MQBench: Towards Reproducible and Deployable Model Quantization Benchmark

Yuhang Li*, Mingzhu Shen*, Jian Ma*, Yan Ren*, Mingxin Zhao*, Qi Zhang*, Ruihao Gong, Fengwei Yu, Junjie Yan
Toolchain Team, SenseTime Research
http://mqbench.tech/

Abstract

Model quantization has emerged as an indispensable technique to accelerate deep learning inference. While researchers continue to push the frontier of quantization algorithms, existing quantization work is often unreproducible and undeployable. This is because researchers do not choose consistent training pipelines and ignore the requirements for hardware deployments. In this work, we propose Model Quantization Benchmark (MQBench), a first attempt to evaluate, analyze, and benchmark the reproducibility and deployability for model quantization algorithms. We choose multiple different platforms for real-world deployments, including CPU, GPU, ASIC, DSP, and evaluate extensive state-of-the-art quantization algorithms under a unified training pipeline. MQBench acts like a bridge to connect the algorithm and the hardware. We conduct a comprehensive analysis and find considerable intuitive or counter-intuitive insights. By aligning up the training settings, we find existing algorithms have about-the-same performance on the conventional academic track. While for the hardware-deployable quantization, there is a huge accuracy gap and still a long way to go. Surprisingly, no existing algorithm wins every challenge in MQBench, and we hope this work could inspire future research directions.

1 Introduction

Modern deep learning is increasingly consuming larger memory and computation to pursue higher performance. While large-scale models can be trained on the cloud, transition to edge devices during deployment is notoriously hard due to the limited resource budget, including latency, energy and memory consumption. For this reason various techniques have been developed to accelerate the deep learning inference, including model quantization [1, 2, 3, 4, 5], pruning [6, 7, 8, 9, 10], neural network distillation [11, 12], lightweight network design [13], and weight matrix decomposition [14].

In this work, we focus on model quantization for efficient inference. Quantization targets to map the (nearly) continuous 32-bit floating-point (FP) numbers into discrete low-bit integers. As a result, the neural networks could rely on the integer-arithmetic units to speed up the inference. In academic research, there is a trend towards steadily reducing the bit-width and maintaining the accuracy across a range of quantized network architectures on ImageNet. It is incredible that the even 3-bit quantization of both weights and activations can reach FP-level accuracy [15]. Exciting though the breakthrough is, there lacks a systematic study that whether these research works can really be applied to practice, and whether the major improvement is brought by the algorithm rather than the training techniques.

We point out two long-neglected key factors in quantization research, namely reproducibility and deployability. First, we observe that the training hyper-parameters can significantly affect the performance of a quantized network. As an example, Esser et al. [15] adopt cosine annealed learning

*Equal Contributions.

rate [16] and better weight decay choice, improving the Top-1 accuracy of 2-bit ResNet-18 [17] by 0.7% and 0.4% on ImageNet. Full precision network pre-training can also boost quantization results [15, 18]. The reproducibility issue has received considerable attention in other areas as well, e.g. NAS-Bench-101 [19]. So far, there lacks a benchmark that unifies training pipelines and compares the quantization algorithms in a thorough and impartial sense.

Second, we find that the majority of the academic research paper does not testify the algorithm on real hardware devices. As a result, the reported performance may not be reliable. For one thing, hardware will fold Batch Normalization (BN) layers [20] into convolutional layers [3] to avoid additional overhead. But most research papers just keep BN layers intact. For another, research paper only considers quantizing the input and weights parameters of the convolutional layers. While in deployment the whole computational graph should be quantized. These rules will inevitably make quantization algorithms less resilient. Another noteless problem is the algorithm robustness: What will happen if one algorithm is applied to per-channel quantization but it is designed to per-tensor quantization at first? The algorithm should incorporate the diversity of quantizers design. All these problems suggest a large gap between academic research and real-world deployments.

In this work, we propose Model Quantization Benchmark (MQBench), a framework designed to analyze and reproduce quantization algorithms on several real-world hardware environments (See Fig. 1 & Table 1). We carefully studied existing quantization algorithms and hardware deployment settings to set up a bridge between the algorithms and hardware. To complete MQBench, we utilize over 50 GPU years of computation time, in an effort to foster both reproducibility and deployability in quantization research. Meanwhile, our benchmark offers some overlooked observations which may guide further research. To our best knowledge, this is the first work that benchmarks quantization algorithms on multiple general hardware platforms.

In the following context of this paper, we first build a benchmark for reproducing algorithms under unified training settings in Sec. 2. We introduce the requirements for hardware deployable quantization in Sec. 3. Then we conduct extensive experimental evaluation and analysis in Sec. 4. Due to the space limit, we put related work as well as the visualization results in the Appendix.

## 2 MQBench: Towards Reproducible Quantization

In this section, we benchmark the reproducibility of quantization algorithms, mainly including Quantization-Aware Training (QAT)\(^2\). We evaluate the performance of algorithms on ImageNet [21] classification task. Other tasks like detection, segmentation and language applications are not considered for now since few baseline algorithms were proposed. MQBench evaluation is performed in 4 dimensions: supported hardware library, algorithm, network architecture, and bit-width.

**Hardware-aware Quantizer.** Throughout the paper, we mainly consider uniform quantization, since the non-uniform quantization requires special hardware design. We use \( w \) and \( z \) to denote the weight matrix and activation matrix in a neural network. A complete uniform quantization process includes quantization operation and de-quantization operation, which can be formulated by:

\[
\bar{w} = \text{clip}( \frac{w}{s} + z, \min, \max ), \quad \hat{w} = s \cdot (\bar{w} - z) \quad (1)
\]

\(^2\)We also build an equally thoroughgoing benchmark for Post-Training Quantization (PTQ) in Appendix. C.
Two scale.

We do not experiment with 3-bit quantization because it is undeployable on general hardware. As for 2-bit quantization, we find most of the algorithm will crash on hardware settings. However, POT scale may offer further speed-up.

Table 2: Comparison of (1) the different hardware we selected and (2) the different QAT algorithms. Infer. Lib. is the inference library; FBN means whether fold BN. * means undeployable originally, but can be deployable when certain requirements are satisfied.

<table>
<thead>
<tr>
<th>Infer. Lib.</th>
<th>Provider</th>
<th>HW Type</th>
<th>Hardware</th>
<th>s Form.</th>
<th>Granularity</th>
<th>Symmetry</th>
<th>Graph</th>
<th>FBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorRT [22]</td>
<td>NVIDIA</td>
<td>GPU</td>
<td>Tesla T4/P4</td>
<td>FP32</td>
<td>Per-channel</td>
<td>Symmetric</td>
<td>2</td>
<td>✓</td>
</tr>
<tr>
<td>ACL [23]</td>
<td>HUAWEI</td>
<td>ASIC</td>
<td>Ascend310</td>
<td>FP32</td>
<td>Per-channel</td>
<td>Asymmetric</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>TVM [25]</td>
<td>OctoML</td>
<td>CPU</td>
<td>ARM</td>
<td>POT</td>
<td>Per-tensor</td>
<td>Symmetric</td>
<td>3</td>
<td>✓</td>
</tr>
<tr>
<td>SNPE [24]</td>
<td>Qualcomm</td>
<td>DSP</td>
<td>Snapdragon</td>
<td>FP32</td>
<td>Per-tensor</td>
<td>Asymmetric</td>
<td>3</td>
<td>✓</td>
</tr>
<tr>
<td>FBGEMM [26]</td>
<td>Facebook</td>
<td>CPU</td>
<td>X86</td>
<td>FP32</td>
<td>Per-channel</td>
<td>Asymmetric</td>
<td>3</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Deployable</th>
<th>Uniformity</th>
<th>Quant. Type</th>
<th>s Form.</th>
<th>Granularity</th>
<th>Symmetry</th>
<th>Graph</th>
<th>FBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSQ [15]</td>
<td>*</td>
<td>Uniform</td>
<td>learning-based</td>
<td>FP32</td>
<td>Per-tensor</td>
<td>Symmetric</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>APoT [27]</td>
<td>✓</td>
<td>Non-uniform</td>
<td>learning-based</td>
<td>FP32</td>
<td>Per-tensor</td>
<td>Symmetric</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>QIL [18]</td>
<td>✓</td>
<td>Uniform</td>
<td>learning-based</td>
<td>FP32</td>
<td>Per-tensor</td>
<td>Symmetric</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>DSQ [28]</td>
<td>*</td>
<td>Uniform</td>
<td>rule-based</td>
<td>FP32</td>
<td>Per-tensor</td>
<td>Symmetric</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>LQ-Net [29]</td>
<td>✓</td>
<td>Non-uniform</td>
<td>rule-based</td>
<td>FP32</td>
<td>Per-tensor</td>
<td>Symmetric</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>PACT [30]</td>
<td>*</td>
<td>Uniform</td>
<td>learning+rule</td>
<td>FP32</td>
<td>Per-tensor</td>
<td>Symmetric</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>DoReFa [31]</td>
<td>*</td>
<td>Uniform</td>
<td>rule-based</td>
<td>FP32</td>
<td>Per-tensor</td>
<td>Symmetric</td>
<td>1</td>
<td>✓</td>
</tr>
</tbody>
</table>

where \( s \in \mathbb{R}_+ \) and \( z \in \mathbb{Z} \) are called scale and zero-point, respectively. \( \lfloor \cdot \rfloor \) rounds the continuous numbers to nearest integers. Eq. (1) first quantizes the weights or activations into target integer range \([N_{min}, N_{max}]\) and then de-quantizes the integers to original range. Given \( t \) bits, the range is determined by \([-2^{t-1}, 2^{t-1} - 1]\). We can divide the quantizer based on several metrics: (1) Symmetric or asymmetric quantization: For symmetric quantization the zero-point is fixed to 0, while the asymmetric quantization has an adjustable zero-point to adapt different range; (2) Per-tensor or per-channel quantization: The per-tensor quantization uses only one set of scale and zero-point for a tensor in one layer while per-channel quantization quantizes each weight kernel independently (i.e. for each row of weight matrix: \( w_i \)); (3) FP32 (32-bit Floating Point) scale or POT (Power of Two) scale: FP32 scale is nearly continuous, while power-of-two scale is much more challenging.

We select 5 general hardware libraries to evaluate the quantization algorithms, including NVIDIA’s TensorRT [22] for Graphics Processing Unit (GPU) inference, HUAWEI’s ACL [23] for Application-Specific Integrated Circuit (ASIC) inference, Qualcomm’s SNPE [24] for mobile Digital Signal Processor (DSP), TVM [25] for ARM Central Processing Unit (CPU), FBGEMM [26] for X86 server-side CPU. We summarize their implementation details for quantization in Table 2 upper side. Each hardware setting corresponds to a unique quantizer design. Thus, the developed algorithm must be robust to adapt different quantizer configurations. We put the detailed setup for hardware environments in Appendix. E.

Algorithm. For quantization-aware training, we compare 6 different algorithms [15, 18, 27, 28, 29, 30, 31]. However, several algorithms cannot be deployable even if we align up the quantizer configuration and the other requirements. We put the summary of them in Table 2 lower side. All these algorithms use per-tensor, symmetric settings. We refer this type as academic setting. We also identify the quantizer type with learning-based which learns the scale and rule-based which directly computes the scale with heuristics. For a detailed description of these algorithms and the reason why they can be extended to deployable quantization, please see Appendix. F.

Network Architecture. We choose ResNet-18 and ResNet-50 [17] as they are most widely used baseline architectures. We also adopt MobileNetV2 [13] which is a lightweight architecture with depthwise separable convolution. In order to quantize EfficientNet [32], we leverage its Lite version [33] that excludes the squeeze-and-excitation block and replaces swish activation to ReLU6 for better integer numeric support on hardware. Finally, we add another advanced architecture RegNetX-600MF [34] with group convolution.

Bit-width. In this paper, we mainly experiment with 8-bit post-training quantization (Appendix. C) and 4-bit quantization-aware training. To test the accuracy of the quantized model, we simulate the algorithm with fake quantization (see difference between fake and real quantization in Sec. 3.1). Unlike the reported results in other paper, 4-bit QAT in our benchmark could be very challenging. We do not experiment with 3-bit quantization because it is undeployable on general hardware. As for 2-bit quantization, we find most of the algorithm will crash on hardware settings.
We should point out that these two graphs may have some tiny and unavoidable disparity, mainly resulting from the difference between FP32 in simulation and real integers in deployments.

2.1 Training Pipelines and Hyper-parameters

Early work like [30, 31] trains the quantized model from scratch, which may have inferior accuracy than fine-tuning [35]. Besides, each paper may have different pre-trained models for initialization. In MQBench, we adopt fine-tuning for all algorithms and each model is initialized by the same pre-trained model, eliminating the inconsistency at initialization.

We adopt standard data prepossessing for training data, including RandomResizeCrop to 224 resolution, RandomHorizontalFlip, ColorJitter with brightness= 0.2, contrast= 0.2, saturation = 0.2, and hue= 0.1. The test data is centered cropped to 224 resolution. We use 0.1 label smoothing in training to add regularization. No other advanced augmentations are further adopted. All models are trained for 100 epochs, with a linear warm-up in the first epoch. The learning rate is decayed by cosine annealing policy [16]. We use the SGD optimizer for training, with 0.9 momentum and Nesterov updates. Other training hyper-parameters can be found in Table 3 aside. We discuss our choice of this set of hyper-parameter in the Appendix. A.

3 MQBench: Towards Deployable Quantization

3.1 Fake and Real Quantization

Given a weight matrix \( \hat{w} \) and a activation matrix \( \hat{x} \), the product is given by

\[
y_{ij} = \sum_{k=1}^{n} \hat{w}_{ik} \hat{x}_{kj} = s_w s_x \sum_{k=1}^{n} (\hat{w}_{ik} \hat{x}_{kj} - z_w \hat{x}_{kj} - z_x \hat{w}_{ik} + z_w z_x),
\]

where \( y \) is the convolution output or the pre-activation. In order to perform QAT on GPU, we have to simulate the quantization function with FP32, denoted as the left Fake Quantize bracket. For the practical inference acceleration, we have to utilize integer-arithmetic-only [3], denoted as the right Real Quantize bracket. In Fig. 2, we draw the computational graph of fake quantization and real quantization to reveal their relationship.

For fake quantization, the weights and input activations are quantized and de-quantized before convolution. The intermediate results as well as the bias term, are all simulated with FP32.

As for deployments in real-world, the computation in the Real Quantize bracket is integer-only and is accumulated using INT32. One can further optimize the convolution kernels by performing the last two terms offline, since \( w \) and \( z_w, z_x \) are determined prior to deployment [36]. For bias parameters, we can keep them in INT32, quantized by \( b = \frac{b}{s_b} \), where \( b \) is INT32 and \( s_b = s_w s_x \). As a result, the bias can be easily fused into Eq. (2). Then, the de-quantization will do the scaling outside the bracket. In the deployment, the de-quantization of the output and the further quantization to integers is fused together, called Requantization, given by

\[
\hat{x}^{t+1} = s_w \cdot s_x \cdot x^{t+1}, \quad \hat{x}^{t+1} = \text{clip}(\frac{x^{t+1}}{s_{\hat{x}^{t+1}}}, N_{\min}, N_{\max})
\]

We should point out that these two graphs may have some tiny and unavoidable disparity, mainly resulting from the difference between FP32 in simulation and real integers in deployments.
ReLU+Quant \( \times \) Conv (a)  
\[ \mathbf{z} + \mathbf{b} \]  
FQuant + \( \times \)  
Update

\( \mathbf{x} \)

Figure 3: Comparison of different Batch Normalization folding technologies. (a) Removing BN layers and directly update \( w_{\text{fold}} \). (b) BN folding without any statistics update. (c) BN folding with two convolutions. (d) folding with running statistics and also requires two convolutions. (e) folding running statistics with an explicitly BN layer in training. Graph (bcde) can be transformed to (a) during inference. FQuant=FakeQuantize.

3.2 Folding Batch Normalization

Batch Normalization (BN) layers are designed to reduce internal covariate shift [20] and also smooth the loss surface [37] for fast convergence. BN introduces a two-step linear transformation for each convolutional layer output. In inference, the linear transformations could be fused into convolutional kernels so that no extra operations are needed, given by:

\[
\mathbf{w}_{\text{fold}} = \mathbf{w} \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}}, \quad \mathbf{b}_{\text{fold}} = \beta + (\mathbf{b} - \mu) \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}},
\]

where \( \mu, \sigma^2 \) are the running mean, variance and \( \gamma, \beta \) are the affine weight, bias, respectively. \( \epsilon \) is for numerical stability (for simplicity, we omit this term in the rest of the paper). If we put quantization after the folding of BN layers, there will be no extra floating-point operations during inference. However, BN folding does not draw much attention in existing literature and causes deployment issues. In this section, we will discuss 5 possible strategies for BN folding. We denote the current batch mean and variance as \( \tilde{\mu}, \tilde{\sigma}^2 \). The diagram of these 5 types are demonstrated in Fig. 3.

Strategy 0: Merge the parameters into weights and bias with Eq. (4), and remove this layer entirely (Fig. 3(a)). We find this choice cannot be trained with large learning rate because of the gradient explosion without BN. Consequently, extensive hyper-parameter searching is necessary.

Strategy 1: Fold BN layers and do not update the running statistics (Fig. 3(b)). Nevertheless, the affine parameters \( \gamma, \beta \) can still be updated with SGD. We find this strategy can still smooth the loss landscape and achieve comparable accuracy even no statistics are updated. This folding strategy also significantly reduces the training time by avoiding statistics synchronization.

Strategy 2: Introduced in [3], this folding strategy can update running statistics (Fig. 3(c)). The convolution will be calculated twice during training, which causes additional overhead. The first time is to compute the batch mean and variance \( \tilde{\mu}, \tilde{\sigma}^2 \) using FP32 weights. Then, the current batch mean, variance are folded into weight kernels. During inference, the weights and biases will be folded using running mean and variance as in Eq. (4).

Strategy 3: Introduced in [1], this option also calculates the convolution twice. The first time is the same with strategy 2, which will estimate \( \tilde{\mu}, \tilde{\sigma}^2 \). However, in this strategy the weights will be folded with running statistics to avoid the undesired fluctuations of the batch statistics. The batch variance factor will be used to re-scale the output after the second convolution, as shown in Fig. 3(d).

Strategy 4: Introduced in PyTorch quantization repo [38], this option does not cost two times convolution but explicitly adds BN layers after the quantized convolution. One of the benefits brought by this strategy is that batch statistics are calculated based on quantized weight. During inference, the re-scaling of output \( \frac{\tildesum}{\gamma} \) can be neutralized by BN, therefore the graph can be transformed to Fig. 3(a).

3.3 Block Graph

Most academic papers only consider quantizing the input and the weight kernels of convolutional or fully connected layers. However, modern neural architectures includes other operations, like
elementwise-add in ResNet [17] and concatenation in InceptionV3 [39]. In addition, different hardware will consider different levels of graph optimization and can propose different solutions to construct a graph for quantized neural networks. In MQBench, we sort out different implementations and summarize them in a schematic diagram (Fig. 4). Note that Fig. 4 only gives an example of a basic block in ResNet-18/-34. The bottleneck block in ResNet-50 can be naturally derived from this diagram. We also put the diagram of the inverted residual bottleneck of MobileNetV2 [13], and concatenation quantization in the Appendix. G.

Fig. 4 left shows the conventional academic implementations for quantizing a basic block. Only the input of convolutional layers will be quantized to 8-bit. (Note that in academic papers the INT8 means both quantization and de-quantization.) The block input and output as well as the elementwise-add all operate at full precision. Consequently, the network throughput hasn’t been reduced, and will significantly affect the latency. In some architectures, this graph even won’t bring any acceleration since the latency is dominated by I/O. Another problem is the separate quantization of the activation in the downsample block, which also brings undesired costs.

Fig. 4 middle presents the NVIDIA’s TensorRT [22] implementation for basic block. We can find that the input and output must be quantized to low-bit to reduce the data throughput. Low bit input can ensure two branches will use the same quantized activation in the downsample block. As for the elementwise-add layer, it will be conducted in a 32-bit mode due to the fusion with one of the former convolutional layer’s bias addition. Thus only one of its inputs will be quantized. Fig. 4 right demonstrates the implementation in other hardware library, such as FBGEMM [26] and TVM [25]. The only difference is that they require all inputs of the elementwise-add to be quantized. In 4-bit symmetric quantization, this can severely affect the accuracy.

4 MQBench Evaluation

In this section, we conduct a thorough evaluation and analysis for quantization algorithms, network architectures, and hardware. We study several evaluation metrics, given by: ① **test accuracy**: the Top-1 accuracy on the ImageNet validation set, it directly reflects the task performance of the algorithm; ② **hardware gap**: the difference between hardware and academic test accuracy, this metric can reflect the impact of deployable quantization on the algorithm; ③ **hardware robustness**: the average test accuracy on 5 hardware environments; ④ **architecture robustness**: the average test accuracy on 5 network architectures. These last two metrics are often neglected by most of the existing literature but may have a significant value. In the Appendix. B, we include more study and provide the diagnostic information for the quantization benchmark.

4.1 Evaluation with Academic Setting.

We first revisit the performance of quantization in academic settings (per-tensor, symmetric quantization without any bn folding, etc.). This setting is predominately adopted in the research paper. We summarize our benchmark results and the originally reported results (in bracket) in Table 4. By aligning up the training hyper-parameters and pipeline, we find several surprising results.
Table 4: Academic setting benchmark for 4-bit quantization-aware training, result in bracket is the reported accuracy in original paper. "NC" denotes not converged.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>70.7</td>
<td>70.5</td>
<td><strong>70.8</strong></td>
<td>70.0</td>
<td>70.5</td>
<td>70.0</td>
</tr>
<tr>
<td>ResNet-50</td>
<td><strong>77.4</strong></td>
<td>77.1</td>
<td>77.2</td>
<td>76.4</td>
<td>76.3</td>
<td>76.4</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>70.6</td>
<td>68.6</td>
<td>70.3</td>
<td>69.6</td>
<td><strong>70.7</strong></td>
<td>NC</td>
</tr>
<tr>
<td>EfficientNet-Lite0</td>
<td>72.6</td>
<td>70.0</td>
<td>72.7</td>
<td>72.6</td>
<td><strong>73.0</strong></td>
<td>NC</td>
</tr>
<tr>
<td>RegNetX-600MF</td>
<td>72.7</td>
<td><strong>73.0</strong></td>
<td>71.6</td>
<td>71.7</td>
<td>72.2</td>
<td>72.9</td>
</tr>
<tr>
<td>Avg. Arch.</td>
<td><strong>72.8</strong></td>
<td>71.9</td>
<td>72.5</td>
<td>72.1</td>
<td>72.5</td>
<td>44.0</td>
</tr>
</tbody>
</table>

1. The difference between algorithms is not as significant as reported in the original paper. As can be seen, the maximum Top-1 accuracy difference of ResNet-18 is 0.8% (= QIL - DSQ), which is much smaller than 3.0% as compared in the original paper. This phenomenon is even more evident in the case of ResNet-50, where the maximum difference as reported is 5.3% while the actual maximum difference is only 1.1%. This suggests that 80% (4/5.3) of the accuracy improvement from DoReFa to LSQ is made from better training techniques, only 20% comes from the superiority of the algorithm. On MobileNetV2, prior work reported PACT and DSQ with only 61.4%, 64.8% accuracy, however, our setting can obtain 70.7%, 69.6% candidates respectively, indicating the importance of the unified training hyper-parameters.

2. No algorithm achieves the best or the worst performance on every architecture. Among 5 different network architectures, there are 4 different winner solutions and 4 different worst solutions. Such an outcome demonstrates that existing algorithms cannot adapt every network architecture very well. We encourage studying the architecture robustness, which is the mean accuracy across architectures. In that case, LSQ achieves the highest robustness. However, the improvement in robustness is also not as evident as we expected before.

3. The rule-based algorithms can achieve comparable performance with learning-based algorithms. DoReFa-Net [31], which simply clips the activation to [0, 1], reaches the same 70.7% test accuracy as LSQ [15] on ResNet-18. It also surpasses the PACT by 0.2%, revealing that even a handcrafted fixed clipping range with the right training pipelines can have state-of-the-art accuracy. Although, DoReFa fails to quantize depthwise conv networks, e.g. MobileNetV2. We believe this is due to the activation range in those networks are much larger than ResNet-family (as can be shown in our diagnostic information in Appendix. B). Nevertheless, we believe rule-based quantization can achieve better performance if a good range can be found in advance.

4.2 Evaluation with BN Folding

We then study the BN folding strategies designed for QAT. We choose LSQ [15] and PACT [30], running on ResNet-18 and MobileNetV2 for ablation study. Here we do not employ any real hardware-aware quantizer but only modify the conventional academic settings (i.e. per-tensor, symmetric) to accommodate BN folding. The results are summarized in Table 5. During our experiments, we have the following observations:

1. BN folding is sensitive to quantization algorithms, and strategy 4 works best generally. We first examine the LSQ with BN folding, where we find the algorithm converges to a similar performance with normal BN QAT, on both ResNet-18 and MobileNetV2. Unlike LSQ, BN folding has a severe impact on PACT quantized models. All folding strategies except 4 fail to converge on ResNet-18. Even using strategy 4 will decrease 2.7% accuracy. For MobileNetV2, the decrease is more significant (9.9%).

2. Strategy 4 does not obtain any significant speed-up than strategy 2, 3 even it only computes one-time convolution. Although strategy 4 is faster than 2.3 in forward computation as it has less computation, but it is much slower in gradient calculation. On ResNet-18 LSQ, we find strategy 4 costs 80% more time than 2.3 to do backpropagation.

3. Updating batch statistics may not be necessary for BN-folding-friendly algorithms like LSQ. As an example, using strategy 1 in LSQ only drops 0.2%~0.3% accuracy than those who update the batch mean and variance. Moreover, strategy 1 can be 30%~50% faster than them since it does not need to compute or synchronize statistics and has much faster backpropagation.
Table 5: Comparison of the accuracy on a 4-bit quantized ResNet-18 and MobileNetV2, using LSQ [15] and PACT [30], given different folding strategies ("-1" denotes normal BN training without folding, others are folding strategies introduced in Sec. 3.2); "NC" denotes Not Converged; "*" denotes asynchronized statistics.

<table>
<thead>
<tr>
<th>Folding Strategy</th>
<th>ResNet-18</th>
<th>MobileNetV2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-1  0  1  2  3  4  4*</td>
<td>-1  0  1  2  3  4  4*</td>
</tr>
<tr>
<td>LSQ</td>
<td>70.7 69.8 70.1 70.2 70.3 <strong>70.4</strong> 70.1 70.6 69.9 70.0 <strong>70.1</strong> 70.1 64.8</td>
<td></td>
</tr>
<tr>
<td>PACT</td>
<td>70.5 NC NC NC NC NC 67.8 65.5 70.7 NC NC NC NC 60.8 NC</td>
<td></td>
</tr>
</tbody>
</table>

4 Synchronization of BN statistics in data-parallel distributed learning can improve accuracy.
   In distributed training with normal BN, asynchronous BN statistics across devices are acceptable and will not affect the final performance as long as the batch size is relatively large. However, in QAT folding BN into weights with asynchronous statistics will produce different quantized weights, further magnifying the training instability. Synchronization needs time, therefore it is an accuracy-speed tradeoff. SyncBN can improve 2.3% accuracy for PACT ResNet-18.

5 Folding BN will incur severe instability in the initial training stage and this can be alleviated effectively by learning rate warm-up. Without warm-up, most QAT with BN folding will fail to converge. Therefore, for all the rest experiments which require BN folding, we employ 1 epoch learning rate linear warm-up, synchronized BN statistics, and strategy 4 to facilitate it.

4.3 Evaluation with Graph Implementation

In this section, we study the effect of different computation graphs for quantization networks and algorithms. Graph 1,2,3 correspond to graph implementations in Fig. 4(a), (b), (c). Following BN folding experiments, we modify the academic settings and only change the graph implementations. PACT and LSQ are selected for this study, conducted on ResNet-18 and MobileNetV2. The results in Table 6 show the final performance. We find: unlike BN folding, which is sensitive to algorithms, the graph implementation is sensitive to the network architecture. For instance, PACT only drops 1.1% accuracy on ResNet-18 by switching graph from 1 to 3. However, the gap can increase to 2.9% on MobileNetV2. The same trend is also observed for LSQ, where 0.4% and 3.6% accuracy degradation are observed on ResNet-18 and MobileNetV2, respectively.

<table>
<thead>
<tr>
<th>Graph</th>
<th>ResNet-18</th>
<th>MobileNetV2</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSQ [15]</td>
<td>70.7 70.7 70.3 70.6 67.5 67.0</td>
<td></td>
</tr>
<tr>
<td>PACT [30]</td>
<td>70.5 70.3 69.4 70.7 68.3 67.8</td>
<td></td>
</tr>
</tbody>
</table>

4.4 4-bit QAT

In this section we establish a major baseline for existing and future work by using our deployable and reproducible benchmark to compare some popular algorithms. We experiment with 4-bit quantization-aware training. Unlike academic settings in Table 4 where 4-bit quantization is near-lossless, we showcase the challenging nature of deployable quantization. Like [19], our intention is not to provide a definite answer to “Which methods work best on this benchmark?”, but rather to demonstrate the utility of a reproducible and deployable baseline. And hopefully, with newly discovered insights, we can guide the future study on quantization algorithms. The major results are presented in Table 7.

Test Accuracy. We apply the algorithm to 5 distinct real-world hardware deployment environments and reported their fake quantization performance. We first visit the absolute test accuracy. As can be seen from the table, we observe no algorithms achieve the best absolute performance on every setting. Among the 25 settings (5 architectures × 5 hardware environments), LSQ [15] obtains the best performance in 52% cases. PACT [30] and DoReFa [31] attain the rest 38% and 10% best practices. Although DSQ [28] does not achieve any best practice, we should never rank it to the last one. In many cases, DoReFa and PACT do fail to converge, while DSQ can have a good performance. We cannot simply rank each algorithm based on a single metric. Moreover, we also define a metric quantization difficulty as the maximum accuracy difference in a specific combination of architecture.
Table 7: 4-bit Quantization-Aware Training benchmark on the ImageNet dataset, given different algorithms, hardware inference libraries, and architectures. "NC" means not converged. Red and Green numbers denotes the decrease and increase of the hardware deployable quantization.

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>Paper Acc.</th>
<th>Academic</th>
<th>TensorRT</th>
<th>ACL</th>
<th>TVM</th>
<th>SNPE</th>
<th>FBGEMM</th>
<th>Avg.</th>
<th>HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>LSQ [15]</td>
<td>71.1 / 70.7</td>
<td>70.7</td>
<td>69.3 (1.4)</td>
<td>70.2 (0.5)</td>
<td>67.3 (3.0)</td>
<td>69.7 (1.0)</td>
<td>69.8 (0.9)</td>
<td>69.3 (0.8)</td>
<td>69.5 (0.75)</td>
</tr>
<tr>
<td>FP: 71.0</td>
<td>PACT [30]</td>
<td>69.2</td>
<td>70.5</td>
<td>69.1 (1.4)</td>
<td>70.4 (0.1)</td>
<td>57.5 (13.0)</td>
<td>69.3 (1.2)</td>
<td>69.7 (0.8)</td>
<td>67.2 (4.87)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DoReFa [31]</td>
<td>68.1</td>
<td>70.7</td>
<td>69.6 (1.1)</td>
<td>70.4 (0.3)</td>
<td>62.8 (2.5)</td>
<td>69.8 (1.8)</td>
<td>69.7 (1.0)</td>
<td>69.4 (0.75)</td>
<td></td>
</tr>
<tr>
<td>ResNet-50</td>
<td>LSQ [15]</td>
<td>76.7</td>
<td>77.4</td>
<td>76.3 (1.1)</td>
<td>76.5 (0.9)</td>
<td>75.9 (1.5)</td>
<td>76.2 (1.2)</td>
<td>76.4 (1.0)</td>
<td>76.3 (0.21)</td>
<td></td>
</tr>
<tr>
<td>FP: 77.0</td>
<td>DSQ [28]</td>
<td>N/A</td>
<td>76.4</td>
<td>74.8 (1.6)</td>
<td>76.2 (0.2)</td>
<td>74.4 (2.0)</td>
<td>75.9 (0.5)</td>
<td>76.0 (0.4)</td>
<td>75.5 (0.72)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PACT [30]</td>
<td>76.5</td>
<td>76.3</td>
<td>76.3 (0.0)</td>
<td>76.1 (0.2)</td>
<td>NC</td>
<td>NC</td>
<td>76.6 (0.3)</td>
<td>55.8 (1.37)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DoReFa [31]</td>
<td>71.4</td>
<td>76.4</td>
<td>76.2 (2.0)</td>
<td>76.3 (0.1)</td>
<td>NC</td>
<td>NC</td>
<td>75.9 (0.5)</td>
<td>45.7 (3.3)</td>
<td></td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>LSQ [15]</td>
<td>66.3</td>
<td>70.6</td>
<td>66.1 (4.5)</td>
<td>68.1 (2.5)</td>
<td>64.5 (6.1)</td>
<td>66.3 (4.3)</td>
<td>65.5 (5.1)</td>
<td>66.1 (1.8)</td>
<td></td>
</tr>
<tr>
<td>FP: 72.6</td>
<td>PACT [30]</td>
<td>61.4</td>
<td>70.7</td>
<td>66.5 (4.2)</td>
<td>70.3 (0.4)</td>
<td>48.1 (22.6)</td>
<td>60.3 (10.4)</td>
<td>66.5 (4.2)</td>
<td>62.3 (7.8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DoReFa [31]</td>
<td>N/A</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>0 ± 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EfficientNet-Lite0</td>
<td>LSQ [15]</td>
<td>N/A</td>
<td>72.6</td>
<td>67.0 (5.6)</td>
<td>65.5 (7.1)</td>
<td>65.0 (7.6)</td>
<td>68.6 (4.0)</td>
<td>66.9 (6.7)</td>
<td>66.6 (1.27)</td>
<td></td>
</tr>
<tr>
<td>FP: 75.3</td>
<td>PACT [30]</td>
<td>N/A</td>
<td>72.6</td>
<td>35.1 (37.5)</td>
<td>69.6 (3.0)</td>
<td>NC</td>
<td>7.5 (65.1)</td>
<td>45.9 (26.7)</td>
<td>31.6 (25.5)</td>
<td></td>
</tr>
<tr>
<td>RegNetX-600MF</td>
<td>LSQ [15]</td>
<td>N/A</td>
<td>72.7</td>
<td>72.5 (0.2)</td>
<td>72.8 (0.1)</td>
<td>70.0 (2.7)</td>
<td>72.5 (0.2)</td>
<td>72.5 (0.2)</td>
<td>72.1 (0.84)</td>
<td></td>
</tr>
<tr>
<td>FP: 73.7</td>
<td>PACT [30]</td>
<td>N/A</td>
<td>71.7</td>
<td>68.6 (2.1)</td>
<td>71.4 (0.3)</td>
<td>64.5 (7.2)</td>
<td>70.0 (1.7)</td>
<td>70.0 (1.7)</td>
<td>68.9 (2.37)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DoReFa [31]</td>
<td>N/A</td>
<td>72.2</td>
<td>72.0 (0.2)</td>
<td>73.3 (1.1)</td>
<td>NC</td>
<td>NC</td>
<td>72.5 (0.3)</td>
<td>43.6 (3.55)</td>
<td></td>
</tr>
</tbody>
</table>

Accuracy reported in [30, 40, 41, 42], respectively.

and hardware. Overall, 20% settings has < 1% difficulty, while 60% settings has > 10% difficulty. This exhibits the diverse settings of our benchmark, and also the complexity of quantization on real-world hardware.

**Hardware Gaps.** We also investigate the hardware gap metric, which means the degradation when transiting from academic setting to hardware setting. The values are marked with colored numbers in the table. Notably, 93% of the experiments encounter accuracy drop. Among them, 25.8% settings drop within 1.0% accuracy, 47.3% settings drop within 3.0% accuracy. Similar to our findings in absolute accuracy, no algorithms achieve the least hardware gap in every setting. LSQ only has a 36% probability to win the hardware gap metric while PACT has a probability of 48%.

**Hardware & Architecture Robustness.** For architecture robustness, the order could be summarized as ResNet-18 > RegNet > ResNet-50 > MobileNetV2 > EfficientNet. On ResNet-18, each algorithm can converge to relatively high accuracy. In comparison, on EfficientNet the average hardware gap reaches 19%. We also explore the robustness of quantization algorithms when they are applied to various hardware. Generally, LSQ has the best hardware robustness. It brings much more stable performance on different hardware. However, we find **LSQ is not suitable for per-channel quantization.** For all 15 per-channel hardware settings, LSQ only wins 23% cases, while 66.7% of the trophy is claimed by PACT. LSQ exhibits exciting superiority in per-tensor quantization, where it wins 90% cases. This result indicates the importance of the hardware robustness metric.

5 Discussion

In this work we have introduced MQBench, a systematic tabular study for quantization algorithms and hardware. To foster reproducibility, we align up the training hyper-parameters and pipelines for quantization algorithms. To foster deployability, we sort out 5 hardware deployments settings and benchmark the algorithms on them across 5 network architectures. We conduct a thorough evaluation, focusing on the less explored aspects of model quantization, including BN folding, graph implementations, hardware-aware quantizer, etc. However, MQBench also has limitations, beyond image classification, quantization faces more challenges in deployments like object detection and NLP application. More advanced algorithms are also needed for a complete benchmark. Be that as it may, we hope MQBench will be the first of a continually improving sequence of rigorous benchmarks for the quantization field.
References


[19] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. 
Nas-bench-101: Towards reproducible neural architecture search. In International Conference 


[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale 

[22] NVIDIA. Tensorrt: A c++ library for high performance inference on nvidia gpus and deep 


Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated end-to-end 
optimizing compiler for deep learning. In 13th {USENIX} Symposium on Operating Systems 

[26] Facebook. Fbgemm (facebook general matrix multiplication) is a low-precision, high-
performance matrix-matrix multiplications and convolution library for server-side inference. 

[27] Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient 
non-uniform discretization for neural networks. In International Conference on Learning 
Representations, 2019.

[28] Ruizhao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei 
Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural 

[29] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization 

Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized 

[31] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: 

networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th 
International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning 

org/2020/03/higher-accuracy-on-vision-models-with-efficientnet-lite. 

[34] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing 
network design spaces. In Proceedings of the IEEE/CVF Conference on Computer Vision 

[35] Jeffrey L McKinstry, Steven K Esser, Rathinakumar Appuswamy, Deepika Bablani, John V 
Arthur, Izet B Yildiz, and Dharmendra S Modha. Discovering low-precision networks close 
2018.


Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes] Yes
   (b) Did you describe the limitations of your work? [Yes] We discuss the limitations in Sec. 6.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Our code will be submitted via a GitHub link.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] For hyper-parameters specification please refer to Sec. 2 and for hyper-parameters justification please refer to Appendix A.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Due to the huge computational resource required by QAT experiments, we do not report error bars. For PTQ experiments we report the standard deviation. However, we intend to report error bars of QAT in the future.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Please see Sec. 2

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Our work mainly investigate the ImageNet [21] classification task.
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [No]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]