DEGREE-QUANT: QUANTIZATION-AWARE TRAINING FOR GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNNs) have demonstrated strong performance on a wide variety of tasks due to their ability to model non-uniform structured data. Despite their promise, there exists little research exploring methods to make them more efficient at inference time. In this work, we explore the viability of training quantized GNNs, enabling the usage of low precision integer arithmetic during inference. For GNNs seemingly unimportant choices in quantization implementation cause dramatic changes in performance. We identify the sources of error that uniquely arise when attempting to quantize GNNs, and propose an architecturally-agnostic and stable method, *Degree-Quant*, to improve performance over existing quantization-aware training baselines commonly used on other architectures, such as CNNs. We validate our method on six datasets and show, unlike previous attempts, that models generalize to unseen graphs. Models trained with Degree-Quant for INT8 quantization perform as well as FP32 models in most cases; for INT4 models, we obtain up to 26% gains over the baselines. Our work enables up to 4.7× speedups on CPU when using INT8 arithmetic.

1 INTRODUCTION

GNNs have received substantial attention in recent years due to their ability to model irregularly structured data. As a result, they are extensively used for applications as diverse as molecular interactions (Duvenaud et al., 2015; Wu et al., 2017), social networks (Hamilton et al., 2017), recommendation systems (van den Berg et al., 2017) or program understanding (Allamanis et al., 2018). Recent advancements have centered around building more sophisticated models including new types of layers (Kipf & Welling, 2017; Velickovic et al., 2018; Xu et al., 2019) and better aggregation functions (Corso et al., 2020). However, despite GNNs having few model parameters, the compute required for each application remains tightly coupled to the input graph size. A 2-layer Graph Convolutional Network (GCN) model with 32 hidden units would result in a model size of just 81KB but requires 19 GigaOPs to process the entire Reddit graph. We illustrate this growth in fig. 1.

One major challenge with graph architectures is therefore performing inference efficiently, which limits the applications they can be deployed for. For example, GNNs have been combined with CNNs for SLAM feature matching (Sarlin et al., 2019), however it is not trivial to deploy this technique on smartphones, or even smaller devices, whose neural network accelerators often do not implement floating point arithmetic, and instead favour more efficient integer arithmetic. Integer quantization is one way to lower the compute, memory and energy budget required to perform inference, without requiring modifications to the model architecture; this is also useful for model serving in data centers.

Although quantization has been well studied for CNNs and language models (Jacob et al., 2017; Wang et al., 2018; Zafrir et al., 2019; Prato et al., 2019), there remains relatively little work addressing GNN efficiency (Mukkara et al., 2018; Jia et al., 2020; Zeng & Prasanna, 2020; Yan et al., 2020). To the best of our knowledge, there is no work explicitly characterising the issues that arise when quantizing GNNs or showing latency benefits of using low-precision arithmetic. The recent work of Wang et al. (2020) explores only binarized embeddings of a single graph type (citation networks). In Feng et al. (2020) a heterogeneous quantization framework assigns different bits to embedding and attention coefficients in each layer while maintaining the weights at full precision (FP32). Due to the mismatch in operands' bit-width the majority of the operations are performed at FP32 after data casting, making it impractical to use in general purpose hardware such as CPUs or GPUs. In

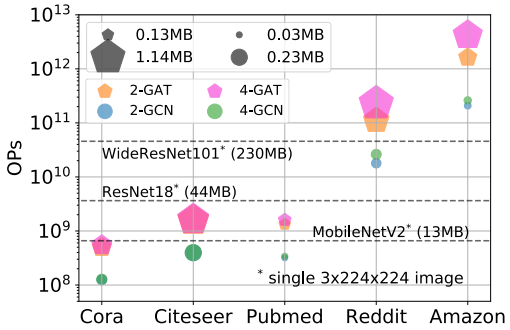

Figure 1: Despite GNN model sizes rarely exceeding 1MB, the OPs needed for inference grows at least linearly with the size of the dataset and node features. GNNs with models sizes 100× smaller than popular CNNs require many more OPs to process large graphs.

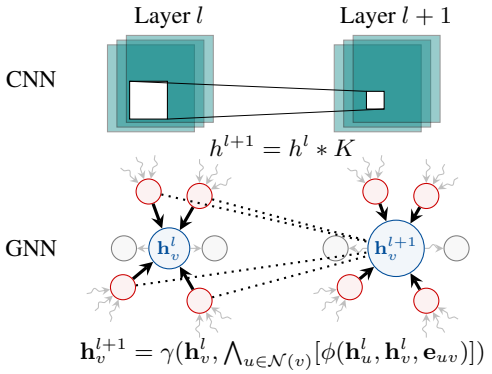

Figure 2: While CNNs operate on regular grids, GNNs operate on graphs with varying topology. A node's neighborhood size and ordering varies for GNNs. Both architectures use weight sharing.

addition they do not demonstrate how to train networks which generalize to *unseen* input graphs. Our framework relies upon uniform quantization applied to all elements in the network and uses bit-widths (8-bit and 4-bit) that are supported by off-the-shelf hardware such as CPUs and GPU for which efficient low-level operators for common operations found in GNNs exists.

This work considers the motivations and problems associated with quantization of graph architectures, and provides the following contributions:

- The explanation of the sources of degradation in GNNs when using lower precision arithmetic. We show how the choice of straight-through estimator (STE) implementation, node degree, and method for tracking quantization statistics significantly impacts performance.
- An *architecture-agnostic* method for quantization-aware training on graphs, *Degree-Quant* (DQ), which results in INT8 models often performing as well as their FP32 counterparts. At INT4, models trained with DQ typically outperform quantized baselines by over 20%. We show, unlike previous work, that models trained with DQ generalize to *unseen graphs*.
- We show that quantized networks achieve up to 4.7× speedups on CPU with INT8 arithmetic, relative to full precision floating point, with 4-8× reductions in runtime memory usage.

2 Background

2.1 Message Passing Neural Networks (MPNNs)

Many popular GNN architectures may be viewed as generalizations of CNN architectures to an irregular domain: at a high level, graph architectures attempt to build representations based on a node's neighborhood (see fig. 2). Unlike CNNs, however, this neighborhood does not have a fixed ordering or size. This work considers GNN architectures conforming to the MPNN paradigm (Gilmer et al., 2017). A graph $\mathcal{G} = (V, E)$ has node features $\mathbf{X} \in \mathbb{R}^{N \times F}$, an incidence matrix $\mathbf{I} \in \mathbb{N}^{2 \times E}$, and optionally D-dimensional edge features $\mathbf{E} \in \mathbb{R}^{E \times D}$. The forward pass through an MPNN layer consists of message passing, aggregation and update phases: $\mathbf{h}_{l+1}^{(i)} = \gamma(\mathbf{h}_l^{(i)}, \bigwedge_{j \in \mathcal{N}(i)}[\phi(\mathbf{h}_l^{(j)}, \mathbf{h}_l^{(i)}, \mathbf{e}_{ij})])$. Messages from node u to node v are calculated using function ϕ, and are aggregated using a permutation-invariant function $\bigwedge$. The features at v are subsequently updated using γ.

We focus on three architectures with corresponding update rules:

1. Graph Convolution Network (GCN): $\mathbf{h}_{l+1}^{(i)} = \sum_{j \in \mathcal{N}(i) \cup \{i\}}(\frac{1}{\sqrt{d_i d_j}} \mathbf{W} \mathbf{h}_l^{(j)})$ (Kipf & Welling, 2017), where d_i refers to the degree of node i.
2. Graph Attention Network (GAT): $\mathbf{h}_{l+1}^{(i)} = \alpha_{i,i} \mathbf{W} \mathbf{h}_l^{(i)} + \sum_{j \in \mathcal{N}(i)}(\alpha_{i,j} \mathbf{W} \mathbf{h}_l^{(j)})$, where α represent attention coefficients (Velickovic et al., 2018).

3. Graph Isomorphism Network (GIN): $\mathbf{h}_{l+1}^{(i)} = f_{\mathbf{\Theta}}[(1+\epsilon)\mathbf{h}_l^{(i)} + \sum_{j \in \mathcal{N}(i)} \mathbf{h}_l^{(j)}]$, where f is a learnable function (e.g. a MLP) and ϵ is a learnable constant (Xu et al., 2019).

2.2 Quantization for Non-Graph Neural Networks

Quantization allows for model size reduction and inference speedup without changing the model architecture. While there exists extensive studies of the impact of quantization at different bit-widths (Courbariaux et al., 2015; Han et al., 2015; Louizos et al., 2017) and data formats (Micikevicius et al., 2017; Carmichael et al., 2018; Kalamkar et al., 2019), it is 8-bit integer (INT8) quantization that has attracted the most attention. This is due to INT8 models reaching comparable accuracy levels to FP32 models (Krishnamoorthi, 2018; Jacob et al., 2017), offer a $4\times$ model compression, and result in inference speedups on off-the-shelf hardware as 8-bit arithmetic is widely supported.

Quantization-aware training (QAT) has become the *de facto* approach towards designing robust quantized models with low error (Wang et al., 2018; Zafrir et al., 2019; Wang et al., 2018). In their simplest forms, QAT schemes involve exposing the numerical errors introduced by quantization by simulating it on the forward pass Jacob et al. (2017) and make use of STE (Bengio et al., 2013) to compute the gradients—as if no quantization had been applied. For integer QAT, the quantization of a tensor x during the forward pass is often implemented as: $x_q = \min(q_{\max}, \max(q_{\min}, \lfloor x/s + z \rfloor))$, where $q_{\min}$ and $q_{\max}$ are the minimum and maximum representable values at a given bit-width and signedness, s is the scaling factor making x span the $[q_{\min}, q_{\max}]$ range and, z is the *zero-point*, which allows for the real value 0 to be representable in x_q. Both s and z are scalars obtained at training time. hen, the tensor is *dequantized* as: $\hat{x} = (x_q - z)s$, where the resulting tensor $\hat{x} \sim x$ for a high enough bit-width. This similarity degrades at lower bit-widths. Other variants of integer QAT are presented in Jacob et al. (2017) and Krishnamoorthi (2018).

To reach performance comparable to FP32 models, QAT schemes often rely on other techniques such as *gradient clipping*, to mask gradient updates based on the largest representable value at a given bit-width; noisy QAT, which stochastically applies QAT to a portion of the weights at each training step (Fan et al., 2020); or the re-ordering of layers (Sheng et al., 2018; Alizadeh et al., 2019).

3 Quantization for GNNs

In this section, we build an intuition for why GNNs would fail with low precision arithmetic by identifying the sources of error that will disproportionately affect the accuracy of a low precision model. Using this insight, we propose our technique for QAT with GNNs, *Degree-Quant*. Our analysis focuses on three models: GCN, GAT and GIN. This choice was made as we believe that these are among the most popular graph architectures, with strong performance on a variety of tasks (Dwivedi et al., 2020), while also being representative of different trends in the literature.

3.1 Sources of Error

QAT relies upon the STE to make an estimate of the gradient despite the non-differentiable rounding operation in the forward pass. If this approximation is inaccurate, however, then poor performance will be obtained. In GNN layers, we identify the aggregation phase, where nodes combine messages from a varying number of neighbors in a permutation-invariant fashion, as a source of substantial numerical error, especially at nodes with high in-degree. Outputs from aggregation have magnitudes that vary significantly depending on a node's in-degree: as it increases, the variance of aggregation values will increase.[1] Over the course of training $q_{\min}$ and $q_{\max}$, the quantization range statistics, become severely distorted by infrequent outliers, reducing the resolution for the vast majority of values observed. This reults in increased rounding error for nodes with smaller in-degrees. Controlling $q_{\min}$ and $q_{\max}$ hence becomes a trade-off balancing *truncation error* and *rounding error*.

We can derive how the mean and variance of the aggregation output values vary as node in-degree, n, increases for each of the three GNN layers. Suppose we model incoming message values for a single output dimension with random variables X_i, without making assumptions on their exact distribution

[1] The reader should note that we are not referring to the concept of estimator variance, which is the subject of sampling based approaches—we are exclusively discussing the variance of values immediately after aggregation.

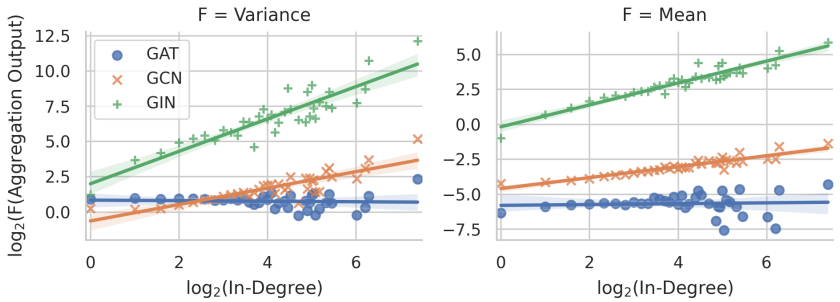

Figure 3: Analysis of values collected immediately after aggregation at the final layer of FP32 GNNs trained on Cora. Generated using channel data collected from 100 runs for each architecture. As in-degree grows, so does the mean and variance of channel values after aggregation.

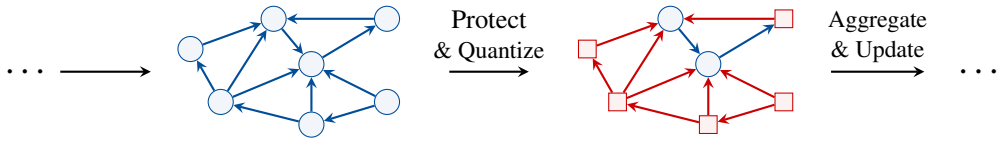

Figure 4: High-level view of the stochastic element of Degree-Quant. Protected (high in-degree) nodes, in blue, operate at full precision, while unprotected nodes (red) operate at reduced precision. High in-degree nodes contribute most to poor gradient estimates, hence they are stochastically protected from quantization more often.

or independence. Further, we use Y_n as the random variable representing the value of node output after the aggregation step. With GIN layers, we have $Y_n = (1+\epsilon)X_0 + \sum_{i=1}^{n} X_i$. It is trivial to prove that $\mathbb{E}(Y_n) = \mathcal{O}(n)$. The variance of the aggregation output is also $\mathcal{O}(n)$ in the case that that $\sum_{i \neq j} \text{Cov}(X_i, X_j) \ll \sum_i \text{Var}(X_i)$. We note that if $\sum_{i \neq j} \text{Cov}(X_i, X_j)$ is large then it implies that the network has learned highly redundant features, and may be a sign of over-fitting. Similar arguments can be made for GCN and GAT layers; we would expect GCN aggregation values to grow like $\mathcal{O}(\sqrt{n})$, and GAT aggregation values to remain constant ($\mathcal{O}(1)$) due to the attention coefficients.

We empirically validate these predictions on GNNs trained on Cora; results are plotted in fig. 3. We see that the aggregation values do follow the trends predicted, and that for the values of in-degree in the plot (up to 168) the covariance terms can be neglected. As expected, the variance and mean of the aggregated output grow fastest for GIN, and are roughly constant for GAT as in-degree increases. From this empirical evidence, it would be expected that GIN layers are most affected by quantization.

By using GIN and GCN as examples, we can see how aggregation error causes error in weight updates. Suppose we consider a GIN layer incorporating one weight matrix in the update function i.e. $\mathbf{h}_{l+1}^{(i)} = f(\mathbf{W}\mathbf{y}_{\text{GIN}}^{(i)})$, where f is an activation function, $\mathbf{y}_{\text{GIN}}^{(i)} = (1+\epsilon)\mathbf{h}_l^{(i)} + \sum_{j \in \mathcal{N}(i)} \mathbf{h}_l^{(j)}$, and $\mathcal{N}(i)$ denotes the in-neighbors of node i. Writing $\mathbf{y}_{\text{GCN}}^{(i)} = \sum_{k \in \mathcal{N}(i)} (\frac{1}{\sqrt{d_i d_k}} \mathbf{W}\mathbf{h}_l^{(j)})$, we see that the derivatives of the loss with respect to the weights for GCN and GIN are:

GIN

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{i=1}^{|V|} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{h}_{l+1}^{(i)}} \circ f'(\mathbf{W}\mathbf{y}_{\text{GIN}}^{(i)}) \right) \mathbf{y}_{\text{GIN}}^{(i)\top}$$

GCN

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{i=1}^{|V|} \sum_{j \in \mathcal{N}(i)} \frac{1}{\sqrt{d_i d_j}} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{h}_{l+1}^{(i)}} \circ f'(\mathbf{y}_{\text{GCN}}^{(i)}) \right) \mathbf{h}_l^{(j)\top}$$

The larger the error in $\mathbf{y}_{\text{GIN}}^{(i)}$—caused by aggregation error—the greater the error in the weight gradients for GIN, which results in poorly performing models being obtained. The same argument applies to GCN, with the $\mathbf{h}_l^{(j)\top}$ and $\mathbf{y}_{\text{GCN}}^{(i)}$ terms introducing aggregation error into the weight updates.

3.2 Our Method: Degree-Quant

To address these sources of error we propose *Degree-Quant* (DQ), a method for QAT with GNNs. We consider both *inaccurate weight updates* and *unrepresentative quantization ranges*.

Stochastic Protection from Quantization to Improve Weight Update Accuracy. DQ aims to encourage more accurate weight updates by stochastically protecting nodes in the network from

Algorithm 1 Degree-Quant (DQ). Functions accepting a protective mask $\mathbf{m}$ perform only the masked computations at full precision: intermediate tensors are *not* quantized. At test time protective masking is disabled. In fig. 11 (in the Appendix) we show with a diagram how a GCN layers makes use of DQ.

procedure TRAINFORWARDPASS($\mathcal{G}, \mathbf{p}$)
  ▷ Calculate mask and *quantized* weights, Θ', which all operations share
  $\mathbf{m} \leftarrow$ BERNOULLI($\mathbf{p}$)
  $\Theta' \leftarrow$ QUANTIZE(Θ)
  ▷ Messages with masked sources are at full precision (excluding weights)
  $\mathcal{M} \leftarrow$MESSAGECALCULATE($\mathcal{G}, \Theta', \mathbf{m}$)
  $X \leftarrow$ QUANTIZE(AGGREGATE($\mathcal{M}, \Theta', \mathbf{m}$), $\mathbf{m}$) ▷ No quantization for masked nodes
  return UPDATE($X, \Theta', \mathbf{m}$) ▷ Quantized weights always used
end procedure

quantization. At each layer a protective node mask is generated; all masked nodes have the phases of the message passing, aggregation and update performed at full precision. This includes messages sent by protected nodes to other nodes, as shown in fig. 4 (a detailed diagram is shown in fig. 11). It is also important to note that the weights used at all nodes are the same quantized weights; this is motivated by the fact that our method is used to encourage more accurate gradients to flow back to the weights through high in-degree nodes. At test time protection is disabled: all nodes operate at low precision.

To generate the mask, we pre-process each graph before training and create a vector of probabilities $\mathbf{p}$ with length equal to the number of nodes. At training time, mask $\mathbf{m}$ is generated by sampling using the Bernoulli distribution: $\mathbf{m} \sim \text{Bernoulli}(\mathbf{p})$. In our scheme p_i is higher if the in-degree of node i is large, as we find empirically that high in-degree nodes contribute most towards error in weight updates. We use a scheme with two hyperparameters, $p_{\min}$ and $p_{\max}$; nodes with the maximum in-degree are assigned $p_{\max}$ as their masking probability, with all other nodes assigned a probability calculated by interpolating between $p_{\min}$ and $p_{\max}$ based on their in-degree ranking in the graph.

Percentile Tracking of Quantization Ranges. Figure 3 demonstrates large fluctuations in the variance of the aggregation output as in-degree increases. Since these can disproportionately affect the ranges found by using min-max or momentum-based quantization, we propose using *percentiles*. While percentiles have been used for post-training quantization (Wu et al., 2020), we are the first (to the best of our knowledge) to propose making it a core part of QAT; we find it to be a key contributor to achieving consistent results with graphs. Using percentiles involves ordering the values in the tensor and clipping a fraction of the values at both ends of the distribution. The fraction to clip is a hyperparameter. We are more aggressive than existing literature on the quantity we discard: we clip the top and bottom 0.1%, rather than 0.01%, as we observe the fluctuations to be a larger issue with GNNs than with CNNs or DNNs. Quantization ranges are more representative of the vast majority of values in this scheme, resulting in less *rounding error*.

We emphasize that a core contribution of DQ is that it is *architecture-agnostic*. Our method enables a wide variety of architectures to use low precision arithmetic at inference time. Our method is also *orthogonal*—and complementary—to other techniques for decreasing GNN computation requirements, such as sampling based methods which are used to reduce memory consumption (Zeng et al., 2020), or weight pruning (Blalock et al., 2020) approaches to achieve further model compression.

4 EXPERIMENTS

In this section we first analyse how the choice of quantization implementation affects performance of GNNs. We subsequently evaluate Degree-Quant against the strong baselines of: FP32, INT8-QAT and, INT8-QAT with stochastic masking of weights (Fan et al., 2020). We refer to this last approach as *noisy* QAT or nQAT. To make explicit that we are quantizing both weights and activations, we use the notation W8A8. We repeat the experiments at INT4. Our study evaluates performance on six datasets and includes both node-level and graph-level tasks. The datasets used were Cora, CiteSeer, ZINC, MNIST and CIFAR10 superpixels, and REDDIT-BINARY. Across all datasets INT8 models trained with Degree-Quant manage to recover most of the accuracy lost as a result of quantization. In some instances, DQ-INT8 outperform the extensively tuned FP32 baselines. For INT4, DQ outperforms all QAT baselines and results in double digits improvements over QAT-INT4 in some settings. Details about each dataset and our experimental setup can be found in appendix A.1.

Dataset	Model Arch.	*vanilla* STE				STE with Gradient Clipping			
		min/max		momentum		min/max		momentum	
		W8A8	W4A4	W8A8	W4A4	W8A8	W4A4	W8A8	W4A4
Cora (Acc. %) ↑	GCN	$\mathbf{81.0 \pm 0.7}$	65.3 ± 4.9	42.3 ± 11.1	49.4 ± 8.8	80.8 ± 0.8	62.3 ± 5.2	66.9 ± 18.2	$\mathbf{77.2 \pm 2.5}$
	GAT	76.0 ± 2.2	16.8 ± 8.5	81.7 ± 1.3	$\mathbf{51.7 \pm 5.8}$	76.4 ± 2.6	15.4 ± 8.1	$\mathbf{81.9 \pm 0.7}$	47.4 ± 5.0
	GIN	69.9 ± 1.9	25.9 ± 2.6	49.2 ± 10.2	$\mathbf{42.8 \pm 4.0}$	69.2 ± 2.3	29.5 ± 3.5	$\mathbf{75.1 \pm 1.1}$	40.5 ± 5.0
MNIST (Acc. %) ↑	GCN	$\mathbf{90.4 \pm 0.2}$	51.3 ± 7.5	90.1 ± 0.5	$\mathbf{70.6 \pm 2.4}$	90.4 ± 0.3	54.8 ± 1.5	90.2 ± 0.4	10.3 ± 0.0
	GAT	$\mathbf{95.8 \pm 0.1}$	20.1 ± 3.3	95.7 ± 0.3	67.4 ± 3.2	95.7 ± 0.1	30.2 ± 7.4	95.7 ± 0.3	$\mathbf{76.3 \pm 1.2}$
	GIN	96.5 ± 0.3	62.4 ± 21.8	$\mathbf{96.7 \pm 0.2}$	$\mathbf{91.0 \pm 0.6}$	96.4 ± 0.4	19.5 ± 2.1	75.3 ± 18.1	10.8 ± 0.9
ZINC (Loss) ↓	GCN	0.486 ± 0.01	0.747 ± 0.02	0.509 ± 0.01	0.710 ± 0.05	0.495 ± 0.01	0.766 ± 0.02	$\mathbf{0.483 \pm 0.01}$	$\mathbf{0.692 \pm 0.01}$
	GAT	0.471 ± 0.01	0.740 ± 0.02	0.571 ± 0.03	$\mathbf{0.692 \pm 0.06}$	0.466 ± 0.01	0.759 ± 0.04	$\mathbf{0.463 \pm 0.01}$	0.717 ± 0.03
	GIN	0.393 ± 0.02	1.206 ± 0.27	$\mathbf{0.386 \pm 0.03}$	$\mathbf{0.572 \pm 0.02}$	0.390 ± 0.02	1.669 ± 0.10	0.388 ± 0.02	0.973 ± 0.24

Table 1: Impact on performance of four typical quantization implementations for INT8 and INT4. The configuration that resulted in best performing models for each dataset-model pair is bolded. Hyperparameters for each experiment were fine-tuned independently. **A major contribution of this work is identifying that seemingly unimportant choices in quantization implementation cause dramatic changes in performance.**

4.1 Impact of Quantization Gradient Estimator on Convergence

The STE is a workaround for when the forward pass contains non-differentiable operations (e.g. rounding in QAT) that has been widely adopted in practice. While the choice of STE implementation generally results in marginal differences for CNNs—even for binary networks (Alizadeh et al., 2019)—it is unclear whether only marginal differences will also be observed for GNNs. Motivated by this, we study the impact of four off-the-shelve quantization procedures on the three architectures evaluated for each type of dataset; the implementation details of each one is described in appendix A.3. We perform this experiment to ensure that we have the strongest possible QAT baselines. Results are shown in table 1. We found the choice quantization implementation to be highly dependent on the model architecture and type of problem to be solved: we see a much larger variance than is observed with CNNs; this is an important discovery for future work building on our study.

We observe a general trend in all INT4 experiments benefiting from momentum as it helps smoothing out the quantization statistics for the inherently noisy training stage at low bitwidths. This trend applies as well for the majority of INT8 experiments, while exhibiting little impact on MNIST. For INT8 Cora-GCN, large gradient norm values in the early stages of training (see fig. 5) mean that these models not benefit from momentum as quantization ranges fail to keep up with the rate of changes in tensor values; higher momentum can help but also leads to instability. In contrast, GAT has stable initial training dynamics, and hence obtains better results with momentum. For the molecules dataset ZINC, we consistently obtained lower regression loss when using momentum. We note that GIN models often suffer from higher performance degradation (as was first noted in fig. 3), specially at W4A4. This is not the case however for image datasets using superpixels. We believe that datasets with Gaussian-like node degree distributions (see fig. 9) are more tolerant of the imprecision introduced by quantization, compared to datasets with tailed distributions. We leave more in-depth analysis of how graph topology affects quantization as future work.

4.2 Obtaining Quantization baselines

Our FP32 results, which we obtain after extensive hyperparameter tuning, and those from the baselines are shown at the top of table 2. We observed large gains on MNIST, CIFAR10 and, ZINC.

For our QAT-INT8 and QAT-INT4 baselines, we use the quantization configurations informed by our analysis in section 4.1. For Citeseer we use the best resulting setup analysed for Cora, and for CIFAR-10 that from MNIST. Then, the hyperparameters for each experiment were fine tuned individually, including noise rate $n \in [0.5, 0.95]$ for nQAT experiments. QAT-INT8 and QAT-INT4 results in table 2 and QAT-INT4, with the exception of MNIST (an easy to classify dataset), corroborate our hypothesis that GIN layers are less resilient to quantization. This was first observed in fig. 3. In the case of ZINC, while all models results in noticeable degradation, GIN sees a more severe 16% increase of regression loss compared to our FP32 baseline. For QAT W4A4 an accuracy drop of over 35% and 47% is observed for Cora and Citeseer respectively. The stochasticity induced by nQAT helped in recovering some of the accuracy lost as a result of quantization for citation networks (both INT8 and INT4) but had little impact on other datasets and harmed performance in some cases.

Quant. Scheme	Model Arch.	Node Classification (Accuracy %) Cora ↑	Citeseer ↑	Graph Classification (Accuracy %) MNIST ↑	CIFAR-10 ↑	Graph Regression (Loss) ZINC ↓
Ref. (FP32)	GCN	81.4 ± 0.7	71.1 ± 0.7	90.0 ± 0.2	54.5 ± 0.1	0.469 ± 0.002
	GAT	83.1 ± 0.4	72.5 ± 0.7	95.6 ± 0.1	65.4 ± 0.4	0.463 ± 0.002
	GIN	77.6 ± 1.1	66.1 ± 0.9	93.9 ± 0.6	53.3 ± 3.7	0.414 ± 0.009
Ours (FP32)	GCN	81.2 ± 0.6	71.4 ± 0.9	90.9 ± 0.4	58.4 ± 0.5	0.450 ± 0.008
	GAT	83.2 ± 0.3	72.4 ± 0.8	95.8 ± 0.4	65.1 ± 0.8	0.455 ± 0.006
	GIN	77.9 ± 1.1	65.8 ± 1.5	96.4 ± 0.4	57.4 ± 0.7	0.334 ± 0.024
QAT (W8A8)	GCN	81.0 ± 0.7	71.3 ± 1.0	90.9 ± 0.2	56.4 ± 0.5	0.481 ± 0.029
	GAT	81.9 ± 0.7	71.2 ± 1.0	95.8 ± 0.3	66.3 ± 0.4	0.460 ± 0.005
	GIN	75.6 ± 1.2	63.0 ± 2.6	96.7 ± 0.2	52.4 ± 1.2	0.386 ± 0.025
nQAT (W8A8)	GCN	81.0 ± 0.8	70.7 ± 0.8	91.1 ± 0.1	56.2 ± 0.5	0.472 ± 0.015
	GAT	82.5 ± 0.5	71.2 ± 0.7	96.0 ± 0.1	66.7 ± 0.2	0.459 ± 0.007
	GIN	77.4 ± 1.3	65.1 ± 1.4	96.4 ± 0.3	52.7 ± 1.4	0.405 ± 0.016
DQ (W8A8)	GCN	81.7 ± 0.7 (+**0.7**)	71.0 ± 0.9 (-**0.3**)	90.9 ± 0.2 (-**0.2**)	56.3 ± 0.1 (-**0.1**)	0.434 ± 0.009 (+**9.8**)
	GAT	82.7 ± 0.7 (+**0.2**)	71.6 ± 1.0 (+**0.4**)	95.8 ± 0.4 (-**0.2**)	67.7 ± 0.5 (+**1.0**)	0.456 ± 0.005 (+**0.9**)
	GIN	78.7 ± 1.4 (+**1.3**)	67.5 ± 1.4 (+**2.4**)	96.6 ± 0.1 (-**0.1**)	55.5 ± 0.6 (+**2.8**)	0.357 ± 0.014 (+**7.5**)
QAT (W4A4)	GCN	77.2 ± 2.5	64.1 ± 4.1	70.6 ± 2.4	38.1 ± 1.6	0.692 ± 0.013
	GAT	55.6 ± 5.4	65.3 ± 1.9	76.3 ± 1.2	41.0 ± 1.1	0.655 ± 0.032
	GIN	42.5 ± 4.5	18.6 ± 2.9	91.0 ± 0.6	45.6 ± 3.6	0.572 ± 0.02
nQAT (W4A4)	GCN	78.1 ± 1.5	65.8 ± 2.6	70.9 ± 1.5	40.1 ± 0.7	0.669 ± 0.128
	GAT	54.9 ± 5.6	65.5 ± 1.7	78.4 ± 1.5	41.0 ± 0.6	0.637 ± 0.012
	GIN	45.0 ± 5.0	34.6 ± 3.8	91.3 ± 0.5	48.7 ± 1.7	0.561 ± 0.068
DQ (W4A4)	GCN	78.3 ± 1.7 (+**0.2**)	66.9 ± 2.4 (+**1.1**)	84.4 ± 1.3 (+**13.5**)	51.1 ± 0.7 (+**11.0**)	0.536 ± 0.011 (+**26.2**)
	GAT	71.2 ± 2.9 (+**16.3**)	67.6 ± 1.5 (+**2.1**)	93.1 ± 0.3 (+**14.7**)	56.5 ± 0.6 (+**15.5**)	0.520 ± 0.021 (+**20.6**)
	GIN	69.9 ± 3.4 (+**24.9**)	60.8 ± 2.1 (+**26.2**)	95.5 ± 0.4 (+**4.2**)	50.7 ± 1.6 (+**2.0**)	0.431 ± 0.012 (+**23.2**)

Table 2: This table is divided into three sets of rows with FP32 baselines at the top. We provide two baselines for INT8 and INT4: standard QAT and stochastic QAT (nQAT). Both are informed by the analysis in 4.1, with nQAT achieving better performance in some cases. Models trained with Degree-Quant (DQ) are always comparable to baselines, and usually substantially better, especially for INT4. **DQ is a stable method which requires little tuning to obtain excellent results across a variety of architectures and datasets.**

Quantization	Model	REDDIT-BIN (Acc. %) ↑
Ref. (FP32)	GIN	92.2 ± 2.3
Ours (FP32)	GIN	92.0 ± 1.5
QAT-W8A8	GIN	76.1 ± 7.5
nQAT-W8A8	GIN	77.5 ± 3.4
DQ-W8A8	GIN	91.8 ± 2.3 (+**14.3**)
QAT-W4A4	GIN	54.4 ± 6.6
nQAT-W4A4	GIN	58.0 ± 6.3
DQ-W4A4	GIN	81.3 ± 4.4 (+**23.0**)

Table 3: Results for DQ-INT8 GIN models perform nearly as well as at FP32. For INT4, DQ offers a significant increase in accuracy.

Device	Arch.	Zinc (Batch=10K) FP32	W8A8	Speedup	Reddit FP32	W8A8	Speedup
CPU	GCN	181ms	42ms	4.3×	13.1s	3.1s	4.2×
	GAT	190ms	50ms	3.8×	13.1s	2.8s	4.7×
	GIN	182ms	43ms	4.2×	13.1s	3.1s	4.2×
GPU	GCN	39ms	31ms	1.3×	191ms	176ms	1.1×
	GAT	17ms	15ms	1.1×	OOM	OOM	-
	GIN	39ms	31ms	1.3×	191ms	176ms	1.1×

Table 4: INT8 latency results run on a 22 core 2.1GHz Intel Xeon Gold 6152 and, on a GTX 1080Ti GPU. Quantization provides large speedups on a variety of graphs for CPU and non-negligible speedups with unoptimized INT8 GPU kernels.

4.3 Comparisons of Degree-Quant with Existing Quantization Approaches

Degree-Quant provides superior quantization for all GNN datasets and architectures. Our results with DQ are highlighted in gray in table 2 and table 3. Citation networks trained with DQ for W8A8 manage to recover most of the accuracy lost as a result of QAT and outperform most of nQAT baselines. In some instances DQ-W8A8 models outperform the reference FP32 baselines. At 4-bits, DQ results in even larger gains compared to W4A4 baselines. We see DQ being more effective for GIN layers, outperforming INT4 baselines for Cora (+24.9%), Citeseer (+26.2%) and REDDIT-BINARY (+23.0%) by large margins. Models trained with DQ at W4A4 for graph classification and graph regression also exhibit large performance gains (of over 10%) in most cases. For ZINC, all models achieve over 20% lower regression loss. Among the top performing models using DQ, ratios of $p_{\min}$ and $p_{\max}$ in $[0.0, 0.2]$ were the most common. Figure 10 in the appendix shows validation loss curves for GIN models trained using different DQ probabilities on the REDDIT-BINARY dataset.

5 DISCUSSION

Latency and Memory Implications. In addition to offering significantly lower memory usage ($4\times$ with INT8), quantization can reduce latency—especially on CPUs. We found that with INT8 arithmetic we could accelerate inference by up to $\mathbf{4.7}\times$. We note that the latency benefit depends on the graph topology and feature dimension, therefore we ran benchmarks on a variety of graph datasets, including Reddit[2], Zinc, Cora, Citeseer, and CIFAR-10; Zinc and Reddit results are shown in table 4, with further results given in the appendix. For a GCN layer with in- and out-dimension of 128, we get speed-ups of: $4.3\times$ on Reddit, $2.5\times$ on Zinc, $1.3\times$ on Cora, $1.3\times$ on Citeseer and, $2.1\times$ on CIFAR-10. It is also worth emphasizing that quantized networks are necessary to efficiently use accelerators deployed in smartphones and smaller devices as they primarily accelerate integer arithmetic, and that CPUs remain a common choice for model serving on servers. The decrease in latency on CPUs is due to improved cache performance for the sparse operations; GPUs, however, see less benefit due to their massively-parallel nature which relies on mechanisms other than caching to hide slow random memory accesses, which are unavoidable in this application.

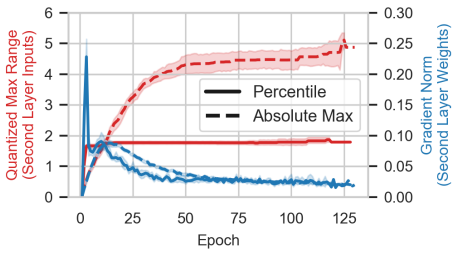

Figure 5: $q_{\max}$ with absolute min/max and percentile ranges, applied to INT8 GCN training on Cora. We observe that the percentile max is half that of the absolute, *doubling* resolution for the majority of values.

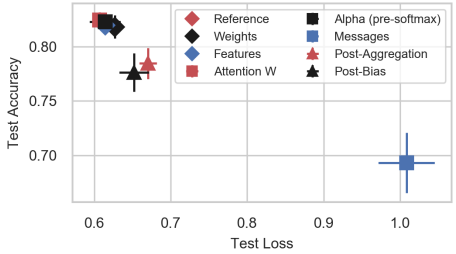

Figure 6: Analysis of how INT8 GAT performance degrades on Cora as individual elements are reduced to 4-bit precision *without DQ*. For GAT the message elements are crucial to classification performance.

Ablation Study: Benefits of Percentile Ranges. Figure 5 shows the value of percentiles during training. We see that when using absolute min/max the upper range grows to over double the range required for 99.9% of values, effectively halving the resolution of the quantized values. DQ is more stable, and we obtained strong results with an order of magnitude less tuning relative to the baselines.

Ablation Study: Source of Degradation at INT4. Figure 6 assesses how INT8 GAT (without DQ) degrades as single elements are converted to INT4, in order to understand the precipitous drop in accuracy in the INT4 baselines; further plots for GCN and GIN are included in the appendix. We observe that most elements cause only modest performance losses relative to a full INT8 model. DQ is most important to apply to elements which are constrained by *numerical precision*, such as the aggregation and message elements in GAT. Weight elements, however, are consistently unaffected.

Ablation Study: Effect of Stochastic Element in Degree-Quant. We observe that the stochastic protective masking in DQ alone often achieves most of the performance gain over the QAT baseline; results are given in table 9 in the appendix. The benefit of the percentile-based quantization ranges is *stability*, although it can yield some performance gains. The full DQ method provides consistently good results on all architectures and datasets, without requiring an extensive analysis as in 4.1.

6 CONCLUSION

This work has presented Degree-Quant, an architecture-agnostic and stable method for training quantized GNN models that can be accelerated using off-the-shelf hardware. With 4-bit weights and activations we achieve $8\times$ compression while surpassing strong baselines by margins regularly exceeding 20%. At 8-bits, models trained with DQ perform on par or better than the baselines while achieving up to $4.7\times$ lower latency than FP32 models. Our work offers a comprehensive foundation for future work in this area and is a first step towards enabling GNNs to be deployed more widely, including to resource constrained devices such as smartphones.

[2]The largest graph commonly benchmarked on in the GNN literature

REFERENCES

R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, 2012.

Milad Alizadeh, Javier Fernández-Marqués, Nicholas D. Lane, and Yarin Gal. A empirical study of binary neural networks' optimisation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJfUCoR5KX.

Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=BJOFETxR-.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning?, 2020.

Zachariah Carmichael, Hamed F. Langroudi, Char Khazanov, Jeffrey Lillie, John L. Gustafson, and Dhireesha Kudithipudi. Deep positron: A deep neural network using the posit number system, 2018.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets, 2020.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations, 2015.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pp. 2224–2232, 2015.

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks, 2020.

Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. Training with quantization noise for extreme model compression, 2020.

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, 2020.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212, 2017. URL http://arxiv.org/abs/1704.01212.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2017.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017.

Sambhav R. Jain, Albert Gural, Michael Wu, and Chris H. Dick. Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks, 2020.

Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. Improving the accuracy, scalability, and performance of graph neural networks with roc. In *Proceedings of Machine Learning and Systems 2020*, pp. 187–198. 2020.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation, 2018.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, Jiyan Yang, Jongsoo Park, Alexander Heinecke, Evangelos Georganas, Sudarshan Srinivasan, Abhisek Kundu, Misha Smelyanskiy, Bharat Kaul, and Pradeep Dubey. A study of bfloat16 for deep learning training, 2019.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In *Proceedings of Machine Learning and Systems 2020*, pp. 230–246. 2020.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning, 2017.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2017.

Anurag Mukkara, Nathan Beckmann, Maleen Abeydeera, Xiaosong Ma, and Daniel Sanchez. Exploiting locality in graph analytics through hardware-accelerated traversal scheduling. In *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-51, pp. 1–14. IEEE Press, 2018. ISBN 9781538662403. doi: 10.1109/MICRO.2018.00010. URL https://doi.org/10.1109/MICRO.2018.00010.

Gabriele Prato, Ella Charlaix, and Mehdi Rezagholizadeh. Fully quantized transformer for machine translation, 2019.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. *arXiv preprint arXiv:1911.11763*, 2019.

Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen, and Mickey Aleksic. A quantization-friendly separable convolution for mobilenets. *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, Mar 2018. doi: 10.1109/emc2.2018.00011. URL http://dx.doi.org/10.1109/emc2.2018.00011.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

Rianne van den Berg, Thomas N. Kipf, and Max Welling. Graph convolutional matrix completion, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, Xiangjian He, Yiguang Lin, and Xuemin Lin. Binarized graph neural network, 2020.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision, 2018.

Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation, 2020.

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, 2017.

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.

Bencheng Yan, Chaokun Wang, Gaoyang Guo, and Yunkai Lou. Tinygnn: Learning efficient graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD '20, pp. 1848–1856, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403236. URL https://doi.org/10.1145/3394486.3403236.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert, 2019.

Hanqing Zeng and Viktor Prasanna. Graphact. *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Feb 2020. doi: 10.1145/3373087.3375312. URL http://dx.doi.org/10.1145/3373087.3375312.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method, 2020.