Output-Directed Dynamic Quantization for DNN Acceleration

Beilei Jiang[1], Xianwei Cheng[1], Yuan Li[1], Jocelyn Zhang[1], Song Fu[1], Qing Yang[1], Mingxiong Liu[2], Alejandro Olvera[1]
{beilei.jiang,yuan.li,song.fu,qing.yang}@unt.edu;mliu@lanl.gov;{xianweicheng,jocelynzhang,alejandroolvera}@my.unt.edu
[1]University of North Texas, [2]Los Alamos National Laboratory

ABSTRACT

Quantization is an effective technique for reducing the number of computations and improving the performance of deep neural networks (DNNs). Weight quantization is popular because weights can be trained beforehand. However, weight quantization only targets the kernel weights and ignores the sensitivity of input features, which can lead to reduced accuracy. Fine-grained input quantization has gained attention as a way to speed up DNNs while maintaining accuracy. Existing approaches determine computation precision based on input sensitivity but do not effectively reduce computations for insensitive outputs or retain the precision of sensitive outputs. These limitations motivate us to develop an ***output-directed dynamic quantization method*** named ***ODQ*** in this paper. ODQ is a two-stage DNN quantization scheme designed to improve performance, reduce energy consumption, and maintain and often improve accuracy, compared with existing quantization methods. Specifically, inputs and weights go through sensitivity prediction and result generation. The high-order 2 bits of input and weight are used to predict output sensitivity. Result generation is performed only for predicted sensitive outputs. We designed an FPGA accelerator to optimize ODQ quantization performance for DNNs. We implement a prototype of ODQ and evaluate its performance using several state-of-the-art DNNs. Compared with a state-of-the-art input-directed quantization approach, ODQ achieves a 67.6% performance speedup and a 66.9% energy saving, with minimal accuracy degradation (≤0.6%).

CCS CONCEPTS

• **Computer systems organization** → **Architectures**; *Neural networks*; Reconfigurable computing.

KEYWORDS

Deep Neural Network, Dynamic Quantization, Sensitivity Prediction, Performance Acceleration, FPGA.

ACM Reference Format:
Beilei Jiang[1], Xianwei Cheng[1], Yuan Li[1], Jocelyn Zhang[1], Song Fu[1], Qing Yang[1], Mingxiong Liu[2], Alejandro Olvera[1]. 2023. Output-Directed Dynamic Quantization for DNN Acceleration. In *52nd International Conference on Parallel Processing (ICPP 2023), August 07–10, 2023, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3605573.3605580

ICPP 2023, August 07–10, 2023, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0843-5/23/08…$15.00
https://doi.org/10.1145/3605573.3605580

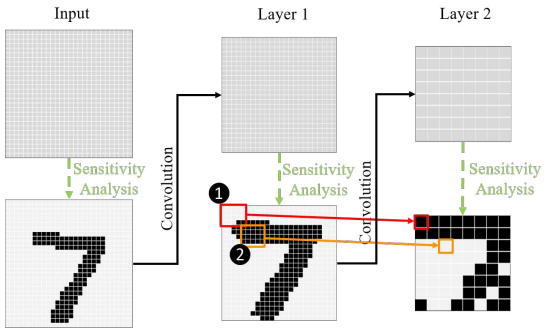

Figure 1: Inefficiency of input-directed quantization. We use LeNet-5 on the MNIST dataset as an illustrating example. Sensitive features are represented by black squares while insensitive features are denoted by gray squares.

1 INTRODUCTION

The past decade has witnessed the wide adoption and success of deep neural networks (DNNs) in various fields, which is attributed to their capability to capture highly complex, nonlinear input/output relationships with unprecedented accuracy. However, as DNNs become increasingly complex for better accuracy, the number of parameters and computations grows drastically, which puts a heavy burden on the underlying computing platforms and runtime systems. This poses a critical challenge in developing pervasive deep learning for real-time inference with low energy consumption on resource-constrained systems, which is imperative for many new fields and applications, such as autonomous driving, smart healthcare monitoring, smart manufacturing, and speech and emotion recognition [5, 7, 16, 21].

Quantization is an effective technique in reducing inference time and energy consumption by mapping a DNN's inputs from a large range to outputs in a smaller range [10, 27]. A number of quantization methods have been presented in the literature, e.g., DRQ [20], OLAccel [15], and FILM-QNN [22], etc. Recent studies focus on weight and/or input quantization [15, 25]. However, many suffer from significant accuracy degradation (≥ 5%). Additionally, weight-focused quantization has progressed to the point where it is challenging to speed up the DNNs while maintaining accuracy standards. Input sensitivity should be considered during quantization to further speed up DNNs and maintain accuracy [2, 17]. However, the sensitivity of accuracy to input/activation has not been well investigated.

Input-based quantization methods, such as DRQ [20], adjust weights and input precision based on input sensitivity. It compares input features, or the sum of input features in a region, with a predefined threshold. If the input features are larger than the threshold, they are sensitive; otherwise, they are insensitive. For sensitive

inputs, high-precision weights and inputs are used, whereas low-precision computations are performed for less sensitive inputs. Although those methods reduce inference time, they lack the following desirable properties. (1) Preserve model accuracy. High-precision computations should be performed for sensitive output features. (2) Improve efficiency. Low-precision computations should be performed for insensitive output features. Please note that outputs that are more sensitive, i.e., those with a larger magnitude, have a greater impact on a model's accuracy.

Figure 1 illustrates these issues. Existing input-directed quantization methods conduct sensitivity analysis on the input feature maps of each layer before convolution. In both cases (❶ and ❷ in Figure 1), the output is generated by mixed-precision computations. In the first case, the sensitive output is calculated from almost all insensitive (i.e., low-precision) inputs, which affects model accuracy, which is property (1). In the second case, the insensitive output is computed from almost all sensitive (i.e., high-precision) inputs, which incurs computation overhead, which is property (2). Both cases indicate that sensitivity of features is not fully exploited by input-directed quantization.

To address these problems, this paper proposes an **output-directed dynamic quantization** (**ODQ**) method, *aiming to effectively explore low-precision computations while preserving accuracy for DNN acceleration.* ODQ uses sensitivity analysis of *output features* to determine the quantization strategy, i.e., low-precision or high-precision computation. An output sensitivity prediction algorithm is developed for ODQ to efficiently and accurately predict sensitive outputs. ODQ streamlines sensitivity prediction and result generation into a single-shot process, which significantly reduces sensitivity analysis overhead and enhances quantization efficiency without compromising model accuracy.

In the sensitivity prediction step, high-order bits of inputs and weights are used to quickly generate partial results to determine the output's sensitivity with respect to model accuracy. The prediction operation is lightweight by design. In the result generation step, the remaining computation related to sensitive outputs is performed. To speed up ODQ execution for real-time applications, an ODQ accelerator is developed on the FPGA. Our ODQ accelerator can be reconfigured to maximize the utilization of on-chip resources and data reuse and mitigate performance degradation caused by processing elements' stalls.

The main contributions of this paper are:

- Qualitative and quantitative characterization of current input-focused DNN dynamic quantization limitations.
- Design of a novel **output-directed dynamic quantization (ODQ)** method that predicts output sensitivity and fine-tunes computation precision at runtime. Sensitive output features are computed with high-precision weights and inputs, while insensitive outputs are produced with low-precision ones.
- Development of a reconfigurable ODQ accelerator in Verilog that includes three types of PE arrays: reconfigurable PE arrays, predictor PE arrays, and executor PE arrays. The reconfigurable PE arrays can be set up as predictor PE arrays or executor PE arrays depending on the percentage of sensitive output features.
- Implementation of a prototype of ODQ and its accelerator and extensive evaluation of performance on ResNet-56, ResNet-20, VGG-16 and DenseNet. ODQ achieves promising results, including a 97.8% performance improvement in terms of execution time and a 97.6% energy consumption reduction compared to static quantization methods. Moreover, ODQ achieves a 67.6% reduction in execution time and a 66.9% energy reduction over DRQ, an input-directed dynamic quantization framework.

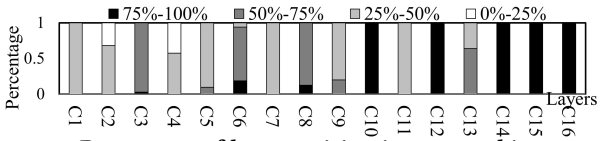

Figure 2: Percentage of low-precision inputs used in generating sensitive outputs via input-directed quantization applied to ResNet-20 on CIFAR-10.

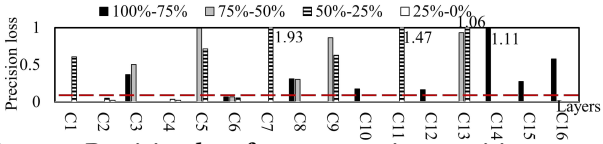

Figure 3: Precision loss from computing sensitive outputs using low-precision inputs via input-directed quantization applied to ResNet-20 on CIFAR-10.

2 MOTIVATION

Quantization can be applied to weights and inputs. Weight-based quantization has advanced to the point where it is difficult to speed up the DNNs while maintaining accuracy standards. Recent studies have found that different input features exhibit varying degrees of model accuracy sensitivity. However, less effort has been devoted to input sensitivity analysis and the design of input-based quantization methods due to high input variability.

Input-directed quantization approaches, such as DRQ [20], typically measure the importance of inputs based on the magnitude of input features. However, they suffer from the following major problems, which motivate this work.

1) Input-directed quantization may compromise the precision of sensitive outputs due to the use of low-precision inputs.

We measure the percentage of low-precision input features (4-bits) involved in computing sensitive outputs in input-directed quantization methods. Figure 2 shows the results from DRQ on ResNet-20. In the figure, each shade represents the percentage of low-precision input features among all input features when calculating one sensitive output feature. We summarize the percentage in four ranges:(0%-25%,25%-50%,50%-75%,75%-100%). We can see that in almost every layer, most sensitive output features are calculated with more than 25% of low-precision input features. In particular, in the convolutional layers C10, C12, C14, C15, and C16, over 75% of low-precision inputs are used in producing all sensitive outputs. In summary, a significant number of sensitive outputs cannot be generated with the desired precision. What is more, the relationship between the precision of sensitive features and the accuracy of DNN model is analyzed in [20], it shows that when the precision loss for sensitive outputs reaches 0.1, more precision loss results in significant accuracy degradation, i.e., more than 10. Therefore, to further demonstrate the impact on the accuracy of DNN models, the precision loss caused by this issue is observed.

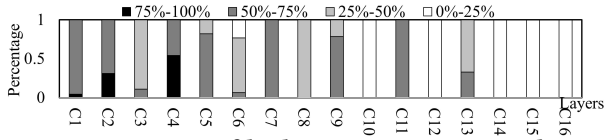

Figure 4: Percentage of high-precision inputs used in generating insensitive outputs via input-directed quantization applied to ResNet-20 on CIFAR-10.

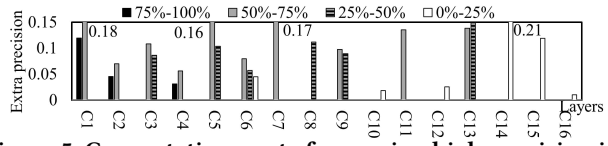

Figure 5: Computation waste from using high-precision inputs to generate insensitive outputs via input-directed quantization applied to ResNet-20 on CIFAR-10.

Figure 3 shows the average precision loss caused by using low-precision inputs to compute sensitive outputs. As can be observed, the amount of noise introduced to most of the layers is greater than 0.1. In particular, the noise added to the layers C7, C11, C13, and C14 is even greater than 1, which leads to a significant perturbation to the DNN model and affects the utilization of smaller bit widths for effective quantization.

2) Input-directed quantization incurs unnecessary computations as high-precision inputs are used to produce insensitive outputs.

We measured the number of high-precision inputs (8-bits) involved in generating insensitive outputs in input-directed quantization methods. Figure 4 shows the DRQ results on ResNet-20. We observed that over 25% of high-precision inputs are involved in calculating all insensitive outputs in multiple layers. In particular, the convolutional layers C1, C2, C4, C7, and C11 use over 50% of high-precision inputs to compute all insensitive outputs, resulting in significant high-precision computation for insensitive features.

To prove this will cause redundant computation, the extra precision (computation waste) that results from using high-precision inputs to produce insensitive outputs are observed, as shown in Figure 5. According to DRQ, insensitive outputs have a higher tolerance for noise. Adding noise with a magnitude of less than 10 to inputs has a negligible effect on prediction accuracy. In Figure 5, we find that for insensitive outputs, using high-precision inputs leads to extra precision of up to 0.21 compared to low-precision inputs. The extra precision can be quantified as follows:

$$Extra\_precision = max(abs(O_{IDQ} - O_{LP\_input})), \quad (1)$$

where O_{IDQ} refers to the insensitive output features produced by input-directed quantization, and O_{LP_input} represents the insensitive output features computed using low-precision inputs. From the figure, we can see that removing the extra precision only causes up to 0.21 orders of magnitude of noise, which can be eliminated for better energy efficiency and speedup.

In short, input-directed quantization not only increases noise in producing sensitive outputs but also causes unnecessary computations for insensitive outputs using high-precision inputs. We designed an output-directed dynamic quantization (ODQ) method to address these problems.

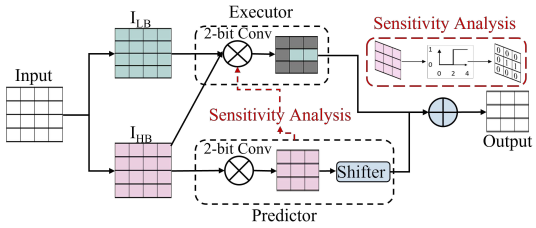

Figure 6: Structure and key components of the proposed output-directed dynamic quantization (ODQ).

3 OUTPUT-DIRECTED DYNAMIC QUANTIZATION (ODQ)

Accurately predicting output sensitivity is imperative for retaining DNN models' accuracy and improving quantization efficiency. However, this process will introduce additional overhead. Therefore, a novel design is required to ensure the performance gain from quantization outweighs the prediction cost.

To this end, we design a two-step output-directed dynamic quantization (ODQ) method that is streamlined into a single-shot process. The structure and workflow of ODQ are presented in Figure 6. The key steps of ODQ are *sensitivity prediction* and *result generation*. It dynamically selects the quantization strategy (low-precision/high-precision computation and bit width) based on the sensitivity prediction of output features on the fly.

Specifically, an output feature is produced by accumulating the multiplications of input features (I) and weight filters (W). That is

$$O(t_o, x, y) = \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} \sum_{t_i=0}^{N-1} W(t_o,\ t_i,\ i,\ j) * I(t_i,\ i+S*x,\ j+S*y), \quad (2)$$

where K denotes the spatial dimension of a filter, S is the step size, and N is the number of input channels. Subscripts t_o, t_i, i and j denote indexes. An input (I) and a weight (W) can be partitioned into high-order bits I_{HBs} and W_{HBs}, and low-order bits I_{LBs} and W_{LBs}, respectively. Equation (2) can be rewritten as follows.

$$\begin{aligned} O = &(\sum I_{HBs} * W_{HBs}) << 2N_{LBs} + \sum I_{LBs} * W_{LBs} \\ &+ (\sum I_{HBs} * W_{LBs}) << N_{LBs} + (\sum I_{LBs} * W_{HBs}) << N_{LBs}, \end{aligned} \quad (3)$$

where N_{LBs} is the bit width of I_{LBs} and W_{LBs}, and << represents a shift operation. According to Equation (3), an output feature can be computed in four parts: ① computation between the high-order bits of weights and the high-order bits of activation; ② computation between the high-order bits of weights and the low-order bits of activation; ③ computation between the low-order bits of weights and the high-order bits of activation; and ④ computation between the low-order bits of weights and the low-order bits of activation. We find that the output is dominated by the result from the high-order input and weight bits, i.e., I_{HBs} and W_{HBs}. Therefore, we design the sensitivity predictor based on the MAC operations on I_{HBs} and W_{HBs}. The rest of the MAC operations are performed in the result generation step.

As shown in Figure 6, given an input feature map, we first quantize the feature map from FP32 to INT4 [27]. Then, we divide it into two sub-input feature maps, one from the high-order bits (i.e., I_{HBs}-2 bits) and the other from the low-order bits (i.e., I_{LBs}-2 bits). The sub-input feature map with I_{HBs} is processed by the sensitivity

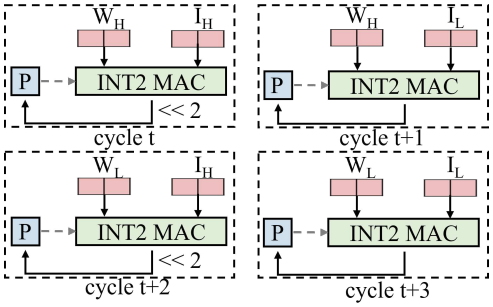

Figure 7: Architecture of a multi-precision PE.

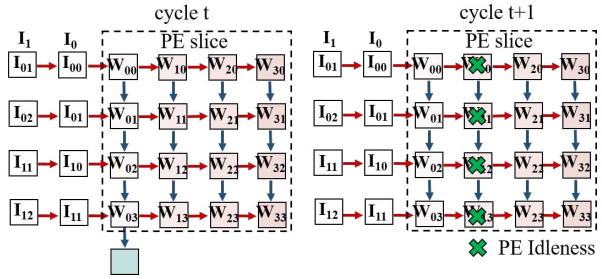

Figure 8: Illustration of PE idleness in existing accelerators with multi-precision PEs.

predictor. The predictor uses convolution results to identify sensitive output features. Specifically, the predictor uses an activation function to create a bit mask by comparing each calculated output feature with a threshold. That is, if the magnitude of a feature exceeds the threshold, it is sensitive. Otherwise, the feature is insensitive. The predicted sensitivity values of features are stored in a bit mask with the same dimension as the output feature map, where"1" indicates a sensitive feature and "0" denotes an insensitive one.

The result executor uses the bit mask produced by the sensitivity prediction to perform the remaining computations for sensitive output. By adding the results from both the sensitivity predictor and the result executor, ODQ produces the final output.

A key parameter in sensitivity prediction is the threshold. ODQ adopts an adaptive approach to finding the optimal value. This process is illustrated with the following example. A network is initially trained with 4-bit weights and 4-bit inputs. Next, ODQ randomly selects N inputs from the test dataset, performs inference using the high-order bits of the inputs (I_{HBs}) and the trained weights (W_{HBs}), and generates the output distribution of each layer. A relatively large initial threshold is chosen based on the output distribution. Weights are retrained after introducing the threshold to the model to capture sensitivity information in the input feature maps. Then, the retrained weights are used for ODQ-based inference. If accuracy meets the expectation, the threshold is selected. Otherwise, ODQ halves the threshold value and repeats the above process until a suitable threshold is found.

4 ODQ ACCELERATOR

ODQ leverages the high-order bits of inputs and weights for output sensitivity prediction, which directs bit width selection for quantized computation. General-purpose processors execute efficiently with FP32, FP16, or INT8 operands, when compared to 2-bit data. Moreover, the sparse characteristics of ODQ can result in resource underutilization and performance degradation. Therefore, we design an accelerator to speed up ODQ for real-time inferences.

ODQ performs mixed-precision computations (i.e., 2-bit and 4-bit MACs). A brute-force approach is to design 4-bit MAC processing elements (PEs) that can handle both 2-bit and 4-bit MACs in a single clock cycle. However, due to the high probability of low-precision computation, a lot of processing resources will be wasted. An efficient design is multi-precision PEs, which can be found in BitFusion and DRQ [19, 20]. The structure of a multi-precision PE is depicted in Figure 7. Multi-precision PEs are applicable to both 2-bit and 4-bit MACs, taking one clock cycle and four clock cycles, respectively. With multi-precision PEs, all PEs in the PE group use one clock cycle to generate the output in the sensitivity prediction without any loss of computing resources. In addition, weight filters and inputs can be reused in calculating output features.

We note that the irregularity and sparsity of insensitive output features can cause idle PEs during the result generation step, challenging on-chip bandwidth management.

4.1 PE Idleness and Bandwidth Management

Existing DNN accelerators, such as [4, 18, 20], usually use a weight-stationary or input-stationary dataflow to take advantage of data reference locality in DNN models. Those accelerators process multiple inputs and weights and generate multiple output features from the same or different output feature maps.

Figure 8 illustrates PE idleness caused by sparsity of insensitive output features. Figure 8 shows a DNN accelerator with weight-stationary dataflow. Let the output feature computed with $W_0([W_{00}, W_{01}, W_{02}, W_{03}])$ and I_0 be sensitive, and the output feature computed with $W_1([W_{10}, W_{11}, W_{12}, W_{13}])$ and I_0 be insensitive. Their sensitivity is used to control PE execution in result generation. At cycle t, the remaining outputs are computed since the output feature with W_0 and I_0 is sensitive. Nonetheless, since the feature with W_1 and I_0 is insensitive, the PE is idle at cycle $t+1$, which cancels out the performance improvement from quantization.

Furthermore, the irregular and sparse insensitive output features hinder data reuse during result generation. The sensitivity predictor and the result generator have different bandwidth requirements. Assume the sensitivity predictor requires N bytes of data every cycle. The result generator, however, has limited data reuse opportunities. As a result, more data must be collected every cycle to avoid PE stalls during result generation. The bandwidth becomes underutilized during sensitivity prediction if we assign the on-chip bandwidth based on the result generator requirements. The PE will stall during result generation, waiting for data if bandwidth is determined by the sensitivity predictor's needs.

4.2 Static vs. Dynamic PE Allocation

The PEs in a column form a PE array. Allocating an equal number of PE arrays to the sensitivity predictor and result generator is inefficient due to their differing computational workloads, leading to pipeline bubbles. With 50% sensitive output features, the result generator has a 1.5x higher computational load than the sensitivity predictor. PE array allocation should follow the same ratio accordingly. Moreover, the sensitivity predictor PEs compute output in one cycle, while the result generator takes three cycles. The result generator does not need to compute insensitive outputs.

Figure 9: Percentage of insensitive output features in ResNet-56 using output-directed dynamic quantization (ODQ).

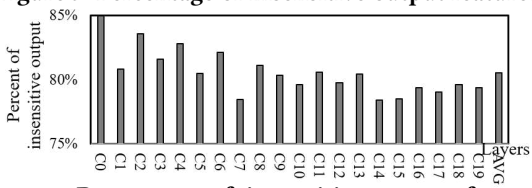

Figure 10: Percentage of insensitive output features in ResNet-20 using ODQ.

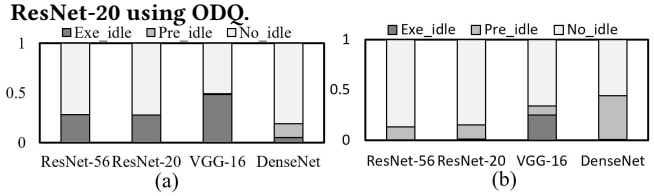

Figure 11: Percentage of idle PEs with Static PE allocation. (a) Executor PE Array: 12, Predictor PE Array: 15. (b) Executor PE Array: 9, Predictor PE Array: 18.

Table 1: Configuration of PE arrays and the corresponding maximum percentage of sensitive output features without causing pipeline bubbles.

#of PE arrays for predictor	#of PE arrays for executor	Maximum percent of sensitive output features (%)
9	18	66
12	15	41
15	12	26
18	9	16
21	6	9

We also need to consider DNN models' characteristics when allocating PEs. Figures 9 and 10 present the percentage of insensitive output features identified by the sensitivity predictor in ResNet-56 and ResNet-20, respectively. The percentage of insensitive output features exhibits considerable variation across layers and DNN models. Consequently, static PE allocation can lead to either over-provisioning or under-provisioning of PEs, impacting resource utilization and/or the performance of ODQ. As shown in Figure 11, the static PE allocation scheme results in 14%-50% of idle PEs (Pre_idle: idle predictor PEs and Exe_idle: idle executor PEs).

From experiments, we observe that the percentage of sensitive outputs in ResNet-20, ResNet-56, VGG16, and DenseNet ranges from 8% to 50%. A PE slice containing 27 PE arrays can be used in resource management. Specifically, based on the percentage of sensitive output features, 12 reconfigurable PE arrays can be dynamically allocated to the sensitivity predictor or the result generator. The predictor uses the leftmost 9 PE arrays, while the generator uses the rightmost 6 PE arrays. Table 1 lists the relation between PE array configuration and the percentage of sensitive output features without causing pipeline bubbles. In the table, we can see that a dynamic allocation scheme allows five alternative PE array configurations. It also indicates that by allowing the percentage of sensitive output features to be between 9% and 66%, we can allocate PEs without pipeline bubbles.

Dynamic PE management is more efficient as it differentiates the allocation of PE arrays to the sensitivity predictor and result generator according to the detected percentage of sensitive output features. This will speed up ODQ and improve resource utilization.

4.3 Reconfigurable Accelerator for ODQ

We design an ODQ accelerator that differentiates the management of the sensitivity predictor and result generator, aiming to reduce PE idleness and maximize data reuse and resource utilization. To this end, the ODQ accelerator allocates resources separately to the two components.

Figure 12 shows the architecture and major components of our reconfigurable ODQ accelerator. PE slices consist of a predictor PE array, an executor PE array, and a reconfigurable PE array. The reconfigurable PE array can be assigned as either the predictor PE array or executor PE array, depending on the percentage of sensitive output features, optimizing resource utilization. During sensitivity analysis, the predictor results are recorded in a bit mask, which is used by the executor. The executor output is stored in an output buffer before being transferred to the off-chip DRAM. The ODQ accelerator employs a global weight and input buffer to hide DRAM access latency. Moreover, our accelerator introduces line buffers to exploit data reuse in DNNs. The Im2col/Pack engine transforms data into the format seen in Figure 17 before packing it into line buffers, enabling input sharing amongst weight filters from various channels stored in PE arrays.

PE Architecture. Our ODQ accelerator has three PE groups: predictor PE, executor PE, and reconfigurable PE. During sensitivity prediction, a PE performs MAC operations only on the high-order bits of inputs and weights. Thus, a basic INT2 MAC is used. Figure 13(a) depicts the architecture of a predictor PE. It comprises two 2-bit registers to store the high-order 2 bits of an input and a weight (i.e., I_h and W_h), and a 4-bit register for a partial sum (P).

The executor handles the remaining calculations. This cannot be done with a single INT2 MAC. Thus, we employ Bitfusion's multi-precision PE architecture [19] to ensure the computation is finished in three clock cycles, as shown in Figure 13(b). In addition, the executor computes sensitive output features sparsely distributed in the feature maps. Sensitive output features are irregular and sparse, which prevents them from being reused. PEs normally request data on demand without input/weight sharing between PEs, which incurs more memory accesses. To prevent PEs from being idle, more data must be retrieved every clock cycle from DRAM. The ODQ accelerator partitions the PE array for the executor into three clusters. Every three clock cycles, a PE will make a new data request because computations take three clock cycles to complete on each

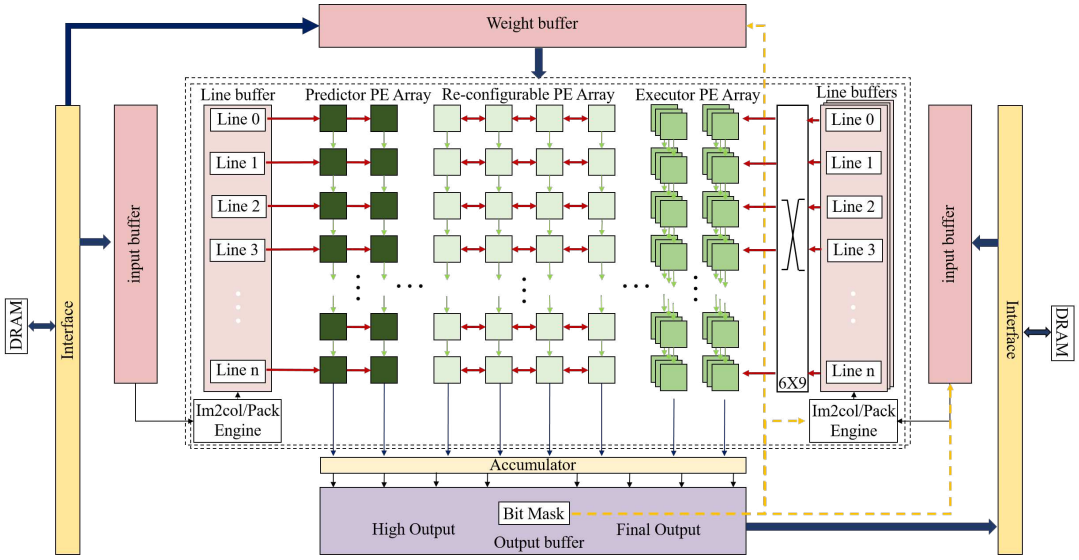

Figure 12: Architecture of reconfigurable ODQ accelerator.

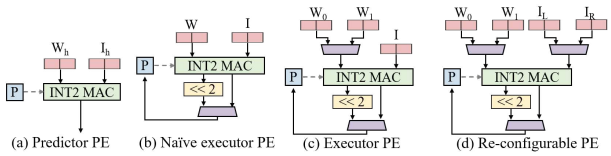

Figure 13: PE architecture of the ODQ accelerator.

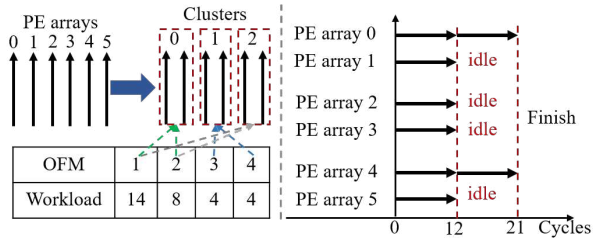

Figure 14: Illustration of PE array execution using static resource allocation.

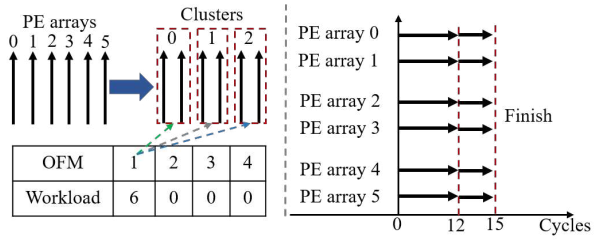

Figure 15: Illustration of PE array execution using dynamic resource allocation.

PE. With three PE clusters, data is delivered to one cluster every cycle, which minimizes the number of memory accesses.

However, workload imbalance across PE arrays may also result in idle PEs. Figure 14 illustrates this problem. In the figure, the four output feature maps (OFMs) exhibit varying workload levels. The computations associated with OFM1 and OFM2 are scheduled to the first two PE arrays in PE cluster 0, and OFM3 and OFM4 are assigned to the two PE arrays in PE cluster 1. As OFM1 and OFM2 involve more workload than the others, their computations are

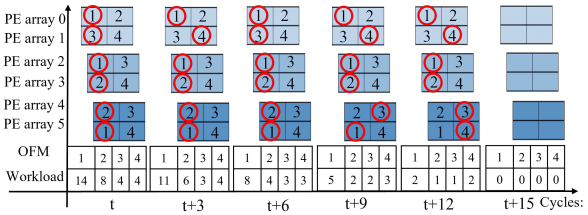

Figure 16: Illustration of the dynamic workload allocation scheme in ODQ.

divided into halves. One is handled by two PE arrays in cluster 0 and the other by two PE arrays in cluster 2. PE arrays 1, 2, 3, and 5 finish their computations after 12 cycles since they process four sensitive output features. PE arrays 0 and 4 need 21 cycles since each of them is assigned three additional sensitive output features. However, PE arrays 1, 2, 3, and 5 are idle for 9 cycles, waiting for the job to be finished. Dynamic workload distribution reallocates OFM1's remaining workload to free PE arrays using a dynamic allocation scheme (Figure 15). This maximizes computational resource utilization but introduces significant hardware overhead.

To better utilize resources and minimize the hardware overhead, we design a fine-grained dynamic workload scheduling method for the ODQ accelerator. In our dynamic scheduling method, (1) Each PE array is responsible for computing several output channels, and a cluster covers all the output channels. This prevents PE from becoming idle by allowing the workload to be allocated to any clusters. For example, in Figure 16, each PE array processes two output channels, e.g., output channels 1 and 2 are assigned to PE array 0, and the clusters handle all output channels, i.e., OFM1, OFM2, OFM3 and OFM4. (2) Computations performed by all clusters should cover as many combinations of output channels as possible, balancing the workload across output channels. Because there are four output channels in this example, six possible combinations exist, i.e., 1 and 2, 1 and 3, 1 and 4, 2 and 3, 2 and 4, 3 and 4. Four possible combinations are for Cluster 0, i.e., 1 and 3, 1 and 4, 2 and 3, 2 and 4. The possible combinations for Cluster 1 and Cluster 2 are complementary to those of Cluster 0. Thus, all of the 6 combinations are covered.

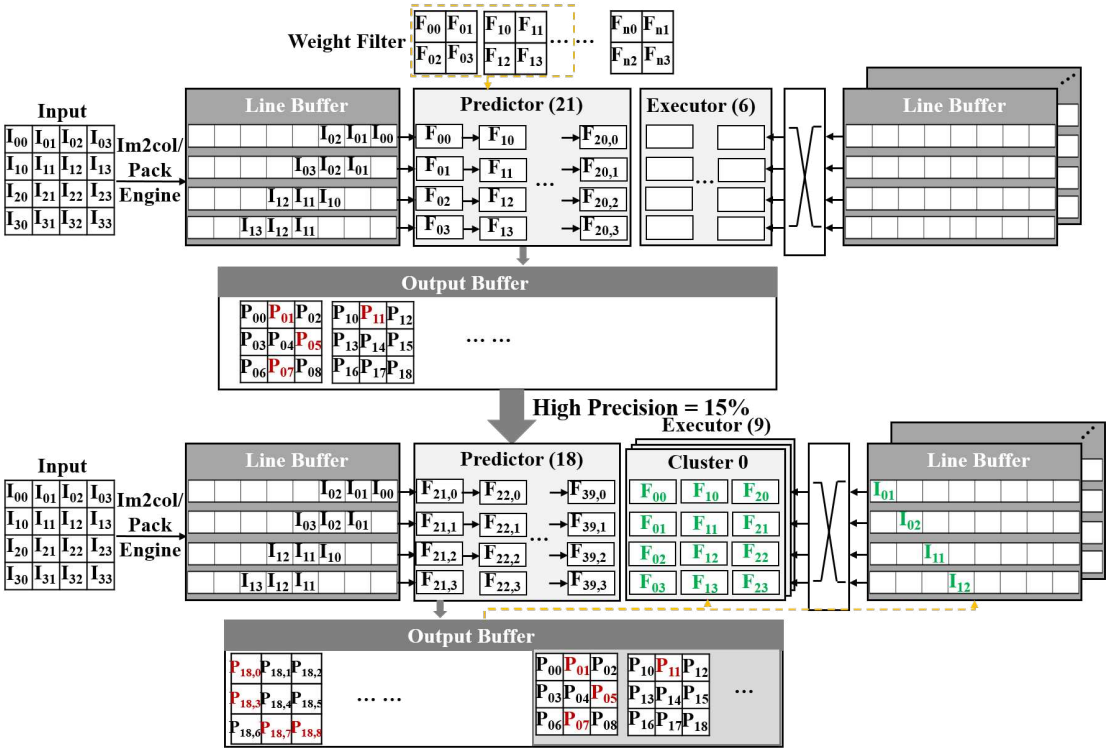

Figure 17: Execution flow of ODQ accelerator.

In each PE array, the output channel with the greatest workload has the highest priority. Data from the winning output channel is sent to the PE array via a crossbar. In Figure 16, the output channels highlighted in red circles are the output channels chosen by PE arrays. For example, as output channel 1 has more workload than output channel 2, the computation from output channel 1 is assigned to PE array 0. In this way, all computations are completed in 15 cycles without wasting resources.

The executor's PEs are designed to match the dynamic workload schedule. Figure 13(c) shows the PE architecture for the executor. Specifically, the PEs are multi-precision PEs that allow two weight sources from two output channels. In addition, two input sources are enabled since a PE can be operated either as a predictor PE or an executor PE, as shown in Figure 13(d).

Workflow of ODQ Accelerator. Figure 17 illustrates the workflow of the ODQ accelerator. In this example, a 4×4 input feature map undergoes a convolution operation with n 2×2 filters, producing n output feature maps (OFMs). The ODQ accelerator first processes the high-order 2 bits of the weight and the input in the predictor. All 12 reconfigurable PE arrays operate as predictor PE arrays, i.e., 21 predictor PE arrays in total, as there is no computation by the executor at the beginning. In Figure 17, the first 21 weight filters are loaded into the PE's registers, and the input is transformed by an Im2col/Pack Engine to match the systolic array convolution before being packed to the line buffer [11]. The output is stored in an output buffer, which is then used for sensitivity analysis. The predictor produces partial results of the sensitive output features, and we use $\mathbf{P_{xy}}$ (partial sum) to denote it in the figure. The red $\mathbf{P_{xy}}$ refers to sensitive output features.

Assuming that after the first 21 OFMs are computed in the predictor, an average of 15% of the high-precision output features are identified. Based on Table 1, to ensure that the predictor and the executor finish their assigned workload almost simultaneously, we reconfigure the PE arrays so that the predictor uses 18 PE arrays and the executor uses the remaining nine PE arrays. The nine PE arrays in the executor are divided into three clusters to relieve the memory access burden. The predictor can calculate the next 18 OFMs using 18 PE arrays. To keep the system stable, we strive to keep the number of OFMs waiting to be processed in the executor equal to 21 in the output buffer. Consequently, in the executor, the first 18 OFMs computed by the predictor will be processed. In this figure, the green color refers to high-precision operators (4-bit). As illustrated in Figure 17, PE array 0 is responsible for the remaining sensitive output feature calculations in the OFM0, while PE array 1 and PE array 2 are responsible for the rest of the sensitive output feature calculations in OFM1 and OFM2, respectively. Note that each PE array in the executor has two sets of PE registers and can hold two different weight filters. For simplicity, we only show the winning candidate in this example. As seen on the right side of the figure, the required inputs for these three PE arrays come from three line buffers. The executor's outputs are then added to the predictor's partial outcomes to form the final output.

5 PERFORMANCE EVALUATION

5.1 DNN Accuracy

We have implemented ODQ in PyTorch [1] and comprehensively evaluated its performance. We have built an ODQ system that leverages DoReFa_Net [27]. Our ODQ system can dynamically

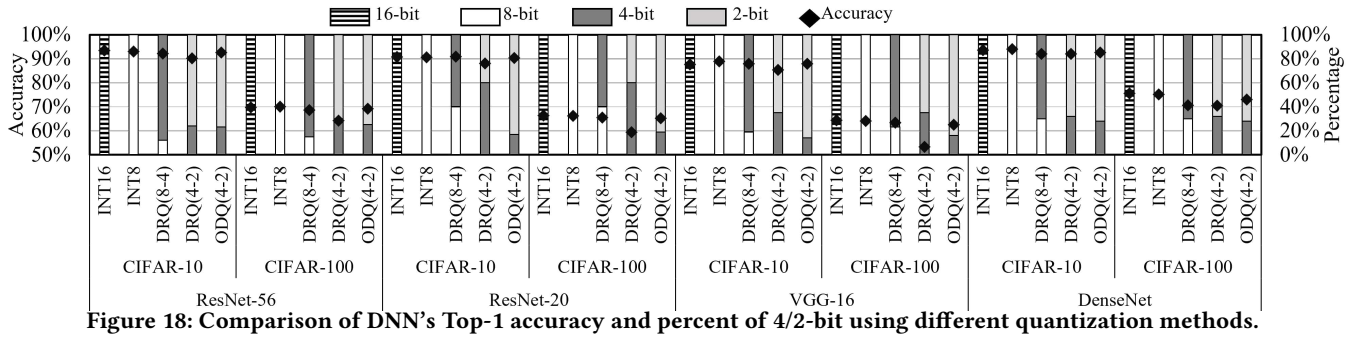

Figure 18: Comparison of DNN's Top-1 accuracy and percent of 4/2-bit using different quantization methods.

Table 2: Configuration of DNN accelerators.

	INT16	INT8	DRQ	ODQ
#PEs	120	1692	1692	4860
Bitwidth	INT16	INT4	INT4	INT2
area (mm^2)	0.17	0.17	0.17	0.17
On-chip memory (MB)	5	5	5	5

adjust the precision of inputs and weights at runtime. Although we explore 4-bit and 2-bit dynamic quantization in the implementation, ODQ is not limited to 4-bit and 2-bit quantization and can be easily extended to support other types of precision, e.g., INT8, etc.

We use CIFAR-10 and CIFAR-100 [12] in the experiments. The CIFAR-10 and CIFAR-100 datasets provide a representative set of images widely used in DNN and accelerator evaluation [3, 19, 20]. CIFAR-10 contains 60,000 images grouped in 10 categories, and CIFAR-100 consists of 60,000 images in 100 classes. We test a number of DNN models, including ResNet-56, ResNet-20, VGG16, and DenseNet [8, 9, 24]. VGG-16 is a classical DNN model, and ResNet and DenseNet contain shortcut connections. In this paper, the accuracy of ODQ-based DNN models is compared against that using INT16, INT8 static quantization, and DRQ (INT8 and INT4).

5.2 ODQ Accelerator

We have implemented the ODQ accelerator and synthesized it at the register transfer level (RTL) using Verilog. Experiments have been conducted to evaluate its performance, such as measuring the area of PE slices under different designs using the Design Compiler and a 45nm TSMC library. CACTI [14] is used to measure power consumption. We also employ Xilinx Vivado [23] to measure the execution time under different ODQ configurations (shown in Table 1). Furthermore, based on the collected data, we have developed a simulator to further analyze our ODQ accelerator. Specifically, we use Pytorch [1] to dump the binary mask maps for inference, which are then fed into our simulator to test a model's inference time.

Table 2 lists the configurations of the accelerators tested in our experiments. The accelerators have the same amount of on-chip memory for caching inputs and weights, which reduces the off-chip memory access latency. For fair comparison, we use the same area budget (i.e., $0.17mm^2$) for ODQ and other state-of-the-art accelerators in the literature. We compare our 4/2-bit ODQ with: (1) INT16 [27], in which weights and inputs are quantized to 16-bits using DoReFa-Net, a static quantization method; (2) INT8 [27], in which weights and inputs are quantized to 8-bits using DoReFa-Net; and (3) DRQ [20], which is a fine-grained input-directed dynamic quantization approach that quantizes the weights and inputs in a network according to input sensitivity.

6 EXPERIMENTAL RESULTS

6.1 DNN Accuracy Results

We evaluate the accuracy of our ODQ system using ResNet-56, ResNet-20, VGG-16, and DenseNet on the CIFAR-10 and CIFAR-100 datasets. The results are shown in Figure 18. Overall, ODQ achieves classification accuracy comparable to DRQ, INT8 DoReFa-Net, and INT16 DoReFa-Net. Compared with INT8-INT4 DRQ, ODQ exhibits negligible accuracy degradation, i.e., ≤0.6%. Moreover, DRQ using INT4-INT2 suffers from high accuracy degradation, ranging from 2.5% to 5%, with the greatest degradation reaching 10% by VGG-16 on CIFAR-100. This is because DRQ introduces noise into computing sensitive output features, as discussed in Section 2. The precision loss may not be that significant for INT8-INT4 DRQ. However, when using low-bitwidth representations, such as INT4-INT2, the impact on precision is considerable, resulting in a deteriorating effect on the model accuracy. In contrast, our output-directed ODQ does not suffer from precision loss and maintains the accuracy of DNN models.

ODQ may incorporate an insensitive output from a previous layer. However, since the output in the current layer is sensitive, the insensitive output from the preceding layer is calculated with high precision (4-bit). In contrast, DRQ only utilizes the high 2-bit (low precision) of the insensitive input, which is the output from the preceding layer, when computing the sensitive output. Our experimental results confirm that ODQ introduces lower precision loss compared to DRQ. When applied to ResNet-20 on CIFAR-10, ODQ gets precision loss for each layer as follows: C1: 0.08, C2: 0.1, C3: 0.04, C4: 0.07, C5: 0.06, C6: 0.04, C7: 0.07, C8: 0.07, C9: 0.02, C10: 0.04, C11: 0.02, C12: 0.02, C13: 0.03, C14: 0.06, C15: 0.04, C16: 0.05. These results demonstrate that ODQ achieves significantly lower precision loss in almost all layers, ensuring accurate model preservation. This is in contrast to the higher precision loss observed in Figure 3 by DRQ.

In addition, we characterize the percentage of high-order bits and low-order bits in different quantization frameworks using ResNet-56, ResNet-20, VGG-16, and DenseNet on the CIFAR-10 and CIFAR-100 datasets. The results are also shown in Figure 18. Both DRQ and ODQ benefit from low-bit quantization, such as INT4-INT2 bits. Input-directed quantization approaches, like DRQ, introduce noise into the sensitive output, reducing accuracy significantly. The accuracy results show that output-directed DRQ frameworks perform

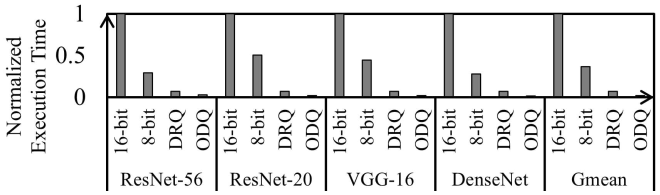

Figure 19: Normalized execution time of four DNNs using different accelerators.

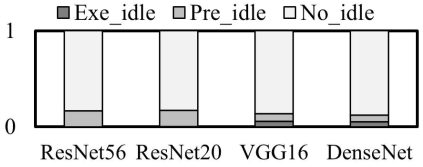

Figure 20: Percentage of idle PEs with ODQ.

better than input-directed DRQ frameworks with a comparable or even lower percentage of high-precision output features (INT4).

6.2 Performance Analysis

Deep learning workloads are time-consuming. To determine the extent of ODQ performance improvement, we evaluate the execution times of four DNNs (ResNet-56, ResNet-20, VGG-16, and DenseNet) on various accelerators. Figure 19 shows the results. Because it incorporates a greater percentage of INT4 precision computation (25% compared to 18% for ResNet-20), the performance increase of ResNet-56 (97% compared to INT16 DoReFa-Net) is lower than that of ResNet-20 (97.7% compared to INT16 DoReFa-Net), balancing the advantage of the extra layers. Overall, ODQ outperforms INT16 DoReFa-Net by 97.8%. This is because ODQ computes with INT4-INT2 precision rather than INT16. When compared to INT8 DoReFa-Net, ODQ improves performance by 95.8% on average. Furthermore, when compared to input-directed quantization with INT8-INT4 mix precision, ODQ shows a performance boost of 67.6%.

Futhermore, we evaluate the PE utilization in the reconfigured ODQ accelerator and Figure 20 presents the results. The figure demonstrates a notable decrease in the percentage of idle PEs, resulting in a significant performance improvement. The highest PE idleness observed is 18%, which is in stark contrast to the 50% idleness observed with the static PE allocation scheme (as shown in Figure 11).

6.3 Energy Efficiency

We evaluate energy savings by using ODQ. Figure 21 illustrates the energy consumption by ODQ compared with INT16 DoReFa-Net, INT8 DoReFa-Net, and DRQ measured by CACTI [14]. As can be seen, ODQ helps reduce energy usage for all four DNNs we evaluated (ResNet-56, ResNet-20, VGG-16 and DenseNet). Compared to INT16 DoReFa-Net, ODQ reduces energy consumption by 97.6% on average. Furthermore, when compared to INT8 DoReFa-Net, ODQ saves 93.5% of energy. When compared to DRQ, a state-of-the-art dynamic quantization framework, ODQ saves 66.9% of energy.

We analyze the energy consumption of ODQ's three key components: DRAM, Buffer (input, weight, and output buffer), and Cores

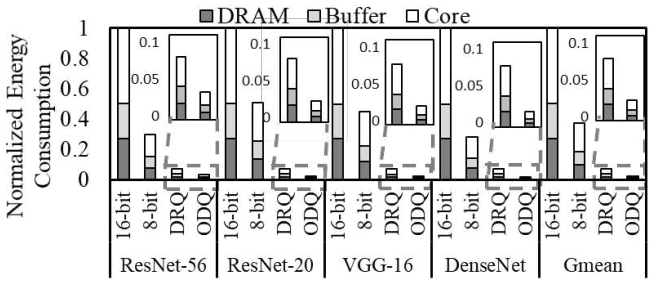

Figure 21: Normalized energy consumption of four DNNs using different accelerators.

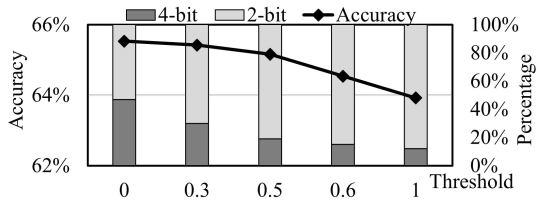

Figure 22: Threshold analysis (ResNet-20).

Table 3: Threshold used in this work.

NN Model	Threshold
ResNet-56	0.5
ResNet-20	0.5
VGG-16	0.3
DenseNet	0.05

(PE slices) to have a better understanding of its energy efficiency, as shown in Figure 21. As can be observed, all of these components (i.e., DRAM, Buffer, and PE slices) help reduce energy consumption. To elaborate, DRAM, Buffer, and PE slices help in the reduction of DNN execution time, which accounts for static energy consumption. The dynamic energy is mainly saved by the PE Slices. ODQ uses low-bitwidth calculation in addition to minimizing the number of operations required between low-bitwidth operators.

In this paper, we evaluate the performance of ODQ on multiple DNNs and find that it offers significant advantages in both performance and accuracy, particularly for DNNs that include convolutional layers. These layers often contribute to the majority of execution time and power consumption. ODQ specifically optimizes convolution operations across different types of DNNs.

6.4 Threshold Analysis

The threshold affects the percentage of sensitive output features that are directly connected to performance and energy efficiency. A higher threshold identifies fewer sensitive output features, leading to substantial performance gains. The threshold is closely tied to the accuracy of DNN models, with a higher threshold causing significant accuracy loss. Therefore, finding a suitable threshold is crucial for balancing accuracy and performance. The influence of threshold on accuracy and the percentage of high(INT4)/low(INT2) precision calculations is evaluated in a real-world DNN model (ResNet-20), and the result is illustrated in Figure 22.

Although the threshold has a negative impact on accuracy, it is beneficial to overall performance. Increasing the threshold from

0 to 1 reduces accuracy by 1.8% while increasing insensitive output features by about 40%. Based on these results, we determine that a threshold of 0.5 optimally balances classification accuracy preservation and performance maximization for ResNet-20. Table 3 lists the thresholds for the DNN models employed in this study and demonstrate that the optimal threshold varies per DNN model. In the same DNN model, we use the same threshold across all layers, which greatly simplifies the design.

The threshold determination time for low-precision data is considered acceptable in this study. We have trained the model three times each for Resnet-20, Resnet-56, and VGG-16, and four times for DenseNet to determine the thresholds. As the threshold is determined offline, time consumption is not a primary concern in this paper. Instead, our focus is on inference time, with a particular emphasis on the convolutional layers. These layers make up the majority of the inference time.

7 RELATED WORK

7.1 Uniform Interval Quantization

Uniform quantization includes binary, ternary, and fixed-point quantization. Binary quantization restricts values to -1 and 1. As such, multiply-and-accumulate operations can be replaced by accumulation operations, resulting in a drastic reduction in computation and memory usage. Binarized neural networks that quantize both inputs and weights can further reduce computation by replacing addition operations with bitwise operations [6]. Ternary quantization includes 0, which requires an extra bit. On the other hand, it increases the expressive power and retains the benefit of not having multiplication in binary weight networks since multiplication and accumulation are not needed for 0.

7.2 Non-Uniform Interval Quantization

Weights, however, are not distributed uniformly. Instead, they follow a Gaussian distribution centered around 0. Power-of-2 quantization, a non-uniform quantization scheme, has higher precision near 0 and lower precision at the ends [13, 26]. This better reflects the distribution of weights. Quantizing weights as powers of 2 replaces multiplications with efficient bit shifts. However, power-of-2 quantization primarily improves precision near 0 and lacks sufficient precision for the tails.

In order to address this flaw, Chang et al. proposed the sum-of-power-of-2 (SP2) quantization scheme [3]. The weights are quantized to a number equivalent to the sum of two powers of 2, which are multiplied with the activations independently, allowing bit shifts to be used instead of multiplication. SP2 quantization is also more precise near 0 and less precise near the ends, but the difference in precision is less extreme compared to that in power-of-2 quantization. As such, it better fits the distribution of the weights.

8 CONCLUSIONS

We present a novel method for deep learning acceleration. ODQ dynamically performs high- and low-precision computations based on the sensitivity of output features. The prediction phase uses the high-order 2 bits of an input and weight. If the result is predicted to be sensitive, the execution phase performs the remaining computation, while the remaining computation is skipped if the output is predicted to be insensitive. We have designed and implemented a reconfigurable deep neural network accelerator based on ODQ. Our experimental results demonstrate that ODQ can significantly improve performance and reduce power consumption for deep learning applications, making it a promising solution for high-performance and energy-efficient deep learning systems.

ACKNOWLEDGMENTS

This work has been supported in part by the U.S. NSF grants CNS-2231519, CNS-2113805, CNS-1852134, OAC-2017564, ECCS-2010332, CNS-2037982, DUE-2225229, and CNS-1828105. We thank the reviewers for their constructive comments, which helped us improve this paper.

REFERENCES

[1] 2018. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. In *https://github.com/pytorch*.
[2] David Bau and et. al. 2017. Network dissection: Quantifying interpretability of deep visual representations. In *CVPR*.
[3] Sung-En Chang and et. al. 2021. Mix and Match: A novel FPGA-centric deep neural network quantization framework. In *HPCA*.
[4] Tianshi Chen and et. al. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News* (2014).
[5] Xiaozhi Chen and et. al. 2017. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*.
[6] Matthieu Courbariaux and et. al. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* (2016).
[7] Li Deng and et. al. 2013. Recent Advances in Deep Learning for Speech Research at Microsoft. In *ICASSP*.
[8] Kaiming He and et. al. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.
[9] G. Huang and et. al. 2017. Densely connected convolutional networks. In *CVPR*.
[10] Beilei Jiang and et. al. 2022. MLCNN: Cross-Layer Cooperative Optimization and Accelerator Architecture for Speeding Up Deep Learning Applications. In *IPDPS*.
[11] Norman P. Jouppi and et. al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *ISCA '17*.
[12] Alex Krizhevsky. 2010. CIFAR-10 and CIFAR-100 datasets. In *https://www.cs.toronto.edu/ kriz/cifar.html*.
[13] Daisuke Miyashita and et. al. 2016. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025* (2016).
[14] N. Muralimanohar and et. al. 2009. CACTI 6.0: A tool to model large caches. In *HP laboratories*.
[15] Eunhyeok Park and et. al. 2018. Energy-efficient neural network accelerator based on outlier-aware low-precision computation. In *ISCA*.
[16] S Preethi and et. al. 2020. Smart Healthcare Monitoring System for War-End Soldiers Using CNN. IGI Global.
[17] Marco Sandri and et. al. 2006. Variable selection using random forests. In *Data analysis, classification and the forward search*.
[18] Murugan Sankaradas and et. al. 2009. A massively parallel coprocessor for convolutional neural networks. In *ASAP*.
[19] Hardik Sharma and et. al. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *ISCA*.
[20] Z. Song and et. al. 2020. Drq: dynamic region-based quantization for deep neural network acceleration. In *ISCA*. IEEE.
[21] Zhuoran Song and et. al. 2020. VR-DANN: Real-Time Video Recognition via Decoder-Assisted Neural Network Acceleration. In *MICRO*.
[22] Mengshu Sun and et. al. 2022. FILM-QNN: Efficient FPGA Acceleration of Deep Neural Networks with Intra-Layer, Mixed-Precision Quantization *(FPGA '22)*.
[23] Xilinx. 2020. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_2/ug888-vivado-design-flows-overview-tutorial.pdf.
[24] Xiangyu Zhang and et. al. 2015. Accelerating Very Deep Convolutional Networks for Classification and Detection. In *CoRR*.
[25] Shixuan Zheng and et. al. 2018. An efficient kernel transformation architecture for binary-and ternary-weight neural network inference. In *DAC*.
[26] Aojun Zhou and et. al. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044* (2017).
[27] Shuchang Zhou and et. al. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).