



CHALMERS
UNIVERSITY OF TECHNOLOGY

PKTQ: Per-Kernel Thresholded Quantization With Size-Constrained Two-Stage Optimization for CNNs

Downloaded from: <https://research.chalmers.se>, 2026-05-18 03:29 UTC

Citation for the original published paper (version of record):

Song, H., Mo, L., Al-Hasan, T. et al (2026). PKTQ: Per-Kernel Thresholded Quantization With Size-Constrained Two-Stage Optimization for CNNs. *Electronics Letters*, 62(1).
<http://dx.doi.org/10.1049/ell2.70562>

N.B. When citing this work, cite the original published paper.

LETTER OPEN ACCESS

PKTQ: Per-Kernel Thresholded Quantization With Size-Constrained Two-Stage Optimization for CNNs

Hengyan Song¹  | Lei Mo¹  | Tamim M. Al-Hasan²  | Minyu Cui³  | Guiyun Liu⁴  | Xiaojun Zhai² 

¹Key Laboratory of Measurement and Control of CSE, Ministry of Education, School of Automation, Southeast University, Nanjing, China | ²School of Computer Science and Electronic Engineering, University of Essex, Colchester, UK | ³Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden | ⁴School of Mechanical and Electrical Engineering, Guangzhou University, Guangzhou, China

Correspondence: Lei Mo (lmo@seu.edu.cn)

Received: 3 October 2025 | **Revised:** 4 December 2025 | **Accepted:** 25 February 2026

ABSTRACT

Convolutional neural networks are challenging to deploy on resource-constrained computing platforms due to their high storage and computational demands. Mixed-precision quantization can mitigate these issues by assigning different bit-widths per layer. However, finding an optimal quantization policy is challenging due to the exponential size of the policy search space and the need for extensive policy evaluations. Additionally, existing works often overlook strict model size constraints and apply per-layer quantization, which ignores intra-kernel variances. We propose a two-stage deep deterministic policy gradient (DDPG) framework. In the first stage, it optimizes accuracy; in the second, it jointly optimizes accuracy and model size with a constraint-aware reward that enforces storage limits during training. On this basis, a per-kernel clipping threshold is introduced to reduce quantization error by adapting the clipping threshold to individual kernel distributions. We perform extensive simulations based on ImageNet with MobileNet-V1/V2 and ResNet-50. The results show that our method outperforms the state-of-the-art baselines under the same compression ratios, achieving minimal accuracy drops with aggressive compression and near-lossless performance at moderate levels.

1 | Introduction

Convolutional neural networks (CNNs), are widely used in multiple scenarios such as computer vision tasks and recommendation systems. However, their deployment in resource-constrained environments remains challenging due to high storage requirements and computational demands. For example, deploying a full-precision model like VGG16 (528 MB) [1] on a smartphone SoC is challenging because limited on-chip memory cannot support its size, which leads to frequent DRAM accesses that increase latency and power consumption.

Quantization technology can mitigate storage and computation bottlenecks by reducing both the model size and the required memory bandwidth, where the full-precision floating-point weights can be represented by reduced-precision integers [2].

To perform CNN quantization, some methods use the same bit-width for all layers, such as [3, 4]. However, a uniform bit-width allocation across all layers ignores the differences among layers, for example, the layer structure. Therefore, uniform bit-width will cause more information loss in the critical layers, leading to significant degradation in model accuracy.

Mixed-precision quantization technology selects an appropriate bit-width for each layer of a CNN, providing system flexibility and adaptivity, compared with fixed-precision quantization. Therefore, mixed-precision quantization can take the characteristics and behaviour of different layers into account [5], when a quantized CNN is applied to a hardware platform. On the other hand, with the recent advances in embedded AI hardware, many platforms support mixed-precision operations, such as the NVIDIA Jetson edge computing platform, which provides

This is an open access article under the terms of the [Creative Commons Attribution](https://creativecommons.org/licenses/by/4.0/) License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2026 The Author(s). *Electronics Letters* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology.

dedicated Tensor Cores to accelerate FP16 and INT8 mixed-precision computation [6]. Therefore, mixed-precision quantized models can be deployed in this resource-constrained platform.

A major challenge in mixed-precision quantization design is efficiently determining the appropriate bit-width for each CNN layer. The search space for bit-width selection grows exponentially with the number of layers, as each layer can be assigned one of multiple candidate bit-widths. For instance, a network with N layers and each layer has M possible bit-widths, an exhaustive search is required to evaluate $\mathcal{O}(M^N)$ combinations [7], which is computationally expensive. In addition, the evaluation of each candidate configuration usually involves re-training or fine-tuning the CNN to estimate its performance, further increasing computational costs.

Some mixed-precision quantization methods have been proposed by existing works, such as [5, 7–11], and the above methods can be generally classified as:

1. **Criterion-based mixed-precision frameworks:** These methods manually set sensitivity evaluation metrics to guide the bit-width allocation for different layers. HAWQ [8] introduces a Hessian-based sensitivity analysis for layer-wise mixed-precision quantization, but requires manual bit-width assignment. HAWQ-V2 [9] automatically allocates bit-width to layers via a Pareto-frontier search method and adopts a new Hessian-trace criterion, improving feasibility. However, this method still relies on heuristic search without a rigorous guarantee of optimality. Heuristic criteria and manual design are not involved in [7]; instead, it provides a mathematically principled optimization formulation, that is, it formulates the mixed-precision quantization as a combinatorial optimization problem.

These criterion-based approaches mainly focus on and rely on manually designed sensitivity metrics, which reduce system adaptivity. In addition, to find a heuristic or optimal solution, it is often required to build a corresponding mathematical formulation, which is changed with constraints such as model size.

2. **RL-based mixed-precision frameworks:** These methods formulate the bit-width optimization problem as a Markov decision process (MDP), where a reinforcement learning (RL) agent sequentially selects the quantization policy for each layer or kernel, and the reward function bridges objectives such as model accuracy and constraints on model size. Compared to criterion-based methods, RL-based approaches are flexible in incorporating multiple optimization objectives into the reward design and can be adapted to different hardware platforms by adjusting the reward function accordingly.

HAQ [5] is a framework that uses RL to search per-layer quantization policies under a single hardware constraint while relying on the measured hardware feedback from the target platform to guide the training process. AutoQ [11] extends HAQ by supporting multiple hardware constraints alongside model accuracy. It employs predictive models for hardware resources and applies per-kernel quantization. ADRL [10], on the other hand, focuses on model compression as a constraint and enhances search efficiency by generating multiple candidate actions at each step of the episode.

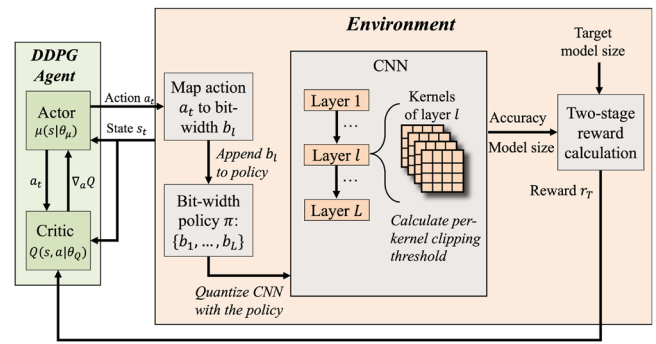


FIGURE 1 | The structure of the proposed PKTQ framework.

Although the above methods consider hardware-related objectives, constraints may be violated during the search process. For example, in HAQ and ADRL, when a learned bit-width configuration violates the target constraint, the quantization policy for the final layers is manually adjusted to compensate. In AutoQ, hardware constraints are weighted terms within the reward function, guiding the RL agent to balance model accuracy with constraint satisfaction. However, this method does not enforce hard constraints, meaning violations may occur during the training process.

Compared to existing approaches, we focus on mixed-precision quantization for CNNs under strict model size constraints imposed by limited storage, to maximize accuracy. Our main contributions are summarized as follows:

- We propose a two-stage DDPG-based framework with per-layer policies to maximize accuracy, while guaranteeing the model size constraint.
- Based on the obtained per-layer policies, we introduce a per-kernel clipping threshold for convolutional layers. This approach offers finer granularity than per-layer quantization and mitigates precision loss.
- We conduct a series of experiments using MobileNetV1/V2 and ResNet-50 on the ImageNet dataset across various compression ratios. The results demonstrate that our method yields higher Top-1 accuracy than existing approaches. Furthermore, we analyse the training dynamics of the agent, illustrating stable convergence in both accuracy and compression ratio.

2 | Mixed-Precision Quantization Approach

Figure 1 illustrates the overall workflow of the proposed PKTQ framework. We model mixed-precision quantization as a DDPG-based two-stage reinforcement learning (RL) problem and solve it to find an optimal quantization policy. The DDPG architecture contains an actor network and a critic network. The actor network predicts the bit-width for each CNN layer. Then, the predicted bit-widths are used as the quantization policy to quantize the CNN model. On this basis, the critic network evaluates the quantization policy by measuring both the accuracy and the model size of the quantized network. Finally, the evaluation results will feed back to update the subsequent

quantization policy. During the training of actor and critic networks, in the first stage, we focus on optimizing accuracy, while in the second stage, we incorporate the model size constraint to ensure the quantized model meets deployment requirements. The details of the PKTQ framework are explained below.

2.1 | Principle of DDPG-Based RL

The RL method consists of an agent interacting with an environment E . The interaction process includes a series of episodes, where each episode contains multiple discrete time steps in which the agent and environment can exchange information. At each time step t , the agent receives an observation x_t , takes an action a_t , and receives a reward r_t from the environment E . This repeated agent–environment interaction within an episode can be modelled as a Markov decision process (MDP), characterized by a state space S and an action space A , and then the behaviour of the agent is governed by a policy $\pi : S \rightarrow A$.

During the interaction process, the agent aims to maximize not only immediate reward r_t , but also long-term return R_t , from action a_t . Let s_t denote the state of step t . To quantify the total benefit in steps $[t, \dots, T]$, where T is the final step, we introduce the return $R_t = \sum_{i=t}^T \gamma^{i-t} r(s_i, a_i)$, indicating the cumulative rewards from initial step t to final step T , discounted by a decay factor γ^{i-t} . The discount factor γ^{i-t} decreases as the time step i moves further from t , which balances near-term gains with long-term effects.

The goal of the agent is to learn a policy π that maximizes the expected return from the start step, that is, $J(\pi) = \mathbb{E}[R_1|\pi]$. To this end, the policy π must be optimized based on interactions between the agent and the environment E . Let $\mathbb{E}[R_t|s_t = s, a_t = a, \pi]$ denote the expected return R_t under the given state s_t , action a_t , where policy π is a set of actions $[a_1, \dots, a_T]$.

2.1.1 | Critic Network

In the DDPG actor-critic framework, the aim of the critic network $Q(s, a|\theta_Q)$ with parameters θ_Q is to estimate the expected return $\mathbb{E}[R_t]$ (i.e., action-value: Q value):

$$Q(s, a|\theta_Q) \approx \mathbb{E}[R_t|s_t = s, a_t = a, \pi], \quad (1)$$

The critic network is trained using the Bellman equation $L(\theta_Q) = \mathbb{E}[(y_t - Q(s_t, a_t|\theta_Q))^2]$ by minimizing the loss between the target Q -value y_t and the estimated Q -value given by the critic network $Q(s_t, a_t|\theta_Q)$, where the target Q -value,

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta_\mu)|\theta_Q), \quad (2)$$

and $r(s_t, a_t)$ is the immediate reward at time step t , $\gamma \in [0, 1]$ is the discount factor controlling the importance of future rewards, $a_{t+1} = \mu(s_{t+1}|\theta_\mu)$ is the action predicted by the actor for the next state s_{t+1} with parameters θ_μ .

2.1.2 | Actor Network

The training objective of the actor network is to maximize the estimated Q -value $Q(s, a|\theta_Q)$ in (1). Since $J(\pi) = \mathbb{E}[R_1|\pi]$ and $R_1 = \sum_{i=1}^T \gamma^{i-1} r(s_i, a_i)$, $J(\pi)$ represents the weighted sum of the estimated Q -value from step 1 to step T . Taking the gradient of $J(\pi)$ with respect to the parameters θ_μ of actor network (i.e., $\nabla_{\theta_\mu} J$), we can maximize $Q(s, a|\theta_Q)$.

Since $Q(s, a|\theta_Q)$ depends on the action a , while a depends on the parameters θ_μ of actor network through $a = \mu(s|\theta_\mu)$, we can apply the chain rule to calculate the gradient $\nabla_{\theta_\mu} J$. Thus, the gradient of $J(\pi)$ with respect to θ_μ is given by

$$\nabla_{\theta_\mu} J \approx \mathbb{E} \left[\nabla_a Q(s, a|\theta_Q) \Big|_{s=s_t, a=\mu(s_t|\theta_\mu)} \nabla_{\theta_\mu} \mu(s|\theta_\mu) \Big|_{s=s_t} \right]. \quad (3)$$

During the training process of actor and critic networks, the critic parameters θ_Q and actor parameters θ_μ are updated iteratively. Once training converges—that is, when the critic network provides stable estimated Q -values $Q(s_t, a_t|\theta_Q)$ and the actor network consistently provides high-value actions a_t —the trained actor network $\mu(s|\theta_\mu)$ can be used to generate the policy π .

2.2 | Modelling for Mixed-Precision Quantization

We use DDPG to learn a per-layer quantization policy π of CNNs, where the environment E , the action a_t , the state s_t (also the observation x_t) are used to represent the CNN, the layer-wise bit-width, and the structural attributes and the historical information of the CNN, respectively. During this process, we introduce a two-stage reward in environment E that reflects the accuracy and model size constraint. Under the given per-layer quantization policy π , we quantize the CNN using per-kernel clipping thresholds, thereby reducing quantization loss. The details are as follows.

2.2.1 | State and Action Definition

For CNN quantization, the environment E is assumed to be fully observable, meaning the state s_t equals the observation x_t , that is, $s_t = x_t$. At each time step t , the agent handles the l -th layer of the CNN. Note that the structures of convolutional and fully connected layers differ in CNNs. Their observations are different as well. The observation for convolutional layers is

$$x_t = (l, c_{\text{in}}, c_{\text{out}}, s_{\text{kernel}}, s_{\text{stride}}, s_{\text{feat}}, n_{\text{params}}, i_{\text{dw}}, a_{t-1}), \quad (4)$$

while the observation for fully connected layers is

$$x_t = (l, h_{\text{in}}, h_{\text{out}}, 0, 0, s_{\text{feat}}, n_{\text{params}}, 0, a_{t-1}). \quad (5)$$

In (4), l denotes the layer index, c_{in} and c_{out} represent the numbers of input and output channels for the convolutional layer, s_{kernel} and s_{stride} are the kernel size and stride for the convolutional layer, s_{feat} is the feature map size, n_{params} is the total number of parameters, i_{dw} is a binary indicator for depthwise convolution, and a_{t-1} denotes the action taken by the agent in the previous step $t-1$.

In (5), h_{in} and h_{out} are the numbers of input and output hidden units for the fully connected layer. Since s_{kernel} , s_{stride} and i_{dw} only reflect the characteristics of convolutional layers, their values are set to 0 in (5). Note that the scales of the elements in (4) and (5) are different. To stabilize the training process, these elements are normalized between [0,1], before being used in the agent network.

Since the observation x_t is used to generate an action a_t that determines the quantization bit-width for layer l , x_t describes the CNN layer from the following three perspectives: (1) l , c_{in} , c_{out} , h_{in} , h_{out} , s_{kernel} , s_{stride} and i_{dw} reflect the role of each layer in CNN; (2) s_{feat} and n_{params} quantify the resource demands of the layer and (3) a_{t-1} encodes the historical policy choices.

Note that the value of a_t is normalized within the range [0,1]. Moreover, DDPG operates in a continuous action domain. Therefore, its output a_t is continuous and bounded by [0,1]. Since the bit-width values of the quantized CNN weights are discrete, to map the action a_t to the desired bit-width value b_l , we perform the following transformation:

$$b_l = \text{round}(b_{\min} - 0.5 + a_t \times (b_{\max} - b_{\min} + 1)), \quad (6)$$

where b_l is bounded by $\{b_{\min}, \dots, b_{\max}\}$, and we set $b_{\min} = 2$ and $b_{\max} = 8$. Equation (6) shows that the mapping process consists of the following three steps: (i) *Scaling*: The action a_t is first scaled to the range $[0, b_{\max} - b_{\min} + 1]$; (ii) *Offset*: A correction factor of $b_{\min} - 0.5$ is then added to adjust the scaling and ensure proper rounding; (iii) *Rounding*: Finally, the value is rounded to the nearest integer to obtain the discrete bit-width b_l .

2.2.2 | Two-Stage DDPG Reward

During the training process of actor and critic networks, we need to take accuracy and model size constraints into account. Therefore, we introduce a two-stage reward r_T . Note that r_T is given by the last step T . In each episode, that is, from step 1 to step T , the agent generates the quantization policy π for all the layers of CNN. Both accuracy and model size can be evaluated only after applying the policy π for all layers to quantize the CNN, and fine-tuning the quantized CNN. At the last step T of each episode, we can get a reward r_T that describes the model accuracy and size; however, the rewards of previous steps are zero, that is, $r_t = 0$ ($1 \leq t < T$). During the training process of DDPG, the final reward r_T is propagated backward to get the target Q-values y_t from step 1 to step $T - 1$ through (2).

This design of DDPG reward r_T aligns with the quantization objective, that is, optimizing the final classification accuracy acc_{quant} , while satisfying the model size constraint $size_{cons}$. Specifically, the reward (7) contains two parts: (i) The difference between the quantized model accuracy acc_{quant} and the original model accuracy acc_{origin} , scaled by a constant λ . This term encourages the agent to reduce the accuracy error to improve model accuracy after quantization; (ii) Size-related penalty P_{size} , depending on the quantized model size $size_{quant}$ and size constraint $size_{cons}$. If $size_{quant}$ exceeds $size_{cons}$, the agent receives a penalty P_{size} , which is proportional to the gap among $size_{quant}$ and $size_{cons}$, and controlled by a hyperparameter β .

To stabilize training, we employ a two-stage reward design based on the episode index $episode$: (i) In the first stage (i.e., $episode < th$, where th is the threshold of stage separation), the reward only considers accuracy $\lambda \times (acc_{quant} - acc_{origin})$, allowing the agent to fully explore the accuracy optimization without considering the model size constraint $size_{cons}$; (ii) In the second stage (i.e., $episode \geq th$), the penalty term P_{size} is introduced, guiding the agent to generate the quantization policy π that satisfies the model size constraint $size_{cons}$.

Because the optimization objectives of model accuracy and model size are often in conflict, the above sequential approach facilitates stable training and yields models suitable for deployment on resource-limited devices. Based on the above analysis, the reward r_T can be formulated as follows:

$$r_T = \begin{cases} \lambda \times (acc_{quant} - acc_{origin}), & episode < th, \\ \lambda \times (acc_{quant} - acc_{origin}) - P_{size}, & episode \geq th. \end{cases} \quad (7)$$

where the size-related penalty P_{size} is given by

$$P_{size} = \begin{cases} 0, & size_{quant} \leq size_{cons}, \\ \beta \times (size_{quant} - size_{cons}), & size_{quant} > size_{cons}. \end{cases} \quad (8)$$

By properly setting the reward coefficients λ and β , any violation of the size constraint can be penalized, thereby enforcing the agent to satisfy the constraint, which allows temporary violations during early training.

2.2.3 | Per-Kernel Clipping Threshold Linear Quantization

When the per-layer quantization policy π is obtained, we next quantize the CNN using per-kernel clipping thresholds. Linear quantization is the process of linearly mapping high-bit floating-point data to low-bit integer data, which can be expressed as follows:

$$\mathbf{w}_{int} = \text{clip}(\text{round}(\mathbf{w}/scale) + z, -2^{b-1}, 2^{b-1} - 1). \quad (9)$$

In (9), \mathbf{w} is the floating-point weight data that needs to be quantized, while \mathbf{w}_{int} is the quantized low-bit integer. z is the zero point, b is the bit-width of the low-bit data, and the quantization scale

$$scale = (w_{\max} - w_{\min}) / (2^b - 1), \quad (10)$$

where w_{\min} and w_{\max} represent the minimum and maximum clipping thresholds, respectively. The function $\text{round}(\cdot)$ is a rounding operation, and the rounding error is within the range $[-1/scale, 1/scale]$. The $\text{clip}(\cdot)$ operation ensures that the quantized values, that is, $\text{round}(\mathbf{w}/scale) + z$, are constrained within the range $[-2^{b-1}, 2^{b-1} - 1]$. Specifically, the floating-point values in \mathbf{w} less than w_{\min} are mapped to -2^{b-1} , and those greater than w_{\max} are mapped to $2^{b-1} - 1$, incurring a clipping error introduced by operation $\text{clip}(\cdot)$.

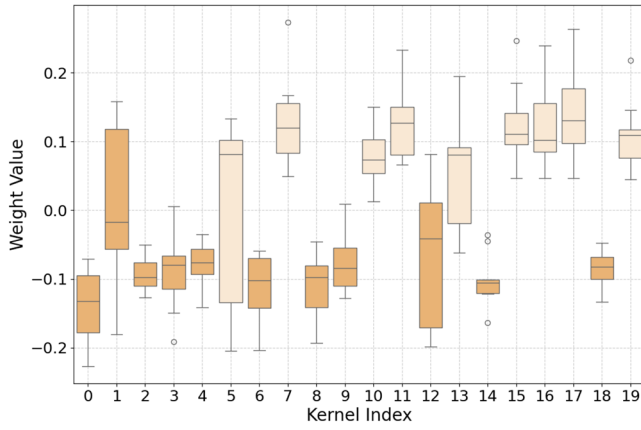


FIGURE 2 | First 20 kernels' weight distribution of layer 'features.4.conv.1.0' in pretrained MobileNetV2.

Note that the quantized low-bit integer \mathbf{w}_{int} can also be mapped back to the floating-point data \mathbf{w}_q through $\mathbf{w}_q = (\mathbf{w}_{\text{int}} - z) \times \text{scale}$. Due to the clipping operation $\text{clip}(\cdot)$ and rounding operation $\text{round}(\cdot)$ in (9), there exists a quantization error between the quantized values \mathbf{w}_q and the original values \mathbf{w} . Since the quantization error is influenced by clipping thresholds w_{max} and w_{min} , according to (9) and (10), the proper values of w_{max} and w_{min} should be selected to reduce the quantization error and improve the accuracy of the model.

In CNNs, different convolutional kernels within a layer often exhibit distinct weight distributions, characterized by different means, variances, and dynamic ranges. Therefore, applying a uniform clipping threshold $[w_{\text{min}}, w_{\text{max}}]$ across all kernels without considering these intrinsic differences, potentially leads to excessive quantization error for some kernels. For example, Figure 2 is the weight distribution difference of the first 20 kernels of the layer 'features.4.conv.1.0' in the pretrained MobileNetV2, which is a 3×3 depthwise convolutional layer with 72 kernels in total. On each box, the central mark indicates the median, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The whiskers extend to the most extreme data points that are not considered outliers, and the outliers are plotted individually using the 'o' symbol. Figure 2 shows that different kernels have very different weight distributions in terms of mean and dynamic range.

Figure 3 illustrates the quantization differences between per-layer and per-kernel clipping thresholds, where $\mathbf{w} = [\mathbf{w}^1, \mathbf{w}^2, \mathbf{w}^3]$ are the weights of a layer including three kernels, and $\min(\mathbf{w})$ and $\max(\mathbf{w})$ are the lower and upper bounds of \mathbf{w} , respectively. If a per-layer threshold $[w_{\text{min}}, w_{\text{max}}]$ is used, as shown in Figure 3a, for \mathbf{w}^1 , the weights in the range $[\min(\mathbf{w}^1), w_{\text{min}}]$ are clipped, while the weights in the range $[w_{\text{min}}, \max(\mathbf{w}^1)]$ are mapped to the integers $\{-2^{b-1}, -2^{b-1} + 1, -2^{b-1} + 2\}$. In contrast, Figure 3b shows the mapping with per-kernel clipping thresholds. \mathbf{w}^1 can be mapped to the whole range of integers, that is, $\{-2^{b-1}, \dots, 2^{b-1} - 1\}$, which is wider than the range $\{-2^{b-1}, -2^{b-1} + 1, -2^{b-1} + 2\}$ given by the per-layer clipping threshold, that is, we can reduce quantization error and improve overall accuracy. Regarding hardware deployment overhead, our per-kernel scheme shares a

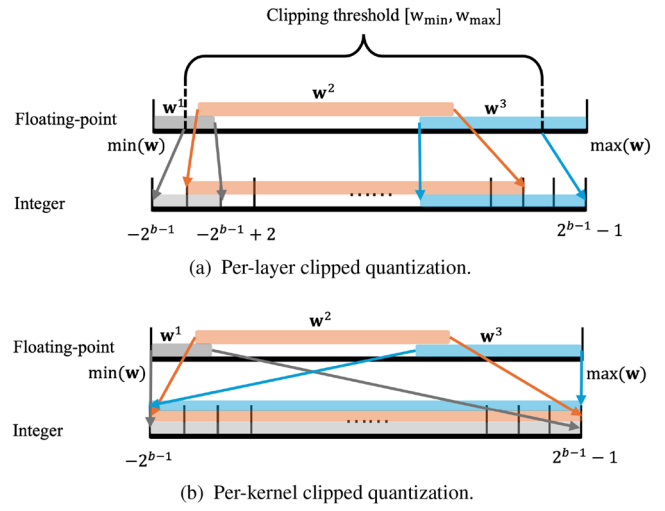


FIGURE 3 | Comparison of per-layer and per-kernel clipped quantization.

single bit-width per layer, similar to per-layer quantization. The only additional overhead is a $c_{\text{out}} \times c_{\text{in}}$ table of scales/zero-points.

To reduce quantization error, we adopt the per-kernel clipping threshold selection strategy. Note that across different convolutional layers, the number of weight samples per kernel varies. In addition, convolutional layers can generally be divided into standard and depth-wise convolutional layers. Therefore, we have

- Standard convolutional layer: Since the number of weights per kernel is typically hundreds, we determine the clipping threshold $[w_{\text{min}}^k, w_{\text{max}}^k]$ of the k^{th} kernel by minimizing Kullback–Leibler (KL) divergence $D_{\text{KL}}(\cdot || \cdot)$ between original and quantized distributions:

$$[w_{\text{min}}^k, w_{\text{max}}^k] = \arg \min_{t_{\text{min}}^k, t_{\text{max}}^k} D_{\text{KL}}(P_{\mathbf{w}^k} || Q_{\mathbf{w}^k}^{(t_{\text{min}}^k, t_{\text{max}}^k)}). \quad (11)$$

In (11), $P_{\mathbf{w}^k}$ is the original weight distribution histogram of the k^{th} kernel, and $Q_{\mathbf{w}^k}^{(t_{\text{min}}^k, t_{\text{max}}^k)}$ is the histogram after the quantization using the clipping threshold $[t_{\text{min}}^k, t_{\text{max}}^k]$, and finally we get $w_{\text{min}}^k = t_{\text{min}}^k$ and $w_{\text{max}}^k = t_{\text{max}}^k$. The KL-divergence method [12] finds an optimal trade-off between removing outliers (reducing rounding error) and preserving important distribution characteristics (reducing clipping error), and is effective only when sufficient samples are available to build a stable histogram.

- Depthwise convolutional layer: Each depthwise kernel typically contains only $K_h \times K_w$ elements, where K_h and K_w are the height and width of kernel, respectively. Note that the values of K_h and K_w are usually small, for example, $3 \times 3 = 9$ in 'features.4.conv.1.0,' which is insufficient to estimate a histogram-based KL divergence reliably. Therefore, we directly adopt the simple min–max scheme to determine the clipping thresholds for depthwise kernels. For the k^{th} kernel with the weights \mathbf{w}^k , we set $w_{\text{min}}^k = \min(\mathbf{w}^k)$ and $w_{\text{max}}^k = \max(\mathbf{w}^k)$. The above min-max method ensures that all values in the kernel map exactly into the available quantization range, without introducing clipping errors.

TABLE 1 | CNN Top-1 accuracy on ImageNet dataset under different methods and compression ratios.

Model	Method	Policy	Top-1		Top-1	
			(Full)	Comp.	(Quant)	(Drop)
MobileNet-V1 [13]	DC [3]	W2	70.90	0.068	37.62	-33.28
	HAQ [5]	MP	70.90	0.068	57.14	-13.76
	Ours	MP	71.38	0.067	61.68	-9.70
	DC [3]	W3	70.90	0.099	65.93	-4.97
	HAQ [5]	MP	70.90	0.098	67.66	-3.24
	Ours	MP	71.38	0.097	68.35	-3.03
	DC [3]	W4	70.90	0.130	71.14	+0.24
	HAQ [5]	MP	70.90	0.128	71.74	+0.84
	Ours	MP	71.38	0.126	72.25	+0.87
MobileNet-V2 [14]	DC [3]	W2	71.87	0.072	58.07	-13.8
	HAQ [5]	MP	71.87	0.071	66.75	-5.12
	Ours	MP	72.34	0.071	68.38	-3.96
	DC [3]	W3	71.87	0.103	68.00	-3.87
	HAQ [5]	MP	71.87	0.103	70.90	-0.97
	Ours	MP	72.34	0.103	71.79	-0.55
	DC [3]	W4	71.87	0.134	71.24	-0.63
	HAQ [5]	MP	71.87	0.133	71.47	-0.40
	Ours	MP	72.34	0.133	72.21	-0.13
ResNet50 [15]	DC [3]	W2	76.15	0.065	68.95	-7.20
	HAQ [5]	MP	76.15	0.065	70.63	-5.52
	Ours	MP	76.64	0.065	71.47	-5.17
	AutoQ [11]	MP/MP	74.80	0.097	72.51	-2.29
	DC [3]	W3	76.15	0.096	75.10	-1.05
	HAQ [5]	MP	76.15	0.095	75.30	-0.85
	Ours	MP	76.64	0.094	75.92	-0.72
	OMSE [4]	W4	77.72	0.126	74.98	-2.74
	DC [3]	W4	76.15	0.127	76.15	+0.00
	HAQ [5]	MP	76.15	0.125	76.14	-0.01
	Ours	MP	76.64	0.124	76.63	-0.01

3 | Experimental Results

We evaluate the model performance on the ImageNet dataset, focusing on quantizing CNN architectures, including MobileNet-V1 [13], MobileNet-V2 [14], and ResNet50 [15]. Compared with MobileNet-V1, which has ‘depthwise–pointwise’ blocks, MobileNet-V2 adopts ‘pointwise–depthwise–pointwise’ blocks. The accuracy of the quantized model is measured via simulated quantization on a server with an A100 GPU. In the following section, we first evaluate the accuracy of the quantized model. Then, we analyse the characteristics of the proposed PKTQ framework.

3.0.1 | Accuracy Evaluation of Quantized Model

Table 1 compares the Top-1 classification accuracy of three CNNs, that is, MobileNet-V1, MobileNet-V2 and ResNet50, on

the ImageNet dataset under different model compression ratios (Comp.) and quantization methods, which include: DC [3], HAQ [5], AutoQ [11], OMSE [4], and our proposed method. In addition, ‘Policy’ represents the quantization strategy (e.g., W2–W4 indicate weight bit-width levels; MP indicates mixed-precision policies). ‘Top-1 (Full)’ is the Top-1 accuracy (%) of the uncompressed baseline model. ‘Comp.’ denotes the model size compression ratio, defined by $size_{\text{quant}}/size_{\text{full}}$, where $size_{\text{quant}}$ and $size_{\text{full}}$ are the storage size of the quantized model and the full-precision model, respectively. ‘Top-1 (Quant)’ is the accuracy after quantization. ‘Top-1 (Drop)’ is the gap between ‘Top-1 (Quant)’ and ‘Top-1 (Full)’, where the negative and positive values represent accuracy loss and accuracy improvement, respectively. The improvement in accuracy is due to our results obtained after 30 epochs of finetuning under the optimal quantization policy.

From Table 1, we can see that across MobileNet-V1 and MobileNet-V2 scenarios, our method achieves higher Top-1 accuracy compared with DC and HAQ methods under similar compression ratios, as our ‘Top-1 drop’ value is larger than others. For instance, in MobileNet-V1 with W2 equivalent compression ratio, our method achieves 61.68% ‘Top-1 (Quant)’ accuracy and -9.70% ‘Top-1 drop’ loss, outperforming DC (37.62% and -33.28%) and HAQ (57.14% and -13.76%) methods. With W3 and W4 equivalent compression ratios, we can get similar results. Note that the value of ‘Top-1 drop’ is positive under the W4 equivalent compression ratio. This is because fine-tuning is performed after quantization, thereby improving the classification accuracy of MobileNet-V1 on the ImageNet dataset.

For MobileNet-V2, our approach also shows the robustness of model accuracy. Under moderate compression (W3 and W4), our method yields a -0.55% ‘Top-1 (Drop);’ even under the highest compression (W2), the ‘Top-1 (Drop)’ remains -3.96% , compared with -13.8% of DC, indicating our quantization scheme effectively mitigates precision loss.

For ResNet50, our method outperforms the state-of-the-art (SoA) techniques, for example, DC, HAQ, AutoQ, and OMSE, regarding model accuracy. Specifically, at aggressive compression (i.e., W2), our ‘Top-1 (Drop)’ is -5.17% , which is smaller than -7.20% of DC and -5.52% of HAQ. At moderate compression W3, our drop further reduces to -0.72% , and for W4, our approach virtually eliminates performance degradation (-0.01% of drop), achieving better performances than the SoA methods such as OMSE and HAQ.

The results summarized in Table 1 show that the proposed method meets the desired system requirements, and outperforms the SoA methods in terms of model accuracy under the same compression ratios, across both lightweight networks (MobileNet-V1 and MobileNet-V2) and larger architectures (ResNet50). In addition, our method can minimize accuracy drops under aggressive compression (W2), with near-lossless performance at moderate compression levels. The reasons why our method can keep model accuracy are that:

- *Mixed-precision bit-width allocation*, which optimally assigns bit-widths at multiple structural levels of the network.
- *Fine-grained per-kernel quantization clipping threshold*, which enables precise scaling for each convolution kernel to minimize quantization error.
- *Automatic search under compression constraints*, which jointly optimizes compression ratio and quantization settings to satisfy size limits while preserving accuracy. In contrast, HAQ does not consider model size during quantization policy search; instead, if a policy violates the size constraint, bit-widths are reduced manually.

3.0.2 | Characteristics Analysis of PKTQ Framework

In Figure 4, we compare the learning behaviours of our two-stage PKTQ method (with stage thresholds $th = \{0, 100, 200\}$)

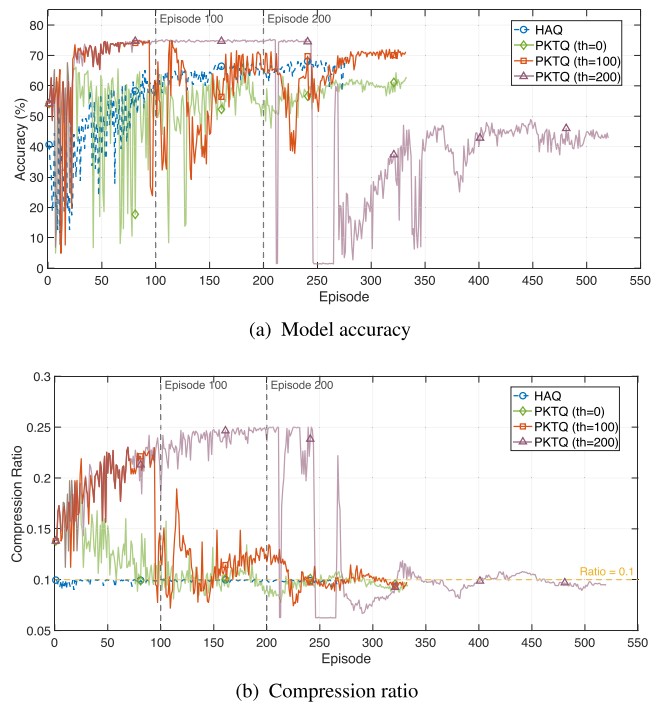


FIGURE 4 | HAQ and PKTQ’s model accuracy and compression ratio of MobileNet-V2 with model size constraint during DDPG training process.

and HAQ [5] during the DDPG-based quantization policy search for MobileNet-V2, and analyse the accuracy (i.e., ‘Top-1 (Quant)’ in Table 1) and the compression ratio (‘Comp.’, i.e., $size_{quant}/size_{full}$) throughout training process until convergence.

In Figure 4a, the accuracy of HAQ increases steadily and remains relatively stable after reaching around 70%. In contrast, the accuracy of PKTQ grows smoothly during the first stage, drops when entering the second stage, and eventually converges. The convergence and the accuracy of PKTQ are sensitive to the choice of th parameter: when the value of th is small (e.g., $th = 0$), PKTQ converges quickly (333 episodes) but accuracy only reaches around 61%; when the value of th is large (e.g., $th = 200$), convergence becomes slow (520 episodes), and the accuracy drops to around 43%; when $th = 100$, the convergence is fast (332 episodes), the accuracy can increase to 72%. In Figure 4b, the compression ratio PKTQ, which reflects the model size, increases during the first stage, then decreases after the model size penalty P_{size} is added to the reward, and finally stabilizes below 0.1, whereas the compression ratio of HAQ remains close to 0.1 throughout the entire process.

The results show that PKTQ’s two-stage design prevents early accuracy loss and enables gradual, stable convergence toward the target compression ratio. In contrast, HAQ does not optimize the model size constraint during training; instead, it manually adjusts the policy in each episode to match the target model size, satisfying the compression ratio requirement, but often degrades accuracy due to ignoring layer-specific characteristics.

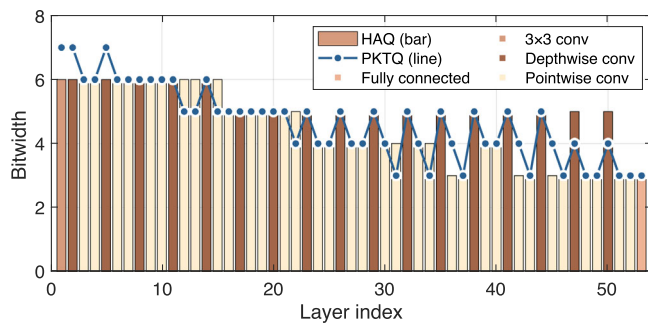


FIGURE 5 | Layer bits comparison of HAQ and PKTQ under identical compression ratio constraint for MobileNet-V2.

3.0.3 | Layer Bit-Width Strategy Evaluation

In Figure 5, we compare HAQ and PKTQ's layer bit-width strategy for MobileNet-V2 under the same compression ratio constraint. These two methods share the property of assigning higher bit-widths to depthwise convolution layers and lower bit-widths to pointwise convolution layers, since depthwise layers contain relatively few parameters, that is, the increase of their bit-width has only a minor impact on the overall model size, whereas pointwise layers are much more parameter-heavy. Compared with HAQ, PKTQ assigns higher bit-widths to the initial layers (layers 1, 2 and 5). This is because these early layers have few parameters but operate directly on raw input features with large dynamic ranges, making them more sensitive to quantization noise. This demonstrates that the two-stage design of PKTQ allows it to allocate bits more efficiently across the network.

4 | Conclusions

In this paper, we presented a DDPG-based framework for mixed-precision quantization of CNNs under the strict model size constraint. By adopting a two-stage optimization strategy—initial accuracy maximization followed by model accuracy and model size joint optimization—we improved policy training stability and avoided performance degradation. On this basis, the proposed per-kernel clipping threshold further reduces the quantization error by making the clipping threshold of each kernel adaptive to its weight distribution. Comprehensive experiments based on MobileNet-V1, MobileNet-V2, and ResNet-50 over the ImageNet dataset demonstrated that our method outperforms existing quantization frameworks in terms of the minimization of accuracy degradation, particularly under high compression ratios. The results also show that our approach strictly enforces the model size constraint during the search for a quantization policy without relying on manual adjustments, ensuring deployment feasibility and adaptivity on resource-constrained edge computing devices.

Author Contributions

Hengyan Song: methodology. **Lei Mo:** conceptualization. **Tamim M. Al – Hasan:** resources. **Minyu Cui:** investigation. **Guiyun Liu:** software. **Xiaojun Zhai:** methodology.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

Data derived from public domain resources.

References

1. K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations (ICLR)*, 2015), 1–14.
2. Y. Li, R. Gong, X. Tan, et al., "BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction," in *International Conference on Learning Representations (OpenReview.net)*, 2021), 1–16.
3. S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks With Pruning, Trained Quantization and Huffman Coding," in *International Conference on Learning Representations (OpenReview.net)*, 2016), 1–14.
4. E. Kravchik, F. Yang, P. Kisilev, and Y. Choukroun, "Low-Bit Quantization of Neural Networks for Efficient Inference," in *IEEE/CVF International Conference on Computer Vision Workshops (IEEE)*, 2019), 3009–3018.
5. K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-Aware Automated Quantization With Mixed Precision," in *IEEE Conference on Computer Vision and Pattern Recognition (IEEE)*, 2019), 8612–8620.
6. *NVIDIA Jetson Platform for Edge AI* (NVIDIA Corporation, 2025), <https://developer.nvidia.com/embedded-computing>.
7. W. Chen, P. Wang, and J. Cheng, "Towards Mixed-Precision Quantization of Neural Networks via Constrained Optimization," in *IEEE/CVF International Conference on Computer Vision (IEEE)*, 2021), 5330–5339.
8. Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "HAWQ: Hessian Aware Quantization of Neural Networks With Mixed-Precision," in *IEEE/CVF International Conference on Computer Vision (IEEE)*, 2019), 293–302.
9. Z. Dong, Z. Yao, D. Arfeen, A. Gholami, M. W. Mahoney, and K. Keutzer, "HAWQ-V2: Hessian Aware Trace-Weighted Quantization of Neural Networks," in *International Conference on Neural Information Processing Systems* (Curran Associates, Inc., 2020), 18518–18529.
10. L. Ning, G. Chen, W. Zhang, and X. Shen, "Simple Augmentation Goes a Long Way: ADRL for DNN Quantization," in *Proceedings of the International Conference on Learning Representations (OpenReview.net)*, 2020), 1–12.
11. Q. Lou, F. Guo, M. Kim, L. Liu, and L. Jiang, "AutoQ: Automated Kernel-Wise Neural Network Quantization," in *Proceedings of the International Conference on Learning Representations (OpenReview.net)*, 2020), 1–11.
12. S. R. Jain, A. Gural, M. Wu, and C. H. Dick, "Trained Quantization Thresholds for Accurate and Efficient Fixed-Point Inference of Deep Neural Networks," *Machine Learning and Systems Conference 2* (2020): 1–17.
13. 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," preprint, arXiv, April 17, 2017, <https://arxiv.org/abs/1704.04861>.
14. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (IEEE)*, 2018), 4510–4520.
15. K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (IEEE)*, 2016), 770–778.