

QwT-v2: Practical, Effective and Efficient Post-Training Quantization

Ningyuan Tang Minghao Fu Hao Yu Jianxin Wu*

National Key Laboratory for Novel Software Technology, Nanjing University, China
School of Artificial Intelligence, Nanjing University, China
tangny@lamda.nju.edu.cn, fumh@lamda.nju.edu.cn, wujx2001@nju.edu.cn

Abstract

Network quantization is arguably one of the most practical network compression approaches for reducing the enormous resource consumption of modern deep neural networks. They usually require diverse and subtle design choices for specific architecture and tasks. Instead, the QwT method is a simple and general approach which introduces lightweight additional structures to improve quantization. But QwT incurs extra parameters and latency. More importantly, QwT is not compatible with many hardware platforms. In this paper, we propose QwT-v2, which not only enjoys all advantages of but also resolves major defects of QwT. By adopting a very lightweight channel-wise affine compensation (CWAC) module, QwT-v2 introduces significantly less extra parameters and computations compared to QwT, and at the same time matches or even outperforms QwT in accuracy. The compensation module of QwT-v2 can be integrated into quantization inference engines with little effort, which not only effectively removes the extra costs but also makes it compatible with most existing hardware platforms.

1 Introduction

The rapid scaling of deep neural networks guarantees their strong performance in various fields, including language (Devlin et al. (2019); Brown et al. (2020)), vision (Dosovitskiy et al. (2021); He et al. (2017)), speech (Zhang et al. (2023)), and multimodal (Radford et al. (2021); Liu et al. (2023)), but also incurs heavy burden on hardware resources such as compute, storage and energy. Network quantization (Liang et al. (2021)) stands out as a highly practical approach to reduce these resource consumptions, highlighting large compression ratio, low accuracy loss, and high hardware friendliness. Existing quantization methods are often categorized into Quantization-Aware Training (QAT) which requires training, and Post-Training Quantization (PTQ) which is training free.

Neither PTQ nor QAT is perfect. PTQ (Li et al. (2021); Xiao et al. (2023)) methods are fast but may lead to considerable accuracy degradation when the bit-width is low, while QAT (Esser et al. (2020); Zhu et al. (2023); Li et al. (2024)) methods are thousands of times slower than PTQ methods. Moreover, existing quantization methods require meticulous design for different models and tasks. Changes in model architecture or task requirements can potentially render these quantization methods ineffective. For example, the issue of handling outliers is not important in quantizing CNNs, but is critical for quantizing Transformers.

Quantization without Tears (QwT) by Fu et al. (2025) is a simple and general quantization method which suits various network architectures and applications. By adding external compensation matrices to a quantized network S^Z (i.e., the output of a PTQ method), QwT can effectively improve network accuracy without severely affecting latency or model size. In addition, parameters of compensation

*Corresponding author.

modules in QwT can be set in closed-form with a small calibration set. This fact enables QwT to train very quickly without any hyperparameter tuning.

But, despite being simple, effective and widely applicable, a few drawbacks in QwT remain, which critically limit its deployment in many other real-world scenarios.

1. **Hardware compatibility.** QwT requires the inference engine to compute mainly in quantized fixed-point format (e.g., INT8), but there are also floating-point computations (e.g., FP16 or BF16) for the compensation modules. Although this capability is available in many GPUs, it is not yet supported well in many other hardware platforms (e.g., simple AI accelerators in a cellphone or an embedded hardware) or software inference engines like vLLM (Kwon et al. (2023)). This incompatibility severely restricts QwT’s usability.
2. **Extra parameters.** QwT compensates a PTQ model with linear projections. In high-bit quantization, compensation matrices only account for a small percentage of the network size. But in a low-bit quantized network, the floating-point compensation matrices become non-negligible and increase model size by around 30%, as shown by Fu et al. (2025).
3. **Lower efficiency.** For each compensated block, QwT receives the block input, calculates floating-point compensation, and adds it to the block output. During this process, the inference engine needs to store all these tensors, which increases both memory footage and latency (due to both more memory accesses and more computations).

In this paper, we show that all these issues can be resolved simultaneously. We propose QwT-v2, which inherits the advantages of QwT and also showcases the following improvements over it:

- **Minimal extra parameter & computation.** We introduce a new form of compensation module, named channel-wise affine compensation (CWAC), which requires significantly less parameters and computations than those in QwT.
- **Easy integration & no floating-point computation.** Furthermore, the CWAC module can be integrated into inference engines and then it incurs zero overhead during inference. The integration requires only few efforts—merely changing quantization parameters without changing the computation graph. After this integration, floating-point computations are *not* required, hence QwT-v2 *suits most hardware platforms*.
- **Fine-grained compensation.** Thanks to the parameter and computation efficiency of CWAC, we can compensate each quantized module individually, rather than compensating for an entire network block. This guarantees better result of QwT-v2 in many settings.

2 Related Works

As aforementioned, network quantization methods can be categorized into two lines: Quantization-Aware Training (QAT) and Post-Training Quantization (PTQ).

QAT methods (Liu et al. (2020); Lee et al. (2021); Esser et al. (2020); Bhalgat et al. (2020); Li et al. (2024)) simulate low-bit computation during training, and minimize accuracy degradation caused by quantization. LSQ (Esser et al. (2020)) updates not only parameters but also scale factors via gradient descent during training. LSQ+ (Bhalgat et al. (2020)) extends LSQ to non-uniform quantization. Q-ViT (Li et al. (2024)) improves quantization performance on Transformers by rectifying information distortion of quantized self-attention maps.

PTQ methods (Li et al. (2021); Yuan et al. (2022); Frantar et al. (2023)) convert a well-trained floating-point model to low-bit fixed-point one with a small calibration set. Several methods (Nagel et al. (2020); Li et al. (2021); Wei et al. (2022)) work well on CNNs but fail to be effective on Transformers, due to different network structure and weight/activation numerical distributions. Methods have also been proposed to quantize Transformers (Yuan et al. (2022); Frantar et al. (2023); Li et al. (2023); Lin et al. (2022); Xiao et al. (2023)). FQ-ViT (Lin et al. (2022)) uses power-of-2 factor and log-int-softmax quantization to tackle the difficulty of quantizing layer norm and softmax activations. RepQ-ViT (Li et al. (2023)) transforms complex quantizers to simple quantizers via scale reparameterization for efficient inference. GPTQ (Frantar et al. (2023)) exploits a single-shot weight quantization technique which utilizes approximate second-order information.

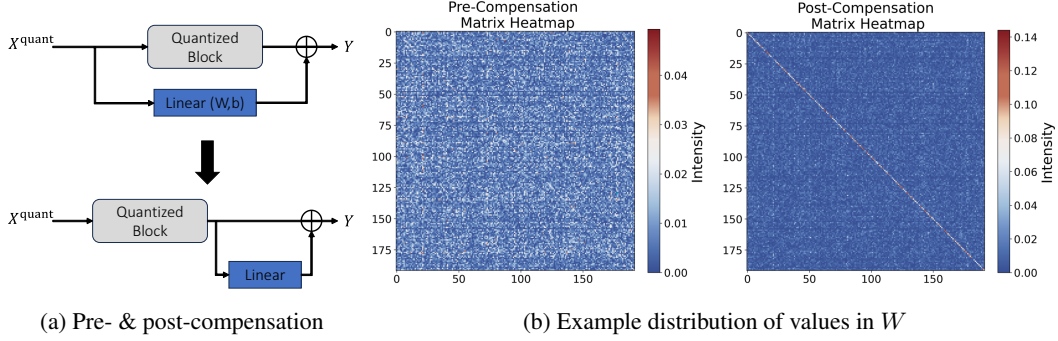


Figure 1: The left figure (a) shows pre-compensation in QwT (top) and our post-compensation in QwT-v2 (bottom), respectively. In pre-compensation, the linear compensation module receives a Transformer block’s input as its input, while in post-compensation, the block’s output is input to the linear compensation module. The right figure (b) shows example compensation matrix W ’s absolute values in QwT and our QwT-v2, respectively (on 4-bit quantized DeiT-tiny).

3 The QwT-v2 Method

QwT by Fu et al. (2025) proposes a simple and general quantization method to improve PTQ methods, and it is effective on both CNNs and Transformers. For example, given a Transformer block (i.e., MHSA+FFN) whose output in the floating-point and quantized networks are Y^{full} and Y^{quant} , respectively; while its input in the quantized version is X^{quant} . QwT noticed that quantization leads to the difference between Y^{full} and Y^{quant} , and reducing this difference helps the quantized model a lot. It linearly regresses $Y^{\text{full}} - Y^{\text{quant}}$ using X^{quant} , whose regression parameters (W, b) can be obtained in closed-form. Then, a compensation term $WX^{\text{quant}} + b$ is implemented as a fully connected (FC) layer and is added to X^{quant} , such that $Y^{\text{quant}} + WX^{\text{quant}} + b$ (compared to Y^{quant} alone) is closer to Y^{full} . The main idea of QwT is illustrated in the top half of Figure 1a.

3.1 Key Ideas in Our QwT-v2

QwT improves PTQ methods’ accuracy significantly. But, since $WX^{\text{quant}} + b$ is computed in floating-point format while other computations are in fixed-point (quantized) format, QwT is not compatible with some hardware platforms. Furthermore, the existence of W adds roughly 30% model size, together with computations and memory accesses to the quantized model. Our goal is to design a PTQ method that suits almost all hardware, is parameter efficient, and also memory efficient.

We start by asking whether QwT is the most suitable compensation method. QwT *linearly* compensates an *entire Transformer block*’s output with its input, but such a block involves high *non-linearity* which cannot be easily modeled by a linear projection. We should *try to minimize the non-linearity* by changing the regressor’s input form the Transformer block’s input to its output, as shown in Figure 1a. We call them pre-compensation (as in QwT) and post-compensation (in our QwT-v2), respectively. Note that in post-compensation, the compensation term $WX^{\text{quant}} + b$ becomes $WY^{\text{quant}} + b$.

This simple change *makes the compensation easier* since it bypasses the complex non-linearity from activation functions or attentions inside the Transformer block. More importantly, we *no longer need to store* the block input during the inference stage for compensation, which is inevitable in QwT.

Then we move on and focus on another difficulty of QwT: the compensation matrix W adds more parameters and latency, especially in low-bit quantization. More importantly, $WX^{\text{quant}} + b$ is computed in floating-point format, rendering QwT incompatible with many AI chips. We need a *lightweight and hardware-friendly* compensation method.

By visualizing the distribution of (absolute) values in W (with an example shown in Figure 1b), we find that the distribution is irregular in QwT’s pre-compensation, but in our post-compensation the weights are *concentrated on the diagonal*. Inspired by this observation, we hypothesize that *using diagonal instead of full matrices can effectively compensate the error as well*, and call this idea channel-wise affine compensation (CWAC). Compared to the compensation module in QwT, CWAC is significantly more efficient in terms of parameter, memory-access, and compute. Furthermore, we

will show that CWAC can be completely absorbed into existing modules by slightly modifying the inference engine (Section 3.4). This means that our method *inserts zero extra inference overhead and is compatible with existing AI chips*.

QwT asks $Y^{\text{quant}} + WX^{\text{quant}} + b$ to be close to Y^{full} . In QwT-v2 we ask $WY^{\text{quant}} + b$, not $Y^{\text{quant}} + WY^{\text{quant}} + b$, to be close to Y^{full} . This key change is because now our compensation is not only post-compensation but also diagonal, hence we do not need the residual connection any more. That is, in Figure 1a the $\oplus$ should be removed and the two branches should become only one.

3.2 Preliminaries

Before presenting details of our QwT-v2, we briefly outline basic concepts in quantization. Uniform quantization is probably the most popular method thanks to its simplicity and hardware-friendliness. Given a quantization bit-width b , uniform quantization turns floating-point tensor x into:

$$x^{\text{quant}} = \text{clip}\left(\left\lfloor \frac{x}{s} \right\rfloor + z, 0, 2^b - 1\right), \quad (1)$$

where x^{quant} is the quantized value of x , $\lfloor \cdot \rfloor$ is the rounding function, and $\text{clip}(x, u, v)$ clips x to the $[a, b]$ range. $s \in \mathbb{R}^+$ is the quantization scale and $z \in \mathbb{Z}$ is zero-point, which are determined by:

$$s = \frac{\max(x) - \min(x)}{2^b - 1}, \quad z = \text{clip}\left(\left\lfloor -\frac{\min(x)}{s} \right\rfloor, 0, 2^b - 1\right). \quad (2)$$

Another commonly used quantization method is log2 quantization, which performs logarithmic compression on the input floating-point tensor before rounding and clipping.

There are usually two ways of quantizing a tensor: per-tensor and per-channel. Per-tensor quantizes the whole tensor with only one s and one z . Per-channel quantization quantizes the i -th channel with s_i and z_i . Modern methods often apply per-channel quantization for weights and per-tensor quantization for activations, which achieves the balance between accuracy and efficiency.

3.3 Channel-Wise Affine Compensation

Given a floating-point model B and its quantized counterpart Q , we propose Channel-Wise Affine Compensation (CWAC) to recover accuracy degradation caused by quantization. For *each* quantized module in Q , we add a channel-wise affine transformation that aligns the quantized outputs with their floating-point counterparts.

Let $M_{\text{full}} : \mathbb{R}^{C_{\text{in}}} \rightarrow \mathbb{R}^{C_{\text{out}}}$ denotes one module in the original network, and $M_{\text{quant}} : \mathbb{R}^{C_{\text{in}}} \rightarrow \mathbb{R}^{C_{\text{out}}}$ its quantized version. Given calibration input $X_{\text{in}} \in \mathbb{R}^{N \times C_{\text{in}}}$ in which N denotes the number of calibration samples, the output would be:

- Floating-point output: $Y^{\text{full}} = M_{\text{full}}(X_{\text{in}})$,
- Quantized output: $Y^{\text{quant}} = M_{\text{quant}}(X_{\text{in}})$.

For each *single* output channel $c \in \{1, \dots, C_{\text{out}}\}$, we compute channel-wise compensation parameters by solving the *one-dimensional* linear regression problem:

$$\min_{\alpha_c, \beta_c} \|Y_{:,c}^{\text{full}} - (\alpha_c \odot Y_{:,c}^{\text{quant}} + \beta_c)\|^2, \quad (3)$$

where $\odot$ denotes element-wise multiplication. This leads to a very simple and easy-to-calculate closed-form solution:

$$\alpha_c = \frac{\text{Cov}(Y_{:,c}^{\text{full}}, Y_{:,c}^{\text{quant}})}{\text{Var}(Y_{:,c}^{\text{quant}})}, \quad (4)$$

$$\beta_c = \mathbb{E}[Y_{:,c}^{\text{full}}] - \alpha_c \mathbb{E}[Y_{:,c}^{\text{quant}}], \quad (5)$$

in which Cov and Var are empirical covariance and variance, respectively. We get $\alpha \in \mathbb{R}^{C_{\text{out}}}, \beta \in \mathbb{R}^{C_{\text{out}}}$ by concatenating all α_c and β_c . The compensated output is then $\alpha \odot Y^{\text{quant}} + \beta$.

In Figure 2, we illustrate our QwT-v2 compensation in a Transformer block. We add our CWAC (channel-wise affine compensation) modules after every quantized linear layer in Vision Transformer, and every quantized convolutional layer in CNN.

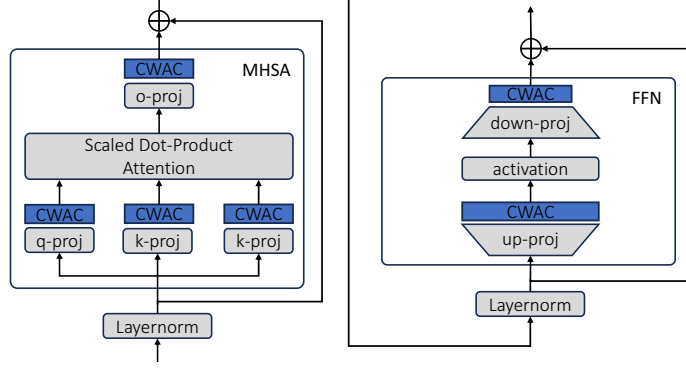


Figure 2: The architecture of QwT-v2’s compensation in a vision Transformer. Compensations are applied after each fully connected layer.

3.4 Integrating into Inference Engine

Although the compensation module in this form is already highly efficient, we will demonstrate that QwT-v2 can further eliminate the computational overhead by integrating parameters and computations into the quantize inference engine.

Here we discuss a potential integration framework following the representation in Jacob et al. (2018). Given a typical quantized linear layer (W_q, b_q) and input x_q , where weights adopt per-channel quantization, and activations adopt per-tensor quantization, we have (S_x, Z_x) , (S_W, Z_W) , (S_r, Z_r) for activations, weights and output, respectively (S for quantization scale and Z for zero-point). Note that S_W, Z_W are d -dimensional vectors (d denotes feature dimension), S_x, S_r are floating-point numbers, Z_x, Z_r are 32-bit integers. Now without considering bias b (which is handled separately in an inference engine), we have $r = Wx$. In the quantized version, it is

$$S_r(r_q - Z_r) = S_W(W_q - Z_W)S_x(x_q - Z_x), \quad (6)$$

which can be rewritten as

$$r_q = Z_r + M(W_q x_q - Z_W \odot x_q - Z_x W_q + Z_x Z_W), \quad (7)$$

where the *multiplier* M is defined as

$$M = \frac{S_x S_W}{S_r} \in \mathbb{R}^d. \quad (8)$$

Now, if we put our compensation module into consideration, we can get:

$$S_r(r_q - Z_r) = \alpha S_W(W_q - Z_W)S_x(x_q - Z_x) + \beta, \quad (9)$$

which can be rewritten as

$$r_q = Z_r + M'(W_q x_q - Z_W \odot x_q - Z_x W_q + Z_x Z_W + \lfloor \frac{\beta}{\alpha S_x S_W} \rfloor), \quad (10)$$

$$M' = \frac{\alpha S_x S_W}{S_r} \in \mathbb{R}^d. \quad (11)$$

This integration procedure can be easily extended to other quantized layers (e.g., convolutional layer) and other hardware friendly quantization methods (e.g., log2 quantization). Notice that the calculation of Equation 7 is already efficiently implemented in inference engines (e.g., by Google Inc. (2024)), and Equation 10 is also fully compatible with inference engine, with only numeric adjustments of M and the offset. During this process, integration of α is lossless, and integration of β is not lossless due to rounding errors. But this information loss is negligible since the magnitude of $\frac{\beta}{\alpha S_x S_W}$ is usually large, which means the effect of rounding error is minimal (Table 8).

4 Experiments

In this section, we evaluate our QwT-v2 on tasks in various domains, including visual classification, detection, generation, language and multimodal following the experimental setting of QwT (Fu et al. (2025)). We compare QwT-v2 with QwT and other quantization methods, plus ablation studies.

Table 1: Results (Top-1 accuracy) on ImageNet recognition. “#-bit” indicates bit-width for both weight and activation. “Size” (MB) denotes model storage cost. “Delta” (MB) denotes extra storage cost from compensation modules.

Network	Method	4-bit			6-bit		
		Size	Delta	Top-1	Size	Delta	Top-1
Swin-T	Floating-point (32-bit)	113.2		81.4			
	RepQ-ViT (Li et al. (2023))	14.9		73.0	21.7		80.6
	RepQ-ViT + QwT (Fu et al. (2025))	19.2	4.3	75.5	26.0	4.3	80.7
	RepQ-ViT + QwT-v2	15.2	0.3	77.1	22.0	0.3	80.9
ViT-B	Floating-point (32-bit)	346.3		84.5			
	RepQ-ViT (Li et al. (2023))	44.9		68.5	66.2		83.6
	RepQ-ViT + QwT	59.1	14.2	76.3	80.4	14.2	83.9
	RepQ-ViT + QwT-v2	45.6	0.7	75.6	66.9	0.7	83.8
ResNet-50	Floating-point (32-bit)	102.2		76.6			
	Percentile (Li et al. (2019))	14.0		68.4	19.9		76.0
	Percentile + QwT	16.0	2.0	74.5	21.9	2.0	76.8
	Percentile + QwT-v2	14.2	0.2	75.1	20.1	0.2	76.7

4.1 Experiments on Image classification

We evaluated QwT-v2 on image classification on ImageNet (Deng et al. (2009)), leveraging various backbone architectures including ViT (Dosovitskiy et al. (2021)), Swin Transformer (Liu et al. (2021)) and ResNet (He et al. (2016)). Following the setting in QwT, we randomly sampled a calibration set with 512 samples from the ImageNet training set. We also performed β rounding in the experiments. (if not specially noted, this setting was adopted in all following experiments).

We tested performance of QwT-v2 on various backbone, and the results are shown in Table 1. On Swin and ViT, we choose RepQ-ViT (Li et al. (2023)) as the baseline method, and Percentile (Li et al. (2019)) for ResNet. Compared to the backbone methods, QwT-v2 shows considerable improvements. We can find that the improvement on 4-bit quantized model (4-8%) is greater than that on 6-bit.

On Swin-T and ResNet, QwT-v2 outperforms QwT, but is slightly lower than QwT on ViT. In terms of extra parameters, QwT-v2 only accounts for about 1%-2% extra parameters compared to QwT’s 30%, which is a significant advantage for QwT-v2 in real-world deployment. Also note these extra parameters and computations can be merged (removed) if we integrate them into the inference engine.

4.2 Experiments on Object Detection

We evaluated our method on object detection and instance segmentation tasks using the COCO 2017 (Lin et al. (2014)) dataset. Following the setting in Fu et al. (2025), we adopted Swin-S with Mask R-CNN (He et al. (2017)), and Swin-S/B with Cascade Mask R-CNN (Cai and Vasconcelos (2018)) as detectors.

Results shown in Table 2 demonstrate QwT-v2’s exceptional ability on detection model’s compensation. QwT-v2 outperforms baseline methods and QwT on all settings. Also, the number of extra parameters and computations required by QwT-v2 is almost negligible compared to those in QwT.

4.3 Experiments on Multimodal Recognition

We conducted experiments on CLIP (Radford et al. (2021)). We test our QwT-v2 on CLIP to verify its ability on multimodal tasks. Following QwT (Fu et al. (2025)), we chose ViT-B/32 as the visual encoder and a 12-block Transformer as the text encoder. We randomly selected 512 image-text pairs from the training data for baseline calibration and QwT-v2 initialization.

Results in Table 3 show that QwT-v2 is also very effective on multimodal model’s compensation. QwT-v2 improves top-1 accuracy over all baselines on all settings. And we can find out that the compensation improvement of compensating both vision and text encoder is greater than compensating vision encoder only. On (vision only) quant setup, result of QwT-v2 is slightly behind QwT on 6/6-bit

Table 2: Detection results on COCO. #Bits indicates the bit-width of weight/activation. We use box average precision (AP^{box}) and mask average precision (AP^{mask}) to assess object detection and instance segmentation accuracy, respectively.

Network	Method	#Bits	Size	AP^{box}	AP^{mask}
Swin-S + Mask R-CNN	Floating-point	32/32	276.5	48.5	43.3
	RepQ-ViT	4/4	36.1	42.6	40.0
	RepQ-ViT + QwT	4/4	44.0	43.1	40.4
	RepQ-ViT + QwT-v2	4/4	36.7	43.4	40.5
	RepQ-ViT	6/6	53.3	47.6	42.9
	RepQ-ViT + QwT	6/6	61.2	48.0	43.1
	RepQ-ViT + QwT-v2	6/6	53.9	48.0	43.1
Swin-S + Cascade Mask R-CNN	Floating-point	32/32	427.8	51.9	45.0
	RepQ-ViT	4/4	56.9	49.3	43.1
	RepQ-ViT + QwT	4/4	64.8	49.9	43.4
	RepQ-ViT + QwT-v2	4/4	57.5	50.0	43.6
	RepQ-ViT	6/6	83.4	51.4	44.6
	RepQ-ViT + QwT	6/6	91.3	51.7	44.8
	RepQ-ViT + QwT-v2	6/6	84.0	52.0	45.0
Swin-B + Cascade Mask R-CNN	Floating-point	32/32	579.9	51.9	45.0
	RepQ-ViT	4/4	76.1	49.3	43.1
	RepQ-ViT + QwT	4/4	90.1	50.0	43.7
	RepQ-ViT + QwT-v2	4/4	77.0	50.3	43.8
	RepQ-ViT	6/6	112.1	51.5	44.8
	RepQ-ViT + QwT	6/6	126.1	51.8	45.0
	RepQ-ViT + QwT-v2	6/6	113.0	51.9	45.1

quantization but slightly better than QwT on 8/8-bit quantization. On (vision & text) setting, QwT-v2 outperforms QwT by a large margin on both bit-widths (1.8% on 6/6-bit and 1.9% on 8/8-bit) with significantly less extra parameters.

4.4 Experiments on Image Generation

We also conducted experiments on quantized diffusion model, to evaluate the generated image quality from full precision DiT-XL/2 model, quantized model, and compensated model, under 256x256 resolution. We utilized the DDIM sampler with 50 sampling steps and applied classifier-free guidance (cfg) of 1.5 to realize between efficiency and precision. We also followed the key assumption from Fu et al. (2025): quantization error mainly depend on input token x , and is minimally affected by timestamp t . We find this assumption practical and chose $t = 0$ for compensation initialization.

We built our QwT-v2 on Q-DiT quantized model. As shown in Table 4, QwT-v2 shows competitive results compared to other PTQ methods and QwT. On W8A8 quantization, FID of QwT is 0.03 higher than QwT, but shows better inception score compared with QwT. On W4A8 quantization, QwT-v2 demonstrates both the best FID score and inception score, with 0.34 advantage on FID and 18.4 advantage on IS compared to second best results from QwT.

4.5 Experiments on Large Language Models

We evaluated our framework on LLaMA3-8B (Dubey et al. (2024)). For PTQ methods, we adopted GPTQ (Frantar et al. (2023)) with INT4 weight quantization only. We conducted a group-wise asymmetric quantization with a group size of 128 and applied activation reordering. In particular, GPTQ take 128 samples from the C4 dataset as calibration sets, and each sample is 2048 tokens long. Like QwT, we use exactly the same calibration set for QwT-v2.

Following the settings of GPTQ, we evaluated the perplexity on the WikiText2 (Stephen et al. (2017)) and C4 (Raffel et al. (2020)) datasets. We further assessed the zero-shot commonsense question answering (QA) ability on eight tasks which are evaluated by QwT (Fu et al. (2025)), including SIQA (Sap et al. (2019)), HellaSwag (Zellers et al. (2019)), PIQA (Bisk et al. (2020)), WinoGrande (Sakaguchi et al. (2021)), ARC (Clark et al. (2018)), BoolQ (Clark et al. (2019)), and OpenBookQA (Mihaylov et al. (2018)).

Table 3: Quantization results of CLIP for zero-shot classification tasks on ImageNet. Note that the ‘Setup’ column differentiates between two strategies: quantizing only the Visual Encoder, and quantizing both the Visual and Text Encoders concurrently.

Setup	Method	#Bits	Size (MB)	Top-1
Vision	Floating-point	32/32	607.2	63.4
	RepQ-ViT	6/6	323.5	59.2
	RepQ-ViT + QwT	6/6	336.8	60.3
	RepQ-ViT + QwT-v2	6/6	324.2	59.8
	RepQ-ViT	8/8	345.3	62.9
	RepQ-ViT + QwT	8/8	359.5	63.0
	RepQ-ViT + QwT-v2	8/8	346.0	63.1
Vision & Text	Floating-point	32/32	607.2	63.4
	RepQ-ViT	6/6	200.8	29.8
	RepQ-ViT + QwT	6/6	221.3	43.5
	RepQ-ViT + QwT-v2	6/6	202.0	45.3
	RepQ-ViT	8/8	232.1	38.7
	RepQ-ViT + QwT	8/8	252.6	54.6
	RepQ-ViT + QwT-v2	8/8	233.3	56.5

Table 4: Results of quantized DiT-XL/2 model. #Bits indicates the bit-width of weight/activation. We use FID and Inception Score (IS) to measure generated image quality. ↓ (↑) means smaller (larger) is better.

Method	#Bits	Size (MB)	FID (↓)	IS (↑)
Floating-point	16/16	1349	5.32	236.17
RepQ-ViT	8/8	677	5.46	234.74
GPTQ	8/8	690	5.90	218.90
Q-DiT	8/8	683	5.45	236.52
Q-DiT + QwT	8/8	707	5.35	236.91
Q-DiT + QwT-v2	8/8	685	5.38	244.34
RepQ-ViT	4/8	339	319.68	2.20
GPTQ	4/8	351	9.94	166.35
Q-DiT	4/8	347	6.75	208.38
Q-DiT + QwT	4/8	361	6.06	215.70
Q-DiT + QwT-v2	4/8	349	5.72	234.06

Table 5: Quantization results among WikiText2 (W2), C4 and eight zero-shot commonsense QA datasets using LLaMA3-8B as the backbone. ↓ (↑) means smaller (larger) is better.

Method	#Bits	Size (GB)	W2 (↓)	C4 (↓)	QA. Avg (↑)
Floating-point	16	16.06	6.24	8.96	66.10
GPTQ	4	5.73	6.65	9.44	64.90
GPTQ + QwT	4	6.80	6.63	9.38	65.18
GPTQ + QwT-v2	4	5.74	6.62	9.40	65.21

Table 5 summarizes the perplexity in WikiText2 (W2), C4 and the average accuracy in eight common sense reasoning datasets. Our QwT-v2 achieves consistent gains over baseline the method GPTQ, and demonstrates close performance compared to QwT.

4.6 Ablation studies

we conducted ablation studies on QwT-v2 to verify the effectiveness of its design choices.

Ablation on varying calibration set size. We changed the size of calibration set in QwT-v2, and report the result on ViT-B and Swin-T quantized by RepQ-ViT. As shown in Table 6, by increasing calibration set size, the result can get slightly better. On ViT-B, the results are more significantly affected (0.5% gap between 32 images and 1024 images), while on Swin-T, the results are less affected (0.2% gap between 32 images and 1024 images). Choosing a calibration set with 512 images can achieve the balance between efficiency and effectiveness.

Ablation on compensation position. To assess the effectiveness of fine-grained compensation (compensating behind each quantized linear/conv layer). We conduct ablation experiment on only compensating the final fc/conv in each block, results are show in Table 7 (ViT-B and Swin-T are quantized by RepQ-ViT, Resnet-50 by Percentile). From the table, we can summarize that our fine-grained compensation (‘all’) consistently provide a gain compared to only compensating final fc/conv (‘post’). And the gain in 4-bit quantization is always greater than 6-bit quantization. The results clearly demonstrated the effectiveness of fine-grained compensation in QwT-v2.

Ablation on rounding β . We further investigated whether rounding β to integers will cause performance drop, and results are shown in Table 8. We can see that there is nearly no gap between rounded β ’s results and its floating-point counterparts.

Table 6: Ablation on varying the size of calibration set in QwT-v2 on 4/4-bit quantized ViT-B and Swin-T model. ‘Top-1’ means Top-1 accuracy on ImageNet validation set.

Network	Calibration Size	Top-1
ViT-B (4/4-bit)	32 images	75.2
	128 images	75.3
	512 images	75.6
	1024 images	75.7
Swin-T (4/4-bit)	32 images	69.9
	128 images	77.0
	512 images	77.1
	1024 images	77.1

Table 7: Ablation on changing the position of compensation modules. ‘all’ denotes compensating behind all quant linear/conv layers. ‘post’ denotes only compensate behind final fc/conv.

Network	#Bits	Position	Top-1
ViT-B	4/4	post	73.5
	4/4	all	75.6
	6/6	post	83.5
	6/6	all	83.8
Swin-T	4/4	post	73.9
	4/4	all	77.1
	6/6	post	80.6
	6/6	all	80.9
Resnet-50	4/4	post	73.1
	4/4	all	75.1
	6/6	post	76.5
	6/6	all	76.7

Table 8: Ablation on the effect of rounding β as described in Sec. 3.4. The $\checkmark$ on the ‘Round’ column denotes the result of rounding β , which is by default adopted in all the experiments.

Network	#Bits	Round	Top-1
ViT-B	4/4		75.6
	4/4	$\checkmark$	75.6
	6/6		83.8
	6/6	$\checkmark$	83.8
Swin-T	4/4		77.1
	4/4	$\checkmark$	77.1
	6/6		90.0
	6/6	$\checkmark$	80.9
ViT-B	4/4		75.3
	4/4	$\checkmark$	75.1
	6/6		76.7
	6/6	$\checkmark$	76.7

5 Conclusions

In this paper, we proposed QwT-v2, which addresses major limitations issues in QwT, including: low hardware compatibility, massive additional parameters, and increased inference latency. These are realized by two key insights based on QwT. First, we change the pre-compensation in QwT to post-compensation, which lowers inference overhead and frees us from learning the complex non-linearity within a Transformer block. Consequently, we can move on to further replace the linear compensation matrices with diagonal ones, which saves parameters and computations to a large extent. More importantly, this compensation module can be integrated into inference engine with minimal effort, and completely eliminate extra parameters and computations. The proposed QwT-v2 has exhibited both low parameter and computation costs and high accuracy compared to QwT in various application: recognition, detection, multimodal, LLM, and image generation.

6 Limitations and Future Work

There are some intriguing drawbacks which are not fully discussed and utilized in QwT-v2. We find that compensation from earlier layers are more prominent and more effective than compensation from deeper layers, and compensation for different modules have different effectiveness. It is worth studying how to make those less effective compensation positions work better.

Another limitation of QwT-v2 is that further finetuning the compensation parameters by 1 epoch is not as effective as that operation in QwT. This is not surprising, as the extra parameter size in QwT-v2 is only around 5-10% of that in QwT. In future work, we are interesting in finding more innovative ways to improve accuracy after finetuning while still maintaining parameter and computation efficiency.

Finally, QwT-v2 and QwT are both built upon an existing PTQ method, which means we need to develop better PTQ methods in areas where they work not so well, e.g., when quantizing both activations and weights in an LLM.

References

- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, pages 4171–4186, 2019.
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS)*, 2020.
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, pages 1–21, 2021.
- Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision*, pages 2980–2988, 2017.
- Dong Zhang, Shimin Li, Xin Zhang, Jun Zhan, Pengyu Wang, Yaqian Zhou, and Xipeng Qiu. SpeechGPT: Empowering large language models with intrinsic cross-modal conversational abilities". In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 15757–15773, 2023.
- Alec Radford, Jong Wook Kim, Chris Hallacy, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763, 2021.
- Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS)*, 2023.
- Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461(C):370–403, 2021.
- Yuhang Li, Ruihao Gong, Xu Tan, et al. BRECK: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations*, pages 1–12, 2021.
- Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.
- Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. In *International Conference on Learning Representations*, pages 1–10, 2020.
- Ke Zhu, Yin-Yin He, and Jianxin Wu. Quantized feature distillation for network quantization. In *AAAI Conference on Artificial Intelligence*, pages 11452–11460, 2023.
- Yanjing Li, Sheng Xu, Baochang Zhang, Xianbin Cao, Peng Gao, and Guodong Guo. Q-ViT: Accurate and fully quantized low-bit Vision Transformer. In *Advances in Neural Information Processing Systems*, pages 34451 – 34463, 2024.
- Minghao Fu, Hao Yu, Jie Shao, Junjie Zhou, Ke Zhu, and Jianxin Wu. Quantization without tears. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.
- Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. ReActNet: Towards precise binary neural network with generalized activation functions. In *European Conference on Computer Vision*, volume 12359 of *LNIP*, pages 143–159, 2020.

- Junghyup Lee, Dohyung Kim, and Bumsub Ham. Network quantization with element-wise gradient scaling. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6444–6453, 2021.
- Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 696–697, June 2020.
- Zhihang Yuan, Chenhao Xue, Yiqi Chen, Qiang Wu, and Guangyu Sun. PTQ4ViT: Post-training quantization for Vision Transformers with twin uniform quantization. In *European Conference on Computer Vision*, volume 13672 of *LNCS*, pages 191–207, 2022.
- Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained Transformers. In *International Conference on Learning Representations*, pages 1–12, 2023.
- Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206, 2020.
- Xiuying Wei, Ruihao Gong, Yuhang Li, Xianglong Liu, and Fengwei Yu. QDROP: Randomly dropping quantization for extremely low-bit post-training quantization. In *International Conference on Learning Representations*, pages 1–12, 2022.
- Zhikai Li, Junrui Xiao, Lianwei Yang, and Qingyi Gu. RepQ-ViT: Scale reparameterization for post-training quantization of Vision Transformers. In *IEEE/CVF International Conference on Computer Vision*, pages 17181–17190, 2023.
- Yang Lin, Tianyu Zhang, Peiqin Sun, Zheng Li, and Shuchang Zhou. FQ-ViT: Post-training quantization for fully quantized Vision Transformer. In *International Joint Conference on Artificial Intelligence*, pages 1173–1179, 2022.
- 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. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- Google Inc. TensorFlow Lite, 2024. URL <https://www.tensorflow.org/lite>. Version 2.16.1.
- Rundong Li, Yan Wang, Feng Liang, Hongwei Qin, Junjie Yan, and Rui Fan. Fully quantized network for object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2805–2814, 2019.
- Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.
- Ze Liu, Yutong Lin, Yue Cao, et al. Swin Transformer: Hierarchical Vision Transformer using shifted windows. In *IEEE/CVF International Conference on Computer Vision*, pages 9992–10002, 2021.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- Tsung-Yi Lin, Michael Maire, Serge Belongie, et al. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*, volume 8693 of *LNCS*, pages 740–755, 2014.
- Zhaowei Cai and Nuno Vasconcelos. Cascade R-CNN: Delving into high quality object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6154–6162, 2018.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, and others. The llama 3 herd of models. *arXiv:2407.21783*, 2024.

- Merity Stephen, Xiong Caiming, Bradbury James, et al. Pointer sentinel mixture models. In *International Conference on Learning Representations*, pages 1–11, 2017.
- Colin Raffel, Noam Shazeer, Adam Roberts, et al. Exploring the limits of transfer learning with a unified text-to-text Transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.
- Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In *Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, pages 4463–4473, 2019.
- Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Annual Meeting of the Association for Computational Linguistics*, page 4791–4800, 2019.
- Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, pages 7432–7439, 2020.
- Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: an adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457*, 2018.
- Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 2924–2936, 2019.
- Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, 2018.
- Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, pages 1–18, 2019.
- Lei Chen, Yuan Meng, Chen Tang, et al. Q-DiT: Accurate post-training quantization for diffusion Transformers. *arXiv:2406.17343*, 2024.

A Implementation Details

In this section, we present full implementation details of the different types of tasks in our experiments.

Image Classification. Following QwT, We selected RepQ-ViT Li et al. (2023) for ViT quantization, and Percentile Li et al. (2019) for ResNet, as the primary baseline PTQ methods to integrate with QwT-v2 method. Following Li et al. (2023), we randomly sampled 32 images from the ImageNet Deng et al. (2009) dataset as the calibration set to initialize the quantized weights for these baseline methods. Additionally, a separate set of 512 randomly selected images from the ImageNet training set was used to initialize the parameters of the QwT modules (excluding PTQ weights). For all networks, the W matrices are in FP32 format (unlike QwT, which set compensation matrices to FP16 to reduce model size, the size of QwT-v2 compensation matrices are extremely small, so FP32 works well here), and b is rounded to INT32 format by default.

All the experiments above on image classification are implemented on a single RTX 3090 GPU with 24GB memory.

Additionally, When fine-tuning the QwT modules along with the classification head for an additional epoch, following Fu et al. (2025), we utilized AdamW Loshchilov and Hutter (2019) as the optimizer. The batch size was set to 32 per GPU (using a total of 4 GPUs), and weight decay was set to 0. The learning rate was configured to $1e-7$ for ViT Dosovitskiy et al. (2021), $5e-6$ for Swin Liu et al. (2021), $1e-5$ for ResNet He et al. (2016).

Object Detection & Instance Segmentation. Following Li et al. (2023), we randomly sampled a single image from the COCO dataset Lin et al. (2014) to initialize the quantized weights for baseline PTQ methods. All other details are consistent with the image classification case. Experiments shown in 2 are also implemented on a single RTX 3090 GPU with 24GB memory.

Image Generation. Consistent with the experimental setup of Q-DiT Chen et al. (2024), we selected the DiT architecture and employed pretrained DiT-XL/2 models at a resolution of 256×256 . Experiments are implemented on 4 RTX 3090 GPUs.

Large Language Models. Experiments on large language models follow the baseline of GPTQ Frantar et al. (2023), and we choose Llama3-8B as our backbone. the computation. the initialization of QwT-v2 is done on 4 RTX 3090 GPUs.

All the experiments are implemented with pyTorch.

B More Results on Image Recognition

We further show results on more backbone in Table 9. We extend the results to DeiT-Tiny, DeiT-Small, Swin-Tiny, Swin-Small, ViT-B, ResNet-50 and ResNet-101. Results shows that QwT-v2 works well on all kinds of backbone, especially on 4-bit quantized backbones.

Table 9: Results (Top-1 accuracy) on ImageNet recognition with more backbones. “#-bit” indicates bit-width for both weight and activation. “Size” (MB) denotes model storage cost. “Delta” (MB) denotes extra storage cost from compensation modules.

Network	Method	4-bit			6-bit		
		Size	Delta	Top-1	Size	Delta	Top-1
DeiT-T	Floating-point (32-bit)	22.9	—	72.2	—	—	—
	RepQ-ViT (Li et al. (2023))	3.3	—	58.2	4.6	—	71.0
	RepQ-ViT + QwT (Fu et al. (2025))	4.2	0.9	61.4	5.5	0.9	71.2
	RepQ-ViT + QwT-v2	3.4	0.1	59.9	4.7	0.1	71.2
DeiT-S	Floating-point (32-bit)	88.2	—	79.9	—	—	—
	RepQ-ViT	11.9	—	69.0	17.2	—	78.9
	RepQ-ViT + QwT	15.4	3.5	71.5	20.7	3.5	79.1
	RepQ-ViT + QwT-v2	12.2	0.3	71.1	17.5	0.3	78.9
Swin-T	Floating-point (32-bit)	113.2	—	81.4	—	—	—
	RepQ-ViT	14.9	—	73.0	21.7	—	80.6
	RepQ-ViT + QwT	19.2	4.3	75.5	26.0	4.3	80.7
	RepQ-ViT + QwT-v2	15.2	0.3	77.1	22.0	0.3	80.9
Swin-S	Floating-point (32-bit)	198.4	—	83.2	—	—	—
	RepQ-ViT	25.8	—	80.2	38.0	—	82.8
	RepQ-ViT + QwT	33.7	7.9	80.4	45.9	7.9	82.9
	RepQ-ViT + QwT-v2	26.4	0.6	80.3	38.6	0.6	83.0
ViT-B	Floating-point (32-bit)	346.3	—	84.5	—	—	—
	RepQ-ViT	44.9	—	68.5	66.2	—	83.6
	RepQ-ViT + QwT	59.1	14.2	76.3	80.4	14.2	83.9
	RepQ-ViT + QwT-v2	45.6	0.7	75.6	66.9	0.7	83.8
ResNet-50	Floating-point (32-bit)	102.2	—	76.6	—	—	—
	Percentile (Li et al. (2019))	14.0	—	68.4	19.9	—	76.0
	Percentile + QwT	16.0	2.0	74.5	21.9	2.0	76.8
	Percentile + QwT-v2	14.2	0.2	75.1	20.1	0.2	76.7
ResNet-101	Floating-point (32-bit)	178.2	—	77.3	—	—	—
	Percentile	23.7	—	74.7	34.3	—	77.1
	Percentile + QwT	28.0	4.3	76.4	38.6	4.3	77.2
	Percentile + QwT-v2	24.1	0.4	76.2	34.7	0.4	77.2

We also show results of QwT-v2 on ImageNet classification when the compensation modules and classification head are fine-tuned on ImageNet training set for one epoch. As shown in Table 10, the compensation accuracy of QwT-v2 were further improved when fine-tuned. But the margin of improvement is not as large as the improvement in QwT. This is not surprising, because the extra parameter account in QwT-v2 is only about 5-10% to that in QwT. By the way, people may prefer not to fine-tune the compensation in most situations, because initial compensation of QwT-v2 only takes about 2-3 minutes, but fine-tuning on ImageNet requires several hours and significant more computation resources.

Table 10: Results with one-epoch fine-tuning on ImageNet recognition. “#-bit” indicates bit-width for both weight and activation. “Size” (MB) denotes model storage cost. “Delta” (MB) denotes extra storage cost from compensation modules. “*” denotes compensation modules and classification head are fine-tuned for one epoch on ImageNet training set.

Network	Method	4-bit			6-bit		
		Size	Delta	Top-1	Size	Delta	Top-1
Swin-T	Floating-point (32-bit)	113.2		81.4			
	RepQ-ViT	14.9		73.0	21.7		80.6
	RepQ-ViT + QwT	19.2	4.3	75.5	26.0	4.3	80.7
	RepQ-ViT + QwT*	19.2	4.3	79.3	26.0	4.3	80.9
	RepQ-ViT + QwT-v2	15.2	0.3	77.1	22.0	0.3	80.9
	RepQ-ViT + QwT-v2*	15.2	0.3	78.1	22.0	0.3	80.9
ViT-B	Floating-point (32-bit)	346.3		84.5			
	RepQ-ViT	44.9		68.5	66.2		83.6
	RepQ-ViT + QwT	59.1	14.2	76.3	80.4	14.2	83.9
	RepQ-ViT + QwT*	59.1	14.2	78.5	80.4	14.2	84.0
	RepQ-ViT + QwT-v2	45.6	0.7	75.6	66.9	0.7	83.8
	RepQ-ViT + QwT-v2*	45.6	0.7	77.5	66.9	0.7	83.8

We also show results for QwT-v2 with PTQ4ViT Yuan et al. (2022) as baseline method. As shown in Table 11, QwT-v2 improves the accuracy compared to baseline by 25.6% on 4-bit quantization and 0.6% on 6-bit quantization.

Table 11: Results with PTQ4ViT Yuan et al. (2022) as baseline method. “#-bit” indicates bit-width for both weight and activation. “Size” (MB) denotes model storage cost. “Delta” (MB) denotes extra storage cost from compensation modules.

Network	Method	4-bit			6-bit		
		Size	Delta	Top-1	Size	Delta	Top-1
ViT-B	Floating-point (32-bit)	346.3		84.5			
	PTQ4ViT-ViT (Yuan et al. (2022))	44.9		30.7	66.2		81.7
	PTQ4ViT-ViT+QwT-v2	45.6	0.7	56.3	66.9	0.7	82.3