An Efficient Segmented Quantization for Graph Neural Networks

Yue Dai[1*], Xulong Tang[1] and Youtao Zhang[1]

[1]Department of Computer Science, University of Pittsburgh, Pittsburgh, 15213, PA, United States.

*Corresponding author(s). E-mail(s): yud42@pitt.com;
Contributing authors: xulongtang@pitt.edu; zhangyt@cs.pitt.edu;

Abstract

Graph Neural Networks (GNNs) are recently developed machine learning approaches that exploit the advances in Neural Networks for a wide range of graph applications. While GNNs achieve promising inference accuracy improvements over conventional approaches, their efficiency suffers from expensive computation and intensive memory access in feature aggregation and combination phases, leading to large inference latency. Recent studies proposed mixed-precision feature quantization to address the memory access overhead. However, its linear approximation and computation complexity become the main constraints for the overall GNN accuracy and performance. In this paper, we propose segmented quantization to partition the feature range into segments and customize linear approximation within each segment based on original value density, and conduct efficient mixed-precision computing between quantized feature and full precision weights. Segmented quantization helps to achieve high inference accuracy while maintaining low computation complexity. We also devise the hardware accelerator to fully explore the benefits of segmented quantization. Our experiments show that up to 5% average accuracy and up to 6.8× performance improvements can be achieved over the state-of-the-art GNN accelerators.

Keywords: Graph Neural Network, Quantization, Accelerator

1 Introduction

Graph Neural Networks (GNNs) are recently developed machine learning approaches that adopt Neural Network processing in graph analysis applications (Hamilton, Ying, & Leskovec, 2017; Kipf & Welling, 2016; Thekumparampil, Wang, Oh, & Li, 2018; Veličković et al., 2017; Xu, Hu, Leskovec, & Jegelka, 2018), e.g., local community detection in social networks, and research correlation in publication databases. While GNNs achieve better accuracy over traditional graph analyzing solutions, they are both memory access intensive and computation-intensive, making it challenging to achieve high processing efficiency — on the one hand, the neural network processing requires intensive parallel computing and weight reusing; on the other hand, the non-euclidean graph topology leads to irregular memory accesses and large memory overhead.

Several prior works have been proposed to reduce the computation and memory overheads in GNNs. Graph sampling reduces the computation strength by processing sampled nodes and edges in the original graph (Hamilton et al., 2017). Developing hardware GNN accelerators

exploits massive parallelism during computation and memory accesses (Liang et al., 2020; Yan et al., 2020). Quantization reduces the memory intensity with reasonable accuracy loss (Feng et al., 2020). Specifically, quantization represents the original weights or features using fewer bits such that less computation and memory accesses are required. However, the approximation from quantization leads to an inevitable accuracy drop than the original models. To balance the trade-off between performance and computing costs, existing GNN quantization schemes quantize features rather than weights (Feng et al., 2020) since the amount of the feature data within GNNs is significantly larger than that of weights, making feature quantization more beneficial for computing cost, which the quantization on weights has much less effect on.

While these GNN quantization methods effectively improve memory performance by reducing the amount of feature data access from memory. It faces two main limitations. One is that GNN quantization adopts linear approximation during quantization, which exhibits significant approximation errors.

On the other hand, due to the power-law distribution of the node degree, node features in GNN might suffer irregular distribution due to the message aggregation. This makes existing non-linear quantization approaches unsuitable for GNN feature quantization. Keeping original weights helps to mitigate the issue but the overall approximation error remains non-negligible due to the feature quantization. Previous studies explore more fine-grained configuration for bit-depth, which potentially leads to the second issue. The other one is that such mixed-precision quantization leads to bit-gaps between different features and weights, and thus demands run-time rematching(i.e. quantization/dequantization) to support computing, which introduces extra computation overhead and slowdowns the overall processing. Therefore, we make our goal to solve two problems: How to decrease approximation error in GNN feature quantization without introducing more bit-depth? How to reduce the rematching overhead in GNN feature quantization?

In summary, We adopt segmented quantization in two considerations: 1) Feature access introduces the majority of computing overhead in GCN, thus raising the demand for quantizing node features instead of both features and weights to achieve the best trade-off between accuracy and efficiency. However, it is challenging to conduct mixed-precision computing without introducing rematching overhead. 2) Linear and non-linear quantization approaches are inadequate to handle the feature quantization in GNN since nodes have irregular connectivities in GCNs and produce the irregular distribution of intermediate results values.

To this end, we propose segmented quantization to address the above issues. Segmented quantization partitions the range of feature data into several segments and adopts different linear approximation functions within each segment. And the pre-computing scheme helps to reduce the rematching overhead from mixed-precision GNN quantization. We summarize our contributions as follows.

- We propose the segmented quantization strategy to address the approximation error in traditional linear quantization. Segmented quantization adopts linear approximation within a smaller range, which reduces the overall quantization error by mitigating the error within each segment.
- We propose to reduce the computation overhead by merging quantization in pre-computation, which combines precision conversion and full-precision multiplication to $O(q)$ shift-add operation where q is the number of quantized bits.
- We devise the hardware enhancements to support the segmented quantization with computation reduction. The proposed hardware is lightweight and can be easily integrated into general GNN accelerators.
- We evaluate the proposed segmented quantization. The experimental results indicate that our approach outperforms the state-of-the-art GNN accelerators with up to 5% average accuracy and up to 6.8× performance improvements.

2 Related Work

2.1 Graph Neural Networks

Graph Neural Networks (GNNs) adopts Neural Networks components in graph processing (Hamilton et al., 2017; Kipf & Welling, 2016; Thekumparampil et al., 2018; Veličković et al., 2017; Xu et al., 2018). To update a node in the

graph, a GNN requires the information not only from the node itself but also neighboring nodes that are one- or multiple- hops away. A typical GNN consists of multiple layers and, within each layer, a node first gathers the information and then combines the information and produces a latent representation, i.e., an aggregation phase and a combination phase. A GNN layer can be represented by the function in Equation (1).

$$H_u^l = \sigma(Comb(Aggr(H_u^{l-1}, H_v^{l-1}, W_a^l), W_c^l)), v \in N(u) \quad (1)$$

where $\sigma(\cdot)$ is the activation function, the H_u^l is the feature of node u in layer l, $N(u)$ is the neighbor set of node u, W_a^l and W_c^l are learnable weights. There are two main functions: the aggregation function $Aggr(\cdot)$, and the combination function $Comb(\cdot)$. The aggregation function accumulates the messages from neighbors to produce intermediate results while the combination function embeds the intermediate results further to the expected output of the layer.

There are two types of aggregation functions: (a) In GraphSage (Hamilton et al., 2017), the aggregation is done based on the graph structure, e.g., max/average/mean. There are no weight parameters in aggregation. (b) In GAT (Veličković et al., 2017), the aggregation can be represented using Equation (2), which contains the learnable attention weights W_a^l in Equation (3).

$$Aggr(H_u^{l-1}, H_v^{l-1}) = \sum_{v \in N(u)} \alpha_{u,v}^l H_v^{l-1} \quad (2)$$

where

$$\alpha_{u,v}^l = W_a^l \cdot (H_u^{l-1} \oplus H_v^{l-1}) \quad (3)$$

The combination function can be presented as Equation (4).

$$H_u^l = W_c^l V_u^l \quad (4)$$

The combination is performed on aggregation results V_u^l and always contains learnable weights W_c^l. H_u^l represents the output features of the current layer. Common combination functions include single-layer perceptron, multilayer perceptron (MLP), and RNNs.

Recent studies revealed that GNN are both computation-intensive and memory access intensive, making it challenging to achieve high processing efficiency (Yan et al., 2020). On the one hand, the neural network processing such as combination functions involves massive matrix multiplication and accumulation with reused weights; on the other hand, the non-euclidean graph topology leads to irregular memory accesses and large memory access latency, specifically, on feature data. For example, GCN has 11.6 DRAM byte per Ops during aggregation phase and 0.06 Dram byte per Ops during combination phase (Yan et al., 2020). The irregular memory access of node features, which suffers from poor locality issues, makes it a major bottleneck in GNN computing.

GNN Accelerators. Hardware-based GNN accelerators, e.g., HyGCN (Yan et al., 2020), EnGN (Liang et al., 2020), and several prior works (Auten, Tomei, & Kumar, 2020; Chen et al., 2021; Geng et al., 2020, 2021; Kiningham, Re, & Levis, 2020; Li, Louri, Karanth, & Bunescu, 2021) have been proposed to accelerate GNN processing. A GNN accelerator usually consists of tens to hundreds of processing units, each of which can operate independently on a subset of features for aggregation or combination processing. To achieve a good tradeoff between high processing precision and die cost, the processing units operate on 16/32-bit fixed-point values, and leave optimization spaces for quantizations.

2.2 Quantization in GNNs

Quantization is a widely adopted technique for achieving efficiency in CNN models (Han, Mao, & Dally, 2015; Hubara, Courbariaux, Soudry, El-Yaniv, & Bengio, 2016; Jacob et al., 2018; Long, Lee, Kim, & Mukhopadhyay, 2020; Marchisio et al., 2020; Qu et al., 2020; Wang, Liu, Lin, Lin, & Han, 2019; Zhu, Han, Mao, & Dally, 2016). Most CNN quantization approaches chose to represent full-precision weights with fewer bits such that the dot product computation can be accelerated at runtime, e.g., BNN (Hubara et al., 2016) represents weights using binary weights such that multiplication can be replaced by shifts in dot product computation.

To quantize the data, similar as those to quantize the inputs and intermediate data in CNNs (Wang et al., 2019), we need value conversion from the original precision to quantized values at runtime. That is, the quantization on H from its full-precision to q bits can be represented in Equation (5).

$$H' = \lfloor \frac{2^q}{a-b}(H-b) \rfloor \quad (5)$$

where H and H' as values before and after quantization, respectively; q is quantized bit-depth, a and b are the empirical/trained maximal and minimal values of H, respectively; and $\lfloor \cdot \rfloor$ is the floor function. It is worth mentioning that the floor function can be replaced by the round function for less quantization error. Quantization can effectively reduce the computing cost, specifically, data sizes, with fewer bits representations, however, it might bring an accuracy drop due to the floor function in Eq.5. To balance the trade-off between accuracy and efficiency, previous studies explored mixed-precision quantization that quantizes the DNN model with different bit-depth assignments to the weights based on sensitivity such as hessian measurement(Dong, Yao, Gholami, Mahoney, & Keutzer, 2019).

However, in GNNs, the importance of the features in different nodes and layers will vary based on the input graph. Therefore, it is challenging to decide the bit-depths of such input-dependent cases properly with hessian measurement.

To address such issue, (Feng et al., 2020) proposed an input aware GNN quantization scheme, that assigns different bit-depth to the feature quantization depends on inputs. However, various bit-depths leave quantized values on different bit-depth representations, which require dequantization first to enable computations between features and weights then quantization to data movement efficient format. Therefore, in this paper we want to: 1) reduce accuracy drop of quantization without involves more bit-depths; 2) accelerate inevitable rematching between feature and weights.

3 Motivation

In this section, we motivate our design by presenting the two major issues in GNN quantization: computation overhead and inference accuracy loss. Fig. 1(a) presents the generalized two-step linear quantization process. Given a set of original values in [b,a], the linear quantization approach first maps the values to a line through scaling $y = \alpha x + \beta$ where α is the slope and β is the zero point of the line. The mapping is linear so that, for a group of values, if they are clustered in [b, a], are also clustered on the scaling line. The linear quantization then divides the destination range to 2^q equally sized sub-ranges and using flooring to map all values in each sub-range to one quantized value. The second step adopts approximation and thus introduces quantization errors. The total error is the sum of the difference between the precise value on the scaling line (blue line in the figure) and the quantized value (black horizontal line in the figure).

- **Non-negligible accuracy loss.** Given quantization is a lossy process in CNN/GNN, it is critical to design good trade-offs between the inference accuracy and the quantization bit depth. Fig. 1(b) reveals the quantization error is proportional to the slope, or the range of y-axis. Given four values along the x-axis, if we use a scaling line with a large slope, which has the four values mapped to 0, 1, 2, and 3, respectively and each value takes the quantized integer value, i.e., there is zero quantization error. However, if we use a scaling line with a small slope, which has the four values mapped to 0 or 1, we would have two values exhibit 25% quantization error while two other values have zero error. This observation matches the naive practice, i.e., using more quantization bits tends to produce better accuracy. We will exploit this property in our design in the next section. While it is feasible to choose one quantization bit-depth for the whole network, recent studies have revealed that it is more beneficial to have mixed quantization. For example, (Wang et al., 2019) showed that different CNN layers have different ideal quantization bit depths, (Dong et al., 2019) designed a hessian-based strategy to assign bit-depth to different parts of the model. For GNN quantization, it is promising to adopt different quantization bits for nodes exhibiting different structural characteristics (Feng et al., 2020). However, mixed-precision quantization requires rematching (i.e. quantization/dequantization) steps to enable computations between values from different bit-depths. The amount of these rematching operations grows with the number of bit-depth deployed. And these scenarios limited the efficiency of arbitrary mixed-precision quantization and drove us to think about how to effectively reduce the error within the same precision. As

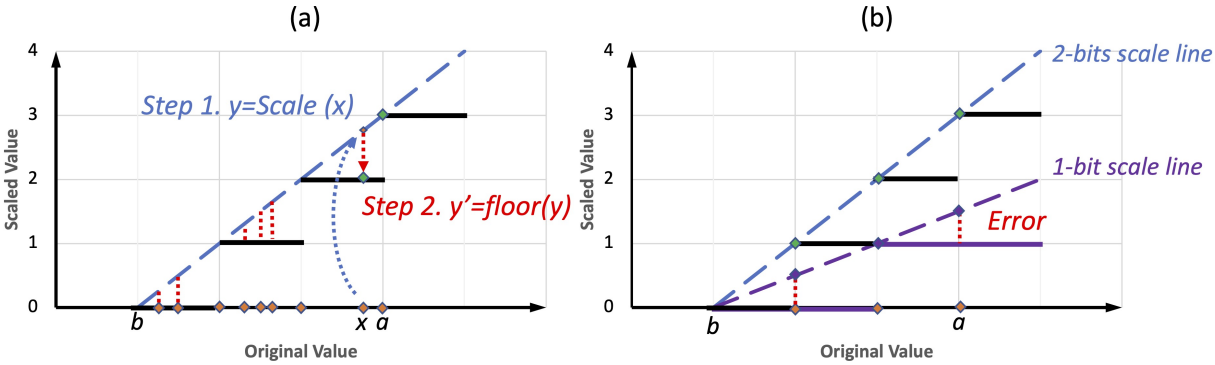


Fig. 1 (a) Linear quantization error. And (b) Comparison between 1-bit and 2-bits quantizations.

an alternative to linear quantization, (Imani et al., 2020) recently proposed non-linear quantization appraoch, to partition the value ranges to several non-equal intervals and represent all values in one interval with a typical value, e.g. cluster center. However, they focused on non-linear quantization on activation functions and deployed pairs of $< input\_table, output\_table >$ lookup tables to enable efficient computation for PIM (processing-in-memory) architecture, which is difficult to be adapted to GNN accelerators with a limited number of processing units. Another critical observation is, the sizes of features and weights are very imbalanced in GNNs. As shown in Fig 2, in two layers GNN models with hidden size as 16, weight sizes only take 0.5%-4% of all on the CORA dataset. Additionally, in GNN, though the feature access is irregular and suffered from poor locality, due to the randomness and sparsity of the graph structure; while the weight access is much regular with better locality since they are shared among all nodes within one layer. This indicates that we can focus on the quantization of the feature while keeping full precision weights to maintain original values as much as possible. Therefore, it could be valuable if we can come up with a quantization method that: on the one hand, uses non-linear quantization to reduce approximation error without introducing more bit-depths; on the other hand, reduces non-linear quantization overhead to fit it better to the memory-bottlenecked GNN feature quantizations.

- **Mixed-precision computation overhead.** GNN quantization adopts linear quantization rather than bit truncation, which tends to introduce extra computation overhead at run-time. To enable general computing, existing GNN quantization schemes (Feng et al., 2020) convert the quantized values to full precision values before computation with full-precision weights, and from the full precision results to quantized data after computation, which achieves better memory efficiency but leads to increased computation overhead at runtime. Additionally, the benefit of low-bit values on computation is under-exploited. Therefore, a computing scheme, that enables efficient computation between quantized features and full-precision weights is desired.

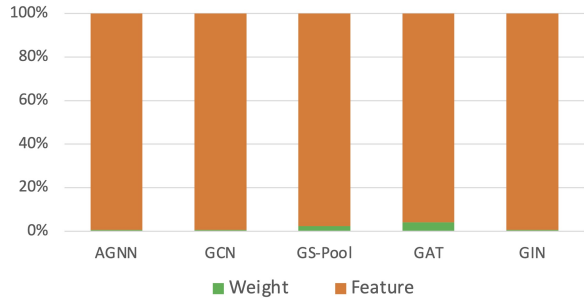


Fig. 2 Comparisons between the data size of features and weights in GNN models. The weights within GNN occupies less than 5% data size.

In summary, to achieve better accuracy and efficiency trade-off within GNN quantization, we expect an approach to i) reduce approximation error introduced by linear quantization and ii) conduct mixed-precision computing between quantized features and full-precision weights.

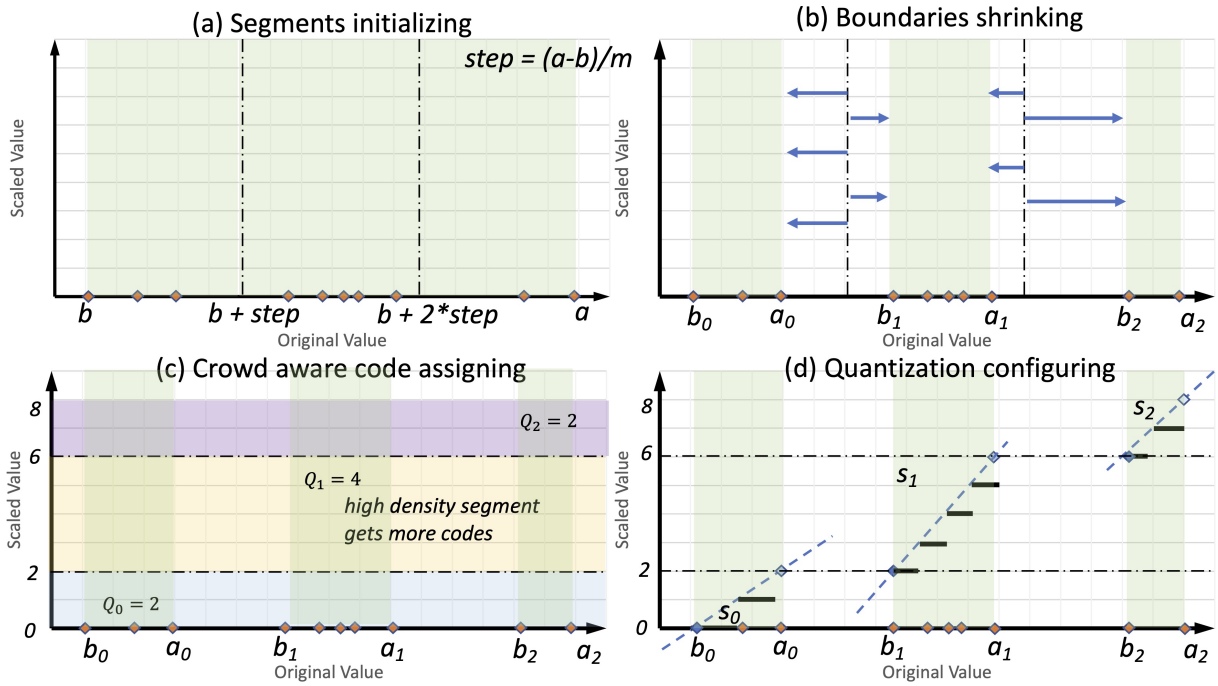


Fig. 3 Segmented quantization when m=3 and q=3. (a) The segment partition. (b) Boundary shrinking. (c) Code Assignment. (d) Determine the scaling line in each segment. There are 3 segments: #1 as leftmost, #2 as middle and #3 as rightmost green field.

4 Design

In this section, we first present the segmented quantization design for reducing quantization errors. We then discuss its application for reducing computation overhead at runtime. At last, we elaborate the hardware design to fully explore segmented quantization in GNN accelerators.

4.1 The Segmented Quantization

As discussed in section 3, non-linear quantization can reduce approximation error effectively yet less suitable for the GNN accelerators. To balance between accuracy and efficiency, we adopt segmented quantization, to leverage the benefit from non-linear scaling while keeping space efficiency of linear quantization. Intuitively, we split original value range and bit levels into smaller segments, and calibrate each segment separately, so that more accurate scaling can be accomplished.

We depict the quantization steps in Fig. 3, our proposed segmented quantization works as follows: a) It first partitions the feature range to m (m >1) segments(shown as Fig. 3(a)). b) Then it shrinks the original value boundary based on the statistical minimum and maximum values (shown as Fig. 3(b)), as the equally splitting does not guarantee the real statistical boundaries. c) Next it adopts crowd aware code assignment based on the density of each segment (shown as Fig. 3(c). d) In the end the quantization scaling lines are determined by boundaries and codes of the segments (shown as Fig. 3(d)). By controlling the hyper-parameter m, we can balance the trade-off between linear and non-linear quantizations. During quantization we using the statistic updated scaling lines to quantize the full precision input values to q-bit quantized values. Assume we have $m = 3$ and $q = 3$ in the following discussion.

For each segment, we record not only its full-precision value range but also the range of the quantized values, e.g., segment #1 is to quantize full-precision values in $[b_1, a_1]$ to integer values in [2,5], which can be used to determine its scaling line. For quantization, given a full-precision value, we use the segment boundary to determine its corresponding segment index and then use the segment's scaling line to determine its quantized integer value. For de-quantization similarly, we reverse the process after finding its segment

index and scaling line. Next, we discuss the design challenges in segmented quantization.

- **Crowd aware code assigning.** It is critical to determine the number of encoded quantization levels for each segment. When we use q-bit quantization, the quantized values from all segments share the 2^k codes, and these codes are equally distributed along the y-axis(scaled original value). However, the number of quantization codes can be dynamically assigned to different segments. For example, we may assign the quantized value ranges [0,1], [2,5], and [6,7] to the three segments, respectively, i.e., values in segment #1 use four codes while values in segment #0 use two codes. Alternatively, we may assign more codes to segment #0 if we have more values in segment #0.

  One key observation in Fig. 1(b) is that the approximation error is linear to the sum of the difference between the scaled value and its quantized value. The error is also linear to the scaled range. Assigning more codes in one segment can effectively reduce the total approximation error in that segment. Therefore, in this paper, we adopt a simple crowd-aware code assignment strategy, i.e. we assign more codes to the segment that contains more full-precision values. Specifically, we stats the number of inputs dropped into each segment during training, and assign the codes based on the value density of each segment. The code number assignment follows Equation (6) during the training,

$$Q_i = Max((\frac{C_i}{N} \times Q_N), 2), \forall i \in 1, 2, ..., m \quad (6)$$

  where the Q_i as code number assigned to segment i, N as the total number of original values and C_i as the number of original values fall into segment i, Q_N as the total number of bits can be used for quantization(i.e. quantization bit depth). Hereby doing this, we can assign more codes to the crowed segments and therefore reduce approximation error from flooring, as more floors are assigned to them. We also use the $Max(\cdot)$ function to ensure that each segment receives at least two code numbers. The reason is that nodes with degrees usually produce (i.e., accumulate) large but infrequent features, which are few but critical for their neighbors. We use the $Max(\cdot)$ function to avoid high errors on these nodes to ensure these features can be appropriately quantized. This strategy works well in our experiments. We shall evaluate other assignment strategies in future work.
- **Determine the scaling line in each segment.** To find the segment index before quantization and decide scaling line, it is important to record the segment boundaries, e.g., x=b+$step$ separates segments #0 and #1. In the segmented quantization, we initialize the segment size to be $1/m$ of the input value range, as shown in Fig. 3(a). Then we shrink the boundaries based on the statistical minimum and maximum values of each segment to get their real original value ranges. Next, we would apply the Equation (6) to assign the codes to each segment. To determine the scaling line for each segment, we utilize the same strategy as linear quantization described in Section III for each segment. As shown in the figure, the scaling line depends on both the smallest and the largest real values in the range and the number of codes in each segment. To ensure full coverage for the future input feature, the segment boundaries for the segment index looking up remain unchanged.

4.2 Quantization-aware Computing

After adopting the linear quantization for features, a quantized GNN contains full precision weight data and quantized feature data such that it needs a dequantization step to convert quantized features to full precision values and then conduct the full precision computation (in particular, multiplication) between weights and features. This process can be formulated as Equation (7).

$$V = W \times H = W(\frac{a-b}{2^q}H' + b) \quad (7)$$

where V is the full precision output; W and H' are the full-precision weight and quantized feature, respectively; q is the bit-depth; and a, b are the maximum and minimum values. Comparing to the multiplication before and after the linear quantization, adding the dequantization step requires extra computation cycles. Assume a GNN accelerator adopts a typical shift-add multiplication implementation on 32-bit fixed-point values,

the full-precision multiplication finishes after 32 shift-add operations while dequantization adds 32+1=33 extra operations.

We study the process and observe that the low efficiency comes from two factors: a) it does not explore the fact that fewer bits are used in quantized values; and b) dequantization is expensive at runtime. We, therefore, propose to reduce computation strength through pre-computation.

Strength reduction through pre-computation. The computation in Equation (7) can be transformed as follows.

$$V = W(\frac{a-b}{2^q}H' + b) = AH' + B \qquad (8)$$
$$A = \frac{a-b}{2^q}W \quad , \quad B = Wb$$

Given a, b, q, and W are constant values, we can pre-compute A and B so that adopting linear quantization would require 32+1=33 operations at runtime, or only one extra operation comparing to the baseline. In addition, the multiplication AH' in the equation involves a full-precision value A and a q-bit quantized value H', which can be implemented using q shift-add operations, rather than 32 operations in the baseline. To summarize, a multiplication with linear quantization can be implemented in q+1 operations, achieving a big improvement from 65 operations in the baseline.

4.3 The Hardware Design

In this section, we fully exploit the potentials from segmented quantization in hardware. Fig. 4 illustrates the extra hardware on top of a generic GNN accelerator EnGN (Liang et al., 2020). The same design principle can be applied to other GNN accelerators (Auten et al., 2020; Chen et al., 2021; Geng et al., 2020, 2021; Kiningham et al., 2020; Li et al., 2021; Yan et al., 2020).

In the figure, the GNN accelerator consists of an array of 128×16 PEs (processing units), a memory controller, and several buffers (feature buffer, weight buffer, edge buffer, and output buffer). Each PE contains one 32-bit ALU and implements multiplication using shift-adding. The memory controller is responsible for scheduling the accesses to the DRAM module and for parallelizing the computation and memory accesses. While each PE consists of several registers to enable faster processing, the accelerators integrate several on-chip

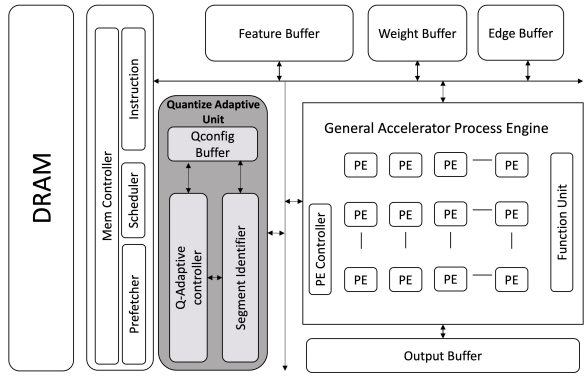


Fig. 4 An overview of our quantization adaptive hardware design. To support the quantization aware computing, the additional Quantization Adaptive Units (Grey Field) are added to the baseline architecture.

global buffers to cache frequently used feature and weight data. The results are temporarily buffered in the output buffer to enable parallel processing.

We enhance the accelerator with three hardware components: Q-config buffer, Q-adaptive controller, and Segment index detector (SID).

- **Q-config buffer**. Given the segment index and its specific approximation line in segmented quantization, we integrate a Q-config buffer so that it holds the meta-data of each segment, i.e., the segment value range in full-precision, its quantized value range, the A and B values in Equation (8).
- **Q-adaptive controller**. Segmented quantization, while converting a large number of multiplication operations using Equation 8, still need to perform many full-precision operations, e.g., the aggregation as shown in Equation 2. For this purpose, we integrate the Q-adaptive controller to determine the correct computing model, control the PEs and operand loading, and the timing to start/stop individual operations.
- **Segment Index Detector (SID)**. After fetching a quantized feature value v from memory, we need SID to determine its segment index according to the quantization for that feature. Since the maximal q and m values in our experiments are 4 and 4, respectively, we implement each SID as a 32-bit register and populate the 2-bit segment index for each of 16 quantized values. We then right-shift $2v$ bits and pick up the lowest order 2-bit as the segment index. Given EnGN can enable parallel processing of 128 new features, we need 128 32-bit registers in SID.

We evaluate the extra hardware cost in the accelerator. The additional hardware cost comes mainly from our Q-buffers, which are relatively tiny compare to overall buffers and PEs. Based on evaluation on Cacti (Thoziyoor, Ahn, Monchiero, Brockman, & Jouppi, 2008), the total overhead is less than 1% and thus is very modest.

5 Experiments

5.1 Experiment Setup

We evaluated our method to answer the following research questions:

- Does the segmented quantization improve inference accuracy in GNN quantization?
- Does the pre-computing scheme improve runtime performance in GNN quantization?
- How much is the overhead brought by our quantization scheme?
- How does the performance affected by the choices of segment numbers?

GNN model and datasets. To evaluate the proposed designs, we implemented five commonly used GNN models — GCN(Kipf & Welling, 2016), GraphSage-Pool(GS-Pool)(Hamilton et al., 2017), AGNN(Thekumparampil et al., 2018), GAT(Veličković et al., 2017), and GIN(Xu et al., 2018). The models were designed for semi-supervised node classification tasks. We used 2 layers as described in the original papers with hidden size as 16. We compared their accuracy and performance using four widely used datasets: CORA (CR), CiteSeer (CS), PubMed (PB), Amazon-Computers (AMAZON) when adopting different full-precision and quantization schemes. The statistical details of the datasets are shown in Table 1. In our experiment, we used segment number $m = 3$. We evaluated the accuracy under i)full-precision model, ii) uniformly quantized model in which both features and weights are quantized linearly, iii) linear quantized model in which only features are quantized linearly and iv) segmented model in which features are quantized by segmented quantization. To fully exploit the potential of different quantization schemes, we employed quantization aware training (Park, Yoo, & Vajda, 2018) for each quantized model. For each training epoch, we follow three steps of computing: (1) Configure quantization parameters based on training data (i.e., detect scale and zero point of each segment). (2) Compute GNN inference output with quantized features. (3) Update GNN weights with gradient calculated in full precision backward propagation.

Accelerator baseline and simulation. For performance evaluation, we used a cycle-accurate in-house simulator. We set EnGN (Liang et al., 2020) as the baseline and then enhanced its hardware components to enable segmented quantization. The system configurations are shown in table 2. For comparison purposes, we simulated inference execution for q=3.

Table 1 Dataset Details

	#Nodes	#Edges	#Features
CORA (CR)	2,708	10,556	1,433
CiteSeer (CS)	3,327	9,104	3,703
PubMed (PB)	19,717	88,648	500
Amazon Computer (AMAZON)	13,381	245,778	767

Table 2 System Configuration

	HyGCN	EnGN
Baselines	1GHz@32 SIMD 16 cores and 32X128 arrays, 22MB+128KB Buffer, 256GB/s HBM 1.0	1GHz@128X16 arrays, 32 PE units VPU, 1600KB Buffer, 256GB/s HBM 2.0
Quantization Units	128KB Q-Buffer, 128 Comparison Units	128KB Q-Buffer, 128 Comparison Units

5.2 Inference Accuracy

We first compared the top-1 inference accuracy when using different schemes, i.e., full-precision `full`, uniform (`uniform`), linear quantization (`linear`), and segmented quantization (`segmented`). We summarized the results in Table 3 when using 3-bit and 4-bit quantization, respectively.

From the table, we observed that `segmented` achieved overall better accuracy over `linear` and `uniform`. `uniform` over 3-bit and 4-bit had higher accuracy drop comparing to `linear` and `segmented`, which indicated the effectiveness of feature quantization in GNNs. `segmented` showed overall better accuracy from `linear`, which indicated the improvement from our proposed method. It is worth mention that the GIN models showed overall lower accuracy since they

Table 3 Accuracy of full-precision, uniform quantization, linear quantization and segmented quantization applied models. The quantized bits are $q = 3$ and $q = 4$.

Dataset	Model	Full	Uniform (3bit)	Linear (3bit)	Segmented (3bit)	Uniform (4bit)	Linear (4bit)	Segmented (4bit)
CR	AGNN	0.812	0.768	0.778	**0.812**	0.8	**0.82**	0.818
	GCN	0.802	0.77	0.776	**0.792**	0.768	0.792	**0.806**
	GS-Pool	0.814	0.77	0.796	**0.804**	0.798	0.8	**0.806**
	GAT	0.828	0.762	**0.822**	0.788	0.812	0.818	**0.83**
	GIN	0.752	0.21	0.538	**0.634**	0.522	0.66	**0.734**
CS	AGNN	0.722	0.686	0.682	**0.692**	0.688	**0.718**	0.704
	GCN	0.73	0.69	**0.702**	0.692	0.702	0.728	**0.732**
	GS-Pool	0.7	0.698	0.702	**0.71**	0.702	0.706	**0.71**
	GAT	0.724	0.674	**0.728**	0.704	0.708	0.718	**0.732**
	GIN	0.612	0.214	0.308	**0.47**	0.29	0.498	**0.56**
PB	AGNN	0.782	0.728	0.752	**0.76**	0.752	0.764	**0.774**
	GCN	0.784	0.698	0.74	**0.764**	0.764	0.772	**0.786**
	GS-Pool	0.762	0.73	0.71	**0.738**	0.734	0.732	**0.752**
	GAT	0.768	0.714	0.722	**0.748**	0.736	0.754	**0.768**
	GIN	0.74	0.414	0.428	**0.598**	0.424	0.612	**0.664**
AMAZON	AGNN	0.808	**0.732**	0.35	0.656	**0.718**	0.488	0.7
	GCN	0.882	**0.836**	0.748	0.806	0.826	0.842	**0.874**
	GS-Pool	0.914	0.438	0.91	0.91	0.392	**0.924**	0.92
	GAT	0.898	0.698	0.846	**0.852**	0.882	**0.892**	0.882
	GIN	0.656	0.268	0.296	**0.346**	0.262	0.372	**0.602**

are originally designed for graph level classifications, therefore less fitted to the datasets of node-classification tasks. However, due to its unique message-passing scheme, we included it in our evaluation.

Comparing to `full`, the 3-bit and 4-bit `segmented` had 3.5%-4.5% and 0.5%-1.8%, respectively, average accuracy loss for citation datasets. These results represented an improvement from 5.9%-9.6% and 2.5%-4.0% accuracy loss in 3-bit and 4-bit with `linear`. The average accuracy improvements on the same dataset achieved up to 5%.

`Linear` had a larger accuracy loss when choosing q=3 for PubMed. This is because PubMed has a higher edge density so that feature values tend to be too close to be differentiated by fewer-bit quantization. As we discussed in Section 3, choosing $q = 4$ helps to gain large improvement in `linear` on densely connected graphs such as that in PubMed. In comparison, `segmented` achieved less accuracy drop when decreasing q=4 to q=3 for PubMed. We further evaluated the sensitivity of accuracy to more bit-depths choices. The results are shown in Fig. 5. It indicated that `segmented` has overall better accuracy on different bit-choices. And `segmented` suffered the least accuracy drop with fewer bits, which indicated it is also more robust to the aggressive quantization compare to `uniform` and `linear`. This is because `segmented` can allocate more codes to the segment that has more values, which reduces the overall approximation error. There is also an observation that in some cases the GCN with quantized features (i.e., `linear` and `segmented`) perform better than the full precision feature. It is because the quantizations potentially remove minor noises in the features. This phenomenon shows the benefit of quantizing features only and further motives mix-ed precision computing within GNN quantizations.

For the Amazon dataset, 4-bit `segmented` exhibits no clear accuracy improvement over `linear` and `uniform`. For the cases that `linear` outperforms `segmented`, `linear` has a higher or close to the baseline accuracy. This indicates that the quantization bit-depth is sufficient, it might be possible to choose a smaller q in practice.

We conduct paired t-tests on uniform, linear, and segmented quantization results. While using 3-bit quantization, the p-value between segment and uniform is 0.006, and the p-value between segment and linear is 0.008; while using 4-bits quantization, the p-value between segment and uniform is 0.005, and the p-value between segment and linear is 0.01. In summary, segmented quantization shows accuracy improvements when bit-level is insufficient to represent original values in crowd ranges, thanks to its dynamically arranged and segmented fitting design.

5.3 Performance

We then compared the runtime performance from different schemes and summarized the results (when q=3) in Fig. 6. Specifically, `uniform` uniformly quantized features and weights with

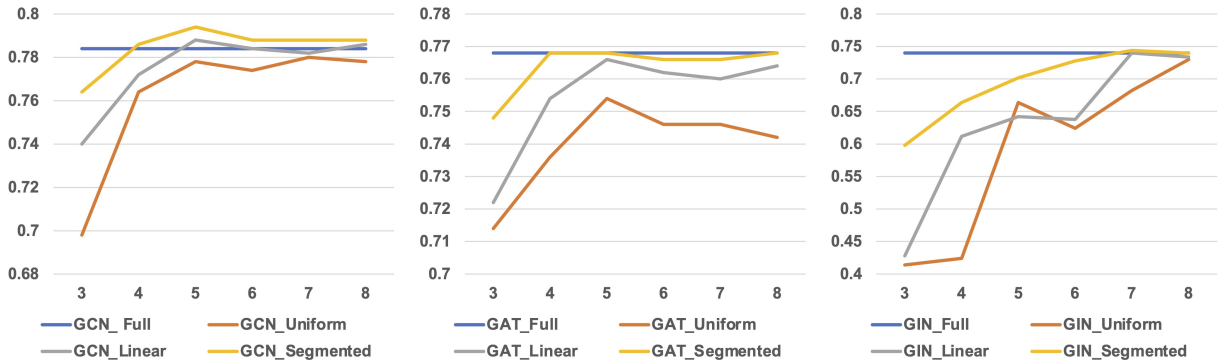


Fig. 5 Accuracy on different quantization bit-depth choices of GCN(left), GAT(middle) and GIN(right) on PubMed(PB) dataset, where y-axis as accuracy, x-axis as bit-depth.

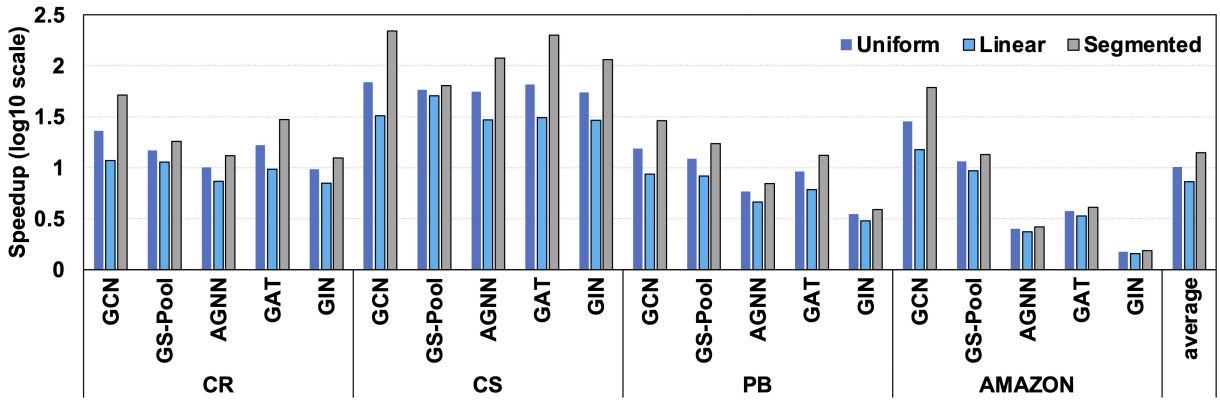


Fig. 6 Performance comparison (when q=3) in speedup (log scale).

no rematching computing during runtime and no quantization-aware computing (i.e., no shift-adding but 32-bit Fixed-point operation as computed by baseline accelerators); `linear` quantized feature only with no pre-computing and thus conducted rematching computing during runtime and conducting full-precision GNN computing; Furthermore, `segmented` models equipped segmented quantization with segmented weights and pre-computing schemes. The results were normalized to the full-precision baseline and plotted on the log scale.

From the figure, `uniform`, `linear` and `segmented` achieved up to 69×, 32× and 219× speedups over the full-precision baseline, respectively. That is, `segmented` achieved 1.1×-6.8× speedups over `linear`. All of the quantization schemes benefited from less bit representation and shown less amount memory access. `linear` performed worse than `uniform`, with 7.3× and 10.2× average speedup. respectively. The reason is that it suffered from additional rematching computations. And `segmented` reduced the computing cost of these rematching and lead to higher speedup than both `linear` with 14.1× average speedup, which demonstrated the effectiveness of the pre-computing mechanism. Also, our quantization-aware computing leads to higher speedup over `uniform` since it costs only three shift-adding operations to calculate full-precision outputs instead of fixed 32-bits operations adopted by baselines.

The improvements for the CiteSeer dataset are larger because this dataset has a large feature-length and low edge density, leading to more multiplication than accumulation operations. In contrast, the improvements for the AMAZON dataset are relatively low because this dataset has high edge density and short feature-length, whose aggregation operations demand more full precision rather than mixed-precision operations. Among models, GCN and GAT showed the best

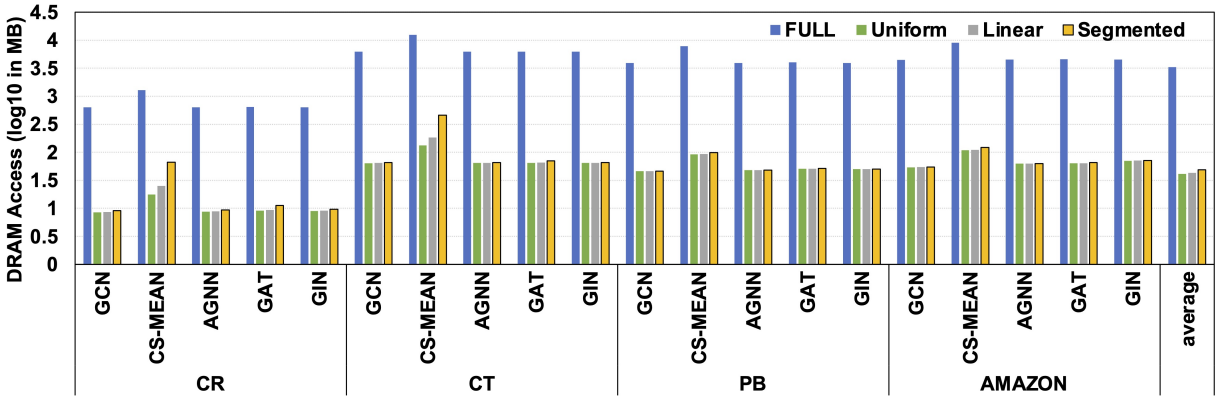


Fig. 7 Performance comparison (when q=3) in DRAM access (log scale).

performance improvements because GAT requires heavy feature/weight computation and thus benefits more from segmented quantization. For GCN, the optimization such as stage re-ordering (Liang et al., 2020) decreases aggregation on the same precision values, leading to more feature combining operations.

We also implemented segmented quantization using HyGCN as the baseline GNN accelerator. We observed 1.6× to 1.9× performance improvements over linear quantization. GCN was designed mainly to speed up GNN models that have skewed aggregation computation so that the pre-computation optimization has fewer opportunities.

To evaluate the overhead of the method, we further look into the DRAM access, shown in Fig. 7. Note we use log values to give a more normalized view of the access amount. `uniform` achieved best memory access reduction with 41.4MB average access compare to the full precision model as `full` with 3327.12MB average access. `linear` and `segmented` had larger amounts of accesses due to full-precision weights with 43.08MB and 48.98MB average access, respectively. Though `segmented` suffers most memory access due to pre-computed weights A and B in Equation. (8), the overhead is small. In terms of access reduction from `full`, `segmented` had 0.2% less reduction compared with the `uniform`) on average. The GAT and GraphSage suffered the most overhead since the GAT takes more weights for multi-head attention, and Sage dataflow is not adaptive to the ENGN reordering dataflow, which is why it had redundant weight access.

Concerning the potential overhead brought by more segments with scaled pre-computed weights, next, we evaluated the sensitivity of runtime and memory access to the choices of the segment number m. We evaluated the speed up and DRAM access changes with different segment number m selections. The results are shown in Fig. 8. Though more segments led to less speedup, the changes were tiny. For example, there were at most 0.1× speedup drop in GCN and less than 0.01× speedup drop in GAT when $m = 8$. The speedup over full precision baseline is 60.89× and 4.08× when $m = 8$, which still outperformed $uniform$ and $linear$ on the same bit-depth. Additionally, the DRAM access increases are also endurable. For example, with $m = 8$ there were 0.49MB and 2.15MB weight accesses in GCN and GAT, respectively. These weight accesses are very few compared to larger feature accesses, which were 54.23MB and 63.99MB in GCN and GAT, respectively. This demonstrates our assumption that the acceptable weight increment in GNNs would not have an unacceptable impact on the performance due to the weight sharing and unbalanced ratio between feature and weight sizes.

To summarize, the proposed segmented quantization with pre-computation optimization effectively reduces the inference latency for a wide range of GNN models and is generally applicable to different GNN accelerators.

6 Discussions

Studies towards quantization have recently gained momentum in reducing the memory and computing overhead of Deep Neural Networks(DNN). The

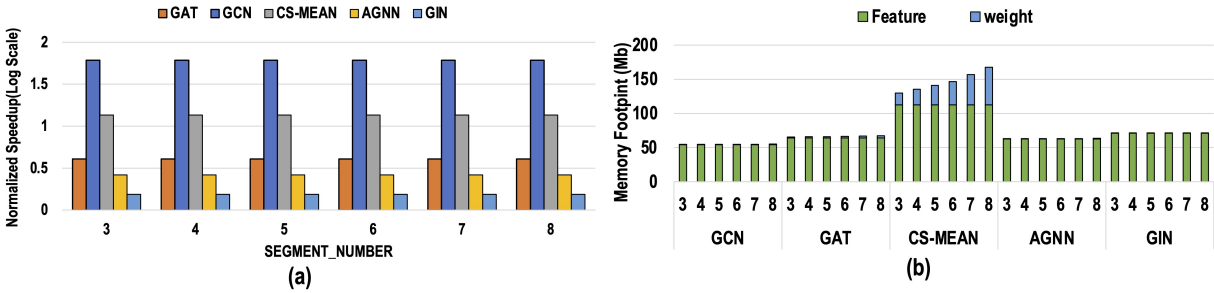


Fig. 8 (a) Speedup(Log10 Scaled) of GNNs using our approach over full-precision baseline and (b) Memory footprint(MB) of GNNs using our approach, in different segment number m.

latest works focus on non-linear quantization to better distribute the given bit levels and cause less approximation loss. (Miyashita, Lee, & Murmann, 2016; Zhou, Yao, Guo, Xu, & Chen, 2017) propose power-of-two quantization approaches which assign more bits around middle points of the original value range. (Fang et al., 2020) divides the original value range into two dense central regions and two sparse tail regions. However, the approaches target the bell-like distributed original data, therefore less adaptive to various data distributions. In contrast, the segmented quantization adapts to various distributions as there is no predefined density for bit-level assignment. (Liu et al., 2021) searches optimal value range among pieces in the original range and adopts mixed-precision quantization between layers. However, the approach only uses the optimal range instead of a set of ranges (i.e., segments), which might introduce inductive biases that limit generalizability. Compared to these approaches, our approach shows better flexibility to various distributions of full-precision feature values and potentially performs better. Moreover, the segmented quantization supports mixed-precision computing between quantized features and full-precision weights of GNNs, which are not considered in prior quantization approaches.

We plan to exploit segment configuration schemes in the future works. Given the differences between nodes within the same graph, recent works adopt different quantizing configurations (e.g. bit after quantization) for different nodes (Feng et al., 2020; Tailor, Fernandez-Marques, & Lane, 2020). These studies demonstrated that the optimal configurations of segments (i.e. how many segments m are used) can also vary across the nodes within the same graph to gain the best efficiency and accuracy trade-off. Therefore, we will further explore the optimized configuration scheme for the segmented quantization by leveraging static node patterns (e.g. degree of the nodes).

7 Conclusion

In this paper, we investigated the challenges in GNN quantization and proposed segmented quantization, a novel GNN quantization method that effectively reduces the quantization error and cost for GNNs. Targeting a more accurate and efficient GNN quantization scheme, we leveraged segmented scaling to exploit more accurate quantized value representation with non-linear scaling, we further applied a pre-computing scheme to improve the performance of mixed-precision quantizations in GNNs. Additionally, we proposed a hardware design to cooperate with our quantization scheme. We demonstrated its general applicability by integrating the proposed design on different GNN accelerators. Our experimental results showed that the scheme achieves significant accuracy and performance improvements over the state-of-the-art.

8 Declarations

All authors certify that they have affiliations with University of Pittsburgh.

References

Auten, A., Tomei, M., Kumar, R. (2020). Hardware acceleration of graph neural networks. *2020 57th acm/ieee design automation conference (dac)* (pp. 1–6).

Chen, X., Wang, Y., Xie, X., Hu, X., Basak, A., Liang, L., ... others (2021). Rubik: A hierarchical architecture for efficient graph neural network training. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Dong, Z., Yao, Z., Gholami, A., Mahoney, M.W., Keutzer, K. (2019). Hawq: Hessian aware quantization of neural networks with mixed-precision. *Proceedings of the ieee/cvf international conference on computer vision* (pp. 293–302).

Fang, J., Shafiee, A., Abdel-Aziz, H., Thorsley, D., Georgiadis, G., Hassoun, J.H. (2020). Post-training piecewise linear quantization for deep neural networks. *European conference on computer vision* (pp. 69–86).

Feng, B., Wang, Y., Li, X., Yang, S., Peng, X., Ding, Y. (2020). Sgquant: Squeezing the last bit on graph neural networks with specialized quantization. *2020 ieee 32nd international conference on tools with artificial intelligence (ictai)* (pp. 1044–1052).

Geng, T., Li, A., Shi, R., Wu, C., Wang, T., Li, Y., ... others (2020). Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing. *2020 53rd annual ieee/acm international symposium on microarchitecture (micro)* (pp. 922–936).

Geng, T., Wu, C., Zhang, Y., Tan, C., Xie, C., You, H., ... Li, A. (2021). I-gcn: A graph convolutional network accelerator with runtime locality enhancement through islandization. *Micro-54: 54th annual ieee/acm international symposium on microarchitecture* (pp. 1051–1063).

Hamilton, W., Ying, Z., Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, *30*.

Han, S., Mao, H., Dally, W.J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y. (2016). Binarized neural networks. *Advances in neural information processing systems*, *29*.

Imani, M., Razlighi, M.S., Kim, Y., Gupta, S., Koushanfar, F., Rosing, T. (2020). Deep learning acceleration with neuron-to-memory transformation. *2020 ieee international symposium on high performance computer architecture (hpca)* (pp. 1–14).

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2704–2713).

Kiningham, K., Re, C., Levis, P. (2020). Grip: a graph neural network accelerator architecture. *arXiv preprint arXiv:2007.13828*.

Kipf, T.N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Li, J., Louri, A., Karanth, A., Bunescu, R. (2021). Gcnax: A flexible and energy-efficient accelerator for graph convolutional neural networks. *2021 ieee international symposium on high-performance computer architecture (hpca)* (pp. 775–788).

Liang, S., Wang, Y., Liu, C., He, L., Huawei, L., Xu, D., Li, X. (2020). Engn: A high-throughput and energy-efficient accelerator for large graph neural networks. *IEEE Transactions on Computers*, *70*(9), 1511–1525.

Liu, Z., Wang, Y., Han, K., Zhang, W., Ma, S., Gao, W. (2021). Post-training quantization for vision transformer. *Advances in Neural Information Processing Systems*, *34*, 28092–28103.

Long, Y., Lee, E., Kim, D., Mukhopadhyay, S. (2020). Q-pim: A genetic algorithm based flexible dnn quantization method and application to processing-in-memory platform. *2020 57th acm/ieee design automation conference (dac)* (pp. 1–6).

Marchisio, A., Bussolino, B., Colucci, A., Martina, M., Masera, G., Shafique, M. (2020). Q-capsnets: A specialized framework for quantizing capsule networks. *2020 57th acm/ieee design automation conference (dac)* (pp. 1–6).

Miyashita, D., Lee, E.H., Murmann, B. (2016). Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*.

Park, E., Yoo, S., Vajda, P. (2018). Value-aware quantization for training and inference of neural networks. *Proceedings of the european conference on computer vision (eccv)* (pp. 580–595).

Qu, S., Li, B., Wang, Y., Xu, D., Zhao, X., Zhang, L. (2020). Raqu: An automatic high-utilization cnn quantization and mapping framework for general-purpose rram accelerator. *2020 57th acm/ieee design automation conference (dac)* (pp. 1–6).

Tailor, S.A., Fernandez-Marques, J., Lane, N.D. (2020). Degree-quant: Quantization-aware training for graph neural networks. *arXiv preprint arXiv:2008.05000*.

Thekumparampil, K.K., Wang, C., Oh, S., Li, L.-J. (2018). Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*.

Thoziyoor, S., Ahn, J.H., Monchiero, M., Brockman, J.B., Jouppi, N.P. (2008). A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. *ACM SIGARCH Computer Architecture News*, *36*(3), 51–62.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

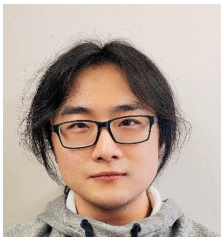
Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S. (2019). Haq: Hardware-aware automated quantization with mixed precision. *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 8612–8620).

Xu, K., Hu, W., Leskovec, J., Jegelka, S. (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Yan, M., Deng, L., Hu, X., Liang, L., Feng, Y., Ye, X., ... Xie, Y. (2020). Hygcn: A gcn accelerator with hybrid architecture. *2020 ieee international symposium on high performance computer architecture (hpca)* (pp. 15–29).

Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y. (2017). Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*.

Zhu, C., Han, S., Mao, H., Dally, W.J. (2016). Trained ternary quantization. *arXiv preprint arXiv:1612.01064*.



Yue Dai received the B.S. degree from Beihang University, Beijing, China in 2014, and M.S. degree from University of Maryland, Maryland, USA in 2017. He is currently pursuing the Ph.D. degree in computer science, University of Pittsburgh. His interests include computer architecture and artificial intelligence.

Xulong Tang is currently an assistant professor in the Department of Computer Science, University of Pittsburgh. He received his Ph.D. degree in Computer Science and Engineering from The Pennsylvania State University in 2019. His research interests include high-performance computing, advanced computer architecture designs, and compilers.

Youtao Zhang (Member, IEEE) received the B.S. and M.E. degrees from Nanjing University, Nanjing, China, in 1993 and 1996, respectively, and the Ph.D. degree in computer science from the University of Arizona, Tucson, AZ, USA, in 2002. He is currently an Associate Professor of computer science with the University of Pittsburgh, Pittsburgh, PA, USA. His current research interests include computer architecture, program analysis, optimization, on-chip interconnection, architectural support for security, new memory technologies, and networks-on-chip. Prof. Zhang was a recipient of the U.S. National Science Foundation Career Award in 2005. He is a member of ACM.