TOPGQ: POST-TRAINING QUANTIZATION FOR GNNS VIA TOPOLOGY BASED NODE GROUPING

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNN) suffer from huge computational and memory costs in processing large graph data on resource-constrained devices. One effective solution to reduce costs is neural network quantization, replacing complex high-bit operations with efficient low-bit operations. However, to recover from the error induced by lower precision, existing methods require extensive computational costs for retraining, which are many times larger than conventional GNN training. In this circumstance, we propose TopGQ, the first post-training quantization (PTQ) framework for GNNs, enabling an order of magnitude faster quantization without backpropagation. We analyze the feature magnitude of vertices and observe that it is correlated to the topology regarding their neighboring vertices. From these findings, TopGQ proposes to group vertices with similar topology information of inward degree and localized Wiener index to share quantization parameters within the group. Then, TopGQ absorbs the group-wise scale into the adjacency matrix for efficient inference by enabling quantized matrix multiplication of node-wise quantized features. The results show that TopGQ outperforms SOTA GNN quantization methods in performance with a significantly faster quantization speed.

1 INTRODUCTION

Graph neural networks (GNNs) attract a great amount of attention due to their ability to process diverse unstructured data. They have achieved success in many areas such as recommendation systems (Pal et al., 2020; Fan et al., 2019; Zhang et al., 2023), molecular interaction (Wale et al., 2008; Borgwardt et al., 2005), transportation networks (Bai et al., 2020; Cao et al., 2020), and social network analysis (Qiu et al., 2018; Arazzi et al., 2023). However, GNNs often suffer from huge computational and memory costs due to increasing demands for processing large graphs, especially on resource-constrained devices.

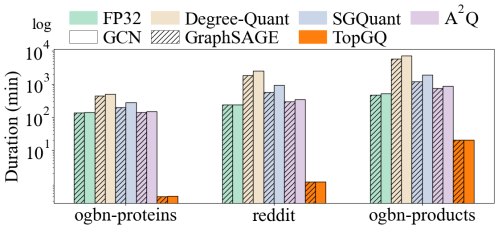


Figure 1: Comparing duration of existing GNN quantization methods against TopGQ.

One promising direction to circumvent this issue is neural network quantization (Choukroun et al., 2019; Zhao et al., 2019; Choi et al., 2021), which efficiently reduces the computation and memory requirements of GNN inference by utilizing reduced numerical precision for computations. However, despite its advantages, applying quantization to GNNs is considered difficult due to the extremely diverse vertex feature magnitudes. This is known to be caused by the necessary message-passing of the GNN algorithm, which aggregates features from neighboring vertices. Since the quantization process is known to be highly sensitive to the magnitude outliers (Wei et al., 2022), such diversity in aggregated features in GNNs results in high quantization errors.

To handle the magnitude outlier problem, several methods (Tailor et al., 2020; Zhu et al., 2022) have been proposed to adopt quantization-aware training (QAT). They reduce quantization error by considering the degree information (Tailor et al., 2020) or applying mixed-precision strategy (Zhu et al., 2022). However, QAT methods accompany huge computation and memory costs for the quantization process, requiring excessive resources and time larger than full-precision pretraining target GNN architecture. Figure 1 reports the time it takes to quantize common GNN architectures

using prior GNN quantization methods (Tailor et al., 2020; Zhu et al., 2022). Measured in wall-clock time, the quantization time easily exceeds 100 minutes, and even up to 4.9 days (ogbn-products, Degree-Quant) when the graph size increases.

To this end, we propose TopGQ, the first accurate post-training quantization (PTQ) framework for GNNs addressing the aforementioned issues of existing GNN quantization methods. TopGQ does not involve any form of gradient computation or parameter updates, which makes the proposed method significantly faster compared to the baseline and requires low memory consumption. Instead, TopGQ focuses on the local topological information of the graphs to determine accurate quantization parameters. As a result, our method TopGQ shortens the quantization time by an order of magnitude, making GNN quantizations much faster and more efficient.

The key to achieving high performance within such a short time (i.e., without gradient computation) is to focus on the local topology of the graphs. We observe that the existing method relying on the indegree of the vertices is insufficient to capture the diversity of the feature magnitudes. Instead, we propose topology-based node grouping. Because the magnitude of a node feature is determined by local neighbors, we arrange the vertices into several quantization groups that share similar indegree and local Wiener index. To perform the grouping within a short time, we devise an efficient algorithm for computing the local Wiener index. Lastly, to utilize the groups for a quantized inference kernel, we additionally provide scale absorption method to enable efficient integer matrix multiplication of node-wise quantized feature matrix. The extensive experimental results show that TopGQ outperforms the existing SOTA method for GNN quantization with up to 358$\times$ speedups with better or comparable accuracy, establishing a new standard of GNN quantization.

Our contributions can be summarized as follows:

- We show that the magnitudes of node features in GNN are correlated with local topological information from degree centrality and Wiener index.
- We propose a topology-based node grouping, which groups vertices with similar topological characteristics to reduce quantization error from high feature magnitude variance of GNN.
- We propose scale absorption to enable efficient integer arithmetic of node-wise quantized GNN operation by absorbing node-wise scale into an adjacency matrix.
- We propose the first PTQ method for GNN, which outperforms the existing training-based quantization method with meaningful margins while bringing up to 358$\times$ less quantization time compared to the baselines.

2 Preliminaries

2.1 Graph Neural Networks

Let graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. Each vertex v_i consists of feature vector h_i and adjacency matrix is $A \in \mathbb{R}^{n \times n}$ for n vertices, where $A_{i,j} = e_{i,j}$, if $e_{i,j} \in E$, else 0. To embed topological information in vertex feature, GNN gathers information from neighboring vertices $u_j \in \mathcal{N}(v_i)$ to update hidden vertex feature h_i of v_i, which is called the message-passing algorithm. The message-passing algorithm consists of two parts: combination and aggregation. Firstly, hidden vertex feature $h_i^{(l)}$ is multiplied with weight matrix $W^{(l)}$ of l-th GNN layer (combination), then the hidden vertex feature $h_i^{(l)}$ of v_i is updated (aggregation) as following:

$$h_i^{(l+1)} = \phi(W h_i^{(l)}, \bigoplus_{j \in \mathcal{N}(i)} e_{i,j} W h_j^{(l)}), \tag{1}$$

where ϕ feature update operator and $\bigoplus$ is a permutation-invariant aggregation, such as sum or mean.

GNN computation can be represented by multiplications of vertex feature matrix $X \in \mathbb{R}^{n \times d_{in}} = [h_1, \cdots, h_n]^T$, weight matrix $W \in \mathbb{R}^{d_{in} \times d_{out}}$, and adjacency matrix $\tilde{A} \in \mathbb{R}^{n \times n}$ as follows:

$$X_{comb}^{(l)} = W \cdot X^{(l)}, \tag{2}$$

$$X^{(l+1)} = \sigma(\tilde{A} \cdot X_{comb}^{(l)}), \tag{3}$$

where σ is nonlinear operation, $\tilde{A}$ may vary with GNN architecture, e.g., GCN (Kipf & Welling, 2016) utilizes normalized graph laplacian matrix $\tilde{A} = D^{-1/2}AD^{-1/2}$, while GIN (Xu et al., 2019) uses binary adjacency matrix $\tilde{A} = A$. GraphSAGE (Hamilton et al., 2017) differs in the aggregation phase by sampling a subset of neighboring vertices instead of considering all neighbors.

2.2 Quantization

Quantization replaces high-bit floating-point operations with low-bit integer operations. We use simple yet effective uniform integer quantization as with scale (s) and zero-point (z) as follows:

$$x^q = Q(x; s, z) = clamp(\lfloor s \cdot x - z \rceil, q_{max}, q_{min}), \tag{4}$$

$$s = (2^k - 1)/(x_{max} - x_{min}), \tag{5}$$

where k is quantization bit, q_{max}, q_{min} is maximum and minimum value of k-bit integer representation, and $\lfloor \cdot \rceil$ is rounding operator. For symmetric quantization, which has the symmetrical representation range centered with zero, $z = 0$. For asymmetric quantization, on the other hand, z is calculated as $z = s \cdot x_{min} + 2^{k-1}$. Also, each row or column may have different quantization parameters (s, z), which are calculated independently according to quantization dimension, called row-wise and column-wise quantization, respectively.

There are two mainstream types of quantization to alleviate the effects of quantization error: Post-training quantization (PTQ) and quantization-aware training (QAT). On the one hand, PTQ methods go through calibration, which denotes the process of adjusting the scale, zero-point, and rounding directions of quantization using only a small set of data. Conversely, QAT methods directly apply gradient-based training to explicitly reduce the quantized network's target loss. The major discrepancy between the two types of methods is that QAT generally incorporates updating weight parameters and optionally some quantization parameters, while PTQ methods focus on quantization parameters without weight updates and is much faster.

2.3 Quantization of Graph Neural Networks

To achieve efficient inference in terms of computational cost and memory requirements, we should consider both the combination phase (Equation (2)) and the aggregation phase (Equation (3)):

$$X^{(l)}_{comb} = Q(W; s_W, z_W) \cdot Q(X^{(l)}; s_{X^{(l)}}, z_{X^{(l)}}), \tag{6}$$

$$X^{(l+1)} = \sigma(Q(\tilde{A}; s_{\tilde{A}}, z_{\tilde{A}}) \cdot Q(X^{(l)}_{comb}; s_{X^{(l)}_{comb}}, z_{X^{(l)}_{comb}})). \tag{7}$$

As we can see in Equation (6) and Equation (7), we have to choose quantization policy for each W, $X^{(l)}$, $X^{(l)}_{comb}$, and $\tilde{A}$. These design choices highly affect the final accuracy and inference efficiency of a quantized network, and many baselines choose different policies.

Prior methods (Tailor et al., 2020; Zhu et al., 2022) each take on a different policy which balances between better accuracy and efficiency. Degree-Quant (Tailor et al., 2020) applies per-tensor quantization for all matrices involved in both combination and aggregation. While this requires the smallest amount of memory, it suffers from quantization outliers as a single high-magnitude channel or element can affect the quantization scale. A^2Q (Zhu et al., 2022) mitigates this issue by applying column-wise quantization for W and X_{comb} while row-wise (node-wise) quantization for X and $\tilde{A}$. As row- and column-wise quantization can separate quantization parameters of each node- and feature-dimension, respectively, it is more robust to outlier vertices/channels at the expense of increased cost.

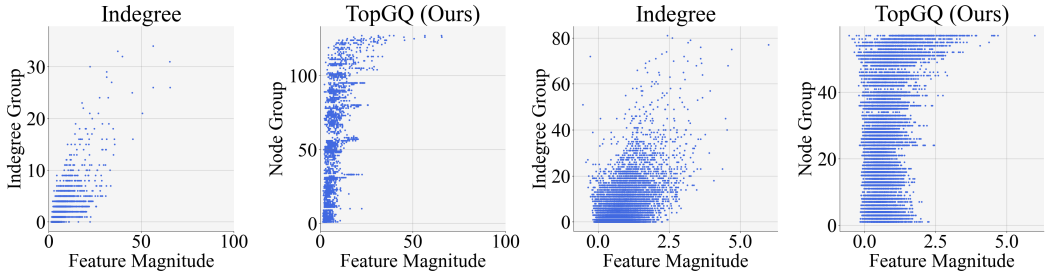
3 Related Work

GNN Quantization is a promising direction to efficiently reduce extensive computational costs and memory requirements of graph neural networks Kipf & Welling (2016); Xu et al. (2019); Veličković et al. (2018), as they suffer from large size of real-world graphs. Many pieces of research (Chen et al., 2022; Phan et al., 2018; Ding et al., 2021; Zhu et al., 2022; Tailor et al., 2020; Feng et al., 2020) aim to quantize GNNs, including weight parameters, node features, and adjacency matrix. Degree-Quant (Tailor et al., 2020) is the first work to quantize GNN architectures, using quantization-aware training (QAT) to allow high-degree vertices to retain full-precision features at training and be

quantized later for inference. SGQuant (Feng et al., 2020) and A^2Q (Zhu et al., 2022) are also QAT methods targeting GNN architectures, but they differ in that they allow mixed-precision to assign higher bitwidth to high-magnitude vertices. Notably, existing GNN quantization methods adopt QAT, i.e., incorporating gradient-based iterative weight updates, which require huge computational overheads and memory requirements (Figure 1). On the other hand, TopGQ is free from such a burden because it only needs to execute inference for a few batches for calibration.

Graph Topology in GNNs is often integrated during training to help the model effectively learn the structural information (Ji, 2019; Zhang & Lu, 2020; Hu et al., 2022; Wu et al., 2018; You et al., 2021; Brasoveanu et al., 2023). For example, Ji (2019) uses degree centrality to find highly central vertices in their pooling layer as they are more important for effective representation learning. Also, Zhang & Lu (2020) uses betweenness centrality to assign weights to each node during aggregation. Wu et al. (2018); Brasoveanu et al. (2023) uses Wiener index from chemoinformatics as inputs of GNNs to enhance its performance on general tasks. However, these methods do not relate topological information with node feature magnitudes, especially for quantization.

4 Motivational Study



(a) GIN, Cora (b) GCN, PubMed

Figure 2: Comparing feature magnitude range of two grouping techniques: indegree (left) and TopGQ (right). For both plots, x-axis denotes feature magnitude and y-axis denotes sorted group index.

For GNNs, the range of the feature values largely depends on each node's structural properties because of their unique message-passing framework. However, there has been a limited effort in leveraging structural properties in GNN quantization, and existing works Feng et al. (2020); Tailor et al. (2020) only utilize node indegree which only takes into account 1-hop neighbors. We find that indegree is a suboptimal measure when it comes to determining the quantization group. In Figure 2, we plot the feature magnitude of each quantization group using indegree as the sole metric, and we compare it against the groups used in our method TopGQ. Using only indegree to group the node features, each group tends to have a large range of values with uneven distribution of nodes among the groups. The extreme spread of values within each group would lead to poor representation of the dense region, leading to large quantization errors. Instead, TopGQ proposes to use a topological feature that can better capture such information. Figure 2 shows that each quantization group of TopGQ has a smaller range with a more even distribution across the groups.

5 Methodology

Quantization of GNN architectures has been studied to some depth (Tailor et al., 2020; Feng et al., 2020; Zhu et al., 2022), but not under the light of PTQ. The goal of TopGQ is to rapidly perform quantization with PTQ, while retaining QAT-like performance on GNNs. For this, we propose topology-based node grouping and scale absorption, which captures the local topology information into GNN quantization.

5.1 Quantization with Topology-based Node Grouping

Figure 3a shows the group generation strategy of TopGQ. Due to the nature of the aggregation phase (Equation (3)), it is evident that the feature magnitudes of a vertex depend greatly on how many and

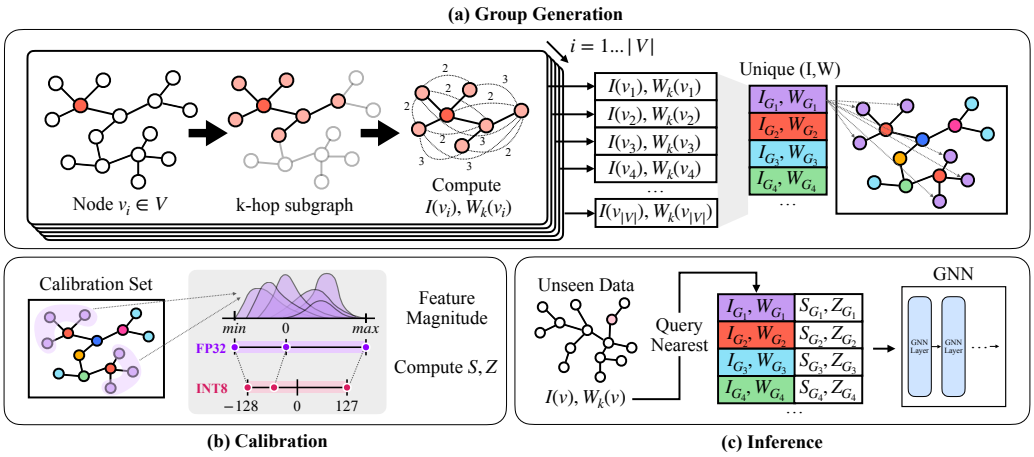


Figure 3: The process of topology-based node grouping. (a) shows group generation using topological characteristics: indegree and Wiener Index. Each color is used to denote each group. (b) shows the calibration process to achieve a set of quantization parameters for each group. (c) demonstrates how inference is done on unseen data by using the quantization parameters of the nearest group.

which vertices the features are being aggregated from. As discussed in Section 4, node indegree is a suboptimal measure because it only accounts for a limited amount of information. Instead, we propose to examine the topology of the local subgraph surrounding each vertex, and group the vertices with the same or similar local topology. Within the same group, we can expect that the vertices aggregate similar values and thus sharing a quantization scale leads to minimal quantized errors.

While there exist several different measures to interpret the topology of a graph, we propose to use Wiener index Graovac & Pisanski (1991) in conjunction with the indegree to better quantify the local structure around each node. Let graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. Wiener index is defined as the sum of shortest lengths between all pairs of vertices:

$$W(G) = \Sigma_{u,v \in V} dis(u, v), \tag{8}$$

where $dis(u, v)$ denotes the shortest path distance between vertices u and v. Because Wiener index is originally a graph-level representation, we make an adaptation to use it as a node-level representation. For each vertex, we extract a k-hop subgraph around each vertex and compute the Wiener index of the subgraph. Formally put, we define the localized Wiener index $W_k(u)$ of vertex u as:

$$W_k(u) = \Sigma_{v,w \in N_k(u)} dis(v, w), \quad N_k(u) = \{v \in V | dis(u, v) \leq k\}, \tag{9}$$

where k is the predefined hop count and N_k is the set of reachable neighbor vertices within k hops.

Once the localized Wiener index values are obtained for the vertices, we consider indegree $I(u)$ together, and vertices with equal $(I(u), W_k(u))$ are assigned to the same quantization group. For example, in Figure 3a, all purple-colored vertices belong to a single group.

After the groups have been generated, the calibration (Figure 3b) takes place. For each group, group-wise quantization parameters (S_G, Z_G) are obtained according to Section 2.2 by measuring min, max values per group. At inference time (Figure 3c), the test set vertices are assigned to the groups according to their (I, W) pair values. When unseen values are found at inference time, they are assigned to the most similar quantization group by first comparing the I and then the W values.

5.2 Accelerated Computation of Localized Wiener Index

Because the localized Wiener index in Equation (9) requires all-pair shortest paths within the subgraphs, its computation can add a considerable overhead. Although several algorithms are known for all-pair shorted paths (Dijkstra, 1959; Floyd, 1962; Warshall, 1962; Bellman, 1958; Ford Jr, 1956), they often require huge computational and space complexity.

For this, we propose a new algorithm to compute the localized Wiener index, shown in Algorithm 1. The key idea is that because we sampled k-hop neighbors of a single vertex to extract a subgraph, its diameter (i.e., the maximum distance between two arbitrary nodes) is capped at $2k$. This can be used to efficiently calculate the localized Wiener indices. For instance, $W_2(u)$ is calculated as

$$W_2(u) = |E_{sub}| + 2|d_2| + 3|d_3| + 4|d_4|, \tag{10}$$

where d_n is a set of distance-n node pairs in a local k-hop subgraph $G_{sub} = (V_{sub}, E_{sub})$ of node u. In practice, obtaining the sets d_i can be time-consuming because this requires costly all-pair shortest paths. Instead, Equation (10) can be restructured in a subtractive manner, using k-hop reachable set $N_i(u)$ that can be easily obtained by simple traversal:

Algorithm 1 Accelerated Wiener Index Computation

```
1:  Input: Local k-hop subgraph G_sub = (V_sub, E_sub)
2:  Output: Wiener index W_k(u) ∈ ℕ
3:
4:  function addNeighbors(node u, set H, depth m)
5:      for v ∈ u.nbr() do
6:          h_m ← h_m ∪ (u, v)
7:          if m > 0 do
8:              addNeighbors(v, H, m − 1)
9:          end if
10:     end for
11: end function
12:
13: W ← k|V_sub|^2
14: H ← {h_l = ∅ | l = 0, ..., k}
15: h_k ← V_sub
16: parallel for node u ∈ V_sub
17:     addNeighbors(u, H, k − 1)
18: end for
19: W ← W − Σ_{l=0}^{k} |⋃_{i=l}^{k} h_i|
```

$$W_2(u) = 4 \cdot \Sigma_{v \in V_{sub}} |N_4(v)| - (\Sigma_{i=0}^{3} \Sigma_{v \in V_{sub}} |N_i(v)|), \tag{11}$$

From the maximum-value case where all vertices are connected in 4 hops, we subtract the number of occurrences in each of i-hop reachable set from the vertices. Additionally, we can substitute some terms trivially obtainable from graph formats such as CSR. The $\Sigma_{v \in V_{sub}} |N_4(v)|$ is simply $|V_{sub}|^2$, $\Sigma_{v \in V_{sub}} |N_1(v)|$ is the number of edges $|E_{sub}|$ and $\Sigma_{v \in V_{sub}} |N_0(v)|$ the number of vertices $|V_{sub}|$:

$$W_2(u) = 4|V_{sub}|^2 - (\Sigma_{v \in V_{sub}} |N_3(v)| + \Sigma_{v \in V_{sub}} |N_2(v)| + |E_{sub}| + |V_{sub}|), \tag{12}$$

The overall process is shown in Algorithm 1. First, we define a function *addNeighbors* (lines 4-11), which recursively adds l-hop reachable node pairs into the set h_l. Then, we initialize Wiener index W with $k|V_{sub}|^2$ (line 13), and h_l with $\varnothing$. As the computation of *addNeighbors* on node n in G_{sub} is independent of each other, we parallelized the computation (line 16). After the computation, the h_l stores a non-overlapping set of l-hop reachable node pairs. By using h_l, we calculate $\Sigma_{v \in V_{sub}} |N_l(v)|$ by $|\bigcup_{i=l}^{k} h_i|$ and obtain the Wiener index result (line 19). Please refer to Table 6 for the experiments on acceleration, compared to Bellman-Ford, Floyd-Warshall, and Dijkstra's algorithm.

5.3 Inference Flow with Scale Absorption

TopGQ maintains the same quantization method for the vertex feature matrix X for the combination and aggregation phases. The preservation of the direction in quantization leads to the preservation of its vertex precision. Assuming symmetric quantization for simplicity, it can be represented as $X \approx S_X \cdot X^Q$, where X^Q is the quantized features and S_X is a diagonal matrix of the scales of each node group.

In the combination phase, the weight parameters are regarded as a single group. Therefore, the quantized form of the combination becomes:

$$X \cdot W \approx S_X \cdot X^Q \cdot W^Q \cdot s_W \tag{13}$$

$$= (s_W \otimes S_X) \cdot (X^Q \cdot W^Q), \tag{14}$$

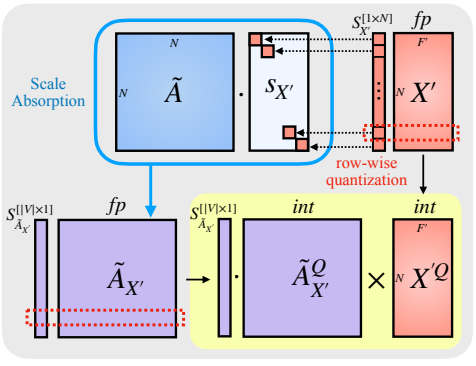


Figure 4: Inference with scale absorption.

where the typical trick of separate scale calculation (Dai et al., 2021; Zhu et al., 2022) can be applied.

For aggregation, using the same quantization method is infeasible, because with quantized $\tilde{A}$ and X,

$$\tilde{A} \cdot X \approx S_A \cdot \tilde{A}^Q \cdot S_X \cdot X^Q, \tag{15}$$

Table 1: Performance on node classification task using large graph datasets.

Dataset	Bit	Method	Type	GCN		GraphSAGE	
				Acc.	Q. Time	Acc.	Q. Time
Reddit	FP32	-	-	90.60	-	94.64	-
	INT4	Degree-Quant	QAT	49.25	(31.18h)	89.86	(42.23h)
		SGQuant	QAT	88.74	(9.19h)	63.73	(15.75h)
		A^2Q	QAT	58.31	(4.92h)	52.65	(5.78h)
		TopGQ (Ours)	PTQ	**83.95**	**(0.02h)**	**93.93**	**(0.02h)**
	INT8	Degree-Quant	QAT	90.91	(30.39h)	90.35	(42.49h)
		SGQuant	QAT	88.67	(9.46h)	69.12	(15.48h)
		A^2Q	QAT	61.15	(4.91h)	76.26	(5.70h)
		TopGQ (Ours)	PTQ	**91.13**	**(0.02h)**	**94.60**	**(0.02h)**
ogbn-proteins	FP32	-	-	56.94	-	73.33	-
	INT4	Degree-Quant	QAT	57.37	(7.68h)	50.02	(8.84h)
		SGQuant	QAT	52.97	(3.46h)	57.77	(4.64h)
		A^2Q	QAT	44.95	(2.35h)	**71.98**	(2.49h)
		TopGQ (Ours)	PTQ	**60.08**	**(0.01h)**	68.93	**(0.01h)**
	INT8	Degree-Quant	QAT	**59.32**	(7.49h)	**73.81**	(8.38h)
		SGQuant	QAT	52.77	(3.32h)	69.30	(4.58h)
		A^2Q	QAT	44.41	(2.35h)	69.38	(2.51h)
		TopGQ (Ours)	PTQ	58.05	**(0.01h)**	73.34	**(0.01h)**
ogbn-products	FP32	-	-	78.41	-	71.65	-
	INT4	Degree-Quant	QAT	**70.58**	(98.38h)	65.05	(121.78h)
		SGQuant	QAT	26.90	(20.03h)	27.38	(37.17h)
		A^2Q	QAT	23.62	(13.16h)	22.21	(14.69h)
		TopGQ (Ours)	PTQ	57.55	**(0.34h)**	**71.02**	**(0.34h)**
	INT8	Degree-Quant	QAT	75.26	(95.95h)	69.18	(118.96h)
		SGQuant	QAT	65.71	(20.18h)	41.71	(31.25h)
		A^2Q	QAT	47.91	(12.57h)	58.26	(14.66h)
		TopGQ (Ours)	PTQ	**76.94**	**(0.34h)**	**73.67**	**(0.34h)**

which contains the S_X matrix inside the multiplication. Instead, we take advantage of the fact that $\tilde{A}$ is a static topology. After calculating the scale diagonal matrix S_X,

$$\tilde{A} \cdot X \approx \tilde{A} \cdot S_X \cdot X^Q = \tilde{A}_X \cdot X^Q \approx S_{A_X} \cdot \tilde{A}_X^Q \cdot X^Q. \tag{16}$$

In the above, the scale diagonal matrix S_X is absorbed into the adjacency matrix $\tilde{A}$ to form $\tilde{A}_X$, which is then row-wisely quantized with the new scale as $S_{A_X} \cdot \tilde{A}_X^Q$. At inference time, this can be pre-calculated as both $\tilde{A}$ and S_X only depend on the topology of the input graph.

6 Experiments

6.1 Experimental Settings

We report evaluation results on two representative graph processing tasks: Node-level classification, graph-level classification. For node-level classification, we compare validation accuracy of Reddit, ogbn-proteins, and ogbn-products, Cora, CiteSeer, and PubMed datasets. For graph-level classification, we evaluate TopGQ on PROTEINS and NCI1 datasets and compare the validation accuracy. We compare TopGQ with three graph quantization baselines using QAT approaches: Degree-Quant (Tailor et al., 2020), SGQuant (Feng et al., 2020), and A^2Q (Zhu et al., 2022). To ensure a fair comparison, we use fixed-precision quantization for both SGQuant and A^2Q when attaining experiment results. We report quantized accuracy of GCN (Kipf & Welling, 2016), GIN (Xu et al., 2019), and GraphSAGE (Hamilton et al., 2017) architectures with 4-bit and 8-bit integer quantization. For a fair comparison, we apply the same bitwidth for all layers, including aggregation and combination. We use $k = 3$ for ogbn-products, PROTEINS, and NCI1 and $k = 2$ for other datasets. For more details on the experimental setting, please refer to the Appendix.

Table 2: Performance on node classification task using smaller graph datasets.

Dataset	Bit	Method	Type	GCN		GIN		GraphSAGE	
				Acc.	Q. Time	Acc.	Q. Time	Acc.	Q. Time
Cora	FP32	-	-	82.08	-	78.54	-	79.58	-
	INT4	Degree-Quant	QAT	79.00	(9.64s)	71.90	(31.47s)	73.50	(15.54s)
		SGQuant	QAT	79.00	(3.20s)	70.20	(4.22s)	75.30	(8.62s)
		A^2Q	QAT	52.70	(2.09s)	64.60	(1.72s)	74.20	**(2.53s)**
		TopGQ (Ours)	PTQ	**81.50**	**(1.40s)**	**78.58**	**(0.99s)**	**79.64**	**(0.87s)**
	INT8	Degree-Quant	QAT	81.80	(9.82s)	74.60	(31.45s)	77.50	(15.52s)
		SGQuant	QAT	80.50	(3.60s)	73.30	(4.53s)	75.30	(8.38s)
		A^2Q	QAT	80.00	(1.60s)	**78.70**	(1.95s)	76.10	(2.48s)
		TopGQ (Ours)	PTQ	**82.08**	**(1.12s)**	78.42	**(1.18s)**	**80.30**	**(0.87s)**
Citeseer	FP32	-	-	72.34	-	70.24	-	71.96	-
	INT4	Degree-Quant	QAT	22.30	(21.72s)	47.90	(90.57s)	17.10	(40.67s)
		SGQuant	QAT	68.10	(5.57s)	46.70	(8.23s)	48.30	(17.91s)
		A^2Q	QAT	54.00	(2.08s)	46.00	(2.67s)	66.20	(3.18s)
		TopGQ (Ours)	PTQ	**71.90**	**(1.17s)**	**70.14**	**(1.14s)**	**71.76**	**(1.05s)**
	INT8	Degree-Quant	QAT	69.70	(22.03s)	58.30	(92.75s)	69.10	(40.63s)
		SGQuant	QAT	68.30	(5.85s)	51.30	(8.56s)	54.10	(18.47s)
		A^2Q	QAT	70.50	(1.77s)	67.30	(2.36s)	66.00	(3.15s)
		TopGQ (Ours)	PTQ	**72.28**	**(1.11s)**	**70.26**	**(1.16s)**	**71.96**	**(1.05s)**
Pubmed	FP32	-	-	80.32	-	78.82	-	78.84	-
	INT4	Degree-Quant	QAT	78.60	(21.33s)	76.60	(108.07s)	78.20	(34.38s)
		SGQuant	QAT	76.10	(5.41s)	65.30	(8.24s)	71.10	(15.86s)
		A^2Q	QAT	69.70	(2.17s)	51.90	(2.60s)	73.90	(3.31s)
		TopGQ (Ours)	PTQ	**79.58**	**(1.21s)**	**77.70**	**(1.18s)**	**79.00**	**(1.12s)**
	INT8	Degree-Quant	QAT	79.20	(21.56s)	**79.70**	(109.59s)	78.40	(34.07s)
		SGQuant	QAT	78.10	(5.31s)	75.20	(8.91s)	73.40	(15.66s)
		A^2Q	QAT	76.40	(1.70s)	76.40	(2.15s)	75.40	(3.24s)
		TopGQ (Ours)	PTQ	**80.30**	**(1.08s)**	78.62	**(1.16s)**	**78.94**	**(1.22s)**

6.2 Node Classification Results

The experimental results of quantization accuracy comparison of node classification task are shown in two settings: larger graphs (Table 1) and more conventional sized graphs (Table 2). The results show that TopGQ performs comparable or significantly better in accuracies, and achieves an order of magnitude faster quantization time. Taking Reddit with 4-bit GraphSAGE as an example, the best-performing baseline is Degree-Quant, with 89.86% accuracy. However, it suffers from almost 90 hours of quantization time. SGQuant and A^2Q are faster on quantization, but suffer from severe accuracy drops. On the other hand, TopGQ achieves a significantly higher accuracy of 93.93%, with only 0.02 hours of quantization time. This is more than 1000$\times$ faster than Degree-Quant, and more than 100$\times$ faster than the low-performing baselines (SGQuant and A^2Q).

Table 2 shows results on the smaller graphs that are more commonly used in existing GNN quantization literature. The results show a similar trend overall. TopGQ shows comparable performance compared to the existing baselines with significantly low overhead for GNN quantization. The quantization times are relatively short for all methods, which comes from small number vertices and edges for the datasets. Nonetheless, TopGQ is the fastest in quantization time in all cases.

6.3 Graph Classification Results

Our experimental results on graph-level classification are depicted in Table 3. The proposed method, TopGQ, demonstrates significant improvements in quantization speed while maintaining competitive classification performance. For instance, Degree-Quant takes almost an hour to quantize the GraphSAGE model on NCI1, with a significant drop in accuracy of 9.0%p.

In contrast, TopGQ achieves remarkable speed improvements with post-training quantization (PTQ), requiring only about a minute for quantization across all datasets and models. This efficiency highlights the superiority of TopGQ, as it achieves a balance between accuracy and quantization speed, making it a practical choice for large-scale graph-level classification tasks.

Table 3: Performance on graph classification task.

Dataset	Bit	Method	Type	GCN		GIN		GraphSAGE	
				Acc.	Q. Time	Acc.	Q. Time	Acc.	Q. Time
PROTEINS	FP32	-	-	76.19	-	74.79	-	72.87	-
	INT4	Degree-Quant	QAT	**75.21**	(2158.47s)	70.44	(1407.09s)	63.72	(1371.54s)
		SGQuant	QAT	59.84	(203.70s)	59.48	(190.28s)	59.66	(249.86s)
		A^2Q	QAT	71.16	(128.52s)	65.59	(116.96s)	**73.59**	(209.23s)
		TopGQ (Ours)	PTQ	70.15	**(4.20s)**	**70.61**	**(3.94s)**	69.67	**(4.21s)**
	INT8	Degree-Quant	QAT	74.93	(2140.48s)	69.72	(1368.98s)	63.61	(1358.99s)
		SGQuant	QAT	72.40	(203.61s)	69.73	(190.71s)	61.99	(261.81s)
		A^2Q	QAT	73.05	(136.03s)	66.85	(129.83s)	70.62	(194.75s)
		TopGQ (Ours)	PTQ	**75.94**	**(4.11s)**	**74.86**	**(3.86s)**	**74.00**	**(4.17s)**
NCI1	FP32	-	-	80.41	-	81.46	-	78.46	-
	INT4	Degree-Quant	QAT	**73.55**	(4588.48s)	76.42	(3110.10s)	69.46	(3585.30s)
		SGQuant	QAT	63.92	(530.27s)	53.09	(571.44s)	66.13	(778.96s)
		A^2Q	QAT	68.81	(668.52s)	**79.08**	(648.44s)	72.38	(656.70s)
		TopGQ (Ours)	PTQ	65.09	**(9.36s)**	78.49	**(8.98s)**	**76.43**	**(9.18s)**
	INT8	Degree-Quant	QAT	75.47	(4493.82s)	77.59	(3025.24s)	69.12	(3449.79s)
		SGQuant	QAT	68.47	(527.19s)	74.36	(572.13s)	67.59	(799.31s)
		A^2Q	QAT	75.64	(648.18s)	79.17	(635.09s)	76.86	(645.50s)
		TopGQ (Ours)	PTQ	**80.91**	**(9.35s)**	**81.88**	**(8.97s)**	**79.16**	**(9.22s)**

Table 4: Ablation study of TopGQ.

Bit	Node Grouping	Scale Absorption	PROTEINS			NCI1		
			GCN	GIN	Graph SAGE	GCN	GIN	Graph SAGE
INT4	✗	✗	57.32	45.51	44.05	53.35	60.66	73.80
	Indegree	✗	56.15	45.04	50.65	60.54	69.71	75.46
	L.Wiener Idx	✗	61.28	47.12	62.76	60.93	72.76	75.63
	L.Wiener Idx	✓	69.94	70.92	68.93	65.88	75.37	75.98
INT8	✗	✗	56.14	55.91	61.25	79.63	81.29	78.30
	Indegree	✗	72.57	71.86	70.48	78.91	81.28	78.32
	L.Wiener Idx	✗	75.64	73.94	73.69	80.89	81.90	79.18
	L.Wiener Idx	✓	75.65	74.34	72.20	79.72	81.36	78.43

6.4 Ablation Study

We conducted an ablation study to show the effect of the proposed quantization groups, which are the influence group and the topological group. The results are shown in Table 4. When a naive version of PTQ is performed without any of the proposed schemes, it suffers from accuracy degradation due to high-variance node-wise magnitude. This phenomenon is especially worse in GIN architecture, as the node features of GIN architecture are larger due to the unnormalized sum aggregation operation (Tailor et al., 2020). Applying the proposed topology grouping with localized Wiener index further boosts the PTQ performance, as it effectively divides quantization groups in a node-wise manner, with the nodes in the group sharing similar magnitudes for the quantization.

6.5 Cost Analysis

Table 5 compares the inference time of the baselines against TopGQ. The inference time is measured on RTX 4090 GPUs with customized kernels. While the forward times are mostly similar due to the same amount of multiplications, the difference in inference time comes from the unseen vertices. While Degree-Quant does not handle unseen nodes any differently, A^2Q has to perform costly nearest neighbor search on the input features. Although TopGQ performs a group search for unseen nodes, this only involves simple I, W comparison before inference.

Table 5: Inference time comparison using GCN.

Bit	Method	Dataset	Reddit	ogbn-products
FP32	-	-	4015.7s	15697.3s
INT8	Degree-Quant	QAT	3167.4s	13795.6s
	A^2Q	QAT	3204.5s	16370.9s
	SGQuant	QAT	3170.3s	13933.4s
	TopGQ	PTQ	3173.0s	13951.7s

Table 6: Comparison of node-wise Wiener index computation time.

Algorithm	Cora	CiteSeer	Pubmed	PROTEINS	NCI1	Reddit	ogbn-proteins	ogbn-products
Bellman-Ford	0.35s	0.46s	19.87s	40.07s	512.13s	4.21h	2.84h	305.50h
Floyd-Warshall	0.15s	0.21s	4.68s	8.50s	11.27s	0.57h	0.41h	35.44h
Dijkstra	0.19s	0.29s	2.70s	12.75s	12.49s	0.16h	0.11h	8.52h
Ours (§5.2)	**0.02s**	**0.01s**	**0.06s**	**4.84s**	**1.78s**	**0.0004h**	**0.0002h**	**0.2855h**
Speed Up	9.77×	31.65×	43.50×	1.76×	6.32×	412.23×	602.30×	29.83×

On the quantization time, TopGQ is orders of magnitude faster as discussed in Section 6.2. A large portion of this is due to the proposed computing method for the localized Wiener index across multiple datasets, as shown in Table 6. We compare the time to compute the Wiener index of k-hop subgraph, using the same k settings that are used in the main experiments. For the baseline methods, we used implementations from the SciPy library. The results show that our method demonstrates significant improvements in computational efficiency compared to traditional algorithms such as Bellman-Ford, Floyd-Warshall, and Dijkstra. Specifically, our approach reduces the Wiener index computation time by up to 4.57× on large-scale datasets like ogbn-proteins, achieving a time reduction from 2.84 hours to 0.02 hours. This trend is consistent across datasets, with speedups ranging from 1.22× on Cora to 4.46× on Reddit. Our method scales significantly better with larger graphs by reducing the computational cost of the Wiener index, achieving superior quantization speed.

6.6 Analysis on Scale Absorption

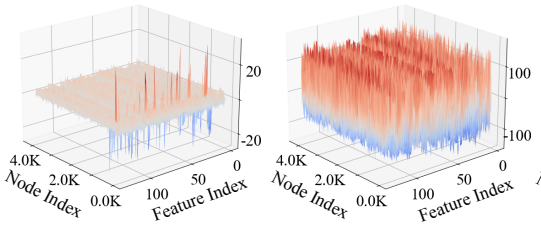
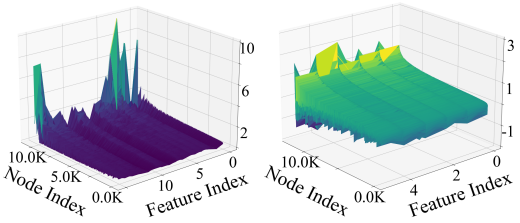


Figure 5: Comparing X_{comb} before (left) and after (right) scale absorption (GCN, PROTEINS).



(a) GIN, Cora (b) GCN, Citeseer

Figure 6: X_{comb} magnitude visualization.

In this section, we compare the activations X_{comb} before and after applying scale absorption to visualize its effect. In Figure 5, the lefthand side denotes activation before scale absorption, where visible outliers can be seen across the channel dimensions. Such distribution makes it difficult for the quantizer to effectively allocate the quantization bins, where a large portion of its representation power would be wasted on the outliers. This is further supported by Figure 6, where the spiky distribution with large outliers is also found in other models and datasets. On the other hand, applying scale absorption leads to an even distribution across the mapped range (-128, 127). Such evenly spread out distribution is much more favorable for quantization, because the values can be mapped evenly across the bins, well utilizing the quantization levels and thus leading to minimized quantization error.

7 Conclusion

In this paper, we propose TopGQ, the first post-training quantization method for graph neural network quantization. TopGQ proposes to group vertices that share similar topological structure, which is measured using an adaptation of Wiener index to capture the local topology around each node. Then, TopGQ proposes the scale absorption method, which merges the scale parameters of quantization groups into a single scale, and the magnitude information is merged into an adjacency matrix for efficient computation. The extensive experimental results show that TopGQ shows better performance while having orders of magnitude faster compared to the baselines.

REFERENCES

Marco Arazzi, Marco Cotogni, Antonino Nocera, and Luca Virgili. Predicting tweet engagement with graph neural networks. In *Proceedings of the 2023 ACM International Conference on Multimedia Retrieval*, pp. 172–180, 2023.

Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. *Advances in neural information processing systems*, 33:17804–17815, 2020.

Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1): i47–i56, 2005.

Andrei Dragos Brasoveanu, Fabian Jogl, Pascal Welke, and Maximilian Thiessen. Extending graph neural networks with global features. In *The Second Learning on Graphs Conference*, 2023.

Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Congrui Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, et al. Spectral temporal graph neural network for multivariate time-series forecasting. *Advances in neural information processing systems*, 33:17766–17778, 2020.

Zhixian Chen, Tengfei Ma, Zhihua Jin, Yangqiu Song, and Yang Wang. Bigcn: A bi-directional low-pass filtering graph neural network. *Analysis and Applications*, 20(06):1193–1214, 2022.

Kanghyun Choi, Deokki Hong, Noseong Park, Youngsok Kim, and Jinho Lee. Qimera: Data-free quantization with synthetic boundary supporting samples. In *Advances in Neural Information Processing Systems*, 2021.

Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *IEEE/CVF International Conference on Computer Vision Workshop*, 2019.

Steve Dai, Rangha Venkatesan, Mark Ren, Brian Zimmer, William Dally, and Brucek Khailany. Vs-quant: Per-vector scaled quantization for accurate low-precision neural network inference. *Proceedings of Machine Learning and Systems*, 3:873–884, 2021.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1 (1):269–271, 1959.

Mucong Ding, Kezhi Kong, Jingling Li, Chen Zhu, John Dickerson, Furong Huang, and Tom Goldstein. Vq-gnn: A universal framework to scale up graph neural networks using vector quantization. *Advances in Neural Information Processing Systems*, 34:6733–6746, 2021.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pp. 417–426, 2019.

Boyuan Feng, Yuke Wang, Xu Li, Shu Yang, Xueqiao Peng, and Yufei Ding. Sgquant: Squeezing the last bit on graph neural networks with specialized quantization. In *2020 IEEE 32nd international conference on tools with artificial intelligence (ICTAI)*, pp. 1044–1052. IEEE, 2020.

Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345–345, 1962.

Lester R. Ford Jr. *Network Flow Theory*. 1956. Technical report, RAND Corporation.

Ante Graovac and Tomaž Pisanski. On the wiener index of a graph. *Journal of mathematical chemistry*, 8(1):53–62, 1991.

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

Man Hu, Dezhi Sun, Fucheng You, and Han Xiao. Hybrid structure encoding graph neural networks with attention mechanism for link prediction. In *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 417–424. IEEE, 2022.

H GaoandS Ji. Graph u-nets. In *Proceedings of the 36th International Conference on Machine Learning, in Proceedings of Machine Learning Research*, 2019.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016.

Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. Pinnersage: Multi-modal user embedding framework for recommendations at pinterest. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2311–2320, 2020.

Anh Viet Phan, Minh Le Nguyen, Yen Lam Hoang Nguyen, and Lam Thu Bui. Dgcnn: A convolutional neural network over large-scale labeled graphs. *Neural Networks*, 108:533–543, 2018.

Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2110–2119, 2018.

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.

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

Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14:347–375, 2008.

Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM (JACM)*, 9(1):11–12, 1962.

Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, Ruihao Gong, Shanghang Zhang, Qi Zhang, Fengwei Yu, and Xianglong Liu. Outlier suppression: Pushing the limit of low-bit transformer language models. *Advances in Neural Information Processing Systems*, 35:17402–17414, 2022.

Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 10737–10745, 2021.

Li Zhang and Haiping Lu. A feature-importance-aware and robust aggregator for gcn. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 1813–1822, 2020.

Yiming Zhang, Lingfei Wu, Qi Shen, Yitong Pang, Zhihua Wei, Fangli Xu, Ethan Chang, and Bo Long. Graph learning augmented heterogeneous graph neural network for social recommendation. *ACM Transactions on Recommender Systems*, 1(4):1–22, 2023.

Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. In *International Conference on Machine Learning*, 2019.

Zeyu Zhu, Fanrong Li, Zitao Mo, Qinghao Hu, Gang Li, Zejian Liu, Xiaoyao Liang, and Jian Cheng. A^2q: Aggregation-aware quantization for graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2022.

Table 7: Sensitivity study of TopGQ

Bit	Datasets	PROTEINS			NCI1		
	Hop size k	GCN	GIN	GraphSAGE	GCN	GIN	GraphSAGE
INT4	$k=1$	73.34	72.88	73.03	80.81	81.60	78.88
	$k=2$	76.05	74.61	74.22	80.86	81.84	79.10
	$k=3$	75.94	74.86	74.00	80.91	81.88	79.16
INT8	$k=1$	60.86	51.04	65.77	62.68	70.91	75.90
	$k=2$	66.06	63.96	67.01	66.14	77.33	76.50
	$k=3$	70.15	70.61	69.67	65.09	78.49	76.43

A CODE

The code, which includes our implementation of this work, is included in a zip archive of the supplementary material. The code is under Nvidia Source Code License-NC and GNU General Public License v3.0.

B ADDITIONAL EXPERIMENTAL SETTINGS

We report evaluation results on two representative graph processing tasks: Node-level classification, graph-level classification. For node-level classification, we compare the validation accuracy of Reddit, ogbn-proteins, and ogbn-products datasets in a transductive setting. Please note that we first conduct GNN quantization experiments on the dataset with this level of scale, thus further enlarging the field of GNN quantization. By following the experimental settings of baselines, we also conduct experiments using Cora, CiteSeer, and PubMed datasets in a transductive setting, which is the common setting for GNN quantization. Lastly, we further conduct a comparison of large-graph processing on Reddit, ogbn-proteins, and ogbn-products datasets. For graph-level classification, we choose PROTEINS and NCI1 datasets to evaluate the inductive inference performance of quantized GNNs.

We compare TopGQ with three graph quantization baselines using QAT approaches: Degree-Quant (Tailor et al., 2020), SGQuant (Feng et al., 2020), and A^2Q (Zhu et al., 2022). To ensure a fair comparison, we use fixed-precision quantization for both SGQuant and A^2Q when attaining experiment results. We report quantized accuracy of GCN (Kipf & Welling, 2016), GIN (Xu et al., 2019), and GraphSAGE (Hamilton et al., 2017) architectures with 4-bit and 8-bit integer quantization. For a fair comparison, we apply the same bitwidth for all layers, including aggregation and combination.

All experiments are conducted on a server with a single A6000 GPU, RTX 4090 GPU, and Intel(R) Xeon(R) Gold 6442Y CPU. We implement our algorithm on PyG library v2.6.0 with PyTorch v2.2.1. In the Wiener index computation time comparison, we use the SciPy library to measure the time of the baseline algorithm to compute the all-pair shortest-path metric.

For the ablation study, we present in Table 4, we first build baseline PTQ method, which applies a min-max quantization strategy to quantize graph neural networks without node grouping and scale absorption. For the case of using Indegree for the node grouping metric, we apply the same strategy with our method that uses the Wiener index by grouping the nodes having the same indegree value and quantizing them to share the same quantization parameters.

C FURTHER COMPARISON ON k-HOP WIENER INDEX

Here, we present further analysis of the Wiener index, including a sensitivity study regarding the hyperparameter k, which determines the diameter of the local subgraph.

Table 7 shows the sensitivity study on quantization accuracy regarding the hop size k. We compare the hop size $k=1$, $k=2$, and $k=3$, where the $k=1$ setting corresponds to the baseline, which is identical to the "indegree" setting in Table 4. For the PROTEINS dataset, increasing the hop size from $k=1$ to $k=2$ led to noticeable improvements across all models.

Table 8: Comparison of node-wise Wiener index computation time

k	Algorithm	Cora	CiteSeer	Pubmed	PROTEINS	NCI1	Reddit	ogbn-proteins	ogbn-products
2	Bellman-Ford	0.35s	0.46s	19.87s	13.02s	33.01s	4.21h	2.84h	34.64h
	Floyd-Warshall	0.15s	0.21s	4.68s	7.91s	2.97s	0.57h	0.41h	4.51h
	Dijkstra	0.19s	0.29s	2.70s	8.78s	2.56s	0.16h	0.11h	1.55h
	Ours	0.02s	0.01s	0.06s	0.29s	0.10s	0.0004h	0.0002h	0.0048h
	Speed Up	9.77×	31.65×	43.50×	27.12×	25.17×	412.23×	602.30×	322.37×
3	Bellman-Ford	3.63s	2.43s	216.76s	40.07s	512.13s	46.63h	30.09h	305.50h
	Floyd-Warshall	0.77s	0.31s	34.62s	8.50s	11.27s	5.75h	3.78h	35.44h
	Dijkstra	0.61s	0.39s	14.71s	12.75s	12.49s	1.02h	0.61h	8.52h
	Ours	0.52s	0.08s	1.82s	4.84s	1.78s	0.0155h	0.0065h	0.2855h
	Speed Up	1.18×	3.76×	8.10×	1.76×	6.32×	65.89×	93.39×	29.83×

Across both precision levels and datasets, a clear trend emerged where increasing the hop size from $k = 1$ to $k = 2$ generally improved performance for all architectures. This effect was particularly noticeable in the PROTEINS dataset under INT4 precision, where all architectures showed consistent gains. In the INT8 configuration, the same trend held, though the magnitude of improvements was more pronounced in some cases. Notably, GIN and GCN showed substantial increases in performance as the hop size increased from $k = 1$ to $k = 3$ for the PROTEINS dataset, while the NCI1 dataset saw more moderate gains. However, increasing the hop size further to $k = 3$ did not always lead to continued improvements.

We also conducted further sensitivity study regarding Wiener index computation time by varying the value of k, as increasing k results in more computational costs due to the larger diameters of each subgraph. Table 8 shows the comparison results of computation time.

For $k = 2$, our method shows remarkable speedups, often outperforming the other algorithms by significant margins, especially for larger graphs where it achieves up to several hundred times faster performance. At $k = 3$, while all methods take longer due to the increased complexity, our method continues to lead in performance, though the speedup is generally lower than for $k = 2$. Nonetheless, it maintains a strong advantage, especially in large-scale cases where traditional methods struggle with execution times. Overall, the trends show that our method provides consistent and substantial speed improvements.

From the experiments, we observe that $k = 2$ often provides the best balances of quantization accuracy and computation time. In addition, for better quantization accuracy, using $k = 3$ is also an option to choose.