SmartDeal: Re-Modeling Deep Network Weights for Efficient Inference and Training

Xiaohan Chen*, Yang Zhao*, Yue Wang, Pengfei Xu, Haoran You, Chaojian Li, Yonggan Fu, Yingyan Lin, and Zhangyang Wang

Abstract—The record-breaking performance of deep neural networks (DNNs) comes with heavy parameter budgets, which leads to external dynamic random-access memory (DRAM) for storage. The prohibitive energy of DRAM accesses makes it non-trivial for DNN deployment on resource-constrained devices, calling for minimizing the movements of weights and data in order to improve the energy efficiency. Driven by this critical bottleneck, we present SmartDeal, a hardware-friendly algorithm framework to trade higher-cost memory storage/access for lower-cost computation, in order to aggressively boost the storage and energy efficiency, for both DNN inference and training.

The core technique of SmartDeal is a novel DNN weight matrix decomposition framework with respective structural constraints on each matrix factor, carefully crafted to unleash the hardware-aware efficiency potential. Specifically, we decompose each weight tensor as the product of a small basis matrix and a large structurally sparse coefficient matrix whose non-zero elements are readily quantized to power-of-2. The resulting sparse and readily-quantized DNNs enjoy greatly reduced energy consumption in data movement as well as weight storage, while incurring minimal overhead to recover the original weights thanks to the required sparse bit-operations and cost favorable computations. Beyond inference, we take another leap to embrace energy-efficient training, by introducing several customized techniques to address the unique roadblocks arising in training while preserving the SmartDeal structures. We also design a dedicated hardware accelerator to fully utilize the new weight structure to improve the real energy efficiency and latency performance.

We conduct experiments on both vision and language tasks, with nine models, four datasets, and three settings (inference-only, adaptation, and fine-tuning). Our extensive results show that: 1) being applied to inference, SmartDeal achieves up to 2.44× improvement in energy efficiency as evaluated via real hardware implementations; 2) being applied to training, SmartDeal can lead to 10.56× and 4.48× reduction in the storage and the training movement cost, respectively, with usually negligible accuracy loss, compared to state-of-the-art training baselines. Our source codes are available at: https://github.com/VITA-Group/SmartDeal.

Index Terms—Efficient Machine Learning, Deep Network Training, Data Movement, Hardware Accelerator

I. INTRODUCTION

A. Background and Core Idea

The performance breakthrough of deep neural networks (DNNs) motivates a growing demand to bring DNNs into storage- and energy-constrained edge devices, such as mobile phones, wearables, and IoT sensors, using domain-specific accelerators. However, the excellent performance comes at a heavy parameter cost, which needs external dynamic random-access memory (DRAM) for storage. The prohibitive energy of DRAM accesses makes DNN deployment on resource-constrained devices non-trivial. Take the accelerator in [1] as an example: >95% of the energy is consumed by DRAM. Thus, it is crucial to minimize weights and data movements in order to improve the energy efficiency of DNNs.

We present a holistic algorithm-hardware co-design framework, called SmartDeal, to aggressively reduce both the energy consumption of data movement and the storage for weights: the two major limiting factors for DNN on-device deployment. Our underlying philosophy is to seek a “smart deal”: trading the more “expensive” memory storage/access for “cheaper” computation, to eliminate the dominant data movement cost. Our technical contributions highlight three major aspects: (1) inference algorithm: whose core is a novel SmartDeal weight representation and a unified optimization framework; (2) inference hardware accelerator, which is co-designed to maximize the benefit of our inference algorithm; and (3) training algorithm, which extends the SmartDeal benefits to energy-efficient training in non-trivial ways. We will go through each of them with more details in Section I-B.

The paper significantly extends our previous conference version [2]. First and Foremost, our prior work [2] only covers the inference algorithm and its hardware accelerator, while this current work proposes SmartDeal-compatible training techniques for the first time, exploring the new horizon of energy-
efficient training. We also extensively benchmark the efficient training performance, in two common edge-based training settings (adaptation and fine-tuning), using a training-specific hardware accelerator. Second, we offer a comprehensive suite of ablation studies to carefully investigate the impact of each proposed component in the SmartDeal weight decomposition. Besides, more datasets and application scenarios are included in experiments, such as a language modeling dataset.

B. Overview of Major Technical Contributions

We start by introducing SmartDeal for inference, whose core is a new weight representation: a layer-wise weight matrix is decomposed as the product of a small basis matrix and a large coefficient matrix (see Fig. 1). We simultaneously enforce two important properties on the latter coefficient matrix:

- **(Structurally) sparse**: most elements are zero. Besides element-wise sparsity, we also exploit structured sparsity \[3\] for more hardware-friendliness.
- **Specially quantized**: the non-zero elements take only power-of-2 values, which not only have compact bit representations, but also (more importantly) turn the dot-product multiplication with the non-zero elements into much lower-cost shift-and-add operations.

It is worth emphasizing that we never claim to have invented any new weight factorization, or quantization, or structured sparsity algorithm in this paper. The essential novelty of SmartDeal lies in its unique angle and motivation, as well as the unified optimization framework dedicated to achieving our goal. The core theme of SmartDeal is trading higher-cost memory storage/access for lower-cost computation: this is beyond simply compressing weights to save storage or FLOPs. Our weight re-structure could be interpreted as an innovative, well-motivated integration of sparsification (or pruning), factorization, and quantization on DNN weights, that can be solved under one unified optimization algorithm.

To fully unleash SmartDeal algorithm’s potential, we further develop a dedicated DNN inference accelerator that takes advantage of the much reduced weight storage and readily-quantized weights resulting from the algorithm to enhance hardware acceleration performance. Experiments show that the proposed accelerator outperforms state-of-the-art DNN inference accelerators in terms of acceleration energy efficiency and latency by up to 6.7× and 19.2×, respectively.

We then take another big leap to extend SmartDeal from inference to DNN training. The current practice of edge-based training typically starts from a pre-trained and pre-loaded model, and then continues tuning the model with more data collected from the same training domain (denoted as fine-tuning), or from a different new domain (denoted as adaptation) due to customization or personalization \[4\], \[5\], \[6\]. Although fine-tuning or adaptation costs much less compared to training from scratch, their large resource consumption stands at odds with the limited computing and energy resources at the edge \[7\]. To enable SmartDeal for energy-efficient on-device training, we aim to preserve the SmartDeal re-structured weight form during training. To do so, we have to cope with two roadblocks in the resulting optimization:

- The weights, and therefore their sparse coefficient matrices, keep changing during training: that was known to be very hardware-unfriendly. To deal with this challenge, we perform a SmartDeal decomposition on the pre-trained weight initializations, and then maintain the (structured) sparsity map (i.e., the locations of nonzero elements) unchanged during training, while their magnitudes can be updated. This greatly saves memory access and energy overhead with little impact on achievable performance \[8\].
- The other challenge arises from our enforced structure that non-zero coefficients could only take discrete values (power-of-2). The gradient descent cannot be directly applied to the discrete domain; meanwhile it is highly inefficient to keep a copy of continuous/higher-precision latent weight for updating. Instead, we design a special and light-weight update rule for the coefficient matrix, that turns floating-point additions into quantization bucket switches, to be decided by gradient signs \[8\].

We evaluate SmartDeal on one state-of-the-art training accelerator \[9\] with necessary modifications to demonstrate the generality where SmartDeal achieves up to 4.48× improvement in energy efficiency over state-of-the-art competitors for training, at the negligible accuracy losses.

II. BACKGROUND AND RELATED WORK

A. A Motivating Example for SmartDeal

Table I shows the unit energy cost of accessing different-level memories with different storage capacities and computing an MAC/multiplication/addition (the main computation operation in DNNs) designed in a commercial 28nm CMOS technology. We can see that the unit energy cost of memory accesses is much higher (\(\geq 9.5\times\)) than that of the corresponding MAC computation. Therefore, it is promising in terms of more efficient acceleration if we can potentially enforce higher-order of weight structures to more aggressively trade higher-cost memory accesses for lower-cost computations, motivating our SmartDeal idea. That is, the resulting higher structures in DNN weights’ decomposed matrices, e.g., \(C_e\) in Figure 1 will enable much reduced memory accesses at a cost of more computation operations (i.e., shift-and-add operations in our design), as compared to the vanilla networks.

|                | DRAM | SRAM | MAC  | multiplier | adder |
|----------------|------|------|------|------------|-------|
| Energy (pJ/8bit) | 100  | 1.36–2.45 | 0.143 | 0.124 | 0.019 |

\[1\]While it is not directly applicable to training from random scratch, such setting is unlikely in resource-constrained training.

B. Basics of Deep Neural Networks

Modern DNNs usually consist of a cascade of multiple convolutional (CONV), pooling, and fully-connected (FC)
layers through which the inputs are progressively processed. The CONV and FC layers can be described as:

\[
O(c_o)[e][f] = \\
C.R.S \\
\sigma \left( \sum_{c_i,k_r,k_s} W[c_o][c_i][k_r][k_s] \cdot I[c_i][eU+k_r][fU+k_s] + B[c_i] \right),
\]

where \( W, I, O, \) and \( B \) denote the weights, input activations, output activations, and biases, respectively. In the CONV layers, \( C \) and \( M, E \) and \( F, R \) and \( S \), and \( U \) stand for the number of input and output channels, the size of input and output feature maps, and the size of weight filters, and stride, respectively; while in the FC layers, \( C \) and \( M \) represent the number of input and output neurons, respectively; with \( \sigma \) denoting the activation function, e.g., a ReLU function \( (ReLU(x) = \max(x, 0)) \). The pooling layers reduce the dimension of feature maps via average or max pooling. The recently emerging compact DNNs (e.g., MobileNet [10] and EfficientNet [11]) introduce depth-wise CONV layers and squeeze-and-excite layers which can be expressed in the above description as well [12].

C. Overview of Efficient Deep Learning

To reduce the large quantity of weight parameters, numerous DNN compression techniques have been proposed to shrink the weight redundancy and accelerate the inference, including matrix decomposition [13], [14], quantization [15], [16], [17], pruning [13], [19], [8], knowledge distillation [20], [21], [17], and dynamic inference [22], [23], [24], [25]. Combinations of two techniques have also been studied, e.g. decomposition with pruning [26], distillation with quantization [27], [28], and distillation with pruning [29]. Some latest works [30], [31] start to combine and jointly optimize three compression ideas, but for different motivations and applications (e.g., compressing robust models, and GANs). To our best knowledge, SmartDeal features a new joint formulation that combines the three ideas of weight pruning, matrix decomposition, and power-of-2 quantization: a unification never being considered easily applied to weight tensors in CONV layers and squeeze-and-excite layers which can be expressed in the above description as well [12].

D. Compression-Aware DNN Accelerators

In general, the three typical compression approaches, weight factorization, data quantization, and weight sparsification, have been exploited in DNN accelerators design to boost energy efficiency. [14] demonstrate DNNs with tensorized factorization using ASIC. For the weight sparsification accelerators, [37], [38], [19] have been proposed. Quantization is widely used by inference accelerators [37], [38], [14], [19]. In comparison, our proposed SmartDeal inference accelerator also unifies three techniques to simultaneously shrink the memory footprint and simplify the computations when recovering the weight matrix during runtime.

III. SmartDeal for Efficient Inference

In this section, we introduce the weight decomposition structure in SmartDeal, as can be naturally applied to feedforward inference to reduce energy and time consumption of data movement as well as storage.

A. Problem Formulation

Given a weight matrix \( W \in \mathbb{R}^{m \times n} \), we seek to decompose it as the product of a coefficient matrix \( C_e \in \mathbb{R}^{m \times r} \) and a basis matrix \( B \in \mathbb{R}^{r \times n} \) where \( r \leq \min\{m, n\} \), such that

\[
W \approx C_e \cdot B.
\]

In practice, \( n \) is usually set to be very small (thus small \( B \)), and \( m \gg n \) (thus much larger \( C_e \)). Here we assume a 2D weight matrix in a FC layer as example for the simplicity of notation. We will later show that SmartDeal algorithm can be easily applied to weight tensors in CONV layers.

In addition to suppressing the reconstruction error (often defined as \( \|W - C_e B\|_F^2 \)), we expect the decomposed matrix factors to display more favorable structures for compression/acceleration. For the much larger \( C_e \), we enforce the following two structures simultaneously: (i) \( C_e \) needs to be highly sparse (a typical goal of pruning); and (ii) the non-zero elements in \( C_e \) are exactly the powers of 2, so that their bit representations can be very compact and their involved multiplications to rebuild the original weights from \( B \) and \( C_e \) are simplified into extremely cheap shift-and-add operations. As a result, instead of storing the whole weight matrix, the new structure requires storing only a very small \( B \); and a large, yet highly sparse and readily quantized \( C_e \). We call this process SmartDeal (SD) decomposition and the resulting \{\( C_e, B \)\} pair the SD form of \( W \).
SmartDeal decomposition hence can be written as the following constrained optimization:

\[
\arg\min_{C_e,B} \|W - C_eB\|^2_F \quad \text{subject to} \sum_{j} \|C_e[:,j]\|_0 \leq S_e, \quad (3)
\]

where \(\Omega_p \equiv \{0, \pm 2^p | p \in P\}\) with \(P\) being a chosen integer set that includes the possible degrees of the power-of-2 numbers and has cardinality no more than \(N_p\), i.e. \(\|P\| \leq N_p\). In Eq. (3), \(S_e\) controls the total number of non-zero elements in \(C_e\), i.e. the sparsity, while \(N_p\) controls the bit-width required to represent an element in \(C_e\). An innovative assumption of SD is to require non-zero elements in \(C_e\) to take one of a few pre-defined, specifically-picked discrete values. That is different from previous compression using weight clustering, whose quantized values are adaptively learned from data \cite{39, 40}. This special design is to facilitate (1) compact storage, and more crucially (2) extremely cheap weight reconstruction using only shift-and-add operations - the latter cannot be fulfilled by other arbitrary quantization.

**B. SmartDeal Decomposition Algorithm**

Solving Eq. (3) is non-trivial due to the nonconvex and integer set constraints. We propose a coordinate descent-type algorithm that iterates between objective fitting and feasible set projection, as outlined in Algorithm 1 and the explanation of three key steps to be iterated are discussed follows. Here, \(\delta(C_e)\) is the difference between two iterates of \(C_e\), \(tol\) is a small number indicating the convergence of the algorithm, and \(\text{max\_iter}\) is the max number of iterations.

**Algorithm 1 SmartDeal decomposition algorithm.**

1. Initialize \(C_e\) and \(B\); \(k = 0\)
2. While \(\|\delta(C_e)\| \geq \text{tol}\) or \(k < \text{max\_iter}\):
   3. \textbf{Step 1:} Quantizing \(C_e\) to powers of 2;
   4. \textbf{Step 2:} Fitting \(B\) and \(C_e\);
   5. \textbf{Step 3:} Promoting (structured) sparsity \(C_e\);
   6. \(k = k + 1\);
7. Re-quantize \(C_e\) and re-fit \(B\).

We empirically find that simply initializing \(C_e = W\) and \(B = I\) can produce robust and good performing decomposition results. Hence, we use this initialization in all experiments. After the initialization, we iteratively perform the following three steps to gradually obtain better decompositions.

**Step 1: Quantizing \(C_e\).** The quantization step projects the nonzero elements in \(C_e\) to \(\Omega_p\). Specifically, we will first normalize each column in \(C_e\) to have a unit norm in order to avoid scale ambiguity. We will then round each non-zero element to its nearest power-of-two value. We define \(\delta(C_e)\) to be the quantization difference of \(C_e\).

**Step 2: Fitting \(B\) and \(C_e\).** We will first fit \(B\) by solving \(\arg\min_B \|W - C_eB\|^2_F\), and then fit \(C_e\) by solving \(\arg\min_{C_e} \|W - C_eB\|^2_F\). When fitting either one, the other is fixed to its current updated value. The step simply deals with two unconstrained least squares.

**Step 3: Sparsifying \(C_e\).** We then prune the non-zero elements in \(C_e\) with smallest magnitudes to promote more sparsity. In practice, we use hard thresholds for element-wise and structured sparsity to zero out small magnitudes in \(C_e\) for implementation convenience. When promoting element-wise sparsity in \(C_e\), as sparsity level \(S_e\) usually needs to be manually adjusted for each layer, we instead use a heuristic threshold \(\theta\) to zero out elements. The structured sparsity is discussed in details in III-D.

After sufficiently iterating between the above three steps (i.e., quantization, fitting and sparsification), we conclude the iterations by re-quantizing the nonzero elements in \(C_e\) to ensure \(C_e[i,j] \in \Omega_p\) and then re-fitting \(B\) with the updated \(C_e\). An example of how the \(C_e\) and \(B\) matrices evolve along the iterations is given in Appendix A.

**C. Applying the SmartDeal Algorithm to DNNs**

1) **SmartDeal algorithm as post-processing:** The selection of the dimensions of the coefficient matrix \(C_e\) and the basis matrix \(B\) is a design knob of SmartDeal for trading-off the achieved compression rate and model accuracy, i.e., a smaller \(r\) (see notations in III-A) favors a higher compression rate yet might cause a higher accuracy loss. Note that \(r\) is equal to the rank of the basis matrix \(B\), i.e., \(r = n\) when \(B\) is a full matrix, otherwise \(r \leq n\). To minimize the memory storage, we set the basis matrix \(B \in \mathbb{R}^{2 \times n}\) to be small. In practice, we choose \(n = R = S\) with \(R \times S\) being the CONV kernel size. Since \(n\) is small, we choose \(r = n = S\) too. We next discuss applying the proposed algorithm to the FC and CONV layers.

- **SmartDeal on FC layers.** Consider a fully-connected layer \(W \in \mathbb{R}^{M \times C}\). We reshape each row of \(W\) into a new matrix \(W_i \in \mathbb{R}^{C/S \times S}\), and then apply SmartDeal algorithm. Specifically, zeros are padded if \(C\) is not divisible by \(S\), and SmartDeal algorithm is applied to \(W_i\), where \(i = 1, \ldots, M\). When \(C \gg S\), the reconstruction error might tend to be large due to the imbalanced dimensions. We alleviate it by slicing \(W_i\) into smaller matrices along the first dimension.

- **SmartDeal on CONV layers.** Consider a convolutional layer \(W\) in the shape \((M, C, R, S)\). Case 1: \(R = S > 1\). We reshape the \(M\) filters in \(W\) into matrices of shape \((S \times C, S)\), on which SmartDeal algorithm is applied. The matrices can be sliced into smaller matrices along the first dimension if \(S \times C \gg S\). Case 2: \(R = S = 1\). The weight is reshaped into a shape of \((M, C)\) and then is treated the same as an FC layer.

The above procedures are easily parallelized along the axis of the output channels for acceleration.

Applying SmartDeal algorithm to a VGG19 network\textsuperscript{2} pretrained on the CIFAR-10 \textsuperscript{11}, with \(\theta = 4 \times 10^{-3}\), \(tol = 10^{-10}\) (introduced in Section III-B), and a maximum iteration of 30, the accuracy drop in the validation set is as small as 3.21\% with an overall compression rate of over 10\times without retraining after the decomposition. The overall compression rate of a network is defined as the ratio between the total number

\textsuperscript{2}https://github.com/chengyangfu/pytorch-vgg-cifar10
of bits to store the weights (including the coefficient matrix \( C_r \), basis matrix \( B \), and encoding overhead) and the number of bits to store the original FP32 weights.

2) Enhancing Accuracy with Re-Training: After a DNN has been post-processed by \emph{SmartDeal} algorithm, a re-training step can be used to remedy the accuracy drop. As the un-regularized re-training will break the desired property of coefficient matrix \( C_r \), we take an empirical approach to alternate between 1) re-training the DNN for one epoch; and 2) applying \emph{SmartDeal} algorithm to ensure the \( C_r \) structure. The default iteration number is 50 for CIFAR-10 [41] and 25 for ImageNet [42]. As shown in the ablation experiments in [VI-B], the alternating re-training process improves the accuracy while maintaining the favorable weight structure. More analytic solutions will be explored in future work, e.g., incorporating \emph{SmartDeal} as a regularization term [40].

3) Time complexity of \emph{SmartDeal}: \emph{SmartDeal} algorithm is efficient in terms of running time with fully parallelized implementation. In practice, running one pass of the parallelized \emph{SmartDeal} algorithm takes less than 30 seconds for VGG-19 and ResNet-19 networks and less than 2 minutes for ResNet-50. In combined with the alternating re-training approach mentioned above, the computational cost of \emph{SmartDeal} is negligible compared to the cost for training on data. The above running time is measured on Intel Xeon Platinum 8168 CPU platform (we only implement \emph{SmartDeal} on CPU currently).

D. \emph{SmartDeal} with Structured Sparsity

Customized accelerators for sparse models can utilize the sparsity to reduce the associated computations and memory accesses [32, 33]. However, the irregularity in element-wise/unstructured sparse models prevents hardware from fully leveraging reduction. Coarse-grained sparsity brings more regular sparsity pattern, making it easier for hardware acceleration [43]. Hereby, when we design the dedicated hardware accelerator for \emph{SmartDeal}, we introduce two types of structured sparsity, \emph{channel-wise} and \emph{vector-wise} sparsity, to \( C_r \):

- We first prune channels whose corresponding scaling factor in batch normalization layers is lower than a threshold which is manually controlled for each layer. In practice, we only apply channel-wise sparsifying at the first training epoch once, given the observation that the pruned channel structure will not change much.
- We then zero out elements in \( C_r \) based on the magnitudes to meet the vector-wise sparsity constraint: \( \sum_{j} \| C_e[j, \cdot] \|_0 \leq S_c \), where \( S_c \) is manually controlled per layer. Structured sparsity being more regular and hardware efficient, it also brings much more aggressive constraints on the model capacity and thus more performance degradation. This is empirically verified in the ablation study in Section [VI-A].

Although vector-wise sparsity has been investigated in [44], we are the first to consider vector sparsity in a unified framework with quantization, sparsification and decomposition.

After we obtain \( C_e \) and \( B \) via \emph{SmartDeal} algorithm, we further desire storage-economic and hardware-friendly representations to store them on-device. We explain the encoding schemes of \( C_e \) and \( B \) in Appendix B. We also discuss more on the rationale behind our algorithm, in Appendix C.

IV. \emph{SmartDeal} for Efficient Training

Extending \emph{SmartDeal} to energy-efficient training is highly non-straightforward. As pointed out in Section II, the dynamic sparsity pattern and the discrete values in \( C_r \) constitute two grand challenges. Also, directly involving optimization like solving Eq. (3) into training is not acceptable due to its own complexity. We hereby discuss how to transplant the methodology of \emph{SmartDeal} to energy-efficient training, with two dedicated techniques presented to address the two challenges. The two key points of the proposed techniques are (1) to optimize the decomposed \( C_e \) and \( B \) matrices in the special discrete space constrained by sparsity and power-of-2 quantization so that the model preserves \emph{SmartDeal} structure and thus the economic memory access throughout training; (2) to avoid using any “shadow weights” i.e., the latent high-precision copies of weights that are actually trained and will introduce large overhead in model storage.

**Basic routine: SD-Training (SD-T).** Considering the practice of on-device learning, we assume a pre-trained DNN to start with, where the goal is either fine-tuning or adaptation. For a layer with pre-trained weight \( W \), we first perform one-pass SD to get the initial \( C_e \) and \( B \). We use \( C_e \) and \( B \) as hidden weights that will be used to reconstruct \( W \) runtime during a feed-forward pass. Note that the reconstruction step introduces little overhead but significantly lowers the data movement cost due to the fixed sparsity structure and the power-of-2 non-zero values in \( C_e \).

During the back-propagation pass, the gradient is passed back to the intermediate \( W \) as usual. The gradients of \( B \) and \( C_e \) are calculated with \( W \)'s gradient and matrix multiplications. We update \( B \) using the standard gradient descent. For updating \( C_e \), we explicitly require \( C_e \) to be within its original feasible domain: a (structurally) sparse matrix with power-of-2 non-zero elements. Such a challenging requirement is met thanks to the next two customized techniques.

#1. For \( C_e \) zeros: fixed sparsity mask. We fix the sparsity pattern in \( C_e \) throughout training: the initial zero entries in \( C_e \) are “frozen” to zero, while the initial non-zero entries can be updated to either zero or non-zero flexibly. Experiments show that such a fixation does not noticeably impact the tuned/adapted model accuracy. This fixed sparsity brings in two-fold advantages: 1) fixing the sparse pattern of \( C_e \) saves the gradient computations of its zero elements during training, which can not be skipped in classic pruning with a dynamic sparse pattern; 2) the static sparsity pattern can be utilized to avoid the dynamic indexing sparse weights and/or adjusting processing schedules. Extensive experiments in Tab. [VII] show that the fixed sparsity implementation is effective across different models and datasets for saving more energy.

#2. For \( C_e \) nonzeros: bucket switch updating. The next dilemma is on updating non-zero power-of-2 elements: if we update \( C_e \) using floating-point add operations, the floating-point numbers have to be recovered before updating incurring overheads. Instead, we propose a \emph{Bucket Switch} updating scheme for \( C_e \) non-zero updating: inspired by [3] showing that taking only gradient signs (i.e., 1-bit gradients) suffices to training DNNs, we refer to gradient signs to guide the switch
of non-zero values, from one discrete “bucket” to another.

Specifically, if the gradient w.r.t. a non-zero element (whose current value is $2^p$) is positive, then we will switch up its value from $2^p$ to $2^{p+1}$ (or no change if $p = P$ already reaches the upper range bound). Similarly, a negative gradient will switch $2^p$ down to $2^{p-1}$, and a zero gradient will not change it. In this way, the non-zero element in $C_e$ is directly updated over the discrete domain, without any overhead of floating-point number operations. Notice that, we never aggregate high-precision updates, and there also exists no high-precision latent weight for $C_e$ in our implementation, because such will go against our goal of saving storage and energy for efficient training. Instead, we only record update directions (signs) and accumulate using an integer counter. Once the counter’s integer record passes a threshold, we “switch the bucket”.

While the above gradient quantization rule could suffer from high variance to learning rates, in practice, we apply two methods to mitigate the effect of noisy gradients. Before taking the sign, gradients whose magnitudes are below a threshold $\theta_g$ are zeroed out. We also adopt an “update delay and aggregation” scheme, in which we only really update an entry in $C_e$ (switch the bucket) when it receives switch signals in the same direction for enough times. The number of required times is controlled by an integer hyperparameter $\theta_e$. That is inspired by the lazy update and trajectory smoothing in gradient-based optimization [45, 46]: applying the similar idea to bucket switch is found to help training stability (as in gradient-based optimization [45, 46]: applying the similar idea to bucket switch is found to help training stability). The detailed description of this strategy is laid out in Appendix D. We note that a similar idea of “bucket switch” and “update decay and aggregation” was coincidentally found useful in another latest work on optimizing binary neural networks (BNNs) [47]. We leave more discussions on the potential theoretical underpinning for future work. We also apply the stochastic weight averaging (SWA) technique [45] that could stabilize training too without incurring noticeable overhead.

V. DEDICATED HARDWARE-ALGORITHM CODESIGN

In this section, we present our proposed SmartDeal accelerator. We first introduce the design principles and considerations (Section V-A) for fully making use of the proposed SmartDeal algorithm’s properties to maximize energy efficiency and minimize latency, and then describe the proposed accelerator (Section V-B) in details.

A. Design Principles and Considerations

**Minimizing overhead of rebuilding weights.** Thanks to the sparse and readily quantized coefficient matrices resulting from the SmartDeal algorithm, the memory storage and data movements associated with these matrices can be greatly reduced (see Table III, e.g., up to $80 \times$). Meanwhile, to fully utilize the advantages of the SmartDeal algorithm, the overhead of rebuilding weights should be minimized. To do so, it is critical to ensure that the location and time of the rebuilding units and process are properly designed. Specifically, for a SmartDeal accelerator 1) the rebuild engine (RE) that restores weights using both the basis matrix and corresponding weighted coefficients should be located closes to the PEs for minimizing the data movement costs of the rebuilt weights. 2) As the basis matrices are reused most frequently, the dataflow for these matrices should be weight stationary, i.e., once being fetched from the memories, they stay in the REs until all the corresponding weights are rebuilt.

**Taking advantage of the structured sparsity.** The enforced vector-wise sparsity in the SmartDeal algorithm’s coefficient matrices offers benefits of 1) vector-wise skipping both the memory accesses and computations of the corresponding activations (see Figure 2(a)) and 2) reduced coefficient matrix encoding overhead (see Figure 2(b)). Meanwhile, there is an opportunity to make use of the vector-wise/bit-level sparsity of activations for improving efficiency.
the percentage of the zero activation bits/rows over the total activation bits/rows. Figure 3 shows the bit-level sparsity of activations w/ and w/o 4-bit Booth encoding in popular DNNs, including VGG11, ResNet50, and MobileNetV2 on ImageNet, VGG19 and ResNet164 on CIFAR-10, and DeepLabV3+ on CamVid. We can see that the bit-level sparsity is 79.8% under an 8-bit precision and 66.0% using the corresponding 4-bit Booth encoding even for a compact model like MobileNetV2; for vector-wise sparsity, it can be widely observed among the CONV layers with $3 \times 3$ kernel size, e.g., up to 27.1% in the last several CONV layers of MobileNetV2 and up to 32.4% in ResNet164.

Support for compact models. The recently emerged compact models, such as MobileNet [10] and EfficientNet [11], often adopt depth-wise CONV and squeeze-and-excite layers other than the traditional 2D CONV layers to restrict the model size, which reduces the data reuse opportunities. Taking a depth-wise CONV layer as an example, it has an “extreme” small number of CONV channels (i.e., 1), reducing the input reuse over the standard CONV layers; for squeeze-and-excite, similar to that of FC layers, there are no weight reuse opportunities in squeeze-and-excite layers. On-device efficient accelerators should consider these features of compact models for their wide adoption and leveraging compact models for more efficient processing.

B. Architecture of the SmartDeal Accelerator

Architecture overview. Figure 4 (a) shows the architecture of the proposed SmartDeal accelerator which consists of a 3D PE array with a total of $dim_M$ PE slices, input/index/output global buffers (see the blocks named Input GB, Weight Index GB, and Output GB, where GB denotes global buffer) associated with an index selector for sparsity (see the blocks named Index sel.), and an controller. The accelerator communicates with an off-chip DRAM through DMA (direct memory access) [38]. Following the aforementioned design principles and considerations (see Section 4-A), the proposed accelerator features the following properties: 1) an RE design which is inserted within PE lines to reduce the rebuilding overhead (see the top part of Figure 4(b)); 2) a hybrid dataflow: an 1D row stationary dataflow is adopted within each PE line for maximizing weight and input reuses, while each PE slice uses an output stationary dataflow for maximizing output partial sum reuses; 3) an index selector (named Index Sel. in Figure 4(a)) to select the none-zero coefficient and activation vector pairs as inspired by [48]. This is to skip not only computations but also data movements associated with the sparse rows of the coefficients and activations. The index selector design in SmartDeal is the same as that of [48] except that Huffman encoding is used here and the index values of 0/1 stand for vector (instead of scalar) sparsity; 4) a data-type driven memory partition in order to use matched bandwidths (e.g., a bigger bandwidth for the weights/inputs and a smaller bandwidth for the outputs) for different types of data to reduce the unit energy cost of accessing the SRAMs which is used to implement the GB blocks [50]. We adopt separated centralized GBs to store the inputs, outputs, weights and indexes, respectively, and distributed SRAMs (see the Weight Buffer unit in Figure 4 (a)) among PE slices to store weights (including the coefficients and basis matrices); and 5) a bit-serial multiplier based MAC array in each PE line to make use of the activations’ bit-level sparsity together with a Booth Encoder as inspired by [49].

PE slices and dataflow. We here describe the design of the PE Slice unit in the 3D PE slice array of Figure 4 (a):

First, the 3D PE Slice array: our SmartDeal accelerator enables parallelized processing of computations associated with the same weight filter using the PE slice array of size $dim_M$ (with each PE slice having $dim_E$ PE lines) and $dim_E$ number of input channels, where the resulting partial sums are accumulated using the adder trees at the bottom of the PE lines (see the bottom right side of Figure 4(a)). In this way, a total of $dim_M$ consecutive output channels (i.e., $dim_M$ weight filters) are processed in parallel to maximize the reuse of input activations. Note that this dataflow is employed to match the...
way we reshape the weights as described in Section III.C.

Second, the PE line design: each PE line in Figure 4 includes an array of \( dim_F \) MACs, one FIFO (using double buffers), and two RE units, where the REs at the left restore the original weights in a row-wise manner. During operations, each PE line processes one or multiple 1D CONV operations, similar to the 1D row stationary in [51] except that we stream each rebuild weight of one row temporally along the MACs for processing one row of input activations. In particular, the 1D CONV operation is performed by shifting the input activations along the array of MACs within the PE line (see Figure 5) via an FIFO; this 1D CONV computation is repeated for the remaining 1D CONV operations to complete one 2D CONV computation in \( \leq (S \times R) \) cycles (under the assumption of w/ sparsity and w/o bit-serial multiplication) with 1) each weight element being shared among all the MACs in each cycle, and 2) the intermediate partial sums of the 2D CONV operations are accumulated locally in each MAC unit (see the bottom right part of Figure 4 (b)).

Third, the RE design: as shown in the bottom left corner of Figure 4 (b), an RE unit includes an RF (register file) of size \( S \times S \) to store one basis matrix and a shift-and-add unit to rebuild weights. The time division multiplexing unit at the left, i.e., MUX1, is to fetch the \( \theta \) coefficient matrices, \( \rho \) basis matrices, or \( \sigma \) original weights. This design enables the accesses of these three types of data to be performed in a time division manner in order to reduce the weight bandwidth requirement by taking advantage of the fact that it is not necessary to fetch these three types of data simultaneously. Specifically, the basis matrix is fetched first and stored stationary within the RE until the associated computations are completed; the weights are then rebuilt in an RE where each row of a coefficient matrix stays stationary until all its associated computations are finished. The third path of MUX1 \( \theta \) for the original weights is to handle DNNs’ layers where SmartDeal is not applied on.

Fourth, the handling of compact models: when handling compact models, we consider an adjusted dataflow and PE line configuration for improving the utilization of both the PE slice array and the MAC array within each PE line. Specifically, for depth-wise CONV layers, since the number of CONV channels is only 1, the \( dim_F \) PE lines will no longer correspond to input channels. Instead, we map the \( R \) number of 1D CONV operations along the dimension of the weight height to these PE lines. For squeeze-and-excite/FC layers, each PE line’s MAC array of \( dim_F \) MACs can be divided into multiple clusters (e.g., two clusters for illustration in the top part of Figure 4 (b)) with the help of the two REs in one PE line (denoted as \( \theta \) and \( \rho \)) and multiplexing units at the bottom of the MAC array, where each cluster handles computations corresponding to a different output pixel in order to improve the MAC array’s utilization and thus latency performance. In this way, the proposed SmartDeal accelerator’s advantage is maintained even for compact models, thanks to this adjustment together with 1) our adopted 1-D row statical dataflow within PE lines, 2) the employed bit-serial multipliers, and 3) the possibility to heavily quantized coefficients.

**Buffer design.** For making use of DNNs’ (filter-/vector-wise or bit-level) sparsity for skipping corresponding computations/memories-accesses, it in general requires a larger buffer (than that of corresponding dense models) due to the unknown dynamic sparsity patterns. We here discuss how we balance between the skipping convenience and the increased buffer size. Specifically, to enable the processing with sparsity, the row pairs of non-zero input activations and coefficients are selected from the Input GB and the Index GB (using the corresponding coefficient indexes), respectively, as inspired by [18], which are then sent to the corresponding PE lines for processing with the resulting outputs being collected to the output GB.

First, input GB: to ensure a high utilization of the PE array, a vanilla design requires \( (dim_C \times dim_F \times bits_{input}) \times \) input activation rows (than that of the dense model counterpart) to be fetched for dealing with the dynamic sparsity patterns, resulting in \( (dim_C \times dim_F \times bits_{input}) \times \) increased input GB bandwidth requirement. In contrast, our design leads to an \( \geq 1/S \) reduction of this required input GB bandwidth, with \( dim_C \times dim_F \times bits_{input} \) inputs for every \( S + \) “Booth encoded non-zero activation bits”) cycles. This is because all the FIFOs in the PE lines are implemented in a ping-pong manner using double buffers, thanks to the fact that 1) the adopted 1-D row stationary dataflow at each PE line helps to relieve this bandwidth requirement, because each input activation row can be reused for \( S \) cycles; and 2) the bit-serial multipliers takes \( \geq 1 \) cycles to finish an element-wise multiplication.

Second, weight/index/output buffer: Similar to that of the input GB, weight/index buffer bandwidth needs to be expanded for handling activation sparsity, of which the expansion is often small thanks to the common observation that the vector-wise activation sparsity ratio is often relatively low. Note that because basis matrices need to be fetched and stored into the RE before the fetching of coefficient matrices and the weight reconstruction computation, computation stalls occur if the next basis matrix is fetched after finishing the coefficient fetching and the computation corresponding to the current basis matrix. Therefore, we leverage the two REs (\( \theta \) and \( \rho \) paths) in each PE line to operate in a “ping-pong” manner to avoid the aforementioned computation stalls. For handling the output data, we adopt an FIFO to buffer the outputs from each PE slice before writing them back into the GB, i.e., a cache between the PE array and the output GB. This is to reduce the required output GB bandwidth by making use of the fact that each output is calculated over several clock cycles.

**VI. Experiments**

In this section, we present a thorough evaluation of SmartDeal. We lay out our plan below.

On the algorithm level, as SmartDeal unifies three mainstream model compression ideas: sparsiﬁcation/pruning, decomposition, and quantization into one framework, we ﬁrst present a carefully designed ablation study in Section VI-A that investigates the effects of different components in SmartDeal and the interactions among them.

We then perform extensive experiments (benchmark over two structured pruning and four quantization, i.e., state-of-the-art compression techniques on four standard DNN models
with two datasets) to validate its superiority. In addition, we evaluate `SmartDeal` on two compact DNN models (MobileNetV2 [52] and EfficientNet-B0 [11]) on the ImageNet [22] dataset, one segmentation model (DeepLabv3+ [53]) on the CamVid [54] dataset, two MLP models on MNIST, and language modeling tasks.

Following the validation of `SmartDeal` inference, we evaluate `SmartDeal` for training on fine-tuning and adaptation tasks given pre-trained models, which is the common practice in edge-based training [55], [56]. Extensive training experiments in Section VI-C1 of two light-weight networks show the storage- and energy-efficiency of `SmartDeal` during training without trading too much model performance. We also provide evaluation over state-of-the-art GPUs in terms of their energy efficiency in Section VI-C2 by using a state-of-the-art training accelerator [9].

On the hardware level, as the goal of the proposed `SmartDeal` is to boost hardware acceleration energy efficiency and speed, we evaluate `SmartDeal`s algorithm-hardware co-design results with state-of-the-art DNN accelerators in terms of energy consumption and latency when processing representative DNN models and benchmark datasets. Furthermore, to provide more insights about the proposed `SmartDeal`, we perform various ablation studies to visualize and validate the effectiveness of `SmartDeal`s component techniques.

### A. An Ablation Study on `SmartDeal`’s Building Blocks

We first investigate the influence of different components of the `SmartDeal` algorithm: 1) Decomposition – decomposing each layer $W$ into $B$ and $C_e$ with no constraint on $C_e$; 2) Quantization – requiring $C_e$ elements to either power-of-2 values or zero; 3) Unstructured Sparsification – promoting element-wise sparsity in $C_e$; 4) Structured Sparsity – incorporating structured sparsity for $C_e$; and 5) Re-Training – iteratively performing re-training and `SmartDeal`. For simplicity, we term the above five components as DE (DEcomposition), Q2 (Quantization to Power-of-2), US (Unstructured Sparsity), SS (Structured Sparsity), and RT (Re-Training), respectively.

Tab. II summarizes the results of ResNet18 [57] on CIFAR-10 dataset [41], with the accuracy, FLOPs, memory size (total memory size to save all parameters including index if necessary), sparsity levels, and normalized energy efficiency results measured on an edge FPGA (i.e., Ultra 96 FPGA [59]) reported. For US, we pick the sparsity threshold $\theta = 8 \times 10^{-3}$. For RT, we iteratively perform re-training and `SmartDeal` algorithm (with fixed $\theta$) for 80 epochs and report best performing models among. For SS, the pruning ratios for different layers are manually tuned. We provide the reproducible details for selecting layer-wise pruning ratios in Appendix B as well as the general rule how we select them. We use Huffman coding representations for $C_e$ (refer to Section B) and 8-bit fixed-point representations for $B$ and input/output activations for `SmartDeal` (Exp. 7, 8, SD, and SD†). Other experiments use standard 32-bit floating-point representations for input/activations. We further use a Ultra96 FPGA [59] platform for real-device energy measurement and report the energy efficiency (normalized to the implementation without any components of the `SmartDeal` algorithm). The difference in utilized computation resource on the FPGA board (i.e., digital signal processing unit (DSP) and look-up table (LUT)) among all implementations is within 5% so that the influence of computation resources is negligible.

- **Quantization.** Comparing Exp. 2 and 3, the power-of-2 quantization has aggressive advantage in reducing the model size, without incurring much accuracy loss: $>3.7\times$ reduction in model size while incurring $<0.6\%$ accuracy loss. Note that because we don’t quantize $B$ in Exp. 3, the equivalent FLOPs is the same as Exp. 2.

- **(Structured) sparsity.** Comparing Exp. 7 with 3, sparsity trades slightly more accuracy drop for higher storage saving and has an immediate influence on FLOPs. Structured sparsity can bring much more aggressive size/FLOPs advantage but will suffer from significant performance loss as shown in Exp. 8. Fortunately, however, model performance can be recovered by re-training in both cases (see last three rows).

- **Decomposition.** The benefit brought by decomposition can be supported by comparing multiple pairs of experiments in different cases. For example, when we compare Exp. 6 and 7 (we compress the models to around the same FLOPs number for fair comparison), the model in Exp. 7 (with decomposition) achieves near 2% higher accuracy with even lower FLOPs.

- **Comparison with sparsity-only models.** Comparing (4+, 5) and (9+, SD) shows that under the same tight storage budget, DE induces a much smaller accuracy drop (93.36% vs. 52.98% and 94.32% vs. 93.02%) at the cost of more FLOPs, which further supports the important role that DE plays in `SmartDeal` by identifying a higher-order structure. The higher FLOPs of Exp. 5 and SD are mainly due to lower sparsity in reconstructed $W$, which has direct influence on FLOPs. Overheads for rebuilding $W$ only account for $<1\%$ of total FLOPs thanks to the high sparsity and readily-quantized-to-2 structure of $C_e$.

- **Energy efficiency on FPGA.** `SmartDeal` trades higher-cost memory storage/access for lower-cost computation. Therefore energy efficiency is a better way than FLOPs to show its power. We can see that SD enjoys near 2% better accuracy than Exp. 9+ and comparable energy efficiency despite of higher FLOPs. When favored with structured sparsity, SD† still outperforms Exp. 9+ with smaller model size and further improves the energy efficiency.

- **Controlling pruning ratios for structured sparsity.** Exp. 8 from Tab. II shows that structured sparsity improves energy efficiency but at the cost of a drastic accuracy drop, although re-training can recover most of the accuracy in Exp. SD†. Here, we use the additional Exp. 8 and SD†, in which we use larger pruning ratios for structured sparsity, to show that we can easily trade-off between storage- and energy-efficiency with accuracy by controlling the pruning ratios.

---

3We use the method in [S8] to compare fixed-point and 32-bit floating-point where the highest bit precision of the operands determines the equivalent floating-point operation. Only CONV and FC layers are considered.
TABLE II
Ablation Study of the Components in SmartDeal using ResNet18 on CIFAR-10 and FPGA energy results. Exp. 7, 8, SD, SD† use Huffman coding representations for \( C_e \) and 8-bit fixed-point representations for \( B \) and activations. Other experiments use 32-bit floating-point representations.

| No. | DE | Q2 | US | SS | RT | Acc | FLOPs* (M) | Size† (MB) | W/C_e | Sparsity | Norm. E.E. |
|-----|----|----|----|----|----|-----|-----------|----------|------|---------|----------|
| 1   |    |    |    |    |    | 94.73% | 1110.8 | 42.59 | 100.0%/ - | 1x       |
| 2   | ✓  | ✓  |    |    |    | 94.73% | 1177.8 | 43.28 | 100.0%/100.0% | 0.79x   |
| 3   | ✓  | ✓  |    |    |    | 94.16% | 1177.8 | 11.34 | 100.0%/100.0% | 2.49x   |
| 4†  | ✓  | ✓  |    |    |    | 52.98% | 212.0  | 9.46  | 19.09%/ - | 2.14x   |
| 5   | ✓  | ✓  |    |    |    | 93.36% | 689.6  | 9.46  | 37.09%/20.60% | 1.71x   |
| 6   | ✓  | ✓  |    |    |    | 90.42% | 440.5  | 3.97  | 39.66%/ - | 3.77x   |
| 7   | ✓  | ✓  |    |    |    | 92.29% | 400.0  | 2.95  | 67.22%/46.44% | 3.63x   |
| 8   | ✓  | ✓  |    |    |    | 43.96% | 290.9  | 2.43  | 41.47%/29.19% | 3.97x   |
| 8a  | ✓  | ✓  |    |    |    | 11.30% | 233.9  | 1.91  | 20.53%/13.76% | 4.26x   |
| 9†  | ✓  | ✓  | ✓  |    |    | 93.02% | 141.7  | 2.18  | 12.75%/ - | 4.31x   |
| SD  | ✓  | ✓  | ✓  |    |    | 94.32% | 261.2  | 1.78  | 22.12%/12.77% | 4.15x   |
| SD− | ✓  | ✓  | ✓  |    |    | 94.60% | 552.2  | 2.99  | 48.27%/36.36% | -       |
| SD† | ✓  | ✓  | ✓  |    |    | 93.30% | 209.8  | 1.74  | 17.16%/10.53% | 4.38x   |
| SD† | ✓  | ✓  | ✓  |    |    | 92.54% | 189.6  | 1.67  | 13.14%/7.5% | 4.40x   |

1‡ Full form of SmartDeal with structured sparsity.
†‡ Representations of \( C_e \) and \( B \) and encoding overheads are all considered for SmartDeal models.
+ The model is pruned to be of the same model size compared with its counterpart.
− A smaller threshold \( \theta \) is used in SmartDeal for performance comparable to the baseline in experiment No. 1.
□ We provide the sparsity of \( C_e \) coefficients as long as DE is involved, in which case we also report the sparsity of rebuilt weight matrices. Sparsity refers to the ratio of non-zeros.
△ Use larger layer-wise structural pruning ratios. See Tab. X.

![Accuracy vs. Model Size Comparison](image)

B. Evaluation of the SmartDeal Algorithm

Experiment settings. To evaluate the algorithm performance of SmartDeal algorithm, we conduct experiments on 1) a total of six DNN models using both the CIFAR-10 and ImageNet datasets, 2) one segmentation model on the CamVid dataset, and 3) two MLP models on the MNIST dataset and compare the performance with state-of-the-art compression techniques in terms of accuracy and model size, including two structured pruning techniques (Network Slimming \[60\] and ThiNet \[61\]), four quantization techniques (Scalable 8-bit (S8) \[62\], FP8 \[63\], WAGEUBN \[64\], and DoReFa \[15\]), one power-of-two quantization technique \[65\], and one pruning and quantization technique \[48\].

SmartDeal vs. existing compression techniques. As SmartDeal unifies the three mainstream ideas of pruning, decomposition and quantization, we evaluate the SmartDeal algorithm performance by comparing it with state-of-the-art pruning-alone and quantization-alone algorithms under four DNN models and two datasets. Note that we use 16-bit floating-point weight representation for the pruning-alone algorithms for more fair comparison. The experiment results are shown in Figure 6. SmartDeal in general outperforms all other pruning-alone or quantization-alone competitors, in terms of the achievable trade-off between the accuracy and the model size. Taking ResNet50 on ImageNet as an example, the quantization algorithm DoReFa \[15\] seems to aggressively shrink the model size yet unfortunately cause a larger accuracy drop; while the pruning algorithm ThiNet \[61\] maintains competitive accuracy at the cost of larger models. In contrast, SmartDeal combines the best of both worlds; it obtains almost as high accuracy as the pruning-only ThiNet \[61\], which is 2.66% higher than the quantized-only DoReFa \[15\]; and on the other hand, it keeps the model as compact as DoReFa \[15\]. SmartDeal also outperforms more recent and competitive model compression works \[a1, a3\]. When compared with the state-of-the-art quantization method LSQ \[66\] that uses 3-bit weight representation, SmartDeal costs much less storage for a ResNet50 network than using LSQ (6.51MB vs. 9.61MB) while inducing less accuracy drop (0.82% vs. 1.10%) on ImageNet.

We re-implement ThiNet \[61\] and report the best numbers that we reproduce in the same compression environment for fair comparison, because we obtain better baseline accuracies than the original ThiNet paper. Note that we report model size in megabytes (MB) (with 16-bit weight representation) while the number of parameters in million (M) is reported in \[61\].

†‡ we did not include decomposition-alone algorithms since their results are not as competitive and also less popular.
the much higher top-1 accuracy (75.31% vs. 73.80%). SmartDeal also outperforms CLIP-Q [67], with a smaller model size than Clip-q (6.51MB vs. 6.70MB) but ageNet. Moreover, SmartDeal also outperforms CLIP-Q [67], a state-of-the-art joint pruning-quantization method, yielding smaller model size than Clip-q (6.51MB vs. 6.70MB) but much higher top-1 accuracy (75.31% vs. 73.80%).

Apart from the aforementioned works, we also evaluate the SmartDeal algorithm with a state-of-the-art power-of-two quantization algorithm [65] based on the same MLP model with a precision of 8 bits: while having a significantly higher compression rate of 188× (vs. 128× in [65]), SmartDeal achieves a comparable accuracy (97.32% vs. 97.35%), even if SmartDeal is not specifically dedicated for FC layers while the power-of-two quantization [65] does. In addition, compared with the pruned and quantized MLP model in [43], SmartDeal achieves a higher compression rate of 66.88× (vs. 40× in [43]) with a comparable accuracy (98.11% vs. 98.42%).

A more extensive set of evaluation results are summarized in Table III in order to show the maximally achievable gains (and the incurring accuracy losses) by applying SmartDeal over the original uncompressed models. In Table III “CR” means the compression rate in terms of the overall parameter size; “Param.,” “B,” and “C_e” denote the total size of the model parameters, the basis matrices, and the coefficient matrices, respectively; “Spar.” denotes the ratio of the pruned and total parameters (the higher the better). Without too much surprise, SmartDeal compresses the VGG networks by 80× to 120×, all with negligible (less than 1%) top-1 accuracy losses. For ResNets, SmartDeal is still able to achieve a solid >10× compression ratio. For example, when compressing ResNet50, we find SmartDeal to incur almost no accuracy drop, when compressing the model size by 15× to 18×.

SmartDeal applied on compact models. Table III seems to suggest that (naturally) applying SmartDeal to more redundant models will have more gains. We thus validate whether the proposed SmartDeal algorithm remains to be beneficial, when adopted for well-known compact models, i.e., MobileNetV2 (MBV2) [52] and EfficientNet-B0 (Eff-B0) [11].

As Table IV indicates, despite the original light-weight design, SmartDeal still yields promising gains. For example, when compressing MBV2 for 7.69× CR, SmartDeal only incurs ~2% top-1 accuracy and 1% top-5 accuracy losses. This result is impressive and highly competitive when placed in the context: for example, the latest work [68] reports 8× compression (4-bit quantization) of MobileNetV2, yet with a 7.07% top-1 accuracy loss.

Extending SmartDeal beyond classification. While model compression methods (and hence co-design works) are dominantly evaluated on classification benchmarks, we demonstrate that the effectiveness of SmartDeal is beyond one specific task setting. We choose semantic segmentation, a popular computer vision task that is well known to be memory/latency/energy-demanding, to apply the proposed algorithm. Specifically, we choose the state-of-the-art DeepLabv3+ [53] with a ResNet50 backbone (output stride: 16), and the CamVid [54] dataset using its standard split. Compared to the original DeepLabv3+, applying SmartDeal can lead to 10.86× CR, with a marginal mean Intersection over Union (mIoU) drop from 74.20% to 71.20% (on the validation split).

Extending SmartDeal beyond computer vision tasks. We additionally perform character-level language modeling experiments with Penn Treebank dataset [69]. Under the same compression ratio, SmartDeal achieves 0.22 better bit-per-character (1.38 vs. 1.60) over standard unstructured pruning.

### C. SmartDeal Training Evaluation

1) Fine-Tuning and Adaptation Study: We present experiment results of SD-T in fine-tuning and adaptation tasks to support the efficacy of SmartDeal for resource-constrained on-device training. We run training experiments in two settings on partitioned CIFAR-10/100 datasets. Our experiments are based on ResNet18 [57] and MobileNetV2 [52]. We compare SD-T (with and without structured sparsity) with standard training. We denote SD-T as SD and SD-T with structured sparsity as SS. Besides accuracies, we also compare the model sizes and normalized energy efficiency.

We consider two datasets split strategies on CIFAR-10/100, corresponding to two scenarios, fine-tuning and adaptation:
### TABLE V
RESULT OF **ResNet18** AND **MobileNet-V2** EXPERIMENTS FOR FINE-TUNING AND ADAPTATION TASKS ON CIFAR-10.

| Task       | Acc   | Norm. Energy Efficiency | Size(MB) |
|------------|-------|-------------------------|----------|
| ResNet18   |       |                         |          |
| Fine Tuning| VA    | 94.67%                  | 1        | 42.59 |
|            | SD    | 93.62%                  | 1.36     | 2.85  |
|            | SS    | 92.40%                  | 1.50     | 2.44  |
| Adaptation | VA    | 95.80%                  | 1        | 42.59 |
|            | SD    | 93.60%                  | 1.13     | 2.52  |
|            | SS    | 93.32%                  | 1.34     | 2.34  |
| MobileNet-V2|       |                         |          |
| Fine Tuning| VA    | 91.26%                  | 1        | 8.63  |
|            | SD    | 90.98%                  | 1.29     | 1.03  |
|            | SS    | 90.38%                  | 1.40     | 0.99  |
| Adaptation | VA    | 93.66%                  | 1        | 8.64  |
|            | SD    | 92.52%                  | 1.31     | 1.00  |
|            | SS    | 92.07%                  | 1.43     | 0.98  |

### TABLE VI
RESULT OF **ResNet18** AND **MobileNet-V2** EXPERIMENTS FOR FINE-TUNING AND ADAPTATION TASKS ON CIFAR-100.

| Task       | Acc   | Norm. Energy Efficiency | Size(MB) |
|------------|-------|-------------------------|----------|
| ResNet18   |       |                         |          |
| Fine Tuning| VA    | 73.87%                  | 1        | 42.76 |
|            | SD    | 72.63%                  | 1.34     | 3.92  |
|            | SS    | 71.40%                  | 1.73     | 3.34  |
| Adaptation | VA    | 92.40%                  | 1        | 42.76 |
|            | SD    | 91.40%                  | 1.50     | 4.04  |
|            | SS    | 91.90%                  | 1.92     | 3.49  |
| MobileNet-V2|       |                         |          |
| Fine Tuning| VA    | 68.71%                  | 1        | 9.07  |
|            | SD    | 69.38%                  | 1.27     | 1.21  |
|            | SS    | 68.75%                  | 1.36     | 1.15  |
| Adaptation | VA    | 89.90%                  | 1        | 9.07  |
|            | SD    | 90.00%                  | 1.33     | 1.57  |
|            | SS    | 90.60%                  | 1.47     | 1.52  |

- **Fine-tuning.** Training samples of all classes are split evenly into two non-overlapping subsets (denoted as sets \(\alpha\) and \(\beta\)). Either \(\alpha\) and \(\beta\) contains all classes. During training, we first pre-train the model with the set \(\alpha\) (not counting into the training energy cost comparison). Then, we fine-tune this same pre-trained model using the different above methods over the set \(\beta\), expecting to continuously grow the performance. We use the original testing set with no split.

- **Adaptation.** The training set is split by classes into two non-overlapping subsets, A and B. We give a 50-50 splitting on CIFAR-10 and a 90-10 splitting on CIFAR-100. During training, we first pre-train the model with the set A (not counting into the training energy cost comparison). Then, starting from this same pre-trained model (except the classification layer being reset to random initialization), we now train it on the set B, using the different above methods, showing how accurately/efficiently the pre-trained model can be adapted to an unseen classification task. For the testing set, we only use the same five class as in B. On CIFAR-100, we adapt from 90 classes the remaining 10.

Results of fine-tuning and adaptation experiments of ResNet18 and MobileNetV2 on CIFAR-10/100 datasets are shown in Tab. V and VI respectively. Refer to the Appendix E for detailed experiment settings. We evaluate the hardware quantified benefits of SmartDeal for training using a state-of-the-art accelerator \([9]\) using the method in \([70]\). We can see SD-T saves energy during fine-tuning/adaption for both networks. When combined with structured sparsity, SD-T can further save the energy cost with tolerable accuracy drop. In Figure 7, we take the adaptation task for ResNet-18 on CIFAR-100 dataset as an example and show the convergence curve of SD-T in terms of testing accuracy on the target classes. We can see that SD-T converges quickly in the first epoch, reaching an accuracy of 77.10%, and then gradually improves the accuracy to 91.40% during the remaining training process. Note that the testing accuracy starts from zero before training starts because we reset the linear layer for the adaptation task.

2) **Hardware quantified benefits of SmartDeal on-device training:** In this set of experiments, we deploy SD-T on a state-of-the-art training accelerator \([9]\) and compare the accelerator’s performance over SOTA GPUs in terms of energy efficiency. Here, we use the state-of-the-art training accelerator \([9]\) instead of designing a dedicated one in order to demonstrate the generality of SD-T algorithm which can potentially improve the performance of training process regardless of the deployment hardware design. We follow the evaluation method in \([70]\) to evaluate the accelerator by implementing a cycle-accurate simulator, aiming to model the Register-Transfer-Level (RTL) behavior of the hardware circuits \([9]\). As shown in Tab. VI by running SD-T-based ResNet18, \([9]\) achieves \(1.3\times\) and \(4.48\times\) improvement in energy efficiency (E.E.) over powerful GPUs with vanilla ResNet18, which are designed in more advanced technologies.

### D. Evaluation of the Dedicated SmartDeal Accelerator.

In this subsection, we present experiments to evaluate the performance of the dedicated SmartDeal accelerator. Specifically, we first introduce the experiment setup and methodology,
and then compare SmartDeal accelerator with four state-of-the-art DNN accelerators (covering a diverse range of design considerations) on seven DNN models (including four standard DNNs, two compact models, and one segmentation model) in terms of energy consumption and latency when running on three benchmark datasets. Details and ablation studies of the accelerator can be referred from Section V of our prior work [2].

**Experiment setup and methodology.** Baselines and configurations: we benchmark the SmartDeal accelerator with four state-of-the-art accelerators: DianNao [1], SCNN [37], Cambricon-X [38], and Bit-pragmatic [73]. These representative accelerators have demonstrated promising acceleration performance, and are designed with a diverse design considerations as summarized in Table VIII. Specifically, DianNao [1] is a classical architecture for DNN inference which is reported to be over 100× faster and over 20× more energy efficient than those of CPUs. While DianNao considers dense models, the other three accelerators take advantage of certain kinds of sparsity in DNNs. To ensure fair comparisons, we assign the SmartDeal accelerator and baselines with the same computation resources and on-chip SRAM storage in all experiments, as listed in Table IX. For example, the DianNao, SCNN and Cambricon-X accelerators use 1K 8-bit non-bit-serial multipliers and SmartDeal and Bit-pragmatic employ an equivalent 8K bit-serial multipliers.

For handling the dynamic sparsity in the SmartDeal accelerator, the on-chip input GB bandwidth and weight GB bandwidth with each PE slice are set to be four and two times of those in the corresponding dense models, respectively, which are empirically found to be sufficient for handling all the considered models and datasets. Meanwhile, because the computation resources for the baseline accelerators may be different from their original papers, the bandwidth settings are configured accordingly based on their papers’ reported design principles. Note that 1) we do not consider FC layers when benchmarking the SmartDeal accelerator with the baseline accelerators (see Figure 8,9,10 for a fair comparison as the SCNN [37] baseline is designed for CONV layers, and similarly, we do not consider EfficientNet-B0 for the SCNN accelerator as SCNN is not designed for handling the squeeze-and-excite layers adopted in EfficientNet-B0; 2) our ablation studies consider all layers in the models can be referred from Section V of our prior work [2].

Benchmark models, datasets, and precision: We use seven representative DNNs (ResNet50, ResNet164, VGG11, VGG19, MobileNetV2, EfficientNet-B0, and DeepLabV3+) and three benchmark datasets (CIFAR-10 [41], ImageNet [42], and CamVid [54]). Regarding the precision, we adopt 1) 8-bit activations for both the baseline-used and SmartDeal-based DNNs; and 2) 8-bit weights in the baseline-used DNNs, and 8-bit precision for the basis matrices and Huffman coding for the coefficient matrices in the SmartDeal-based DNNs.

Technology-dependent parameters: For evaluating the performance of the SmartDeal accelerator, we implemented a custom cycle-accurate simulator, aiming to model the RTL behavior of synthesized circuits, and verified the simulator against the corresponding RTL implementation to ensure its correctness. Specifically, the gate-level netlist and SRAM are generated based on a commercial 28nm technology using the Synopsys Design Compiler and Arm Artisan Memory Compilers, proper activity factors are set at the input ports of the memory/computation units, and the energy is calculated using a state-of-the-art tool PrimeTime PX [74]. Meanwhile, thanks to the clear description of the baseline accelerators’ papers and easy representation of their works, we followed their designs and implemented custom cycle-accurate simulators for all the baselines. In this way, we can evaluate the performance of both the baseline and our accelerators based on the same commercial 28nm technology. The resulting designs operate at a frequency of 1GHz and the performance results are normalized over that of the DianNao accelerator, where the DianNao design is modified to ensure that all accelerators have the same hardware resources (see Table IX). We refer to [70] for the unit energy of DRAM accesses, which is 100pJ per 8 bit, and the unit energy costs for computation and SRAM accesses are listed in Table I.
we compare its latency of processing one image (i.e., batch size is 1) over that of the baseline accelerators on various DNN models and datasets, as shown in Figure 10. We can see that the SmartDeal accelerator achieves the best performance over all the considered DNN models and datasets, achieving a latency improvement ranging from 8.8× to 19.2×. Again, this experiment validates the effectiveness of SmartDeal’s algorithm-hardware co-design effort to reduce the latency on fetching both the weights and the activations from the memories to the computation resources. Since the SmartDeal accelerator takes advantage of both the weights’ vector-wise sparsity and the activations’ bit-level and vector-wise sparsity, it has a higher speedup over all the baselines that make use of only one kind of sparsity. Specifically, the SmartDeal accelerator has an average latency improvement of 3.8×, 2.5×, and 2.0× over SCNN [57] and Cambricon-X [58] which consider unstructured sparsity, and Bit-pragmatic [54] which considers the bit-level sparsity in activations, respectively.

VII. CONCLUSION AND FUTURE WORK

The trade-off between the model performance and its speed/computational cost pervasively exists in the research community and industry. In this paper, we propose a more smart trade-off strategy, SmartDeal, an algorithm-hardware co-design framework with a unique goal to trade higher-cost memory storage/access for lower-cost computation, in order to achieve storage- and energy-efficient DNN inference and training. Extensive experiments including both algorithm and hardware aspects show that SmartDeal can effectively resolve the practical bottleneck of high-cost memory storage/access and can aggressively trim down energy cost and model size, while incurring minimal accuracy drops, for both inference and training. Moreover, SmartDeal provides a handy way for the users to freely balance such trade-off by controlling hyperparameters in SmartDeal to favor either better performance or higher speed. Our immediate future goal is to automate the hyper parameter tuning in SmartDeal via AutoML.

ACKNOWLEDGMENT

The work is supported by the National Science Foundation (NSF) through the Real-Time Machine Learning program (Award number: 1937592, 1937588).
APPENDIX A
SmartDeal Decomposition Evolution.

To give an example of the decomposition evolution of the SmartDeal algorithm, we take one weight matrix \( W \in \mathbb{R}^{192 \times 3} \) from the second CONV layer of the second block in a ResNet164 network pre-trained on CIFAR-10. The SmartDeal algorithm decomposes \( W = C_e B \), where \( C_e \in \mathbb{R}^{192 \times 3} \) and \( B \in \mathbb{R}^{3 \times 3} \). Figure 11 shows the evolution of the reconstruction error, sparsity ratio in \( C_e \), and the distance between \( B \) and its initialization (identity). We can see that the sparsity ratio in \( C_e \) will increase at the beginning at the cost of an increased reconstruction error. But the SmartDeal algorithm remedies the error over iterations while maintaining the sparsity. Also, \( B \) will gradually become more different from the initialization.

APPENDIX B
Encoding \( C_e \) and \( B \) in SmartDeal

After we obtain \( C_e \) and \( B \) via SmartDeal algorithm, we introduce their storage-economic and hardware-friendly representations. We use the representations explained below for all SmartDeal models throughout this paper.

\( a) \) Encoding \( B \): Note that \( B \) is a small dense matrix without special structures. We use 8-bit fixed-point representations for \( B \) because (1) the low bit-width further reduces the model size and (2) the fixed-point representation can fully exploit the benefits of the bit-shifting operations due to power-of-2 coefficients in \( C_e \) and thus minimize the overheads when rebuilding the weights from \( C_e \) and \( B \). The quantization on \( B \) can be easily implemented by adding a quantization step after each fitting step of \( B \) in Algorithm 1.

\( b) \) Encoding \( C_e \): When encoding \( C_e \), an efficient indexing method is used to store the non-zero coefficient in a compact way as in [48]. The “indices” are coded with 1-bit: “0” for zero coefficients and “1” for non-zero coefficients. By using such index, we merely have to store the non-zero coefficients. The cost of the indexing part can be further minimized by exploiting the structured sparsity in \( C_e \) when it is enforced.

For the non-zero elements in \( C_e \), we apply Huffman coding representation following [75], as we empirically observe highly non-uniform distributions of the non-zero elements. We take from the experiments in Section VI-A a ResNet18 model with SmartDeal applied (the “SD” row in Table II) and plot the distribution of the powers with base 2 of the non-zero elements in \( C_e \) within a SmartDeal ResNet18 model trained on the CIFAR-10 dataset.

APPENDIX C
More Discussions on the SmartDeal Rationale

\( a) \) Rationale of SD: From an algorithmic perspective, our SD potentially explores higher-order weight structures, as observed in [26], [30]. That is, rather than enforcing sparsity, quantization, etc. on the original weight directly, it is often more effective to do so on corresponding decomposed matrix factors (either additive or multiplicative). From a hardware perspective, the resulting \( B \) and \( C_e \) in SD enables much reduced memory accesses at a extra light cost of shift-and-add operations. Considering that the unit energy cost of memory accesses is much higher [51], it is promising for energy efficiency and acceleration by trading higher cost memory accesses for lower cost shift-and-add computations.

\( b) \) Comparison with model compression: Although the proposed SmartDeal algorithm includes popular techniques such as pruning, quantization and matrix decomposition that are intensively investigated and used in model compression, there are major differences between SmartDeal and classic model compression works. SmartDeal is motivated by the high-cost memory access consumed by DRAM, which is totally neglected by previous model compression works that only pursue high compression ratio in model storage. To achieve this goal, we design a special weight structure that combines pruning and weight decomposition and is DRAM-friendly, and propose SmartDeal algorithm to enforce that structure on a trained DNN with minimal performance loss, at the cost of inducing extra computations during inference. However, this cost can be further decreased by introducing the power-of-2 quantization and reduce expensive floating point multiplications to cheap bit-shifting operations. In other words, SmartDeal is a novel algorithm that aims at “smartly” trade for high-cost DRAM access for cheap bit-shifting operations, an opportunity largely overlooked by classic model compression.
APPENDIX D
IMPLEMENTATION DETAILS FOR BUCKET SWITCH UPDATING IN SECTION [IV]

During the training process, we utilize “bucket switch updating” and “update decay and aggregation” for \( C_e \)’s non-zeros as discussed Section [IV]. In practice, we will apply a threshold \( \theta_c \) to suppress the effect of noisy gradients for “bucket switch updating”: any gradient whose magnitude is smaller than \( \theta_c \) will be treated as zero. Intuitively, a small update on float-point weights will likely be ignored after quantization. We also allow for an “update delay and aggregation” scheme, in which we can set a “counter” to accumulate the switch decisions in a time window, before actually executing them to update the weight. For example, an element might receive three “switch up”, one “switch down” and one “unchanged” pointers in the last five iterations, while the element is never “actually” updated. We can now aggregate those five and record +2 in the counter. We only switch up or down when the value in the counter reaches \( +\theta_c \) or \( -\theta_c \), where \( \theta_c \) is a positive integer hyper parameter.

APPENDIX E
DETAILED SETTINGS OF FINE-TUNING AND ADAPTATION EXPERIMENTS IN SECTION [VI-C]

A. Training settings

In both tasks, ResNet18 and MobileNetV2 are pre-trained with an initial learning rate 0.1. The Stochastic Weight Averaging (SWA) \([45]\) will start to be applied at the 140th epoch with a learning rate of 0.05. After pre-training, one pass of SmartDeal algorithm (with and without structured sparsity) is applied.

Then, we will train SD-T (with or without structured sparsity) models on the target training subset (B or \( \beta \)). All networks are trained for 80 epochs and SWA starts at the 40th epoch but we use different initial learning rates and SWA-stage learning rates for different combinations of models and tasks. Settings that work well on CIFAR-10 are found to work well on CIFAR-100 dataset, too.

- **Initial learning rates.** For the fine-tuning task, we set the initial learning rate to \( 10^{-2} \). For the adaptation task, the networks usually need a more significant change, especially for the linear layers (we reset linear layers of models trained on adaptation tasks). Therefore, we select an initial learning rate of \( 10^{-1} \).

- **SWA learning rates.** For the fine-tuning task, we use a universal SWA-stage learning rate \( 2 \times 10^{-3} \) for both models on both datasets. For the adaptation task, ResNet18 uses \( 5 \times 10^{-3} \) for the SWA-stage learning rate and MobileNetV2 uses \( 10^{-2} \).

B. SmartDeal settings

In the one-pass SmartDeal algorithm after pre-training as mentioned above, we use a threshold \( \theta = 7 \times 10^{-3} \) for both ResNet18 and MobileNetV2.

When we train SD-T models, we select \( \theta_c = 7 \) and \( \theta_g = 5 \times 10^{-3} \) for ResNet18. The selection of \( \theta_g \) for MobileNetV2 is dependent on the task – we select \( 5 \times 10^{-3} \) and \( 10^{-2} \) for the fine-tuning and adaptation tasks, respectively.

C. Data splitting

For the fine-tuning task, the splitting strategy is the same on CIFAR-10 and CIFAR-100 datasets: 50% of the data are selected as the source or the \( \alpha \) set, on which models are pre-trained; the rest are selected as the target or the \( \beta \) set.

However, it is different for the adaptation task on the two datasets. We use a 50%-50% splitting on CIFAR-10 but a 90%-10% splitting on CIFAR-100, in which the 90% part is the source set and the remaining 10% part is the target.

D. Layer-wise structured pruning ratios for ResNet18

We set the layer-wise structural pruning ratios for ResNet18 as in Tab. [X]. The ratio selection follows the following rules: 1) we select smaller ratios in shallow layers; 2) we select larger ratios for larger layers (more redundancy); 3) we select smaller ratios for shortcuts (1x1 CONV layers); 4) we simply skip linear layers, since ResNet18 has a very small linear layer that hardly affects performance; 5) we choose smaller pruning ratios on CIFAR-100 dataset than we do on CIFAR-10 dataset.

E. Layer-wise structured pruning ratios for MobileNetV2

We set the layer-wise structural pruning ratios for MobileNetV2 as in Tab. [XI]. In addition to the ratio selection principles described in Section [E-D], we also skip the depth-wise separable convolutional layers in MobileNetV2.

APPENDIX F
TRAINING ACCELERATOR CONFIGURATION FOR SmartDeal IN SECTION [VI-C2]

The selected state-of-the-art training accelerator \([9]\) can be configured for feed-forward pass, back propagation, and weight gradient calculations with balanced computation and memory bandwidth requirements. We mainly modify this training accelerator for two parts: inserting rebuilding engines for the original weights and inserting counters to store the last five switch directions for bucket switch updating of \( C_e \)’s non-zero elements. In addition, we reuse the gradients calculations for coefficient and basis matrices.
### TABLE X
Pruning ratios for different layers in ResNet18 in CIFAR-10 and CIFAR-100 experiments. \( C_p \) means the \( p \)th convolutional layer. SC represents shortcut layers. \( \delta \) The larger structural pruning ratios used in experiments marked with \( \delta \) in Table [II].

| Layer     | Shape               | C-10 | C-100 |
|-----------|---------------------|------|-------|
| CONV1     | 64×3×3×3            | 0.2  | 0.2   |
| L1-0-C1   | 64×64×3×3           | 0.2  | 0.2   |
| L1-0-C2   | 64×64×3×3           | 0.2  | 0.2   |
| L1-1-C1   | 64×64×3×3           | 0.2  | 0.2   |
| L1-1-C2   | 64×64×3×3           | 0.2  | 0.2   |
| L2-0-C1   | 128×64×3×3          | 0.2  | 0.2   |
| L2-0-C2   | 128×128×3×3         | 0.3  | 0.2   |
| L2-0-SC   | 128×64×1×1          | 0.1  | 0.1   |
| L2-1-C1   | 128×128×3×3         | 0.4  | 0.2   |
| L2-1-C2   | 128×128×3×3         | 0.4  | 0.2   |
| L3-0-C1   | 256×128×3×3         | 0.3  | 0.2   |
| L3-0-C2   | 256×256×3×3         | 0.3  | 0.2   |
| L3-0-SC   | 256×128×1×1         | 0.2  | 0.2   |
| L3-1-C1   | 256×256×3×3         | 0.4  | 0.2   |
| L3-1-C2   | 256×256×3×3         | 0.5  | 0.2   |
| L4-0-C1   | 512×256×3×3         | 0.5  | 0.2   |
| L4-0-C2   | 512×512×3×3         | 0.5  | 0.2   |
| L4-0-SC   | 512×256×1×1         | 0.5  | 0.2   |
| L4-1-C1   | 512×512×3×3         | 0.5  | 0.2   |
| L4-1-C2   | 512×512×3×3         | 0.5  | 0.2   |
| LINEAR    | \( d_{out}\)×512     | 0.0  | 0.0   |

### TABLE XI
Pruning ratios for different layers in MobileNetV2 in CIFAR-10 and CIFAR-100 experiments, where \( C_p \) means the \( p \)th convolutional layer and SC represents the shortcut layers.

| Layer     | Shape               | C-10 | C-100 |
|-----------|---------------------|------|-------|
| CONV1     | 32×3×3×3            | 0.1  | 0.1   |
| L0.C1     | 32×32×1×1           | 0.4  | 0.2   |
| L0.C2     | 32×1×3×3            | 0    | 0     |
| L0.C3     | 16×32×1×1           | 0.8  | 0.4   |
| L0.SC     | 16×32×1×1           | 0.2  | 0.2   |
| L1.C1     | 96×16×1×1           | 0.2  | 0.2   |
| L1.C2     | 96×1×1×1            | 0    | 0     |
| L1.C3     | 24×96×1×1           | 0.5  | 0.4   |
| L1.SC     | 24×16×1×1           | 0.05 | 0.05  |
| L2.C1     | 144×24×1×1          | 0.2  | 0.2   |
| L2.C2     | 144×1×3×3           | 0    | 0     |
| L2.C3     | 24×14×1×1           | 0.4  | 0.2   |
| L3.C1     | 192×26×1×1          | 0.1  | 0.1   |
| L3.C2     | 144×1×3×3           | 0    | 0     |
| L3.C3     | 32×14×1×1           | 0.1  | 0.1   |
| L4.C1     | 192×32×1×1          | 0.2  | 0.2   |
| L4.C2     | 192×1×3×3           | 0    | 0     |
| L4.C3     | 32×192×1×1          | 0.4  | 0.4   |
| L5.C1     | 192×32×1×1          | 0.2  | 0.2   |
| L5.C2     | 192×1×3×3           | 0    | 0     |
| L5.C3     | 32×192×1×1          | 0.4  | 0.4   |
| L6.C1     | 192×32×1×1          | 0.2  | 0.2   |
| L6.C2     | 192×1×3×3           | 0    | 0     |
| L6.C3     | 64×192×1×1          | 0.4  | 0.4   |
| L7.C1     | 384×64×1×1          | 0.2  | 0.2   |
| L7.C2     | 384×1×3×3           | 0    | 0     |
| L7.C3     | 64×384×1×1          | 0.4  | 0.4   |
| L8.C1     | 384×64×1×1          | 0.3  | 0.2   |
| L8.C2     | 384×1×3×3           | 0    | 0     |
| L8.C3     | 64×384×1×1          | 0.5  | 0.4   |
| L9.C1     | 384×64×1×1          | 0.5  | 0.4   |
| L9.C2     | 384×1×3×3           | 0    | 0     |
| L9.C3     | 64×384×1×1          | 0.6  | 0.4   |
| L10.C1    | 384×64×1×1          | 0.6  | 0.4   |
| L10.C2    | 384×1×3×3           | 0    | 0     |
| L10.C3    | 96×384×1×1          | 0.6  | 0.4   |
| L10.SC    | 96×64×1×1           | 0.1  | 0.1   |
| L11.C1    | 576×96×1×1          | 0.5  | 0.4   |
| L11.C2    | 576×1×3×3           | 0    | 0     |
| L11.C3    | 96×576×1×1          | 0.6  | 0.4   |
| L12.C1    | 576×96×1×1          | 0.6  | 0.4   |
| L12.C2    | 576×1×3×3           | 0    | 0     |
| L12.C3    | 96×576×1×1          | 0.6  | 0.4   |
| L13.C1    | 576×96×1×1          | 0.4  | 0.4   |
| L13.C2    | 576×1×3×3           | 0    | 0     |
| L13.C3    | 160×576×1×1         | 0.6  | 0.4   |
| L14.C1    | 960×160×1×1         | 0.6  | 0.4   |
| L14.C2    | 960×1×3×3           | 0    | 0     |
| L14.C3    | 160×960×1×1         | 0.8  | 0.4   |
| L15.C1    | 960×160×1×1         | 0.8  | 0.4   |
| L15.C2    | 1280×320×1×1        | 0.8  | 0.4   |
| LINEAR    | \( d_{out}\)×1280   | 0.0  | 0.0   |