IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 31 March 2025, accepted 21 April 2025, date of publication 2 May 2025, date of current version 12 May 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3566307

RESEARCH ARTICLE

WAPS-Quant: Low-Bit Post-Training Quantization Using Weight-Activation Product Scaling

GEUNJAE CHOI[1,2], **KAMIN LEE**[1,2], **AND NOJUN KWAK**[1], **(Senior Member, IEEE)**
[1]Graduate School of Convergence Science and Technology, Seoul National University, Gwanak-gu, Seoul 08826, Republic of Korea
[2]LG Electronics, Seocho-gu, Seoul 06772, Republic of Korea

Corresponding author: Nojun Kwak (nojunk@snu.ac.kr)

This work was supported in part by Institute of Information and communications Technology Planning and Evaluation (IITP) funded by Ministry of Science and Information and Communication Technology (MSIT) of Korean Government under Grant RS-2021-II211343, and in part by NRF funded by MSIT of Korean Government under Grant 2021R1A2C3006659.

ABSTRACT Post-Training Quantization (PTQ) has been effectively compressing neural networks into very few bits using a limited calibration dataset. Various quantization methods utilizing second-order error have been proposed and demonstrated good performance. However, at extremely low bits, the increase in quantization error is significant, hindering optimal performance. Previous second-order error-based PTQ methods relied solely on quantization scale values and weight rounding for quantization. We introduce a weight-activation product scaling method that, when used alongside weight rounding and scale value adjustments, effectively reduces quantization error even at very low bits. The proposed method compensates for the errors resulting from quantization, thereby achieving results closer to the original model. Additionally, the method effectively reduces the potential increase in computational and memory complexity through channel-wise grouping, shifting, and channel mixing techniques. Our method is validated on various CNNs, and extended to ViT and object detection models, showing strong generalization across architectures. We conducted tests on various CNN-based models to affirm the superiority of our proposed quantization scheme. Our proposed approach enhances accuracy in 2/4-bit quantization with less than 1.5% computational overhead, and hardware-level simulation confirms its suitability for real-time deplo1yment with negligible latency increase. Furthermore, hardware-level simulation on a silicon-proven ASIC NPU confirms that our method achieves higher accuracy with negligible latency overhead, making it practical for real-time edge deployment.

INDEX TERMS Post-training quantization (PTQ), low-bit quantization, weight-activation product scaling, channel-wise grouping, ASIC, channel-wise grouping, CNN.

I. INTRODUCTION

Quantization is a model compression technique, alongside methods like pruning and knowledge distillation [1], [2], [3], [4], that significantly reduces memory demands and computational costs in deep learning models [5], [6]. This is achieved by lowering the precision of the model's weights and activations. Uniform quantization is particularly recognized for its compatibility with hardware [7], [8], its seamless integration with batch normalization, and its ability to minimize expensive floating-point operations by employing varied quantization scales for each output channel.

From a training perspective, there are two primary approaches to quantization in deep learning: Quantization Aware Training (QAT) [9], [10], [11], [12] and Post-Training Quantization (PTQ) [13], [14], [15], [16], [17]. QAT trains a model with quantized weights and activations, which allows the model to adapt to the quantization process during training. In contrast, PTQ applies quantization to a pre-trained model, resulting in faster model compression but potentially causing a more significant loss in performance.

PTQ is particularly advantageous for compressing large models or when access to the original training data is

The associate editor coordinating the review of this manuscript and approving it for publication was Mohammad Hossein Moaiyeri.

limited. Several studies, such as GPTQ [18], AWQ [19], and SqueezeLLM [20], have demonstrated that weights-only quantization can significantly reduce the bit-width of weights without substantial performance degradation. However, weight-only quantization performs the most computation-intensive multiply-accumulate (MAC) operations in the floating-point domain. This requirement increases the chip area and power consumption, making it less suitable for edge devices where cost and power efficiency are critical.

Several studies [15], [16], [21], [22] have shown that simultaneously quantizing weights and activations are more effective in minimizing performance loss. These methods focus on minimizing the difference between the activation means of the full precision model and the quantized model, known as reconstruction loss. They utilize quantization scaling values and weight rounding to optimize this reconstruction loss and determine whether to round up or down after quantizing each weight.

However, several bottlenecks remain in low-bit PTQ that hinder practical deployment. First, quantization at extremely low bit-widths causes substantial accuracy degradation due to the accumulated quantization error across layers. Second, existing second-order optimization methods often require floating-point operations, which compromise hardware efficiency. Third, channel-wise scaling approaches may cause fragmentation that is incompatible with the parallel structure of MAC arrays.

In cases of extreme low-bit quantization, minimizing reconstruction error is challenging when relying solely on rounding and quantization scaling due to the inherent errors of low-bit quantization. To address this, we introduce new parameters in PTQ that adjust the scale and bias on a channel-by-channel basis, aligning more closely with the original model. While biasing can add computational load at inference, we can reduce this overhead. These parameters are integrated into the quantized model with minimal additional computation by rearranging and grouping the input channels and using shift operations. Using a limited dataset for PTQ, we effectively optimized the layer and verified that performance can be enhanced solely by employing shift operations, ensuring that all calculations remain within the integer domain.

Our main contributions are as follows:

1. We introduce parameters to mitigate the distribution disparities between full-precision and quantized models during quantization. The optimal group is identified by analyzing group probabilities and second-order errors.

2. Our proposed Weight-Activation Product Scaling Quantization (WAPS-Quant.) achieves efficient channel-wise quantization with less than 1.5% computation overhead in most networks and a marginal 2.4% area increase when implemented on ASIC.

3. Our method integrates seamlessly with existing second-order error-based PTQ techniques [15], [21], [22], enhancing their performance in low-bit quantization scenarios.

In edge devices, where internal memory and bandwidth are severely constrained, achieving sufficient performance for deep learning inference is often challenging. Low-bit quantization offers a promising solution by significantly reducing memory footprint and data movement cost. However, the most critical issue in low-bit quantization is the loss of accuracy, particularly under extreme bit-widths. To address this, our work proposes a hardware-friendly solution that mitigates accuracy degradation using simple integer operations and minimal logic changes. This makes it feasible to deploy quantized models on real-world edge hardware without sacrificing performance.

The rest of this paper is organized as follows: Section II discusses related work and background knowledge. Section III describes the proposed optimization method. Section IV presents experimental results comparing our method with existing techniques across various networks. Finally, Section V concludes the paper.

II. RELATED WORK AND BACKGROUND

This chapter explores the background theory and related research on Post-Training Quantization (PTQ) methods, identifying their limitations and challenges. **Notations** Scalars or vectors are represented in lowercase, while matrices and tensors are denoted in uppercase. For a real variable a, $\bar{a}$ and $\hat{a}$ signify the integer coded bit and quantized value, respectively. The quantized value $\hat{a}$ is the reconstructed approximation of the original variable a, i.e., $\hat{a} \approx a$. parenthesized superscript $a^{(\cdot)}$ indicates the layer index or the domain to which it belongs. Among superscripts, x, w, y, and z represent values for the input feature, weight, accumulated output feature, and output feature after batch normalization, activation, and re-quantization. Subscripts are used to indicate an element of a vector or matrix.

A. UNIFORM CHANNEL-WISE QUANTIZATION

Uniform channel-wise quantization [7], [8], [23], [24] apply a uniform quantization strategy separately to each output channel of a convolutional or fully-connected layer. This can be implemented using only integer operations. Given a real-valued r and the coded bit $\bar{r}$, the quantized value $\hat{r}$ can be expressed as[1]

$$\bar{r} = Q(r) = \mathtt{clamp}(\lfloor r/s \rceil + z; min, max), \quad (1)$$

$$\hat{r} = (\bar{r} - z) \times s, \quad (2)$$

where the constants $s \in \mathbb{R}$ and $z \in \mathbb{R}$ represent the scale and zero-point of quantization, respectively. $Q(\cdot)$ is the quantization function, $\lfloor \cdot \rceil$ denotes rounding to the nearest integer, and $\mathtt{clamp}(v; a, b)$ constrains the value v within the range $[a, b]$.

The weight and activation must be appropriately quantized to operate a neural network on a quantized device. For

[1] As in previous works [8], we omit nonlinear activations such as ReLU in the equation.

a simpler hardware implementation with improved performance, typically, all weights corresponding to the same output channel share the same scale value with a zero-point of zero (symmetric quantization) [24], and all features share the same scale and zero-point (asymmetric quantization). When considering a convolution or fully connected layer, where the weight matrix $W \in \mathbb{R}^{m\times c}$ is multiplied with the input feature $x \in \mathbb{R}^c$ to produce the output feature vector $y = Wx \in \mathbb{R}^m$, the feature of the k-th output channel, denoted as y_k, can be computed as follows:

$$y_k = \sum_i^c \hat{w}_{k,i}\hat{x}_i = \sum_i^c (s_k^{(w)}\bar{w}_{k,i}) \cdot (s^{(x)}(\bar{x}_i - z^{(x)})) \tag{3}$$

$$= s_k^{(w)}s^{(x)}\sum_i^c \bar{w}_{k,i}\bar{x}_i - z^{(x)}s_k^{(w)}s^{(x)}\sum_i^c \bar{w}_{k,i}, \tag{4}$$

where $s^w \in \mathbb{R}^m$ and $s^x \in \mathbb{R}$ are the scales of the weight and input feature, respectively. Note that the second term $z^x s_k^w s^x \sum_i^c \bar{w}_{k,i}$ can be pre-calculated before inference time. Eq. (3) is for symmetric weight quantization, and if this is not the case, i.e. $z^w \neq 0$, an additional multiplication and addition operation will be required. This expression shows that quantization transforms the original operation into a scaled integer dot-product, plus a correction term based on the input zero-point.

1) CHANNEL-WISE SCALING

SYQ [25] proposed a method to learn the convolution weights' pixel/row-wise scaling value. However, this method requires using an FP32 accumulator instead of an integer accumulator because it uses real-valued scaling values. In contrast, our approach utilizes integer shift operations, which use 24 times less power on ASIC (45nm) [26].

Both DFQ [27] and SmoothQuant [28] focus on equalizing weight ranges by transforming input activations and weights into a more amenable form to quantization. Compared to these previous works, our method leverages the partial product of weight activation. This approach allows for broader data range adjustments, providing a better chance of aligning closely with the original model. This is because directly applying scaling up/down to individual weights or activations after quantization leaves very little information in low bits, making adjusting difficult. However, the weight-activation product typically accumulates 32-bit integer information, allowing more precise results when scaling or shifting is applied.

2) QUANTIZATION GROUPING

To optimize quantization, some studies extend scaling from output channels to input channels. Methods such as [13], [29], [30] divide the input channels into multiple groups and find the appropriate scale value for each group. However, this approach necessitates floating-point adders for cross-group accumulation, as integer operations are restricted to within-group computations.

RPTQ [31] calculates the min/max of each channel to form groups, often resulting in more than eight groups and cases where only one channel exists in a group. The typical MAC accumulator structure computes multiple input channels simultaneously, requiring several operations for different scale input channels, which consumes additional time. Additionally, fine-grained grouping increases parameter information for each group, leading to higher memory requirements.

Methods like GPTQ [18] and AWQ [19] used grouping but focused solely on weight quantization. While effective in compressing weight bit-width, these methods still require floating-point operations for the weight-feature multiplication, resulting in increased power consumption and area compared to integer-only operations.

Our approach maintains the most computationally intensive parts of the adder tree in integers without requiring additional memory usage. Furthermore, instead of computationally complex floating-point operations, it only requires integer shift operations outside the adder tree. This ensures efficient and power-saving operations suitable for edge devices where cost and power efficiency are critical.

B. SECOND-ORDER ERROR MINIMIZING

Recent studies have shown significant improvements in post-training quantization performance [15], [17], [21], [22] even with a small calibration set by minimizing the mean reconstruction error of the intermediate feature maps. The loss increment caused by quantization can be analyzed by the second-order Taylor series expansion as follows:

$$\mathbb{E}_x[\Delta\mathcal{L}] = \mathbb{E}_x[\mathcal{L}(w + \Delta w)] - \mathbb{E}_x[\mathcal{L}(w)] \tag{5}$$

$$\approx \Delta w^T \bar{g} + \frac{1}{2}\Delta w^T \bar{H}\Delta w, \tag{6}$$

where $\bar{g} := \nabla_w \mathbb{E}_x[\mathcal{L}]$ and $\bar{H} := \nabla_w^2 \mathbb{E}_x[\mathcal{L}]$.

This shows that the loss increase due to quantization can be approximated by the curvature (Hessian) of the original model loss surface, which motivates the use of second-order methods for minimizing quantization error. Assuming that the gradient of the pre-trained model is nearly zero and the effect of higher order terms is negligible due to small Δw, the change in loss can be sufficiently approximated by the Hessian term, i.e., $\mathbb{E}_x[\Delta\mathcal{L}] \approx \frac{1}{2}\Delta w^T \bar{H}\Delta w$. Given a large number of parameters in deep learning models, directly calculating the Hessian matrix can be challenging and under some assumptions on Hessian and CNN architecture, it becomes

$$\Delta w^T \bar{H}\Delta w = \sum_l \Delta w^{(l)T}\bar{H}^{(l)}\Delta w^{(l)} \tag{7}$$

$$= \sum_l ||\Delta W^{(l)}x^{(l-1)}||_2^2 = \sum_l ||\Delta y^{(l)}||_2^2. \tag{8}$$

Here, Δw is the vectorized version of ΔW and $y^{(l)} = W^{(l)}x^{(l-1)}$ for the $(l-1)$-th layer's feature map $x^{(l-1)}$ and l-th layer's weight matrix $W^{(l)}$. Thus, minimizing the loss

TABLE 1. Comparison of representative PTQ methods and their characteristics.

Method	Weight Quant.	Activation Quant.	Second-Order	Mixed-Precision	HW Friendly	LLM Support
GPTQ	✓	✗	✓	✗	✗	✓
AWQ	✓	✗	✗	✗	✗	✓
SmoothQuant	✓	✓	✗	✗	✓	✓
Mr.BiQ	✓	✓	✓	✗	✗	✗
DF-MPC	✓	✓	✓	✓	✗	✗
WAPS (Ours)	✓	✓	✓	✗	✓	✗

boils down to minimizing the block reconstruction error of each layer's output feature map [15].

1) SECOND-ORDER ERROR BASED PTQ

Previous studies, which aimed to minimize reconstruction errors, employed techniques such as rounding up or down weights methods [15], [16], [21], [22] to find optimal values. AdaQuant [32] took it further by using the optimal batch normalization bias. However, in extremely low-bit quantization scenarios, simply relying on weight rounding and bias proved inadequate for reconstruction. Mr.BiQ [17] employed non-uniform quantization, while DF-MPC [33] employed mixed precision to improve reconstruction, but these approaches posed challenges for efficient utilization on standard systolic or MAC array.

We have improved this process by scaling and biasing both the input and output channels, with the goal of reducing reconstruction error and minimizing the distributional gap. Additionally, our method is compatible with existing scale and round-adjusting methods.

C. OPTIMIZED HARDWARE SOLUTIONS FOR IMPROVED QUANTIZATION IN NEURAL NETWORKS

Recent studies [34], [35], [36] have proposed methods to enhance quantization performance by modifying the internal stages of multiplication and accumulation. However, these approaches encounter several issues when applied to practical Neural Processing Unit (NPU) development. For instance, they may require additional parameters, utilize data differently across intervals requiring specialized data reading patterns, or demand processing with specialized data types, all of which introduce significant logic overhead and complicate generalization. These challenges increase implementation complexity and reduce efficiency in NPUs operating across various models.

In contrast, our method enhances low-bit quantization performance without modifying the basic multiplication and adder tree stages or altering the input data. By employing minimal logic adjustments at the final accumulation stage, we significantly improve performance while maintaining the existing MAC structure. This approach preserves the simplicity and generalizability of the NPU architecture, ensuring efficient and effective operation across various models.

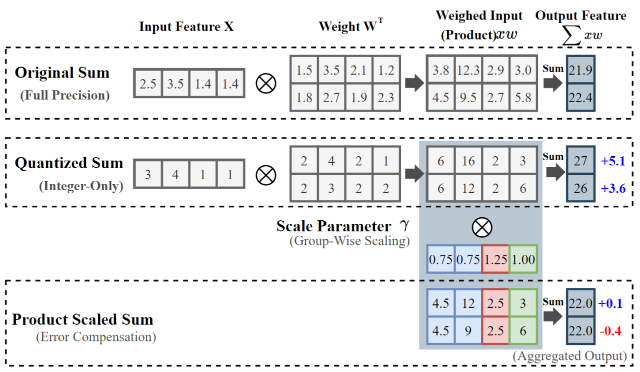


FIGURE 1. Comparison of standard quantized MAC operations and quantized MAC operations with input channel-wise scaling factors applied.

Table 1 summarizes representative post-training quantization (PTQ) methods, comparing them across key aspects such as the scope of quantization, use of second-order optimization, hardware compatibility, and applicability to large language models (LLMs). While most methods are not designed to explicitly support mixed-precision quantization, their layer-wise optimization structure inherently allows flexibility for bit-level adjustments per layer. For instance, WAPS-Quant performs block-level optimization and can be adapted to mixed-precision schemes by assigning different bit-widths per layer if needed. In the LLM Support column, although our method is primarily evaluated on CNNs and ViTs, the underlying design is compatible with transformer architectures, and results on object detection and vision transformers suggest potential for extension to LLMs.

III. PROPOSED METHOD

The distribution of the multiplied quantized weights and inputs can diverge from the actual values due to accumulated rounding and clipping errors from the clamp function, as well as quantization errors from the previous layers [27], [32], [37], [38]. The approaches presented in [15], [16], [39], and [21] sought to minimize the sum of squared differences between original output features and quantized features by solely adjusting weight rounding. Nevertheless, as quantization modifies both the weights and input features, solely relying on rounding in low-bit quantization poses limitations in achieving close approximations to the full-precision values.

Fig. 1 shows an example of a 2-bit weight/activation quantization calculation process with $\bar{x}, \bar{w} \in \{1, 2, 3, 4\}$. Input $X \in \mathbb{R}^{1\times 4}$ and weight $W \in \mathbb{R}^{2\times 4}$ perform an element-wise multiplication, followed by accumulation. The results of the operations before quantization are 21.9 and 22.4, which are the layer outputs. In this example, when quantization is applied, X and W are quantized to values between 1 and 4. After performing the element-wise multiplication and summing each channel, the final values obtained are 27 and 26, showing differences of 5.1 and 3.6 from the original values. This is due to the accumulation of quantization errors from each input X and weight W, which significantly alters the accumulated result.

WAPS-Quant searches for the channels to scale down and scale up in each input channel-wise to optimize the loss. In the example, the first and second input channels are scaled down, the third channel is scaled up, and the fourth channel remains unchanged. By multiplying each product by the scale parameter γ and then summing, we get values closer to the original output features, with differences of 0.1 and -0.4, compared to the results after quantization.

In addition to product scaling, we will also adjust rounding and output channel-wise bias to minimize errors due to network quantization. Rounding and output channel-wise bias adjustments can be simultaneously searched along with product scaling, making the process effective. We can pre-compute these operations offline to minimize additional computations during actual inference. We will explain in Appendix A-C methods to maximize MAC utilization and minimize fragmentation caused by grouping while keeping additional computations to a minimum during inference.

WAPS-Quant can be easily integrated with many existing PTQ methods [15], [21], [32] that adjust rounding to perform PTQ. Rounding-based PTQ approaches typically learn to minimize the output for each block or layer while deciding on rounding up or down. During this process, our channel scale-up/down technique can be implemented alongside.

In our implementation, scaling group determination, γ, and rounding are trained simultaneously to ensure an accurate reconstruction of the full precision value. It's important to note that scaling solely on features or weights can lead to losing scaling information in low-bit quantization. However, partially accumulated products provide a wider range, allowing for more precise adjustments and a better representation of the full precision values.

A. INPUT CHANNEL-WISE PRODUCT SCALING GROUP(ISG)

On top of the output channel-wise quantization, we use different scales and c input channels. As described in Sec. II-A, employing output channel-wise quantization allows the representation of the sum of quantized values as an affine form of full precision values $\alpha_k \sum_i^c \bar{w}_{k,i}\bar{x}_i + \beta_k$. By introducing per-input channel scaling factors, denoted as $\gamma^{(y)} \in \mathbb{R}^c$, it becomes $\alpha_k \sum_i^c (\gamma_i^{(y)} \bar{w}_{k,i}\bar{x}_i) + \beta_k$.

However, previous studies that utilize input channel-wise grouping require floating-point adders to calculate the multiplication results in the floating-point domain. Simply dividing groups and performing accumulation while incorporating a distinct scale value for each input channel inevitably introduces floating-point operations due to the varying resolutions across different input channels. This approach also requires additional memory, significantly increasing computational complexity. To address this issue, we use input-channel grouping and integer shift/sum scaling, which separates input channels into small groups of different scales. Each input channel belongs to one of the scale groups, $\{G_1, \cdots, G_g\}$ with scale values $\{\gamma_1^G, \cdots, \gamma_g^G\}$. The channels within each group are configured to carry out integer multiplication and accumulation operations together.

B. ADJUSTING CHANNEL-WISE BIAS

Instead of employing input-channel-wise biasing, we indirectly bias the product using the output channel-wise scaling and biasing approach presented in AdaQuant [32]. We take an affine transformation of the data before re-quantization using the trainable scale $\gamma^{(z)} \in \mathbb{R}^m$ and bias $\varphi^{(z)} \in \mathbb{R}^m$ resulting in

$$z_k = \gamma_k^{(z)} \alpha_k \sum_i^c \gamma_i^{(y)} \bar{w}_{k,i}\bar{x}_i + \gamma_k^{(z)} \beta_k + \varphi_k^{(z)}, \tag{9}$$

$\gamma^{(z)}$ and φ^z adjust the distribution of the activation outputs to better align with the real values before the activation function and re-quantization.

C. COMPUTATIONAL COMPLEXITY AT INFERENCE

1) PRECOMPUTATION

To perform accumulation within each group, computing each layer becomes

$$z_k = \gamma_k^{(z)} \alpha_k \sum_{p=1}^{g} \gamma_p^G \sum_{i \in G_p} \bar{w}_{k,i}\bar{x}_i + \gamma_k^{(z)} \beta_k + \varphi_k^{(z)} \tag{10}$$

$$= \alpha'_k \sum_{p=1}^{g} \gamma_p^G \sum_{i \in G_p} \bar{w}_{k,i}\bar{x}_i + \beta'_k. \tag{11}$$

In Eq. (10), the last two terms in the middle are merged into one variable β'_k and $\alpha'_k := \gamma_k^{(z)} \alpha_k$. The reparametrized constants α'_k and β'_k enable pre-computation of affine transformations, avoiding runtime floating-point operations.

The real-typed variables $\gamma^{(z)} \in \mathbb{R}^m$ and $\varphi^{(z)} \in \mathbb{R}^m$ are known before inference time, thus $\alpha'_k (= \gamma_k^{(z)} \alpha_k)$ and $\beta'_k (= \gamma_k^{(z)} \beta_k + \varphi_k^{(z)})$ can be pre-computed. To reduce the computational cost associated with high-cost floating-point operations, we set each $\gamma^G \in \mathbb{R}^g$ value to be computable by integer shift/sum operations, using values such as 1.0 ± 2^{-n}, where $n \in \{0, 1, 2, \ldots\}$.

2) CHANNEL PERMUTATION

When performing accumulation across input channels, the hardware may not perform optimally if the values are mixed in an unstructured arrangement, and additional memory is needed to store information about the selected groups. By pre-adjusting the order of the output channels of the preceding layer, we can alter the order of the input channels for the subsequent layer. By performing operations in independent layers for each group of input channels and adding independent accumulation values at the accumulation point, we can achieve computational speeds nearly identical to the original speeds, provided that there are not too many groups and that the groups are not excessively small.

TABLE 2. Computational complexity: MAC operations for different approaches on ResNet-18 layer1.conv1 layer, which have [3 × 3 × 64] weights per one quantized output feature. (*g* is the number of groups for our method. *Quant* refers to the conventional method of output channel quantization, while *Scaling* denotes the process of input channel-wise scaling without the use of grouping.)

Approach	Floating Point		Integer		Other
	Mul.	Add.	Mul.	Add.	Shift
Baseline	3×3×64	3×3×64-1	-	-	-
+*Quant.*	1	1	3×3×64	3×3×64-1	-
+*Scaling*	64	64-1	3×3×64	-	-
WAPS(Ours)	1	1	3×3×64	3×3×64-1+g	g

3) COMPUTATIONAL COST PER APPROACH

Table 2 illustrates the computational cost associated with each approach for generating a single output feature. In the absence of quantization, the convolution layer with $k_h \times k_w$ kernel requires $k_h \times k_w \times c$ multiplication operations and $k_h \times k_w \times c - 1$ addition operations, executed via the MAC process. The conventional output channel-wise quantization requires integer operation for MAC and an additional floating MAC operation due to re-quantization. Previous works that implemented scaling on an input channel-wise basis resulted in products becoming floating-point data, requiring the use of floating-point multipliers and adders. In contrast, our method keeps the product within the integer domain for calculations, requiring only a small number of shifts and additions.

D. OPTIMIZATION PROCESS

1) WEIGHT QUANTIZATION

For the purpose of weight quantization, a rounding policy introduced in Adaround [16] is adopted:

$$Q(W_k) = \texttt{clamp}(\left\lfloor \frac{W_k}{s} \right\rfloor + h(V_k); min, max), \quad (12)$$

$$\text{where } h(V_k) = \texttt{clamp}(\sigma(V_k)(\zeta - \tau) + \tau; 0, 1). \quad (13)$$

Here, $V \in \mathbb{R}^{c\times m}$ is a learnable parameter utilized for rounding, while the constants ζ and τ are employed for stretching the sigmoid function $\sigma(\cdot)$ and are conventionally set to 1.1 and -0.1, respectively. The function $h(V)$, introduced in [40], guides values to converge towards either 0 or 1. It employs a clipping mechanism to prevent differentiation at extreme values, thus guaranteeing convergence to either 0 or 1. During the calibration process, this function is critical in minimizing the reconstruction error between rounding down and up. Instead of the conventional on/off rounding in standard quantization, the floor function is employed, with the on/off rounding being learned via the rectified sigmoid function,

2) PRODUCT GROUPING

The scaling factor γ_p^G is defined before calibration, and each input channel finds an optimal group based on the probability $s(R_{i,p})$. During calibration, the scale γ_i^y for the input channel i is calculated and used as follows,

$$\gamma_i^{(y)} = \sum_p^g \left(\gamma_p^G s(R_{i,p}) / \sum_j^g s(R_{i,j}) \right), \quad (14)$$

$$s(R_{i,p}) = \texttt{clamp}\left(\frac{\sigma(R_{i,p})}{\sum_j^g \sigma(R_{i,j})} (\zeta - \tau) + \tau; 0, 1 \right), \quad (15)$$

where the learnable parameters $R \in \mathbb{R}^{c\times g}$ are used to find the probability that the input channel belongs to a certain scale group $G_p^{(l)}$. The rectified softmax function $s(R_{i,p})$ is utilized to express the probability of inclusion in each group. Similar to $h(V)$, if the probabilities are not fixed during calibration, Eq. (14), the γ_G^p values of the different groups are combined in proportion to their probabilities. However, if the value of $s(R_{i,p})$ reaches 1, the γ_i^y is fixed to a specific γ_p^G value. Eq. (14)implements a rectified softmax to stabilize group assignment probability, ensuring clear convergence during calibration.

3) REGULARIZATION

The values of $h(V)$ used in weight quantization and the value of $s(R_{i,p})$, which represents the probability of inclusion in each group in input channel grouping, should converge to 0 or 1 during calibration with regularization term presented in [16] as follows:

$$f_{reg}(V, R; \lambda_r, \lambda_g) = \lambda_r \sum_i^m \sum_j^c (1 - \left|2h(V_{i,j}) - 1\right|^\beta) + \lambda_g \sum_i^c \sum_p^g (1 - \left|2s(R_{i,p}) - 1\right|^\beta),$$

where λ_r and λ_g are regularization hyperparameters. β is initially assigned a high beta value, which encourages convergence to only the extreme values of 0 or 1. Towards the end of the calibration, we implement normalization by assigning lower β values to allow values near 0.5 to converge as well. This loss formulation jointly optimizes product scaling, rounding, and grouping while regularizing for discrete decisions.

4) LOSS FUNCTION FOR OPTIMIZATION

To identify the optimal parameters V, R, γ^zs and φ^z using a small calibration set, we utilize the block reconstruction loss as described in Sec. II-B. Our optimization proceeds at the block level, and parameters V and R are subjected to regularization. The final loss function can be represented as

$$\underset{V,R,\gamma^z,\varphi^z}{\arg\min} \left\| Z - \hat{Z} \right\|_F^2 + f_{reg}(V, R; \lambda_r, \lambda_g), \quad (16)$$

where Z represents the real-valued output feature while $\hat{Z}$ denotes the approximated reconstructed output feature from Eq. (10), and $\|\cdot\|_F^2$ denotes the Frobenius norm. Note that $\gamma^G \in \mathbb{R}^g$ is a hyperparameter which determines γ^y value. Similar to other round-only quantization methods [15], [16], [21], [32], groups and rounding are determined for each block

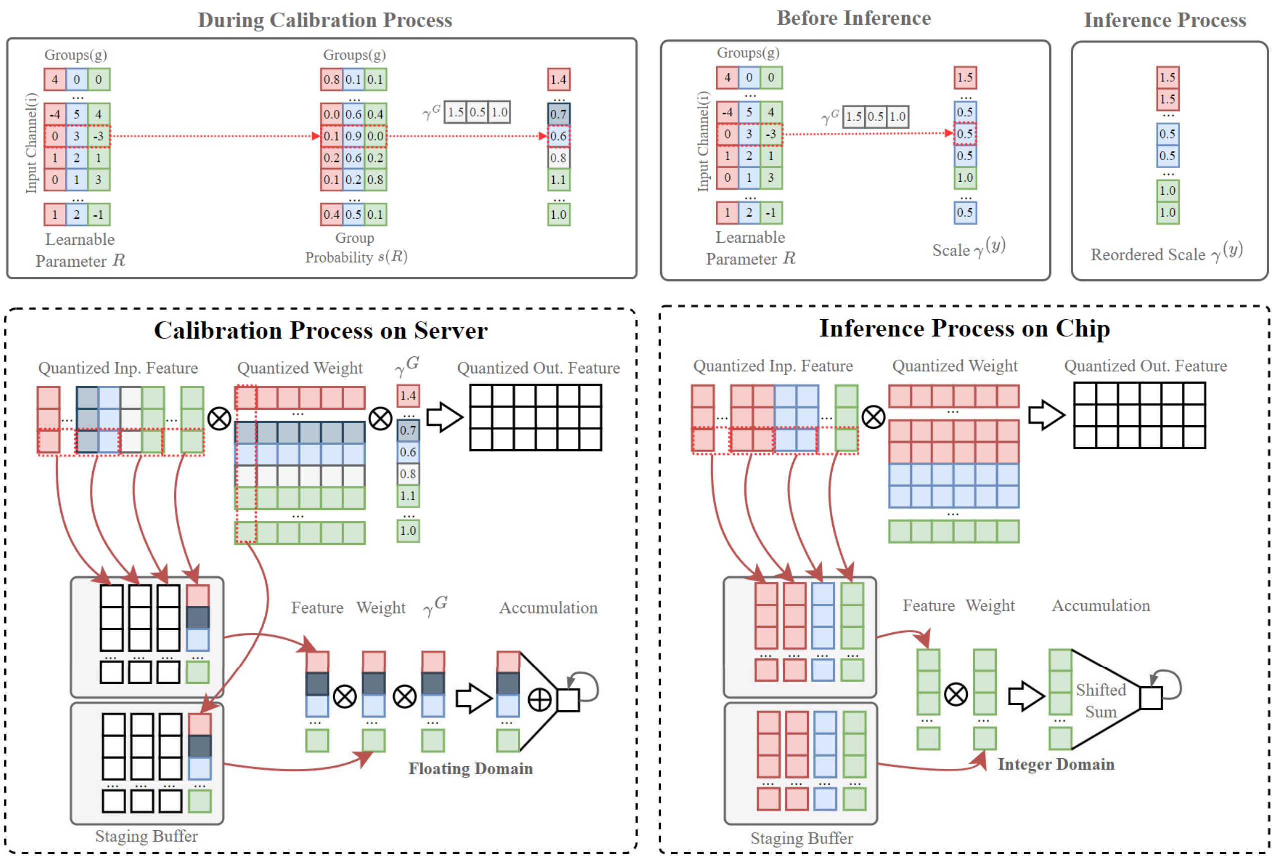


FIGURE 2. **Calibration and inference pipeline of WAPS-Quant. Group-wise scaling is optimized during calibration, and inference is performed using integer-only operations with shift-based compensation.**

to minimize loss. Fig.2 illustrates the processes computed during the overall calibration and inference stages in our method. During the calibration process, each input is channel-wise grouped, and in the inference stage, it operates with minimal computational cost.

IV. EXPERIMENTS

We evaluate the proposed method on various vision models to assess its performance. Our code is based on an open-source BRECQ [15] for our implementation. To minimize the impact of hyperparameters, we set $\lambda_r = \lambda_g$. Since too much fragmentation could lead to inefficient processing in hardware, we designated the number of groups to be 3, and the group scale values γ^G were set to $(1.0, 1.0{-}2^{-4}, 1.0{+}2^{-4})$. Consistent with BRECQ, we used a randomly selected calibration set of 1,024 samples from the ImageNet [41] training set, the first and last layers employed 8-bit precision like [15], [17], [21], and [22], and the weight tuning method was configured identically to that of AdaRound [16]. The calibration process involves 35,000 iterations, taking around 40 minutes to calibrate a ResNet-18 model on a single RTX 3090 GPU.

A. MAIN RESULTS

Table 3 and Table 4 show the results of weight-only and weight-feature quantization experiments conducted on various CNN-based architectures. We conducted quantization on a variety of network architectures, including ResNet-18,50 [43], MobileNetV2 [44], MNasNet_v2 [45], and RegNet [46]. We report the mean and the standard deviation on five trials for ours.

For 2-bit weight quantization, our proposed method improves by approximately 0.5-1.5%p compared to the baseline methods AdaRound and BRECQ. As the quantization bit increases to 3-bit and 4-bit, the difference in performance gradually decreases. This can be attributed to the fact that as the quantization bit becomes sufficiently large, the accumulated bits can be adequately represented, reducing the importance of adjusting the scaling and bias. Additionally, our method shows better results than BRECQ in experiments with 2-bit weight and 2 or 4-bit feature quantization. We also achieved up to 3%p absolute improvement in feature and weight quantization than QDROP.

Carried out on detection tasks and the transformer-based VIT-B model, we achieved a maximum increase of 0.011 in mAP compared to other low-bit quantization methods, and we also observed a performance improvement of 0.64%p in the VIT-B model with 6-bit weight activation.

1) GENERALIZATION TO TRANSFORMERS AND LLMS

In addition to CNNs, we also evaluated our method on ViT-B and object detection tasks, showing notable improvements

TABLE 3. Evaluation of top-1 accuracy (%) in weight only quantization on the ImageNet validation set.; * denotes the number are from BRECQ [15] and FlexRound [42].

Methods	Bits(W/A)	ResNet-18	ResNet-50	MobileNetV2	RegNet-600MF	RegNet-3.2GF	MnasNet-2.0
FullPrec.	32/32	71.08	77.00	72.49	73.71	78.36	76.68
AdaRound*	4/32	68.71	75.23	69.78	71.97	77.12	74.87
AdaQuant*	4/32	68.82	75.22	44.78	-	-	-
BRECQ*	4/32	70.70	76.29	71.66	73.02	78.04	76.00
FlexRound*	4/32	70.28	75.95	70.82	-	-	-
Ours	4/32	**70.72**$\pm$0.08	**76.37**$\pm$0.09	**72.08**$\pm$0.05	**73.16**$\pm$0.06	**78.20**$\pm$0.05	**76.13**$\pm$0.04
AdaRound*	3/32	68.07	73.42	64.33	67.71	72.31	69.33
AdaQuant*	3/32	58.12	67.61	12.56	-	-	-
BRECQ*	3/32	69.81	75.61	69.50	71.48	77.22	74.58
FlexRound*	3/32	68.65	74.38	66.87	-	-	-
Ours	3/32	**70.02**$\pm$0.11	**75.77**$\pm$0.06	**70.43**$\pm$0.08	**72.02**$\pm$0.13	**77.60**$\pm$0.07	**74.96**$\pm$0.09
AdaRound*	2/32	55.96	47.95	32.54	25.66	24.70	30.60
AdaQuant*	2/32	0.30	0.49	0.11	-	-	-
BRECQ*	2/32	66.30	72.40	59.67	65.83	73.88	67.13
FlexRound*	2/32	62.57	63.67	46.04	-	-	-
Ours	2/32	**67.22**$\pm$0.11	**72.96**$\pm$0.13	**62.16**$\pm$0.24	**67.01**$\pm$0.29	**74.97**$\pm$0.07	**68.18**$\pm$0.17

TABLE 4. Evaluation of top-1 accuracy (%) in weight and feature quantization on the ImageNet validation set.; * denotes the number are from BRECQ [15] and FlexRound [42], while † denotes results from each paper's official open-source repository.

Methods	Bits(W/A)	ResNet-18	ResNet-50	MobileNetV2	RegNet-600MF	RegNet-3.2GF	MnasNet-2.0
FullPrec.	32/32	71.08	77.00	72.49	73.71	78.36	76.68
ZeroQ*	4/4	21.71	2.94	26.24	28.54	12.24	3.89
LAPQ*	4/4	60.30	70.00	49.70	57.71	55.89	65.32
AdaQuant*	4/4	67.50	73.70	34.95	-	-	-
BRECQ†	4/4	68.93	77.85	67.43	70.44	76.40	72.33
QDROP†	4/4	69.14	74.99	67.95	70.87	76.46	73.04
FlexRound*	4/4	69.32	74.56	63.74	-	-	-
Ours	4/4	**69.64**$\pm$0.06	**75.12**$\pm$0.07	**68.47**$\pm$0.14	**71.00**$\pm$0.09	**76.58**$\pm$0.08	**73.63**$\pm$0.05
ZeroQ*	2/4	0.08	0.08	0.10	0.10	0.05	0.12
LAPQ*	2/4	0.18	0.14	0.13	0.17	0.12	0.18
AdaQuant*	2/4	0.21	0.12	0.10	-	-	-
BRECQ†	2/4	63.92	69.58	52.37	61.53	71.05	60.42
QDROP†	2/4	64.46	69.71	53.61	62.81	71.88	61.87
Ours	2/4	**66.00**$\pm$0.05	**71.09**$\pm$0.19	**56.10**$\pm$0.12	**63.60**$\pm$0.25	**72.47**$\pm$0.15	**63.42**$\pm$0.26

under low-bit quantization (see Appendix B).These results demonstrate that WAPS is not limited to convolutional architectures. However, it is important to note that our method relies on product-level scaling to reduce bit-level quantization error, which is more effective in ultra-low-bit regimes (e.g., 2 to 4 bits). However, recent QAT approaches such as [47] have achieved ternary weight quantization in LLMs, most PTQ methods for transformers still operate in higher-bit settings (e.g., W8A8 or W4A8). Therefore, the accuracy benefit of our method may not be fully realized under current LLM quantization practices. However, if lower-bit PTQ becomes more viable for LLMs and generative models in the future, the hardware-friendly structure of WAPS and the error compensation mechanism could be highly beneficial.

B. ABLATION STUDY

In Table 5, we investigate the effect of each proposed component using ResNet-18 on the W2A4 settings.

TABLE 5. Ablation study of Product Grouping and Output Channel-Wise scaling/bias on the ResNet-18 model using ImageNet calibration data. Top-1 accuracy comparison on ImageNet test dataset.

Method	V	γ^z, φ^z	γ^y	R	W Quant.	W&A Quant.
Round-Only	O				66.41(+0.00)	64.80(+0.00)
Inp. Scale	O		O		67.18(+0.77)	66.04(+1.24)
+Out. Off.&Scale	O	O	O		**67.21**(+0.80)	**66.05**(+1.25)
Inp. Scale Grouped	O			O	67.02(+0.61)	65.88(+1.08)
+Out. Off.&Scale	O	O		O	**67.19**(+0.78)	**65.97**(+1.17)

1) EFFECT OF GROUPING

To compare the effect of grouping, we looked at the results with and without grouping when applying product scaling. Without grouping, we saw the highest accuracy because the model could find the optimal scale value γ^y for each input channel. However, we can see that even with grouping, the performance drop is less than 0.2%p, which shows

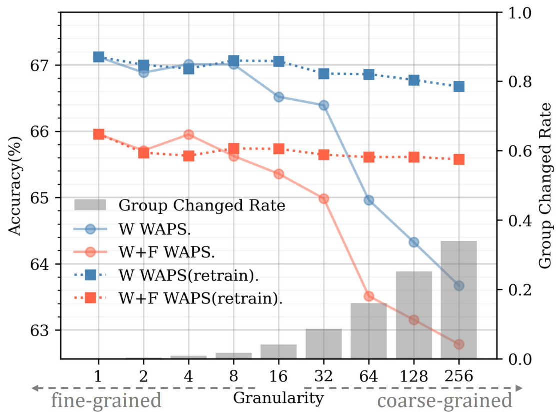


FIGURE 3. Performance change(top-1 accuracy) on ResNet-18 model using ImageNet for granularity variation of the input channel group.

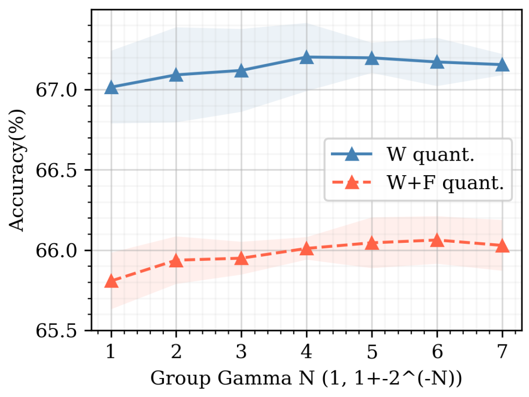


FIGURE 4. Accuracy comparison on ResNet-18 model using ImageNet as the γ^G value changes.

that the proposed method does not significantly sacrifice performance.

2) EFFECT OF OUTPUT CW. SCALING/BIASING

We compare the performance with and without each parameter to examine the effect of scale factor γ^z and offset φ^z. As shown in Table 5, using output channel-wise scaling and biasing with our method compensates for lost accuracy due to grouping and reduces the gap between methods with and without grouping.

C. INPUT CHANNEL GROUP

1) INPUT CHANNEL GROUP GRANULARITY

Input channel grouping has the disadvantage that hardware performance can be degraded if too many groups become fragmented. Hardware architectures designed for MAC operations, such as the systolic array, simultaneously perform a specific amount of MAC operations on input and output channels. This feature is crucial for computationally intensive tasks. Server hardware like Tensor Processing Units (TPUs [48]) can perform 256 $\times$ 256 operations simultaneously, while deep learning accelerators for edge devices can handle at least 16 $\times$ 16 operations [49], [50] at a time. This indicates that if the input channel grouping fragments the channels, the hardware cannot efficiently utilize MAC in the direction of the input channels.

Our initial approach was to evaluate the accuracy by varying the granularity of the input channel groups, with grouped channels prioritized. Higher granularity signifies that the number of channels per group is not limited. Each channel can be assigned to its optimal input scale group G_g. On the other hand, coarse granularity means that the number of channels per group is more balanced, making it more compatible with certain hardware. As depicted in Fig. 3, we can observe that decreasing the granularity led to a rise in the divergence from the optimal group (learned when granularity is the highest). However, by leaving the granularity-constrained groups intact and fine-tuning the remaining variables, we outperformed the traditional rounding method even at a granularity of 256.

TABLE 6. Increased giga integer operations by ISG for each model.

Model	Baseline	ISG	Total
ResNet-18	3.005	0.006	3.011 (+0.20%)
ResNet-50	5.576	0.013	5.589 (+0.24%)
MobileNetV2	0.338	0.014	0.345 (+4.03%)
RegNet-600MF	0.777	0.010	0.782 (+1.28%)
RegNet-3.2GF	4.442	0.030	4.472 (+0.68%)
MnasNet-2.0	2.217	0.025	2.242 (+1.11%)

2) INPUT CHANNEL SCALE FACTOR

In Sec III, we showed that multiplying the $\hat{w}\hat{x}$ by a γ^y value can bring it closer to the real value distribution. To see how performance varies with γ^y, we varied γ^y group from $(1, 1 \pm 2^{-1})$ to very small scale values such as $(1, 1 \pm 2^{-7})$ and compared the accuracy. As shown in Fig. 4, the weights quantization result was best when changing the value of $\hat{w}\hat{x}$ by about 6%(2^{-4}), and weight&feature quantization cases improve accuracy when changing $\hat{w}\hat{x}$ by about 1.5%(2^{-6}).

D. COMPUTATION COSTS FOR MODELS

Table 6 shows the increase in integer operations when ISG is applied to each model. To calculate each model's cost, we counted the number of integer operations in its fully connected and convolutional layers. We then evaluated the increase in computation due to ISG compared to the baseline. We also counted the shift operation as a single integer operation. For depth-wise convolution [51], our method is unnecessary because each input channel corresponds to an individual output channel, and each channel can use γ^z individually. Except for MobileNetV2, the computation cost increased by less than 1.3%. The cost savings of our method are greater when the convolution filter size is large. MobileNetV2 mostly uses 1 $\times$ 1 convolution. It increased by about 4% compared to the other models. However, even this

TABLE 7. Inference latency, throughput, and top-1 accuracy measured on hardware.

Model	BRECQ			WAPS (Ours)		
	Latency (µs)	FPS (img./s)	Accuracy (%)	Latency (µs)	FPS (img./s)	Accuracy (%)
ResNet-50	1762	567.54	69.58	1780	561.78	71.09
MobileNetV2	7994	125.09	52.37	8187	122.14	56.10

TABLE 8. ASIC synthesis area of each method and latency of convolution layer.

Methods	FP16 Model	integer Quant.	with WAPS
Total Area(um^2)	882,233	429,342	439,762
Latency(us)	1129.315	1129.278	1129.283

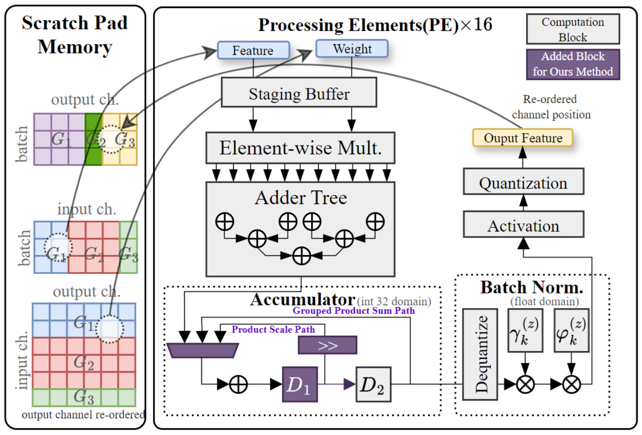


FIGURE 5. Overview of the processing elements in our method. The purple sections indicate the additional hardware logic required to support our approach.

number can be made much smaller for hardware that supports shift operations since shift operations can be made with simple wire connections compared to addition operations.

To evaluate real-world efficiency, we performed register-transfer level (RTL) simulations of our quantized models using an internally developed inference accelerator. The simulations were conducted on a modified version of our silicon-proven TV chip NPU, where only minimal changes were made to support WAPS-specific integer operations. The NPU consists of four 16×16 processing element (PE) arrays synthesized at 800 MHz. Latency and throughput (FPS) were measured using this RTL-level setup. The PE array supports fully integer-only execution, ensuring fair comparison with other PTQ methods. Detailed specifications of the NPU core architecture can be found in Section V-D.

As shown in Table 7, WAPS achieves higher accuracy than BRECQ with comparable latency and FPS, demonstrating its suitability for low-power edge NPUs without sacrificing accuracy or requiring additional hardware logic.

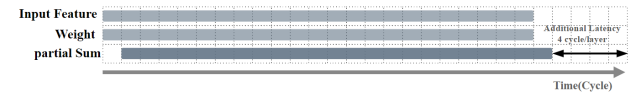


FIGURE 6. Timing diagram of the standard accumulation process.

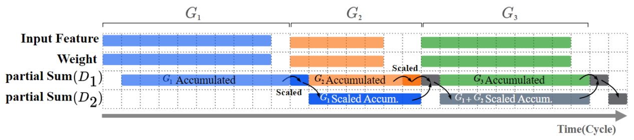


FIGURE 7. Timing diagram of the accumulation process with our method applied.

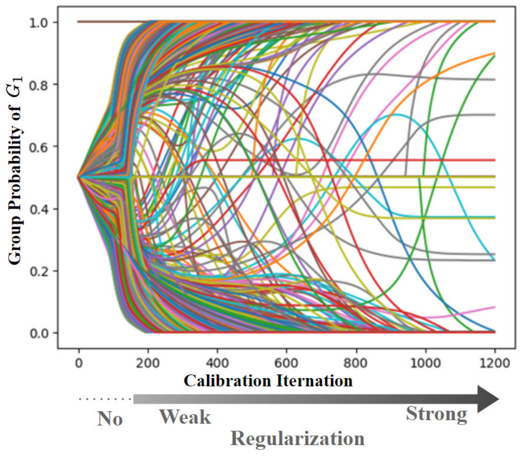


FIGURE 8. Convergence of input channel group probability over calibration iterations.

E. ASIC RESULTS

When implemented in ASIC hardware, to check for an increase in cost, we designed and synthesized the proposed method and confirmed the amount of area for Shift and Add operations and control logic. Additionally, we evaluated the impact on the latency of operations through simulation. According to Table 8, despite the addition of WAPS, the internal structure of the systolic array was not changed, resulting in a small area required for control operations. As a result, the area increase due to shifting and adding in the integer domain and control logic was just 2.4%. In all three implementations, pipelining was used, so the increase in latency is only added as a Flip-Flop stage due to timing issues. However, when computing the scaling value in floating point 16, a significant increase in area was observed. Detailed information on the area, structure, and synthesis environment of each module of these methods is described in Appendix A-D.

V. CONCLUSION

In real-world edge devices, constrained memory and limited bandwidth often hinder the deployment of large neural networks. Low-bit quantization effectively addresses these limitations by reducing storage and computational demands.

TABLE 9. A comparison of quantization performance when applying our method to BRECQ [15], QDROP [21], and NWQ [22].

Quantized Bits		method	resnet18	resnet50	mobilenetv2	regnetx600m	regnetx3200m	mnasnet
weight	**activation**							
2	**4**	BRECQ	63.922	69.582	52.368	61.532	71.050	60.424
		BRECQ+Ours	**64.572**	**70.452**	**54.414**	**62.804**	**71.846**	**61.428**
		QDrop	64.462	69.710	53.608	62.806	71.870	**61.876**
		QDrop+Ours	**65.198**	**70.790**	**55.446**	**63.852**	**72.474**	61.870
		NWQ	64.508	69.768	53.474	62.328	71.604	61.260
		NWQ +Ours	**64.828**	**70.374**	**54.830**	**63.030**	**72.200**	**62.256**
	2	BRECQ	47.616	**48.640**	5.118	**27.878**	**41.784**	10.310
		BRECQ+Ours	**47.664**	47.994	**8.562**	27.022	37.854	**19.668**
		QDrop	51.960	55.442	10.108	39.336	54.586	21.970
		QDrop+Ours	**52.976**	**55.874**	**15.344**	**41.264**	**55.862**	**26.454**
		NWQ	49.742	52.538	9.066	33.746	50.810	20.854
		NWQ +Ours	**50.872**	**52.770**	**10.672**	**35.494**	**51.196**	**27.162**
3	**3**	BRECQ	**64.980**	70.324	51.304	62.626	**70.984**	61.594
		BRECQ+Ours	64.766	**70.492**	**54.628**	**62.926**	70.530	**63.122**
		QDrop	65.616	71.332	54.836	64.688	71.764	64.276
		QDrop+Ours	**65.774**	**71.440**	**57.010**	**65.292**	**72.328**	**64.878**
		NWQ	65.128	70.728	**53.750**	63.362	71.564	62.758
		NWQ +Ours	**65.242**	**70.888**	55.390	**63.880**	**71.874**	**63.820**

TABLE 10. Performance comparison when quantizing the backbone (ResNet50) of Faster RCNN using each method.

Bits	-	mAP	mAP50	mAP75	mAPs	mAPm	mAPl
	BASE	0.403	0.610	0.44	0.24	0.441	0.515
4/4	BRECQ	**0.378**	**0.583**	**0.410**	0.219	0.414	**0.489**
	BRECQ+WAPS	**0.378**	0.581	0.409	**0.222**	**0.415**	**0.489**
	QDROP	0.379	0.584	0.412	0.220	0.415	**0.489**
	QDROP+WAPS	**0.381**	**0.585**	**0.414**	**0.225**	**0.416**	**0.489**
3/3	BRECQ	0.341	0.535	0.364	**0.192**	0.374	0.451
	BRECQ+WAPS	**0.342**	**0.538**	**0.365**	0.189	**0.375**	**0.452**
	QDROP	0.346	0.545	0.372	0.197	0.378	0.455
	QDROP+WAPS	**0.352**	**0.552**	**0.379**	**0.203**	**0.386**	**0.458**
2/4	BRECQ	0.357	0.556	0.386	0.201	0.391	0.469
	BRECQ+WAPS	**0.365**	**0.566**	**0.392**	**0.213**	**0.400**	**0.474**
	QDROP	0.360	0.560	0.387	0.206	0.396	0.474
	QDROP+WAPS	**0.369**	**0.571**	**0.399**	**0.215**	**0.406**	**0.480**

However, the accompanying accuracy drop remains a major bottleneck. Our method alleviates this issue by introducing a lightweight, integer-only scaling mechanism that compensates for quantization errors with negligible hardware overhead. This balance of efficiency and accuracy enables practical deployment of quantized models in bandwidth- and memory-limited environments.

This paper introduced a novel approach to post-training quantization (PTQ). Our method directly applies scaling to the product of weight and feature, finding the optimal group in a probabilistic manner that minimizes second-order error. Our research indicates that our method can be computed using simple integer operations. It requires less than a 1.3% increase in computational resources and, importantly, does not require additional memory.

Our approach uses hardware-friendly groupings, which makes it more efficient than existing methods. Thanks to our hardware co-design architecture, the computational cost of operations has been minimized. The MAC (Multiply-Accumulate) units, which account for most computation and power consumption, remain unchanged, ensuring minimal additional memory is required for scaling. This allows us to boost PTQ with minimal hardware support for product scaling. We precompute various operations to absorb the computational load, which helps maintain performance without significantly increasing hardware resources. By staging buffers and using a 16-channel design, our method optimizes data flow and reduces the need for storing large accumulation values separately.

Furthermore, our method addresses the issue of output channel-wise quantization errors, which can be mitigated using techniques like batch normalization or quantization. However, input channel-wise corrections are only possible at the accumulator and adder tree stages. Unlike other methods that require grouped quantization in the floating domain, our approach ensures that values remain in the integer domain, even with small groups. This enhances performance using second-order methods while minimizing calculation costs in hardware. Our method provides a way to increase performance with minimal computational overhead, requiring only an additional four cycles, which is negligible compared to the overall layer computation. Keeping the number of groups low reduces the likelihood of fragmentation, and output channels are already reordered in the weight, simplifying the implementation.

APPENDIX A PROPOSED HARDWARE ARCHITECTURES THAT ADOPT OUR METHOD

A. HARDWARE PROCESSING ELEMENT

When implementing our proposed method in hardware, the structure appears as shown in Fig. 5. The software, operating offline, receives input features and weights sorted into groups and reordered weights to match the next layer's output channels. After loading the features and weights, they are temporarily stored in a staging buffer and then undergo element-wise multiplication. The resulting values pass through an adder tree for accumulation.

In the accumulator, a single register is originally needed to store values, represented as the D_2 register in Fig. 5. During the computations within the same group, values are continuously accumulated in the D_1 register. When computations for a different group are needed, the value stored in the D_1 register is temporarily stored in the D_2 register. If the value in the D_1 register belongs to a group that requires scaling up or down, it is added by a fixed shift operation before being accumulated in the D_2 register. This ensures that the accumulated value of the input channel group is effectively scaled up or down before being stored back in the D_1 register.

Finally, the value stored in the D_2 register undergoes batch normalization, activation, and quantization sequentially before being stored in the scratchpad memory as the output feature. As depicted, the additional computation units do not significantly alter the existing path. They require only the fixed shifter, essentially a wire connection, a 3-to-1 multiplexer, and registers necessary for accumulation. Therefore, this method can be adopted with minimal changes to the basic MAC array architecture, enhancing performance without substantial hardware modifications.

B. ADDITIONAL CYCLES FOR COMPUTATION

Fig. 6 and 7 show the timing diagrams of a typical accumulation process and our method's accumulation process, respectively. Generally, after loading the features and weights, the MAC operation is performed, and values are accumulated one by one, consuming cycles proportional to the length of the input channel.

As shown in Fig. 7, values are first accumulated within each group in our method. For G_1, the values are accumulated in the D_1 register, and then one cycle is consumed for scaling up or down before temporarily storing them in the D_2 register. The same process is repeated for G_2, where values are accumulated within the group, one cycle is consumed for scaling up or down, and another cycle combines the values from G_1 and G_2. For G_3, since it does not require scaling, only one cycle is needed at the end to combine the values from G_1 and G_2. Our method requires only an additional 4 cycles throughout the entire MAC accumulation process to be effectively applied.

C. GROUP SIZE DETERMINATION METHOD

Many PTQ methods have attempted to improve performance using grouped quantization, but the granularity of the groups has rarely been considered. Fine-grained quantization groups can significantly improve performance by grouping similar-performing channels. However, MAC processors are designed to read multiple input channels simultaneously for parallel processing. If the groups are too numerous or if a group contains too few channels, the MAC processors cannot be fully utilized, which hampers efficient computation.

Fig. 8 illustrates the process of determining the groups for each channel in our method. Each line represents an input channel and shows the probability of being included in G_1. For example, if the Group Probability is 1.0, it indicates the channel belongs to G_1, while probabilities closer to 0 indicate the channel is assigned to G_2. Although our actual method uses three groups, this simplified illustration uses two groups for clarity in the calibration results. Most channels are assigned to a specific group within a few calibration iterations. Some groups start at a probability of 1.0, indicating no further changes are needed, such as when all the weights in a channel are zero. Certain channels are quickly assigned to G_1 or G_2 based on their feature and weight values. This process accelerates as the regularization parameter strengthens over iterations, rapidly pushing probabilities towards 0 or 1. However, some channels are less definitively assigned to either group, indicating difficulty in group determination.

As explained in Eq. 14, once a channel is assigned a probability of 0 or 1, its gradient ceases to change, and its group assignment becomes fixed. Rapid group determination for channels suggests that being in that group effectively reduces loss. Conversely, channels that are slowly assigned to a group may have less impact on loss reduction, making their optimal group assignment less clear. We calculate the order in which each channel is definitively assigned to a group. When forcibly assigning groups, we prioritize quickly determined channels, ensuring the groups are more effective at reducing loss while maintaining appropriate granularity.

D. ASIC SYNTHESIS ENVIRONMENTS

We conducted experiments using the structure of a batch-norm fused systolic array from an NPU developed for edge devices to compare overhead and latency in ASICs. The synthesis was done using the TSMC 12nm 0.9 process and 400 Mhz(2.5ns per cycle) timing conditions, and the unit of measurement is square micrometers. The MAC has a 16×16 structure, and WAPS, batch norm, and activation can process 8 data in 1 cycle. The sequence is MAC($\rightarrow$WAPS)$\rightarrow$ BN$\rightarrow$ACT$\rightarrow$Quant. The performance time, or latency, was tested only on the resnet-50 layer with 28×28x128 feature $3 \times$ 3x128 $\times$ 128 kernel layer. In FP16 models, the computational blocks were modified to use floating point operators, and due to timing issues, each operator required additional flip flops.

The performance of each module is as follows:

- MAC: A 16(input channel) x 16(output channel) MAC array.
- WAPS-Quant.: Consists of 8 integer adders and 8 integer shifters. Additional logic for WAPS-Quant.
- NORM: Includes 8 floating-point adders and multipliers.

- NORM_LUT: Memory allocated for normalization coefficients.
- Act: Contains 8 activation logic units.
- Quant: Comprises 8 quantization modules.

APPENDIX B ADDITIONAL RESULTS

E. IMAGENET CLASSIFICATION WITH RECENT ROUND-BASED METHOD

Table 9 presents the results from experiments designed to evaluate whether applying our proposed method to contemporary state-of-the-art round-based techniques results in performance enhancements within the low-bit quantization. We achieve performance enhancement in low-bit quantization by reducing the reconstruction error. We used the publicly available GitHub code for BRECQ [15] and QDrop [21]. We implemented and tested the Activation Regularization, Annealing Softmax, and Annealing Mixup as proposed in NWQ [22]. Our method, applied to three different techniques, particularly in low bits, demonstrated performance improvements of up to 9.3%p.

F. DETECTION TASK

Table 10 is the detection task result with applying our WAPS to the backbone (resnet50) of the faster rcnn coco model. WAPS operates orthogonally to the conventional rounding scheme of PTQ research (BRECQ [15], QDrop [21]). We used regularization weight 0.01 following QDrop and the result is superior or equivalent to the baselines. Especially, for extreamely low bit quantization, the performance gain is prominent.

G. REGULARIZATION PARAMETER LAMBDA

In our study, we established the values of λ_r and λ_g equal to simplify the hyperparameter tuning process. The role of λ_r is to regularize the rounding up or down, essentially influencing the convergence rate for rounding probability. Conversely, λ_g governs the speed at which the determination of the input channel-wise scale value reaches convergence. Since rounding and WAPS mutually affect one another, the rate at which each probability converges can produce varying outcomes. It is plausible that using identical λ values may not consistently deliver optimal results.

Maintaining identical λ_r and λ_g values does not necessarily guarantee optimal performance. The rounding regulating factor, λ_r, exhibits higher sensitivity, whereas most of the λ_g values produced satisfactory results at a value of 0.01. If λ_r is not set optimally, determining the appropriate λ_g becomes more challenging.

REFERENCES

[1] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[2] X. Dong, S. Chen, and S. J. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2017, pp. 1–11.

[3] X. Dong and Y. Yang, "Network pruning via transformable architecture search," in *Proc. Adv. Neural Inf. Process. Syst.*, Jan. 2019, pp. 1–12.

[4] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, no. 6, pp. 1789–1819, Jun. 2021.

[5] Y. Choi, M. El-Khamy, and J. Lee, "Towards the limit of network quantization," 2016, *arXiv:1612.01543*.

[6] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave Gaussian quantization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5406–5414.

[7] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.

[8] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021, *arXiv:2106.08295*.

[9] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," 2019, *arXiv:1902.08153*.

[10] C. Xu, J. Yao, Z. Lin, W. Ou, Y. Cao, Z. Wang, and H. Zha, "Alternating multi-bit quantization for recurrent neural networks," 2018, *arXiv:1802.00150*.

[11] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8BERT: Quantized 8Bit BERT," in *Proc. 5th Workshop Energy Efficient Mach. Learn. Cognit. Comput. - NeurIPS, Ed., (EMC2-NIPS)*, Dec. 2019, pp. 36–39.

[12] J. Kim, K. Yoo, and N. Kwak, "Position-based scaled gradient for model quantization and pruning," in *Proc. Adv. Neural Inf. Process. Syst.*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., New York, NY, USA: Curran Associates, Jan. 2020, pp. 20415–20426.

[13] P. Stock, A. Joulin, R. Gribonval, B. Graham, and H. Jégou, "And the bit goes down: Revisiting the quantization of neural networks," 2019, *arXiv:1907.05686*.

[14] P. Wang, Q. Chen, X. He, and J. Cheng, "Towards accurate post-training network quantization via bit-split and stitching," in *Proc. Int. Conf. Mach. Learn.*, vol. 1, Jul. 2020, pp. 9847–9856.

[15] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, "BRECQ: Pushing the limit of post-training quantization by block reconstruction," 2021, *arXiv:2102.05426*.

[16] M. Nagel, R. A. Amjad, M. V. Baalen, C. Louizos, and T. Blankevoort, "Up or down? Adaptive rounding for post-training quantization," in *Proc. Int. Conf. Mach. Learn.*, Jan. 2020, pp. 7197–7206.

[17] Y. Jeon, C. Lee, E. Cho, and Y. Ro, "Mr.BiQ: Post-training non-uniform quantization based on minimizing the reconstruction error," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 12319–12328.

[18] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," 2022, *arXiv:2210.17323*.

[19] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "AWQ: Activation-aware weight quantization for LLM compression and acceleration," 2023, *arXiv:2306.00978*.

[20] S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer, "SqueezeLLM: Dense-and-Sparse quantization," 2023, *arXiv:2306.07629*.

[21] X. Wei, R. Gong, Y. Li, X. Liu, and F. Yu, "QDrop: Randomly dropping quantization for extremely low-bit post-training quantization," 2022, *arXiv:2203.05740*.

[22] D. Zheng, Y. Liu, and L. Li, "Leveraging inter-layer dependency for post-training quantization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 6666–6679.

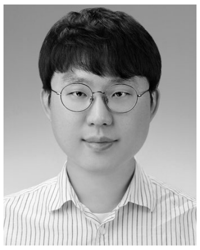
[23] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Jan. 2018, pp. 373–390.

[24] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," 2020, *arXiv:2004.09602*.

[25] J. Faraone, N. Fraser, M. Blott, and P. H. W. Leong, "SYQ: Learning symmetric quantization for efficient deep neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4300–4309.

[26] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, Feb. 2014, pp. 10–14.

[27] M. Nagel, M. V. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1325–1334.
[28] G. Xiao, J. Lin, M. Seznec, J. Demouth, and S. Han, "SmoothQuant: Accurate and efficient post-training quantization for large language models," in *Proc. Int. Conf. Mach. Learn.*, Jan. 2022, pp. 38087–38099.
[29] M. G. D. Nascimento, R. Fawcett, and V. A. Prisacariu, "Dsconv: Efficient convolution operator," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Jun. 2019, pp. 5148–5157.
[30] B. D. Rouhani et al., "Pushing the limits of narrow precision inferencing at cloud scale with Microsoft floating point," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, Nov. 2020, pp. 10271–10281.
[31] Z. Yuan, L. Niu, J. Liu, W. Liu, X. Wang, Y. Shang, G. Sun, Q. Wu, J. Wu, and B. Wu, "RPTQ: Reorder-based post-training quantization for large language models," 2023, *arXiv:2304.01089*.
[32] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, "Improving post training neural quantization: Layer-wise calibration and integer programming," 2020, *arXiv:2006.10518*.
[33] J. Chen, S. Bai, T. Huang, M. Wang, G. Tian, and Y. Liu, "Data-free quantization via mixed-precision compensation without fine-tuning," *Pattern Recognit.*, vol. 143, Nov. 2023, Art. no. 109780.
[34] W. Li, A. Hu, N. Xu, and G. He, "A precision-scalable deep neural network accelerator with activation sparsity exploitation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 1, pp. 263–276, Jan. 2024.
[35] J. Yang, F. Tu, Y. Li, Y. Wang, L. Liu, S. Wei, and S. Yin, "GQNA: Generic quantized DNN accelerator with weight-repetition-aware activation aggregating," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 10, pp. 4069–4082, Oct. 2022.
[36] S. Ryu, H. Kim, W. Yi, E. Kim, Y. Kim, T. Kim, and J.-J. Kim, "BitBlade: Energy-efficient variable bit-precision hardware accelerator for quantized neural networks," *IEEE J. Solid-State Circuits*, vol. 57, no. 6, pp. 1924–1935, Jun. 2022.
[37] A. Finkelstein, U. Almog, and M. Grobman, "Fighting quantization bias with bias," 2019, *arXiv:1906.03193*.
[38] X. Sun, J. Choi, C. Y. Chen, N. Wang, S. Venkataramani, V. Srinivasan, X. Cui, W. Zhang, and K. Gopalakrishnan, "Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, Jan. 2019, pp. 4900–4909.
[39] Z. Yuan, C. Xue, Y. Chen, Q. Wu, and G. Sun, "PTQ4 ViT: Post-training quantization for vision transformers with twin uniform quantization," 2021, *arXiv:2111.12293*.
[40] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," 2017, *arXiv:1712.01312*.
[41] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[42] J. Hyun Lee, J. Kim, S. Jung Kwon, and D. Lee, "FlexRound: Learnable rounding based on element-wise division for post-training quantization," 2023, *arXiv:2306.00317*.
[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[44] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[45] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2815–2823.
[46] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 10425–10433.
[47] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, and F. Wei, "The era of 1-bit LLMs: All large language models are in 1.58 bits," 2024, *arXiv:2402.17764*.
[48] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.
[49] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," 2019, *arXiv:1911.09925*.
[50] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "DLAU: A scalable deep learning accelerator unit on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 3, pp. 513–517, Mar. 2017.
[51] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.



GEUNJAE CHOI received the B.S. and M.S. degrees in electrical engineering and computer science from Seoul National University, in 2009 and 2011, respectively, where he is currently pursuing the Ph.D. degree, with a focus on deep neural network optimization and lightweight design research. Since 2014, he has been with LG Electronics, where he has been involved in designing mobile application processors and deep neural network accelerator hardware.

KAMIN LEE received the B.S. and M.S. degrees in information electronics engineering and electrical engineering from Ewha Womans University, in 2009 and 2011, respectively. She is currently pursuing the Ph.D. degree with Seoul National University. Since 2011, she has been with LG Electronics, where she has been involved in developing gesture recognition and continual learning algorithms. Her research interests include continual, multimodal, and on-device learning.

NOJUN KWAK (Senior Member, IEEE) was born in Seoul, South Korea, in 1974. He received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, in 1997, 1999, and 2003, respectively. From 2003 to 2006, he was with Samsung Electronics, Seoul. In 2006, he joined Seoul National University as a BK21 Assistant Professor. From 2007 to 2013, he was a Faculty Member with the Department of Electrical and Computer Engineering, Ajou University, Suwon, South Korea. Since 2013, he has been with the Graduate School of Convergence Science and Technology, Seoul National University, where he is currently a Professor. His current research interests include feature learning by deep neural networks and their applications in various areas of pattern recognition, computer vision, and image processing.