (GCN)** [12] is the most basic and popular GNN architecture. It has been widely adopted in node classification, graph classification, and link prediction tasks. Besides, it is also the key backbone network for many other GNNs, such as GraphSage [15], and Diffpool [24].

Attention-based Graph Neural Network (AGNN) [13] aims to reduce the parameter size and computation by replacing the fully connected layer with specialized propagation layers. **Graph Attention Network (GAT)** [14] is a reference architecture for many other advanced GNNs with more edge properties, which can provide state-of-the-art accuracy performance on many GNN tasks. Details of their configurations are shown in Table I.

2) **Datasets:** We select two categories of graph datasets to cover the vast majority of the GNN inputs. The first type includes the most typical datasets (*Citeseer*, *Cora*, and *Pubmed*) used by many GNN papers [12], [13], [15]. They are usually small in the number of nodes and edges, but come with high-dimensional feature embedding. The second type (*Amazon-computer*, and *Reddit*) are large graphs [12], [20] in

TABLE III: Overall Quantization Performance.

Dataset	Network	Accuracy (%)	Average Bits	Memory Size (MB)	Saving
Cora	GCN (Full-Precision)	82.2	32	15.42	-
	GCN (Reduced-Precision)	81.72	1.22	0.59	26.1 \times
	AGNN (Full-Precision)	83.16	32	15.94	-
	AGNN (Reduced-Precision)	82.75	2.15	1.07	14.90 \times
	GAT (Full-Precision)	82.50	32	16.21	-
	GAT (Reduced-Precision)	82.10	2.58	1.31	12.37 \times
Citeseer	GCN (Full-Precision)	71.82	32	51.06	-
	GCN (Reduced-Precision)	71.54	1.01	1.6	31.9 \times
	AGNN (Full-Precision)	71.58	32	50.01	-
	AGNN (Reduced-Precision)	71.18	1.08	1.69	29.59 \times
	GAT (Full-Precision)	71.10	32	59.49	-
	GAT (Reduced-Precision)	70.70	2.42	3.82	13.2 \times
Pubmed	GCN (Full-Precision)	80.36	32	43.71	-
	GCN (Reduced-Precision)	80.28	2.9	4.01	10.9 \times
	AGNN (Full-Precision)	80.44	32	43.46	-
	AGNN (Reduced-Precision)	80.31	3.07	4.17	10.42 \times
	GAT (Full-Precision)	78.00	32	44.48	-
	GAT (Reduced-Precision)	77.30	3.77	5.26	8.47 \times
Reddit	GCN (Full-Precision)	81.07	32	328.70	-
	GCN (Reduced-Precision)	80.36	3.72	38.25	8.59 \times
	AGNN (Full-Precision)	74.63	32	643.92	-
	AGNN (Reduced-Precision)	74.40	4	113.92	5.65 \times
	GAT (Full-Precision)	92.66	32	311.85	-
	GAT (Reduced-Precision)	92.23	4.07	39.70	7.86 \times
Amazon-Computer	GCN (Full-Precision)	89.57	32	44.58	-
	GCN (Reduced-Precision)	89.39	3.29	4.59	9.72 \times
	AGNN (Full-Precision)	77.69	32	44.16	-
	AGNN (Reduced-Precision)	77.33	4	5.99	7.37 \times
	GAT (Full-Precision)	93.10	32	45.71	-
	GAT (Reduced-Precision)	92.60	7.53	10.75	4.25 \times

the number of nodes and edges. Details of the above datasets are listed in Table II.

B. Overall Performance

In this section, we demonstrate the benefits of SGQuant by evaluating its impact of accuracy loss and memory saving. As shown in Table III, our specialized quantization method can effectively reduce the memory consumption up to 31.9 \times , 29.59 \times , and 13.2 \times on GCN, AGNN, and GAT respectively, meanwhile limiting the accuracy loss by 0.34%, 0.31%, 0.47% on average compared with the original full-precision model for GCN, AGNN, and GAT.

Moreover, there are several noteworthy observations. **Across different datasets:** On datasets with smaller sizes, such as *Cora* and *Citeseer*, our specialized quantization method can reduce the memory size more aggressively while maintaining accuracy by selecting relatively low average bits, such as 1.22 for GCN on Cora. This is because the smaller datasets with limited size of nodes and edge connections make the quantization precision loss less significant. **Across different models:** we find that to maintain the accuracy, SGQuant would select higher average bits for more complex models. For example, on *Amazon-computer* dataset, GAT model locates 7.53 as the average bit, while the AGNN and GCN locate 4 bit and 3.29 bit, respectively. We observe similar pattern on all other datasets that we evaluated. The major reason is that more complex GNN models would involve more intricate computations that would easily enlarge the accuracy loss of quantization and require higher bits to offset such loss. For instance, GAT has to first compute neighbor-specific attention values and scale them with the number of attention heads

before the combination component. Instead, AGNN and GCN have simpler combination component that requires much less effort in computation, getting its loss of quantization well under-controlled even with lower bits.

What also worth mentioning is that on datasets with large size, such as *Reddit*, the absolute memory size saving is significant, which reduces up to 530MB memory occupation. This can also demonstrate the potential of SGQuant to make the GNNs happen on memory-constrained device more easily.

C. Breakdown Analysis of Multi-granularity Quantization

In this experiment, we break down the benefits of multi-granularity quantization. Specifically, we apply GAT on Cora. We first evaluate the performance of uniform quantization (Uniform) and layer-wise quantization (LWQ). Then, we evaluate more fine-grained granularity by combining LWQ with component-wise quantization (CWQ) and apply different quantization bits to individual components at each layer. For example, for GAT with 2 layers and 2 components of aggregation and combination at each layer, the quantization configuration with LWQ+CWQ has 4 quantization bits (*i.e.*, $q_{1,agg}, q_{1,com}, q_{2,agg}, q_{2,com}$). We additionally impose the topology-aware quantization (TAQ) to study the performance of SGQuant when considering LWQ, CWQ, and TAQ, simultaneously.

Figure 7 shows the error rate of each quantization granularity at each memory size. Specifically, Uniform shows the highest error rate under each memory size. This error rate increases significantly when we compress the model to be smaller a certain size (2.5MB). Compared with Uniform, LWQ achieves lower error rate, due to the flexibility in selecting

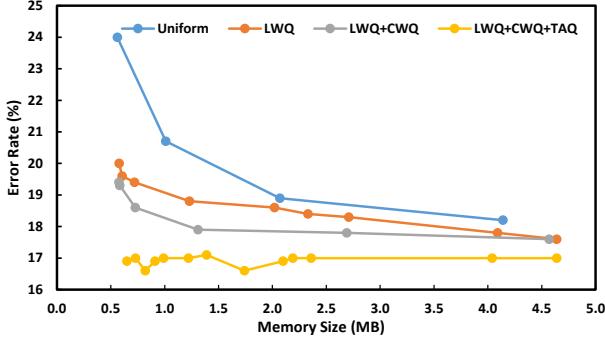


Fig. 7: Breakdown Analysis of Multi-granularity Quantization.

different bits for different layers. Moreover, we observe that *LWQ+CWQ* further mitigates such accuracy degradation when reducing the model memory footprint aggressively. The reason is that *LWQ+CWQ* takes the properties of different layers and different components in to consideration, which can strike a good balance between the memory saving and the accuracy. Finally, this experiment also shows that, by incorporating the node information (degree) with *LWQ+CWQ+TAQ*, our SGQuant can achieve even lower error rate at each memory size. The major reason is that high-degree nodes would intrinsically gather more information from its neighbors compared with the nodes with limited number of neighbors. In other words, applying more aggressive quantization on high-degree nodes would cause minor information loss.

TABLE IV: Optimal Quantization Bit of GAT on Cora.

Quantization Method	Configuration@Mem-Size=2MB	Error Rate
Uniform	$q = 4$	18.90%
LWQ	$q_1 = 4, q_2 = 1$	18.60%
LWQ + CWQ	$q_{1,att} = 2, q_{1,com} = 4$ $q_{2,att} = 2, q_{2,com} = 2$	17.90%
LWQ + CWQ + TAQ	$q_{1,Dj} = [4, 3, 2, 1]$ $q_{2,Dj} = [1, 1, 1, 1]$	16.70%

As a case study, Table IV shows the allocated bit-width and error rate of GAT on Cora with different granularity, with the memory size around 2MB. We observe similar trend as Figure 7 that fine-grained granularities generally lead to lower error rate at a given memory size. One interesting observation is that LWQ achieves a lower error rate than Uniform, while LWQ chooses lower quantization bit than Uniform at layer 2. The insight is that the low quantization bit may introduce the regularization effect and prevent overfitting in the training procedure. Also, LWQ usually assigns higher bits to leading layers, as discussed in Section IV-C. For the *LWQ+CWQ*, we assign smaller quantization bits to the attention component, since attention component is more robust to the numerical error in the GNN quantization, as discussed in Section IV-A. The most fine-grained granularity is *LWQ+CWQ+TAQ*, where we can reduce error rate by 2.2% under the same memory size, compared with the uniform quantization.

D. Effectiveness of Auto-Bit Selection

In this experiment, we evaluate auto-bit selection (ABS) with the machine learning (ML) cost model. As discussed in Section V, we iteratively select and evaluate quantization

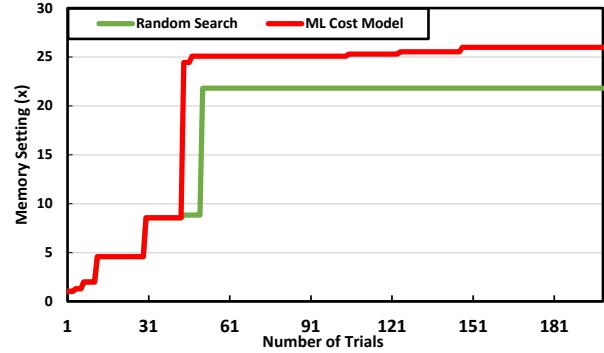


Fig. 8: Benefit of ML Cost Model.

configurations. Among these evaluated quantization configurations, we only select configurations that shows negligible accuracy drop (< 0.5%) compared to the full-precision GNN. Among remaining models, we exhibit its memory saving compared to the full-precision GNN. We compare our ABS with the random search approach, which randomly picks 200 quantization configurations and selects the one with lowest memory size while also showing negligible accuracy drop.

Figure 8 exhibits the results on AGNN and Cora dataset, while similar trend can be observed on other GNNs and datasets. Overall, our ML cost model converges within 200 trials of quantization configurations and achieves two advantages over the random search approach. First, ML cost model can locate the appropriate quantization bits more swiftly compared with naive random search solution. Second, for the final results, ML cost model can pinpoint a more "optimal" value for bits that offers higher memory saving ($25\times$) compared with random search ($20\times$). The major reasons behind such a success are two folds. First, we build our initial model based on several key features (configuration parameters) of SGQuant, which can effectively capture the core relation between their value and the final quantization performance (accuracy). Second, the ML cost model are iteratively updated as it sees more data samples, which helps it refine itself by providing solution more wisely. Besides, we observe similar performance between ML cost model and random search at the first 40 trials. The reason is that, starting with no training data, our ABS randomly samples and profiles N_{sample} (=40) configurations at the beginning, where we can expect similar performance as the random search approach.

VII. CONCLUSION

In this paper, we propose and implement a specialized GNN quantization scheme, SGQuant, to resolve the memory overhead of GNN computing. Specifically, our multi-granularity quantization incorporates the layer-wise, component-wise, and topology-aware quantization granularities that can intelligently compress the GNN features while minimizing the accuracy drop. To efficiently select the most appropriate bits for all these quantization granularities, we further offer a ML-based automatic bit-selecting (ABS) strategy that can minimize the users' efforts in design exploration. Rigorous experiments show that SGQuant can effectively reduce the memory size up to $31.9\times$ under negligible accuracy drop. In sum, SGQuant paves a promising way for GNN quantization that can facilitate their deployment on resource-constraint devices.

REFERENCES

- [1] R. Kaspar and B. Horst, *Graph classification and clustering based on vector space embedding*. World Scientific, 2010, vol. 77.
- [2] J. Gibert, E. Valveny, and H. Bunke, “Graph embedding in vector spaces by node attribute statistics,” *Pattern Recognition*, vol. 45, no. 9, pp. 3072–3083, 2012.
- [3] A. G. Duran and M. Niepert, “Learning graph representations with embedding propagation,” in *Advances in neural information processing systems (NIPS)*, 2017, pp. 5119–5130.
- [4] H. Chen, X. Li, and Z. Huang, “Link prediction approach to collaborative filtering,” in *Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL)*. IEEE, 2005, pp. 141–142.
- [5] J. Kunegis and A. Lommatzsch, “Learning spectral graph transformations for link prediction,” in *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, 2009, pp. 561–568.
- [6] T. Tyrenda, R. Angelova, and S. Bedathur, “Towards time-aware link prediction in evolving social networks,” in *Proceedings of the 3rd workshop on social network mining and analysis*, 2009, pp. 1–10.
- [7] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM international conference on Knowledge discovery and data mining (SIGKDD)*, 2016, pp. 855–864.
- [8] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations,” in *Proceedings of the 20th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. Association for Computing Machinery, 2014, p. 701710.
- [9] D. Luo, F. Nie, H. Huang, and C. H. Ding, “Cauchy graph embedding,” in *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011, pp. 553–560.
- [10] D. Luo, C. Ding, H. Huang, and T. Li, “Non-negative laplacian embedding,” in *2009 Ninth IEEE International Conference on Data Mining (ICDM)*. IEEE, 2009, pp. 337–346.
- [11] D. Cheng, Y. Gong, X. Chang, W. Shi, A. Hauptmann, and N. Zheng, “Deep feature learning via structured graph laplacian embedding for person re-identification,” *Pattern Recognition*, vol. 82, pp. 94–104, 2018.
- [12] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” *International Conference on Learning Representations (ICLR)*, 2017.
- [13] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” in *International Conference on Learning Representations (ICLR)*, 2019.
- [14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Li, and Y. Bengio, “Graph attention networks,” in *International Conference on Learning Representations (ICLR)*, 2018.
- [15] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Advances in neural information processing systems (NIPS)*, 2017, pp. 1024–1034.
- [16] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.
- [17] R. Banner, Y. Nahshan, and D. Soudry, “Post training 4-bit quantization of convolutional networks for rapid-deployment,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 7948–7956.
- [18] C. Zhu, S. Han, H. Mao, and W. J. Dally, “Trained ternary quantization,” *arXiv preprint arXiv:1612.01064*, 2016.
- [19] D. Lin, S. Talathi, and S. Annapureddy, “Fixed point quantization of deep convolutional networks,” in *International Conference on Machine Learning (ICML)*, 2016, pp. 2849–2858.
- [20] J. Leskovec and A. Krevl, “SNAP Datasets: Stanford large network dataset collection,” <http://snap.stanford.edu/data>, Jun. 2014.
- [21] M. Fey and J. E. Lenssen, “Fast graph representation learning with PyTorch Geometric,” in *ICLR Workshop on Representation Learning on Graphs and Manifolds (ICLR)*, 2019.
- [22] J. Shao, *Mathematical Statistics*, ser. Springer Texts in Statistics. Springer, 2003.
- [23] W.-Y. Loh, “Classification and regression trees,” *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 1, pp. 14–23, 2011.
- [24] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, “Hierarchical graph representation learning with differentiable pooling,” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS)*, Red Hook, NY, USA, 2018, p. 48054815.