# A APPENDIX

## A.1 EXPERIMENTAL SETUP

As baselines we use the architectures and results reported by Fey & Lenssen (2019) for citation networks, Dwivedi et al. (2020) for MNIST, CIFAR-10 and ZINC and, Xu et al. (2019) for REDDIT-BINARY. We re-implemented the architectures and datasets used in these publications and replicated the results reported at FP32. Models using GIN layers learn parameter $\epsilon$. These models are often referred to as GIN-$\epsilon$. The high-level description of these architectures is shown in table 5. The number of parameters for each architecture-dataset in this work are shown in table 6.

Our infrastructure was implemented using PyTorch Geometric (PyG) (Fey & Lenssen, 2019). We generate candidate hyperparameters using random search, and prune trials using the asynchronous hyperband algorithm (Li et al., 2020). Hyperparameters searched over were learning rate, weight decay, and dropout (Srivastava et al., 2014) and drop-edge (Rong et al., 2020) probabilities. The search ranges were initialized centered at the values used in the reference implementations of the baselines. Degree-Quant requires searching for two additional hyperparameters, $p_{\text{min}}$ and $p_{\text{max}}$, these were tuned in a grid-search fashion. We report our results using the hyperparameters which achieved the best validation loss over 100 runs on the Cora and Citeseer datasets, 10 runs for MNIST, CIFAR-10 and ZINC, and 10-fold cross-validation for REDDIT-BINARY.

Our experiments ran on several machines in our SLURM cluster using Intel CPUs and NVIDIA GPUs. Each machine was running Ubuntu 18.04. The GPU models in our cluster were: V100, RTX 2080Ti and GTX 1080Ti. (Jain et al., 2020)

| Model | # Layers | | | | | # Hidden Units | | | | | Residual | | | | | Output MLP | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Arch. | Cit | M | C | Z | R | Cit | M | C | Z | R | Cit | M | C | Z | R | Cit | M | C | Z | R |
| GCN | 2 | 4 | 4 | 4 | - | 16 | 146 | 146 | 145 | - | × | ✓ | ✓ | ✓ | - | × | ✓ | ✓ | ✓ | - |
| GAT | 2 | 4 | 4 | 4 | - | 8 | 19 | 19 | 18 | - | × | ✓ | ✓ | ✓ | - | × | ✓ | ✓ | ✓ | - |
| GIN | 2 | 4 | 4 | 4 | 5 | 16 | 110 | 110 | 110 | 64 | × | ✓ | ✓ | ✓ | × | × | ✓ | ✓ | ✓ | ✓ |

**Table 5:** High level description of the architectures evaluated for citation networks (Cit), MNIST (M), CIFAR-10 (C), ZINC (Z) and REDDIT-BINARY (R). We relied on Adam optimizer for all experiments. For all batched experiments, we used 128 batch-sizes. All GAT models used 8 attention heads. All GIN architectures used 2-layer MLPs, except those for citation networks which used a single linear layer.

| Model | Node Classification | | Graph Classification | | | Graph Regression |
|---|---|---|---|---|---|---|
| Arch. | Cora | Citeseer | MNIST | CIFAR-10 | REDDIT-BIN | ZINC |
| GCN | 23063 | 59366 | 103889 | 104181 | - | 105454 |
| GAT | 92373 | 237586 | 113706 | 114010 | - | 105044 |
| GIN | 23216 | 59536 | 104554 | 104774 | 42503 | 102088 |

**Table 6:** Number of parameters for each of the evaluated architectures

For QAT experiments, all elements of each network are quantized: inputs to each layer, the weights, the messages sent between nodes, the inputs to aggregation stage and its outputs and, the outputs of the update stage (which are the outputs of the GNN layer before activation). In this way, all intermediate tensors in GNNs are quantized with the exception of the attention mechanism in GAT; we do not quantize after the softmax calculation, due to the numerical precision required at this stage. With the exception of Cora and Citeseer, the models evaluated in this work make use of Batch Normalization (Ioffe & Szegedy, 2015). For deployments of quantized models, Batch Normalization layers are often *folded* with the weights (Krishnamoorthi, 2018). This is to ensure the input to the next layer is within the expected $[q_{\text{min}}, q_{\text{max}}]$ ranges. In this work, for both QAT baselines and QAT+DQ, we left BN layers unfolded but ensure the inputs and outputs were quantized to the appropriate number of bits (i.e. INT8 or INT4) before getting multiplied with the layer weights. We leave as future work proposing a BN folding mechanism applicable for GNNs and studying its impact for deployments of quantized GNNs.

The GIN models evaluated on REDDIT-BINARY used QAT for all layers with the exception of the input layer of the MLP in the first GIN layer. This compromise was needed to overcome the severe degradation introduced by quantization when operating on nodes with a single scalar as feature.

### A.2 DATASETS

We show in Table 7 the statistics for each dataset either used or referred to in this work. For Cora and Citeseer datasets, nodes correspond to documents and edges to citations between these. Node features are a bag-of-words representation of the document. The task is to classify each node in the graph (i.e. each document) correctly. The MNIST and CIFAR-10 datasets (commonly used for image classification) are transformed using SLIC (Achanta et al., 2012) into graphs where each node represents a cluster of perceptually similar pixels or superpixels. The task is to classify each image using their superpixels graph representation. The ZINC dataset contains graphs representing molecules, were each node is an atom. The task is to regress a molecular property (constrained solubility (Jin et al., 2018)) given the graph representation of the molecule. Nodes in graphs of the REDDIT-BINARY dataset represent users of a Reddit thread with edges drawn between a pair of nodes if these interacted. This dataset contains graphs of two types of communities: question-answer threads and discussion threads. The task is to determine if a given graph is from a question-answer thread or a discussion thread.

We use standard splits for MNIST, CIFAR-10 and ZINC. For citation datasets (Cora and Citeseer), we use the splits used by Kipf & Welling (2017). For REDDIT-BINARY we use 10-fold cross validation.

| Dataset | Graphs | Nodes | Edges | Features | Labels |
|---|---|---|---|---|---|
| Cora | 1 | 2,708 | 5,278 | 1,433 | 7 |
| Citeseer | 1 | 3,327 | 4,552 | 3,703 | 6 |
| Pubmed | 1 | 19,717 | 44,324 | 500 | 3 |
| MNIST | 70K | 40-75 | 564.53 (avg) | 3 | 10 |
| CIFAR10 | 60K | 85-150 | 941.07 (avg) | 5 | 10 |
| ZINC | 12K | 9-37 | 49.83 (avg) | 28 | 1 |
| REDDIT-BINARY | 2K | 429.63 (avg) | 497.75 (avg) | 1 | 2 |
| Reddit | 1 | 232,965 | 114,848,857 | 602 | 41 |
| Amazon | 1 | 9,430,088 | 231,594,310 | 300 | 24 |

**Table 7:** Statistics for each dataset used in the paper. Some datasets are only referred to in fig. 1

### A.3 QUANTIZATION IMPLEMENTATIONS

In section 4.1 we analyse different readily available quantization implementations and how they impact in QAT results. First, vanilla STE, which is the reference STE (Bengio et al., 2013) that lets the gradients pass unchanged; and gradient clipping (GC), which clips the gradients based on the maximum representable value for a given quantization level. Or in other words, GC limits gradients if the tensor's magnitudes are outside the $[q_{\min}, q_{\max}]$ range.

$$x_{\min} = \begin{cases} \min(X) & \text{if step} = 0 \\ \min(x_{\min}, X) & \text{otherwise} \end{cases} \tag{1}$$

$$x_{\min} = \begin{cases} \min(X) & \text{if step } = 0 \\ (1-c)x_{\min} + c\min(X) & \text{otherwise} \end{cases} \tag{2}$$

The quantization modules keep track of the input tensor's min and max values, $x_{\min}$ and $x_{\max}$, which are then used to compute $q_{\min}$, $q_{\max}$, *zero-point* and *scale* parameters. For both vanilla STE and GC, we study two popular ways of keeping track of these statistics: *min/max*, which tracks the min/max tensor values observed over the course of training; and *momentum*, which computes the moving averages of those statistic during training. The update rules for $x_{\min}$ for STE *min/max* and STE *momentum* are presented in eq. (1) and eq. (2) respectively, where $X$ is the tensor to be quantized and $c$ is the momentum hyperparameter, which in all our experiments is set to its default $0.01$. Equivalent rules apply when updating $x_{\max}$ (omitted).

For stochastic QAT we followed the implementation described in Fan et al. (2020), where at each training step a binary mask sampled from a Bernoulli distribution is used to specify which elements of the weight tensor will be quantized and which will be left at full precision. We experimented with block sizes larger than one (i.e. a single scalar) but often resulted in a sever drop in performance. All the reported results use block size of one.

### A.4 Degree-Quant and Graph Level Summarization

The percentile operation in our quantization scheme remains important for summarizing the graph when doing graph-level tasks, such as graph regression (Zinc) or graph classification (MNIST, CIFAR-10 and REDDIT-BINARY). Since the number of nodes in each input graph is not constant, this can cause the summarized representation produced from the final graph layer to have a more tailed distribution than would be seen with other types of architectures (e.g. CNN). Adding the percentile operation reduces the impact of these extreme tails in the fully connected graph-summarization layers, thereby increasing overall performance. The arguments regarding weight update accuracy also still apply, as the $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_{l+1}^{(i)}}$ term in the equations for the GCN and GIN should be more accurate compared to when the activations are always quantized before the summarization. This phenomenon is also noted by Fan et al. (2020).

### A.5 Degradation Studies

Figures 7 and 8 show the results of the ablation study conducted in section 5 for GCN and GIN. We observe that GCN is more tolerant to INT4 quantization than other architectures. GIN, however, requires accurate representations after the update stage, and heavily suffers from further quantization like GAT. The idea of performing different stages of inference at different precisions has been proposed, although it is uncommon (Wang et al., 2018).

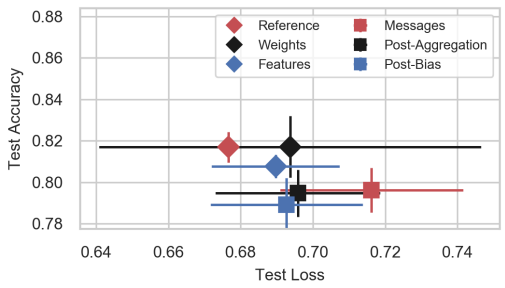


**Figure 7:** Degradation of INT8 GCN on Cora as individual elements are converted to INT4 *without Degree-Quant*.

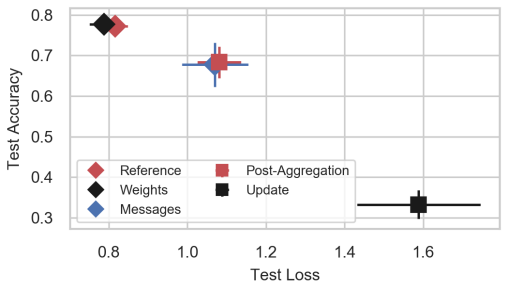


**Figure 8:** Degradation of INT8 GIN on Cora as individual elements are converted to INT4 *without Degree-Quant*.

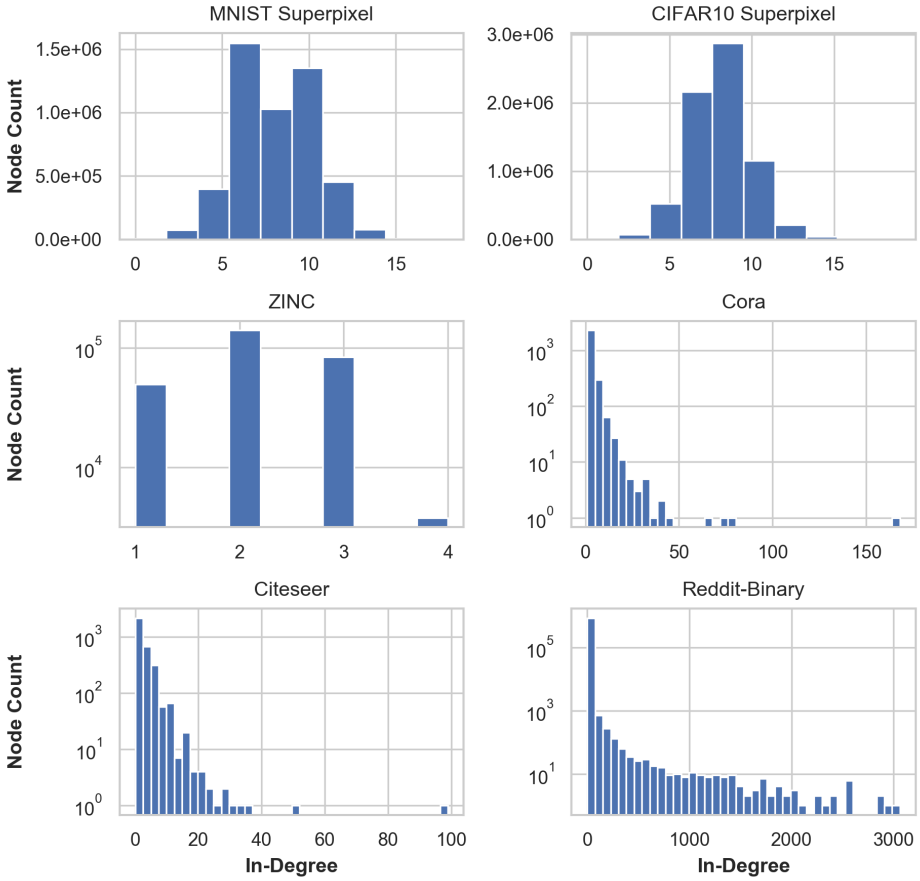


**Figure 9:** In-degree distribution for each of the six datasets assessed. Note that a log $y$-axis is used for all datasets except for MNIST and CIFAR-10.

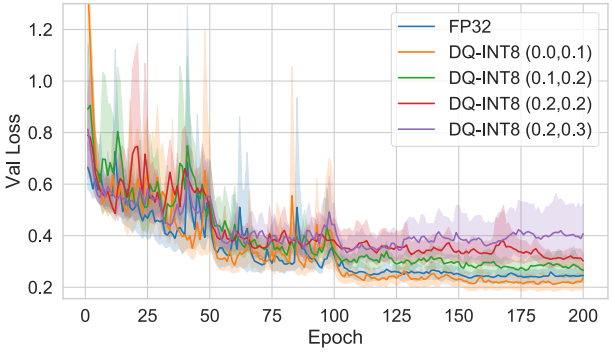


**Figure 10:** Validation loss curves for GIN models evaluated on REDDIT-BINARY. Results averaged across 10-fold cross-validation. We show four DQ-INT8 experiments each with a different values for ($p_{\min}$,$p_{\max}$) and our FP32 baseline.

| Quantization | Model | REDDIT-BIN ↑ |
|---|---|---|
| Ref. (FP32) | GIN | $92.2 \pm 2.3$ |
| Ours (FP32) | GIN | $92.0 \pm 1.5$ |
| DQ-INT8 (0.0, 0.1) | GIN | $91.8 \pm 2.3$ |
| DQ-INT8 (0.1, 0.2) | GIN | $90.1 \pm 2.5$ |
| DQ-INT8 (0.2, 0.2) | GIN | $89.0 \pm 3.0$ |
| DQ-INT8 (0.2, 0.3) | GIN | $88.1 \pm 3.0$ |

**Table 8:** Final test accuracies for FP32 and DQ-INT8 models whose validation loss curves are shown in fig. 10

| Quantization Scheme | Model Arch. | Node Classification Cora ↑ | Citeseer ↑ | Graph Regression ZINC ↓ |
|---|---|---|---|---|
| QAT-INT8 + DQ Masking | GCN | $81.1 \pm 0.6$ | $71.0 \pm 0.7$ | $0.468 \pm 0.014$ |
| | GAT | $82.1 \pm 0.1$ | $71.4 \pm 0.8$ | $0.462 \pm 0.005$ |
| | GIN | $78.9 \pm 1.2$ | $67.1 \pm 1.7$ | $0.347 \pm 0.028$ |
| QAT-INT4 + DQ Masking | GCN | $78.5 \pm 1.4$ | $62.8 \pm 8.5$ | $0.599 \pm 0.015$ |
| | GAT | $64.4 \pm 9.3$ | $68.9 \pm 1.2$ | $0.529 \pm 0.008$ |
| | GIN | $71.2 \pm 2.9$ | $56.7 \pm 3.8$ | $0.427 \pm 0.010$ |
| nQAT-INT4 + Percentile | GCN | $75.6 \pm 2.5$ | $64.8 \pm 3.8$ | $0.633 \pm 0.012$ |
| | GAT | $70.1 \pm 2.8$ | $51.4 \pm 3.4$ | $0.596 \pm 0.008$ |
| | GIN | $63.5 \pm 2.0$ | $46.3 \pm 4.1$ | $0.771 \pm 0.058$ |

**Table 9:** Ablation study against the two elements of Degree-Quant (DQ). The first two rows of results are obtained with only the stochastic element of Degree-Quant enabled for INT8 and INT4. Percentile-based quantization ranges are disabled in these experiments. The bottom row of results were obtained with noisy quantization (nQAT) at INT4 with the use of percentiles. DQ masking alone is often sufficient to achieve excellent results, but the addition of percentile-based range tracking can be beneficial to increase stability. We can see that using nQAT with percentiles is not sufficient to achieve results of the quality DQ provides.

| Device | Arch. | CIFAR-10 FP32 | CIFAR-10 W8A8 | CIFAR-10 Speedup | Cora FP32 | Cora W8A8 | Cora Speedup | Citeseer FP32 | Citeseer W8A8 | Citeseer Speedup |
|---|---|---|---|---|---|---|---|---|---|---|
| CPU | GCN | 182ms | 88ms | 2.1× | 0.94ms | 0.74ms | 1.3× | 0.97ms | 0.76ms | 1.3× |
| | GAT | 500ms | 496ms | 1.0× | 0.86ms | 0.78ms | 1.1× | 0.99ms | 0.88ms | 1.1× |
| | GIN | 144ms | 44ms | 3.3× | 0.85ms | 0.68ms | 1.3× | 0.95ms | 0.55ms | 1.7× |
| GPU | GCN | 2.1ms | 1.6ms | 1.3× | 0.08ms | 0.09ms | 0.9× | 0.09ms | 0.09ms | 1.0× |
| | GAT | 30.0ms | 27.1ms | 1.1× | 0.57ms | 0.64ms | 0.9× | 0.56ms | 0.64ms | 0.9× |
| | GIN | 20.9ms | 16.2ms | 1.2× | 0.09ms | 0.07ms | 1.3× | 0.09ms | 0.07ms | 1.3× |

**Table 10:** INT8 latency results run on a 22 core 2.1GHz Intel Xeon Gold 6152 and, on a GTX 1080Ti GPU. All layers have 128 in/out features. For CIFAR-10 we used batch size of 1K graphs.

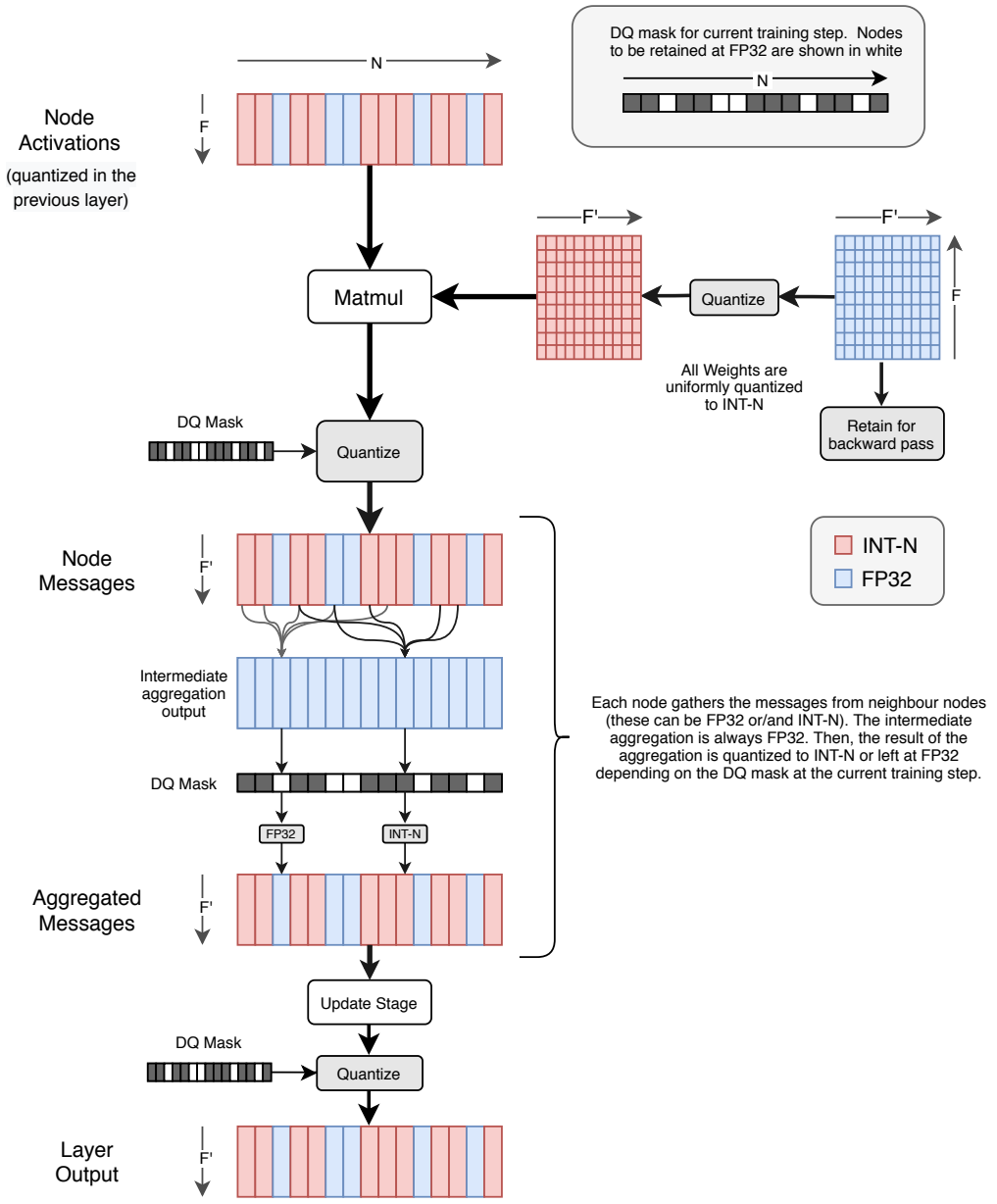


**Figure 11:** Diagram representing how DQ makes use of a topology-aware quantization strategy that is better suited for GNNs. The diagram illustrates this for a GCN layer. At every training stage, a degree-based mask is generated. This mask is used in all quantization layers located after each of the stages in the message-passing pipeline. By retaining at FP32 nodes with higher-degree more often, the noisy updates during training have a lesser impact and therefore models perform better, even at INT4.

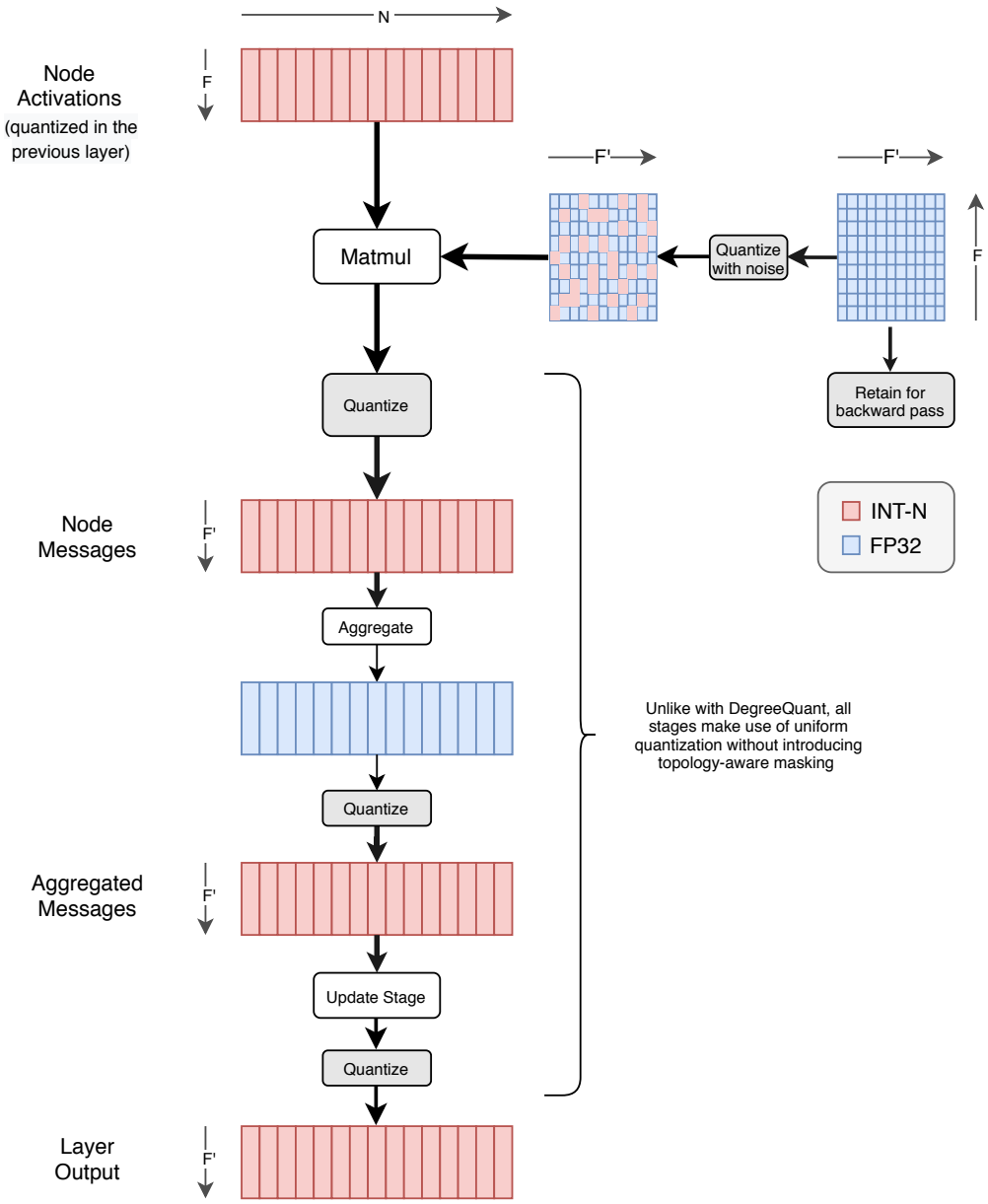


**Figure 12:** Diagram representing how nQAT is implemented for GNNs. The diagram illustrates this for a GCN layer. The stochastic stage only takes place when quantizing the weights, the remaining of the quantization modules happen following a standard QAT strategy. A QAT diagram would be similar to this one but fully quantizing the weights.

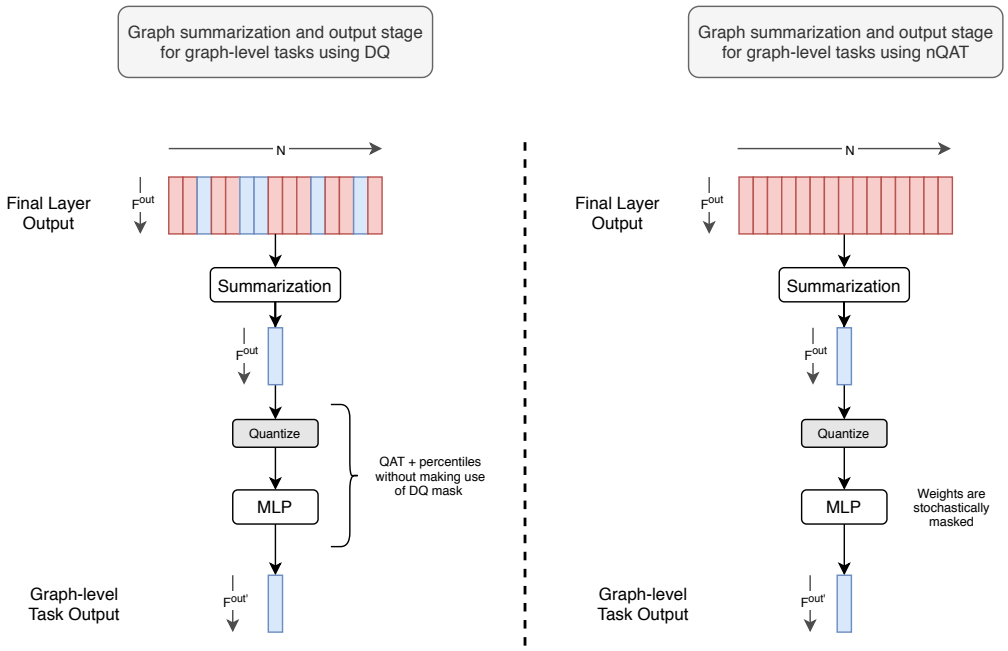


**Figure 13:** Diagrams representing how the output graph-summarization stages for graph-level tasks (e.g. graph classification, graph regression) are implemented when making use of DQ (left) and nQAT (right). GNNs making use of DQ during the node-aggregation stages (see fig. 11), do not use the stochastic element of DQ in the output MLP layers but still make use of percentiles. For models making use of nQAT, the final MLP still makes use of stochastic quantization of weights.