



# Sensitivity-Aware Post-training Quantization for Deep Neural Networks

Zekang Zheng<sup>1</sup>, Haokun Li<sup>1</sup>, Yaofu Chen<sup>1</sup>(✉) , Mingkui Tan<sup>1,2,3</sup> ,  
and Qing Du<sup>1</sup>

<sup>1</sup> South China University of Technology, Guangzhou, China  
chenyaofu@scut.edu.cn

<sup>2</sup> Pazhou Laboratory, Guangzhou, China

<sup>3</sup> Key Laboratory of Big Data and Intelligent Robot, Ministry of Education,  
Guangzhou, China

**Abstract.** Model quantization reduces neural network parameter precision to achieve compression, but often compromises accuracy. Existing post-training quantization (PTQ) methods employ iterative parameter updates to preserve accuracy under high compression ratios, incurring significant computational complexity and resource overhead, which limits applicability in resource-constrained edge computing and real-time inference scenarios. This paper proposes an efficient PTQ method guided by parameter sensitivity analysis. The approach prioritizes quantization of high-sensitivity parameters, leveraging unquantized low-sensitivity parameters to compensate for quantization errors, thereby mitigating accuracy degradation. Furthermore, by exploiting column-wise clustering of parameter sensitivity, the method introduces a row-parallel quantization framework with a globally shared inverse Hessian matrix update mechanism, reducing computational complexity by an order of magnitude. Experimental results on ResNet-50 and YOLOv5s demonstrate a 20–200-fold quantization speedup over the Optimal Brain Quantization baseline, with mean accuracy loss below 0.3%, confirming the method’s efficacy in balancing efficiency and accuracy.

**Keywords:** Model Quantization · Post-Training Quantization · Taylor approximation · Hessian Matrix · Sensitivity Ranking

## 1 Introduction

The widespread application of deep learning models in domains such as computer vision [10, 18] and natural language processing [3, 17, 24, 25] has led to a continual increase in their scale and computational complexity. This poses significant challenges for deploying these models on resource-constrained devices. Model quantization, a crucial model compression technique, reduces the numerical precision of model parameters and activations. This not only significantly diminishes model storage requirements and computational overhead but also enables

efficient model deployment and inference. However, the quantization process, being inherently an approximation, inevitably leads to performance degradation. Consequently, achieving efficient and rapid quantization while preserving model accuracy has become a prominent research focus of model compression.

Model quantization methods are generally categorized into two types: Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). While QAT methods can achieve higher quantization accuracy, they complicate and prolong the training process due to the introduction of quantization operations during training. In contrast, PTQ methods offer greater flexibility, obviating the need for model retraining and allowing direct quantization of pre-trained models, which facilitates deployment in practical applications. Nevertheless, PTQ methods often encounter the issue of post-quantization accuracy degradation. Thus, designing PTQ algorithms that are both efficient and maintain high accuracy presents a significant contemporary challenge. Most existing PTQ methods are based on a layer-wise quantization approach, where each layer is quantized independently. For instance, AdaRound [22] employs gradient optimization to learn rounding strategies, while the classical Optimal Brain Surgeon (OBS) framework [14] utilizes Taylor expansion to minimize layer-wise quantization error. However, these methods still exhibit limitations such as considerable accuracy loss, high computational overhead, and extended quantization times. Particularly in edge computing scenarios, where model updates are frequent, the efficiency of the quantization process is paramount. Although methods like Optimal Brain Quantizer (OBQ) [7] demonstrate good accuracy, their row-wise parameter update scheme limits parallelism, leading to prolonged quantization times and rendering them unsuitable for the real-time demands of edge environments.

To address the aforementioned challenges, this paper proposes a sensitivity-guided efficient post-training quantization algorithm. The core idea is to first resort the columns of the parameter matrix and subsequently perform column-wise quantization based on this sorted order. To circumvent the problem of repetitive Hessian matrix computations encountered in existing methods, we introduce a parallel parameter quantization algorithm operating along the row dimension. This design allows all rows to share and update a single Hessian matrix (or its inverse, more specifically), significantly reducing computational overhead. Furthermore, we observe that values exhibiting large quantization errors within the model often follow a long-tail distribution. Based on this observation, we propose a column-wise quantization algorithm under the pre-sorted parameter matrix. By prioritizing the quantization of columns associated with larger errors, we can more effectively leverage the compensation mechanism inherent in OBS-like algorithms [14], thereby enhancing overall quantization accuracy. By minimizing unnecessary computations and prioritizing more sensitive parameters, our method achieves an effective balance between quantization efficiency and accuracy. The proposed method requires no retraining and completes the quantization process in a relatively short time, making it suitable for scenarios with stringent real-time requirements, such as edge computing.

**Contributions:** 1) We designed a sensitivity-guided model parameter quantization strategy, which effectively mitigates accuracy loss during the quantization process. 2) We proposed a parallel parameter quantization algorithm along the row dimension. By establishing a shared inverse Hessian update mechanism, parameter quantization operations are executed in parallel along the row dimension, significantly reducing computational overhead and substantially decreasing quantization time. 3) Extensive experimental results show that our method significantly reduces quantization time and memory footprint across various models, achieving nearly lossless accuracy compared to current SoTA methods.

## 2 Related Work

Model quantization, a critical technique for neural network compression, aims to map weights and activations, typically represented as floating-point numbers, to lower-precision discrete values (e.g., fixed-point integers), thereby significantly reducing memory footprint, computational latency, and power consumption [8]. This technique is vital for deploying increasingly large and complex deep learning models in resource-constrained environments, such as edge devices and mobile platforms. Empirical evidence shows that converting floating-point representations to 4-bit or 8-bit fixed-point integers can achieve 4x to 8x reductions in memory and latency, with theoretical reductions up to 16x, while striving to preserve the model’s original accuracy.

**Post-Training Quantization (PTQ)** [1, 4, 12, 15, 20, 22] offers an efficient method to convert pre-trained floating-point models to low-precision formats. PTQ primarily involves quantizing the weights and activations of a trained model, providing significant advantages for rapid deployment on resource constrained devices by reducing memory usage and computational demands. A common PTQ strategy is to compute the minimum and maximum values of model parameters to determine clipping ranges [12]. This approach effectively reduces the data representation range while maintaining acceptable accuracy. Additionally, a small set of unlabeled data can be used as a calibration set [11, 12] to fine-tune quantization parameters. In practice, PTQ typically maintains good model performance at higher bit widths (e.g., 8 bits) with minimal accuracy loss. However, significant performance degradation may occur at lower bit widths. To address accuracy loss in PTQ, researchers have proposed several strategies, including correcting biases in the mean and variance of quantized weights [1, 21] to better align with the model’s requirements, balancing weight ranges across layers or channels to reduce cumulative quantization errors, and optimizing the L2 distance between quantized and floating-point tensors [4]. Techniques such as outlier channel splitting [28] also mitigate the impact of anomalous channels. Recent advancements, such as output reconstruction-based PTQ methods like AdaRound [22] and AdaQuant [12], have emerged as prominent trends, enhancing PTQ accuracy and stability through output optimization.

**Quantization-Aware Training (QAT)** [13, 19, 23, 26] introduces pseudo-quantization operations during the training or fine-tuning phase, simulating

the effects of quantization on forward propagation and gradient computation. QAT involves retraining on the training dataset using gradient descent to adjust model parameters, mitigating accuracy loss due to quantization. Although QAT requires more computational resources and time compared to PTQ, it significantly improves prediction accuracy at lower quantization bit widths, making it indispensable for high-accuracy applications. During QAT, floating-point computations are used for forward and backward propagation to ensure accuracy, while pseudo-quantization operators are inserted after gradient updates to simulate quantization effects. Handling non-differentiable operations, such as rounding and clipping, during backpropagation is a core challenge. To address this, researchers have proposed solutions like designing quantization operations with smooth, non-zero derivatives (e.g., LQ-Nets [27] and DSQ [9]). However, the most widely adopted approach is the Straight-Through Estimator (STE) [2], which approximates non-differentiable operators with an identity function. Despite its simplicity, STE has proven effective in numerous experiments and applications, remaining a cornerstone of QAT.

### 3 Problem Definition and Background

**Layer-Wise Compression Problem.** Let  $f_\ell(X_\ell, W_\ell)$  be a layer, where  $W_\ell$  represents the weights and  $X_\ell$  denotes the input. The goal of layer-wise compression is to find a compressed version of the weights,  $\hat{W}_\ell$ , that performs as closely as possible to the original weights  $W_\ell$ . Specifically, the compressed weights  $\hat{W}_\ell$  should minimize the expected change in layer output, as measured by a loss function  $\mathcal{L}$ , while satisfying a general compression constraint  $C(\hat{W}_\ell) > C$ , where  $C(\cdot)$  is tailored to the type of compression:

$$\arg \min_{\hat{W}_\ell} E \left( f_\ell(X_\ell, W_\ell), f_\ell(X_\ell, \hat{W}_\ell) \right), \quad \text{subject to } C(\hat{W}_\ell) > C. \quad (1)$$

The weights  $W_\ell$  form a  $d_{\text{row}} \times d_{\text{col}}$  matrix (for convolutional layers,  $d_{\text{col}}$  is the total number of weights in a single filter), the input  $X_\ell$  has dimensions  $d_{\text{col}} \times N$ , the function  $f_\ell(X_\ell, W_\ell)$  is defined as  $W_\ell X_\ell$ , and the error function  $E$  is defined as  $\|f_\ell(X_\ell, W_\ell) - f_\ell(X_\ell, \hat{W}_\ell)\|_2^2$ . Thus, the quantization objective becomes:

$$\arg \min_{\hat{W}_\ell} \|W_\ell X_\ell - \hat{W}_\ell X_\ell\|_2^2, \quad \text{subject to } C(\hat{W}_\ell) > C. \quad (2)$$

**Optimal Brain Surgeon Method [6].** By approximating the target function  $E$  using a Taylor series expansion, a perturbation  $\delta \mathbf{w}$  in the parameter vector induces the following change in the error function. For a neural network, the change in error can be expressed via Taylor expansion as:

$$\delta E = \left( \frac{\partial E}{\partial \mathbf{w}} \right)^T \cdot \delta \mathbf{w} + \frac{1}{2} \delta \mathbf{w}^T \cdot \mathbf{H} \cdot \delta \mathbf{w} + \mathcal{O}(\|\delta \mathbf{w}\|^3), \quad (3)$$

where  $\frac{\partial E}{\partial \mathbf{w}}$  is the first-order derivative (gradient) of the error function  $E$  with respect to the weights  $\mathbf{w}$ ,  $\mathbf{H}$  is the Hessian matrix containing all second-order

derivatives of  $E$  with respect to  $\mathbf{w}$ ,  $\delta\mathbf{w}$  represents the weight perturbation, and  $\mathcal{O}(\|\delta\mathbf{w}\|^3)$  denotes higher-order terms that are neglected. For a neural network trained to a local minimum, the first-order derivative is zero, so the first term is omitted; higher-order terms are also neglected, leaving only the second-order term. Thus, the Taylor expansion simplifies to:

$$\delta E = \frac{1}{2} \delta\mathbf{w}^T \cdot \mathbf{H} \cdot \delta\mathbf{w}. \quad (4)$$

To minimize the error increase after setting a weight to zero, the following constraint is introduced:

$$\mathbf{e}_q^T \cdot \delta\mathbf{w} + w_q = 0, \quad (5)$$

where  $\mathbf{e}_q$  is a unit vector in the weight space corresponding to the scalar weight  $w_q$ , which is the weight to be set to zero. Thus, the optimization objective becomes:

$$\min_q \left\{ \min_{\delta\mathbf{w}} \left( \frac{1}{2} \delta\mathbf{w}^T \cdot \mathbf{H} \cdot \delta\mathbf{w} \right) \mid \mathbf{e}_q^T \cdot \delta\mathbf{w} + w_q = 0 \right\}. \quad (6)$$

To solve this optimization problem, a Lagrangian function is constructed:

$$\mathcal{L} = \frac{1}{2} \delta\mathbf{w}^T \cdot \mathbf{H} \cdot \delta\mathbf{w} + \lambda (\mathbf{e}_q^T \cdot \delta\mathbf{w} + w_q), \quad (7)$$

where  $\lambda$  is the Lagrange multiplier. By taking derivatives with respect to  $\delta\mathbf{w}$  and  $\lambda$  and combining with the constraint, the following solution is obtained:

$$\delta\mathbf{w} = -\frac{w_q}{[\mathbf{H}^{-1}]_{qq}} \mathbf{H}^{-1} \cdot \mathbf{e}_q, \quad (8)$$

and the error increase after setting the weight to zero is:

$$L_q = \frac{w_q^2}{2[\mathbf{H}^{-1}]_{qq}}. \quad (9)$$

Here,  $L_q$  is referred to as the sensitivity of weight  $q$ , representing the error increase introduced by quantizing that weight. A more sensitive weight results in a larger error after quantization. Thus, the quantization process focuses on minimizing the growth of  $L_q$  while quantizing all parameters.

**The Optimal Brain Quantizer Method [7].** This method is extended to the quantization domain. Suppose the weights of a layer are to be quantized on a fixed grid with width  $\Delta$  while minimizing the loss. To map OBS to quantization, the Lagrangian constraint in Eqn. 7 is set to  $(\text{quant}(w_p) - w_p)$ , where  $\text{quant}(w_p)$  is the quantized weight after rounding.

With the update for the remaining parameters given by:

$$\delta w_p = -\frac{w_p - \text{quant}(w_p)}{[\mathbf{H}^{-1}]_{pp}} \cdot \mathbf{H}^{-1}_{:,p}. \quad (10)$$

The introduced quantization error is:

$$L_q = \frac{(\text{quant}(w_p) - w_p)^2}{2[\mathbf{H}^{-1}]_{qq}}. \quad (11)$$

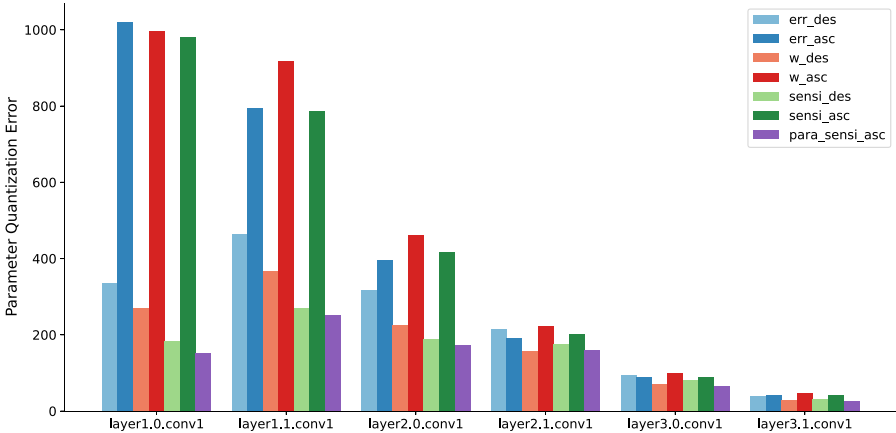


Fig. 1. Quantization error under various sorting strategies.

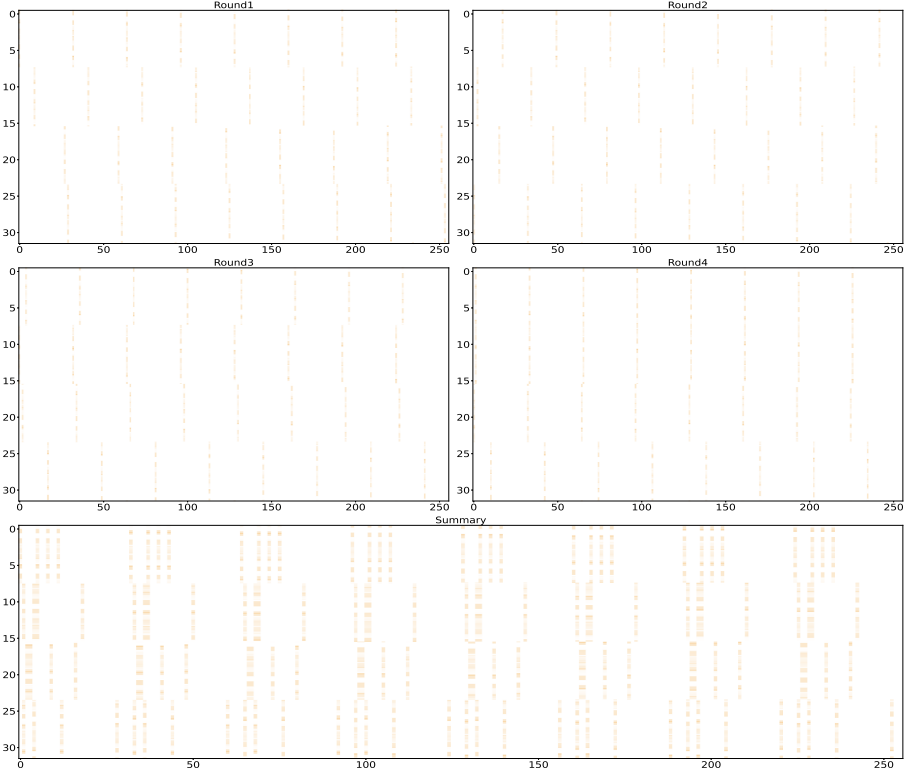
## 4 Sensitivity Guided Post-training Quantization

### 4.1 Parameter Quantization Guided by Sensitivity Ranking

Adapting model pruning techniques, such as quantizing parameters row-wise in ascending order of pre-computed sensitivity  $L_q$ , is suboptimal for quantization. Pruning removes low-sensitivity parameters with minimal impact, but quantization processes all parameters within a layer. Quantizing in ascending  $L_q$  order, which prioritizes low-sensitivity parameters, defers critical high-sensitivity parameters to later stages. In Fig. 1, this increases quantization error, as fixed low-sensitivity parameters limit compensation for errors from high-sensitivity ones. The figure uses colors to represent quantization orders, with prefixes (*err*, *w*, *sensi*) denoting sorting by quantization error, parameter norm, or sensitivity  $L_q$ , and suffixes (*des*, *asc*) indicating descending or ascending order.

To address this, we propose a sensitivity-guided quantization strategy that prioritizes parameters with the highest  $L_q$ . Quantizing high-sensitivity parameters first introduces primary errors early, when many unquantized low-sensitivity parameters remain. Per Eqn. 10, these low-sensitivity parameters collectively compensate for larger errors, unlike ascending quantization, which introduces errors late with limited adjustment capacity. Figure 1 shows that descending  $L_q$  sorting significantly reduces layer-wise quantization error compared to other strategies, with  $L_q$ -based descending order performing best.

The proposed algorithm operates as follows: for a layer, compute  $L_q$  for all parameters using Taylor expansion, sort them in descending  $L_q$  order, and quantize sequentially. This approach mitigates the limitations of ascending quantization by leveraging low-sensitivity parameters for error compensation, achieving lower global quantization error and better model performance.



**Fig. 2.** Row-wise quantization positions on ResNet18 (layer 1.0.conv1).

## 4.2 Row-Parallel Efficient Quantization Method

Building on the analysis above, we design a quantization strategy inspired by Optimal Brain Quantization (OBQ). It processes weight matrix  $W$  row-wise, quantizing parameters based on Hessian-derived sensitivity. After quantizing  $w_{ij}$  to  $q(w_{ij})$ , OBQ updates unquantized parameters and the inverse Hessian  $H^{-1}$ :

$$W \leftarrow W - H^{-1} \cdot \frac{1}{[H^{-1}]_{kk}} \cdot \delta_k \quad (12)$$

$$H^{-1} \leftarrow H^{-1} - \frac{1}{[H^{-1}]_{kk}} H^{-1}_{:,k} H^{-1}_{k,:} \quad (13)$$

where  $\delta_k$  is the quantization error vector (non-zero at index  $k$  for  $w_{ij}$ ).

From Eqn. 13, the  $H^{-1}$  update depends on index  $k$  (position  $(i, j)$ ). In OBQ, quantizing low-sensitivity parameters independently per row leads to unsynchronized column indices  $j$  across rows  $i$ , requiring a distinct  $H^{-1}$  for each row.

This requires storing a  $[d_{\text{row}}, d_{\text{col}}, d_{\text{col}}]$  tensor for  $H^{-1}$  matrices, with memory scaling quadratically with  $d_{\text{col}}$  and linearly with  $d_{\text{row}}$ . Sequential row-specific

updates hinder parallelization, increasing time complexity. A strategy quantizing the same column across rows could use a single  $H^{-1}$ , reducing overhead.

Based on sensitivity-guided ranking, we analyzed parameter sensitivity distribution. In Fig. 2, we visualize a ResNet-18 convolutional layer, showing high-sensitivity parameter columns in early rounds (truncated to 256 columns). The results show column-wise clustering, suggesting synchronized quantization is feasible without altering sensitivity ranking.

In Fig. 1, we present the per-layer error (“para\_sensi\_des”), comparable to the non-parallel “sensi\_des” method, validating that column-wise parallel quantization leverages sensitivity clustering effectively with minimal error increase.

Based on these observations and validations, we propose an efficient post-training quantization algorithm guided by sensitivity. The core idea shifts from parameter-level to column-level processing, guided by aggregated sensitivity. Specifically, given a weight matrix  $W$  to be quantized:

1. Compute the sensitivity  $L_q$  for all individual parameters  $w_{ij}$ , typically derived from the diagonal elements of the inverse Hessian matrix  $H^{-1}$ .
2. Instead of sorting individual parameters, compute an aggregated sensitivity score for each *column*  $j$  by summing the sensitivities  $L_q$  of all parameters  $w_{ij}$  in that column:  $S_j = \sum_{i=1}^{d_{\text{row}}} L_q(w_{ij})$ .
3. Sort the columns of the weight matrix  $W$  based on these aggregated sensitivity scores  $S_j$  in descending order, prioritizing columns with higher total sensitivity. Accordingly, permute the rows and columns of the single  $H^{-1}$  matrix to match the new column order.
4. Perform quantization column-wise in the determined order. In each step, all parameters  $w_{ij}$  in the current column  $j$  (across all rows  $i = 1, \dots, d_{\text{row}}$ ) are quantized simultaneously.
5. After quantizing an entire column  $j$ , perform a single update step, similar to Eqns. 12 and 13, to reflect the collective impact of quantizing that column. This single update modifies the remaining unquantized parameters and the shared inverse Hessian matrix  $H^{-1}$ .

Let the computational cost associated with a single inverse Hessian update (Eqn. 13) be denoted as  $f(H^{-1})$ . The original OBQ method, due to its row-wise processing of potentially different column indices, requires approximately  $d_{\text{row}} \times f(H^{-1})$  computations for Hessian updates after processing one parameter per row. In contrast, the proposed column-wise method performs only one  $f(H^{-1})$  operation per column quantization. This consolidation of updates significantly reduces the computational complexity of Hessian management, potentially by a factor proportional to  $d_{\text{row}}$ . Additionally, the need to store and manage only a single inverse Hessian matrix greatly reduces the memory footprint.

## 5 Algorithm Complexity Comparison and Analysis

The FastOBQ algorithm, is analyzed here for a single layer. The complexity analysis consists of the following components: computing parameter sensitivity,

sorting parameters by column, parameter quantization and updates, and inverse Hessian matrix updates. Assume the parameters to be quantized form a  $d_{\text{row}} \times d_{\text{col}}$  matrix. The computational complexity of each component is as follows:

1. Computing Parameter Sensitivity: This involves calculating the inverse of the Hessian matrix. In practice, this is achieved by first performing Cholesky decomposition, with a time complexity of  $O(d_{\text{col}}^3)$ . Computing the inverse of the decomposed result also has a time complexity of  $O(d_{\text{col}}^3)$ . Thus, the total time complexity for this step is  $O(d_{\text{col}}^3)$ .
2. Sorting Parameters by Column: For a matrix with  $d_{\text{col}}$  columns, the sorting time complexity is  $O(d_{\text{col}} \log d_{\text{col}})$ .
3. Parameter Quantization: Quantization is a linear operation, and parallelization across rows results in a time complexity of  $O(d_{\text{col}})$ .
4. Parameter Updates: This involves element-wise vector multiplication, also parallelized, with a computational complexity of  $O(d_{\text{row}}d_{\text{col}})$ .
5. Inverse Hessian Matrix Updates: This involves matrix multiplication of dimensions  $[d_{\text{col}}, 1]$  and  $[1, d_{\text{col}}]$ , yielding a computational complexity of  $O(d_{\text{col}}^2)$ .

Since steps (2), (3), (4), and (5) are performed sequentially for each column, the overall complexity of the FastOBQ algorithm is:

$$O(d_{\text{col}}^3) + O(d_{\text{col}}) \times (O(d_{\text{col}} \log d_{\text{col}}) + O(d_{\text{col}}) + O(d_{\text{row}}d_{\text{col}}) + O(d_{\text{col}}^2)),$$

which simplifies to  $O(d_{\text{col}}^3) + O(d_{\text{row}}d_{\text{col}}^2)$ . Compared to the OBQ’s complexity of  $O(d_{\text{row}}d_{\text{col}}^3)$ , FastOBQ reduces the complexity by an order of magnitude.

## 6 Experiments

### 6.1 Experimental Setup

**Datasets.** We use ImageNet [5] and COCO [16] to validate the proposed row-parallel post-training quantization algorithm under sensitivity-guided ordering. ImageNet (ILSVRC2012) contains 1.28 million training and 50,000 validation images across 1,000 categories, ideal for image classification. COCO, with 118,000 training and 5,000 validation images annotated for 80 object categories, supports object detection and segmentation tasks.

**Models.** We employ ResNet [10] series (ResNet-18, 11.7M parameters; ResNet-34, 21.8M; ResNet-50, 25.6M) for classification, leveraging their residual structures to assess quantization sensitivity across depths. For detection, YOLOv5 variants (YOLOv5s, 7.2M parameters; YOLOv5m, 21.2M) are used, adjusting complexity via width and depth to evaluate multi-scale feature quantization.

**Evaluation Metrics.** For ImageNet classification, Top-1 Accuracy measures the proportion of correct predictions. For COCO detection, mean Average Precision (mAP) is used, with mAP@0.5 at an IoU threshold of 0.5 and mAP@0.5:0.95 averaged over IoU thresholds from 0.5 to 0.95 (step size 0.05).

**Table 1.** Comparison of different methods under 4-bit weight quantization. “Time” denotes quantization times. “Mem.” denotes peak memory usage.

Model	Bit Width	Method	Layer-wise Quant.	Accuracy	Time (s)	Mem. (MB)
ResNet-18	FP32	-	-	69.76	-	-
	W4A32	BRECQ		70.94	1789	5391
		Bias Correction	✓	53.76	-	-
		AdaRound	✓	68.52	1225	3834
		AdaQuant	✓	67.01	341	4789
		Bit-split	✓	69.11	3191	10803
		OBQ	✓	69.33	7784	6502
		FastOBQ	✓	<b>69.37</b>	<b>58</b>	<b>3069</b>
ResNet-50	FP32	-	-	76.13	-	-
	W4A32	BRECQ		76.463	5558	10295
		Bias Correction	✓	63.52	-	-
		AdaRound	✓	75.26	3766	4517
		AdaQuant	✓	75.22	1127	7760
		Bit-split	✓	75.58	5032	10856
		OBQ	✓	75.71	9287	6859
		FastOBQ	✓	<b>75.77</b>	<b>80</b>	<b>3683</b>

**Implementation Details.** Following the practices of OBQ, the quantization calibration process uses a subset of training data. Specifically, for the ImageNet classification task, 0.1% of the original training set (approximately 2,048 samples) is randomly sampled to form the calibration dataset, augmented with standard data augmentation techniques (random flipping and cropping) to expand the dataset size by tenfold. For object detection tasks, no data augmentation is applied. Additionally, to ensure numerical stability and invertibility of the Hessian matrix  $H$ , a damping value of 0.1 is uniformly added to its diagonal elements. After the quantization process, a post-processing step tunes the Batch Normalization (BN) layers in the model. This step adjusts the running statistics (mean and variance) of BN layers to adapt to the activation distribution resulting from quantized weights, mitigating potential negative impacts on model accuracy. The procedure involves resetting the running statistics of all BN layers, performing several forward passes on the quantized model using the calibration dataset, and re-estimating and updating the running mean and variance of the BN layers. During final evaluation, the model uses these tuned BN statistics, which better reflect the data distribution characteristics after quantization.

## 6.2 Comparative Experiments

To evaluate the proposed quantization method, experiments compare layer-wise and non-layer-wise approaches on ImageNet and COCO tasks, with results in

**Table 2.** Comparison of different methods under YOLOv5 model quantization. “Time” denotes quantization times. “Mem.” denotes peak memory usage.

Model	Bit Width	Method	Layer-wise	mAP@0.5	mAP@0.5:0.95	Time (s)	Mem. (MB)
YOLOv5s	FP32	-	-	37.46	56.73	-	-
	W6A32	Bias Correction	✓	26.52	45.69	-	-
		OBQ	✓	<b>37.01</b>	<b>56.46</b>	611	1966
		FastOBQ	✓	36.60	56.25	<b>17</b>	<b>1966</b>
	FP32	-	-	37.46	56.73	-	-
	W8A32	Bias Correction	✓	26.95	45.89	-	-
OBQ		✓	<b>37.41</b>	<b>56.74</b>	766	1966	
FastOBQ		✓	37.36	56.70	<b>22</b>	<b>1966</b>	
YOLOv5m	FP32	-	-	45.16	63.88	-	-
	W6A32	Bias Correction	✓	35.33	53.96	-	-
		OBQ	✓	<b>44.83</b>	<b>63.81</b>	3289	4350
		FastOBQ	✓	44.37	63.44	<b>36</b>	<b>3007</b>
	FP32	-	-	45.16	63.88	-	-
	W8A32	Bias Correction	✓	35.56	53.99	-	-
OBQ		✓	<b>45.09</b>	63.87	3282	4350	
FastOBQ		✓	45.08	<b>63.88</b>	<b>39</b>	<b>3007</b>	

Tables 1, and 2. On ImageNet, the method matches top methods at 4 bits, outperforming the non-layer-wise BRECQ with significantly lower quantization time (tens to hundreds of times faster) and minimal resource usage. On COCO, at 6-bit and 8-bit settings, it achieves near-identical accuracy to OBQ (0.3% difference) with hundreds of times faster quantization and reduced resource demands, confirming its effectiveness and efficiency.

### 6.3 Ablation Studies

**Effectiveness of Sensitivity-Guided Parameter Quantization.** We conduct experiments on three ResNet models of different scales to validate the effectiveness of the proposed sensitivity-guided parameter quantization. The results are shown in Table 3. To ensure fair comparison, all results based on sensitivity-guided ordering are reproduced under identical settings. For parameter sorting methods in ascending order, the quantized model accuracy significantly decreases, with an average reduction of 0.5% compared to descending order sorting. Among the various parameter sorting methods tested, the sensitivity-based sorting method achieves a slight accuracy advantage under specific ordering rules, comparable to the heuristic OBQ algorithm. These results indicate that the proposed method, by considering parameter sensitivity and leveraging Taylor expansion-based minimization of layer-wise quantization errors for parameter compensation, better accounts for the importance of parameters to be quantized, thereby enhancing the quantization performance.

**Table 3.** Accuracy of different parameter quantization orders on ResNet models.

Bits	Method	Parallel	ResNet-34			ResNet-50			Avg. Acc.
			Accuracy	Time	Memory	Accuracy	Time	Memory	
W4A32	err_des		72.64	13564	7276	75.49	9208	6859	74.07
	err_asc		72.16	13704	7276	75.32	9199	6859	73.74
	w_des		72.63	13502	7276	75.57	9166	6859	74.10
	w_asc		72.04	13140	7276	75.49	9280	6859	73.77
	sensi_des		<b>72.83</b>	13511	7275	<b>75.58</b>	9112	6859	<b>74.21</b>
	sensi_asc		72.24	13138	7275	75.03	9263	6859	73.64
	No Sorting	✓	72.65	90	3842	75.52	70	3683	74.09
	err_des	✓	73.00	100	3842	75.62	79	3683	74.31
	w_des	✓	72.93	90	3842	75.66	77	3683	74.30
	sensi_des	✓	<b>73.12</b>	93	3842	<b>75.67</b>	79	3683	<b>74.40</b>

**Effectiveness of Row-Parallel Matrix Parameter Quantization.** As shown in Table 3, introducing the row-parallel quantization algorithm results in no significant accuracy loss compared to non-parallel quantization experiments across the sorting methods tested. Moreover, significant improvements are observed in quantization time and memory usage. These results demonstrate that, when performing row-parallel parameter quantization under specified rules, the proposed method accounts for the similarity in quantization order across rows guided by specific metrics, enabling efficient row-parallel quantization and substantially reducing the time and resource consumption of quantization.

## 7 Conclusions

We propose FastOBQ, a sensitivity-guided post-training quantization algorithm that enhances efficiency while preserving accuracy. By prioritizing high-sensitivity parameters and using row-parallel quantization with a shared Hessian matrix, FastOBQ reduces quantization error and computational overhead. Experiments on ResNet and YOLOv5 show order-of-magnitude speedups with accuracy comparable to state-of-the-art methods. Our approach is extensible to cross-layer and hybrid-precision quantization, potentially matching retraining-based methods. Future work will explore these extensions, particularly for large-scale models.

**Acknowledgments.** This work was partially supported by the Joint Funds of the National Natural Science Foundation of China (Grant No. U24A20327), Key-Area Research and Development Program Guangdong Province 2018B010107001, the Major Key Project of Peng Cheng Laboratory (PCL) PCL2023A08, Postdoctoral Fellowship Program of CPSF (Grant No. GZC20251043), and TCL Science and Technology Innovation Fund, China.

## References

1. Banner, R., Nahshan, Y., Soudry, D.: Post training 4-bit quantization of convolutional networks for rapid-deployment. In: Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS), vol. 32 (2019)
2. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint [arXiv:1308.3432](https://arxiv.org/abs/1308.3432) (2013)
3. Brown, T.B., et al.: Language models are few-shot learners. In: Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS) (2020)
4. Choukroun, Y., Kravchik, E., Yang, F., Kisilev, P.: Low-bit quantization of neural networks for efficient inference. In: Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), pp. 3009–3018. IEEE (2019)
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 248–255. IEEE (2009)
6. Dong, X., Chen, S., Pan, S.: Learning to prune deep neural networks via layer-wise optimal brain surgeon. In: Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS), vol. 30 (2017)
7. Frantar, E., Alistarh, D.: Optimal brain compression: a framework for accurate post-training quantization and pruning. In: Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS), vol. 35, pp. 4475–4488 (2022)
8. Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M.W., Keutzer, K.: A survey of quantization methods for efficient neural network inference. In: Low-Power Computer Vision, pp. 291–326. Chapman and Hall/CRC (2022)
9. Gong, R., et al.: Differentiable soft quantization: bridging full-precision and low-bit neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 4852–4861 (2019)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)
11. He, X., Cheng, J.: Learning compression from limited unlabeled data. In: European Conference on Computer Vision (ECCV), pp. 752–769 (2018)
12. Hubara, I., Nahshan, Y., Hanani, Y., Banner, R., Soudry, D.: Accurate post training quantization with small calibration sets. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 4466–4475. PMLR (2021)
13. Jacob, B., et al.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2704–2713 (2018)
14. LeCun, Y., Denker, J., Solla, S.: Optimal brain damage. In: Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS), vol. 2 (1989)
15. Li, Y., et al.: BRECQ: pushing the limit of post-training quantization by block reconstruction. In: Proceedings of the International Conference on Learning Representations (ICLR) (2021)
16. Lin, T.-Y., et al.: Microsoft COCO: common objects in context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8693, pp. 740–755. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-10602-1\\_48](https://doi.org/10.1007/978-3-319-10602-1_48)

17. Liu, A., et al.: DeepSeek-V3 technical report. arXiv preprint [arXiv:2412.19437](https://arxiv.org/abs/2412.19437) (2024)
18. Liu, Z., et al.: Swin transformer: hierarchical vision transformer using shifted windows. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 10012–10022 (2021)
19. Liu, Z., et al.: LLM-QAT: data-free quantization aware training for large language models. In: Ku, L., Martins, A., Srikumar, V. (eds.) Findings of the Association for Computational Linguistics (ACL), pp. 467–484. Association for Computational Linguistics (2024)
20. Liu, Z., Wang, Y., Han, K., Zhang, W., Ma, S., Gao, W.: Post-training quantization for vision transformer. In: Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS), vol. 34, pp. 28092–28103 (2021)
21. Meller, E., Finkelstein, A., Almog, U., Grobman, M.: Same, same but different: recovering neural network quantization error through weight factorization. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 4486–4495. PMLR (2019)
22. Nagel, M., Amjad, R.A., Van Baalen, M., Louizos, C., Blankevoort, T.: Up or down? Adaptive rounding for post-training quantization. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 7197–7206. PMLR (2020)
23. Nagel, M., Fournarakis, M., Bondarenko, Y., Blankevoort, T.: Overcoming oscillations in quantization-aware training. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 16318–16330. PMLR (2022)
24. OpenAI: GPT-4 technical report. arXiv preprint [arXiv:2303.08774](https://arxiv.org/abs/2303.08774) (2023)
25. Touvron, H., et al.: Llama: open and efficient foundation language models. arXiv preprint [arXiv:2302.13971](https://arxiv.org/abs/2302.13971) (2023)
26. Wang, Z., Lu, J., Tao, C., Zhou, J., Tian, Q.: Learning channel-wise interactions for binary convolutional neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 568–577 (2019)
27. Zhang, D., Yang, J., Ye, D., Hua, G.: LQ-Nets: learned quantization for highly accurate and compact deep neural networks. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11212, pp. 373–390. Springer, Cham (2018). [https://doi.org/10.1007/978-3-030-01237-3\\_23](https://doi.org/10.1007/978-3-030-01237-3_23)
28. Zhao, R., Hu, Y., Dotzel, J., De Sa, C., Zhang, Z.: Improving neural network quantization without retraining using outlier channel splitting. In: Proceedings of the International Conference on Machine Learning (ICML), pp. 7543–7552. PMLR (2019)