BBS: Bi-directional Bit-level Sparsity for Deep Learning Acceleration

Yuzong Chen
Cornell University
New York, NY, USA
yc2367@cornell.edu

Jian Meng
Cornell University
New York, NY, USA
jm2787@cornell.edu

Jae-sun Seo
Cornell University
New York, NY, USA
js3528@cornell.edu

Mohamed S. Abdelfattah
Cornell University
New York, NY, USA
mohamed@cornell.edu

***Abstract*—Bit-level sparsity methods skip ineffectual zero-bit operations and are typically applicable within bit-serial deep learning accelerators. This type of sparsity at the bit-level is especially interesting because it is both orthogonal and compatible with other deep neural network (DNN) efficiency methods such as quantization and pruning. Furthermore, it comes at little or no accuracy degradation and can be performed completely post-training. However, current bit-sparsity approaches lack practicality because of (1) load imbalance from the random distribution of zero bits, (2) unoptimized external memory access because all bits are fetched from off-chip memory, and (3) high hardware implementation overhead, including large multiplexers and shifters to support sparsity at the bit level.**

In this work, we improve the practicality and efficiency of bit-level sparsity through a novel algorithmic bit-pruning, averaging, and compression method, and a co-designed efficient bit-serial hardware accelerator. On the algorithmic side, we introduce bi-directional bit sparsity (BBS). The key insight of BBS is that we can leverage bit sparsity in a symmetrical way to prune either zero-bits or one-bits. This significantly improves the load balance of bit-serial computing and guarantees the level of sparsity to be more than 50%. On top of BBS, we further propose two bit-level binary pruning methods that require no retraining, and can be seamlessly applied to quantized DNNs. Combining binary pruning with a new tensor encoding scheme, BBS can both skip computation and reduce the memory footprint associated with bi-directional sparse bit columns. On the hardware side, we demonstrate the potential of BBS through *BitVert*, a bit-serial architecture with an efficient PE design to accelerate DNNs with low overhead, exploiting our proposed binary pruning. Evaluation on seven representative DNN models shows that our approach achieves: (1) on average $1.66\times$ reduction in model size with negligible accuracy loss of $< 0.5\%$; (2) up to $3.03\times$ speedup and $2.44\times$ energy saving compared to prior DNN accelerators.

***Index Terms*—Deep learning accelerator, bit-serial computing, hardware-software co-design, sparsity, model compression**

I. Introduction

Deep neural networks (DNNs) have demonstrated remarkable accomplishments in many important fields such as computer vision and natural language processing. However, the growth of DNN model size and complexity continues to outpace the scaling of compute performance in existing hardware platforms [12]. Bridging this performance gap is very desirable for wider adoption of DNNs, particularly in edge scenarios that demand both high performance and energy efficiency. Codesigning novel DNN compression algorithms, together with accelerators for the efficient deployment of the compressed models, is a promising way to achieve this goal.

Numerous efficiency algorithms [21], [30], [31] and hardware prototypes [6], [13], [14], [42], [43] have been proposed to leverage *value-based sparsity* in DNNs to reduce the cost of storing and deploying DNNs. Yet the degree of such value sparsity, which depends on the underlying model architecture, can strongly limit the resulting hardware performance. For instance, recent transformer-based DNNs show limited or no activation sparsity with GeLU and sigmoid activation functions [7], [9]. Even for single-sided sparse accelerators that target weight sparsity, plenty of time and cost are spent on retraining the model to balance the degree of sparsity and accuracy loss. Unfortunately, in many real-world cases, retraining may become impractical for end users due to cost constraints and lack of access to the original training dataset [3], [39]. This challenge is particularly pronounced in recent large language models [40], [47] that contain billions of parameters, making retraining even more resource and data intensive. Hence, there is a strong need to further enhance the efficiency of DNN accelerators *without imposing retraining*.

Another line of DNN compression research focuses on *post-training quantization* (PTQ), which represents DNN operands in lower precision without retraining the model [15], [24], [25], [32], [36], [44], [45]. For example, researchers have designed new quantization data types such as the Microscaling format [36], where a group of low-precision operands can share an 8-bit exponent to balance the accuracy and memory footprint. However, Microscaling still requires a floating-point pipeline to handle the shared 8-bit exponent, resulting in higher hardware cost than integer quantization. On the other hand, state-of-the-art PTQ algorithms can already reduce the operand precision to 8-bit integer with negligible accuracy loss [24], [32], [44]. Unfortunately, a quantized 8-bit DNN shows extremely low value sparsity (less than 5% as will be shown in the next section), since it tries to utilize all quantization levels as much as possible to reduce the quantization error. This fundamental quantization-sparsity tension poses a big performance bottleneck in existing value-based DNN accelerators [16], [38].

In order to jointly exploit the efficiency of quantization and sparsity, a series of bit-serial DNN accelerators exploit *bit-level sparsity* [1], [19], [20], [26], [37], [39]. Unlike coarse-grained value sparsity that is incompatible with quantization, the bit-level sparsity targets the abundant *zero bits* in the

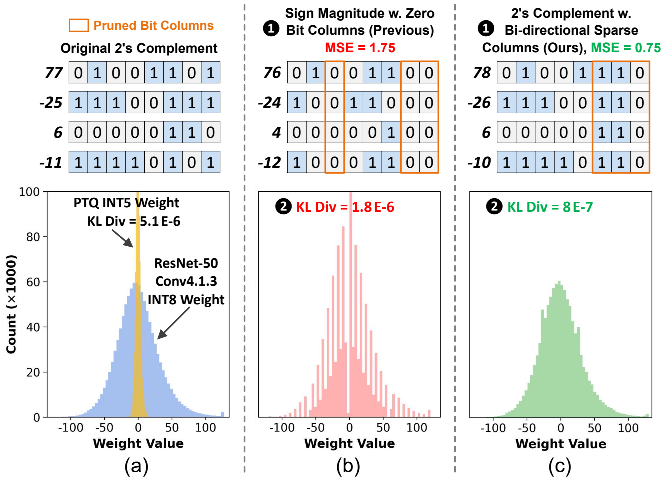

Fig. 1: Comparison of different model compression approaches. (a) Example of a 4-value group and the weight distribution of a ResNet-50 layer before and after PTQ. (b) ❶ Bit-sparsity enhancement by generating three zero bit columns using sign-magnitude format, ❷ achieving lower KL divergence than PTQ but still losing many quantization levels. (c) ❶ BBS generates three bi-directional sparse bit columns and is able to preserve all quantization levels of 8-bit precision, ❷ leading to much lower KL divergence.

binary representation of operands, thus is both compatible and orthogonal to other forms of DNN redundancy. Stripes [19] is an early bit-serial prototype that uses reduced precision for DNNs to scale the performance. Pragmatic [1], Laconic [37] and Bitlet [26] propose to skip zero-bit operations from different perspectives. However, the distribution of zero bits is generally random, whether in an individual operand or a group of operands, leading to significant workload imbalance. A direct consequence is that these accelerators must still fetch all data bits from off-chip memory, and use sophisticated hardware schedulers to skip zero-bit operations as much as possible during on-chip computation. The latter usually incurs non-trivial hardware overhead.

To reduce both memory access and scheduling overhead of bit-serial computing, BitWave [39] employs a bit-column-serial approach, which examines the sparsity of the same bit significance across a group of operands. If a bit column contains all zero-bits, then it does not need to be stored in memory. Moreover, BitWave proposes a bit-sparsity-enhancing technique based on sign-magnitude formatted weights to selectively flip bits to zero. With this *bit-flip* technique, BitWave is able to further compress a quantized 8-bit DNN by generating more zero bit columns. As a result, it has demonstrated the potential to achieve higher performance than other bit-serial accelerators [1], [19], [26].

Despite these approaches exploring bit sparsity at varying degrees, they still suffer from one significant drawback: bit sparsity is only limited to zero bits. To demonstrate this problem, consider Figure 1(a) that shows a group of four INT8 values, as well as the INT8 weight distribution of a layer in ResNet-50. If we want to further reduce the bit-width to, *e.g.*, 5-bit, conventional PTQ needs coarse-grained clipping and re-scaling so that the quantization mean square error (MSE) is minimized. Nevertheless, no matter what PTQ algorithm is used, the resulting distribution can only have $2^5 = 32$ discrete quantization levels, resulting in large KL divergence, a common metric to quantify the difference between two distributions [17]. On the other hand, previous bit-sparsity-aware works [23], [35], [39] leverage sign-magnitude format to prune bit columns at the group level as shown in Fig. 1(b). Given that DNN weights are typically small, many inherent zero bit columns exist (*e.g.*, the third bit columns in Fig. 1(b)), leading to less sparse columns enforced (*e.g.*, the seventh and eighth bit columns in Fig. 1(b)) to achieve the effective 5-bit data width. As a result, they can preserve more quantization levels and achieve lower KL divergence and better accuracy than PTQ. However, if there is no inherent sparse bit column in a group, all lower significant bit columns must be flipped to zero, leading to reduced quantization levels especially in intervals with large absolute values (*e.g.*, $> |50|$ in Fig. 1(b)).

Our focus: this work proposes a novel sparsity concept called *bi-directional bit-level sparsity* (BBS) and the associate bit-serial accelerator design named *BitVert*. The key insight of BBS is that the bit-level sparsity can be explored in a symmetrical way, where less zero-bits implies more one-bits, and vice versa. This ensures that any bit vector can exhibit at least 50% BBS, which significantly improves the load balance of bit-serial computing while minimizing the number of ineffectual bit operations. Due to the balanced workload, BBS eliminates the expensive bit synchronization mechanism that is typically associated with prior bit-serial accelerators [1], [20], [26]. Furthermore, unlike previous bit-sparsity-aware works that only prune zero bit columns, BBS offers a new opportunity for model compression—it permits pruning a bit column with entirely zero-bits or entirely one-bits, which we call *bi-directional sparse bit columns*. As shown in Fig. 1(c), by looking for an optimal way to generate 3 bi-directional sparse columns, we can achieve much lower MSE compared to merely pruning zero bit columns with the same compression ratio. Additionally, since BBS allows any bit significance to be one, it preserves all quantization levels of the original INT8 weight and yields much lower KL divergence w.r.t. the original numerical distribution pre-compression. Finally, the balanced nature of BBS can be exploited in a hardware-friendly manner to improve the performance and energy efficiency of bit-serial accelerators. The main contributions of this work are summarized as follows:

1) We introduce the new BBS concept, and demonstrate that BBS significantly improves the load balance of bit-serial accelerators.
2) We propose two bit-level *binary pruning* strategies to enhance structured BBS. The binary pruning employs a new encoding scheme to reduce the memory footprint of a quantized DNN without the need of retraining.
3) We design *BitVert*, a bit-serial accelerator to exploit BBS for DNN acceleration. *BitVert* adopts an efficient processing element (PE) with low hardware overhead for bit skipping, along with a channel-reordering mechanism to support binary pruning.

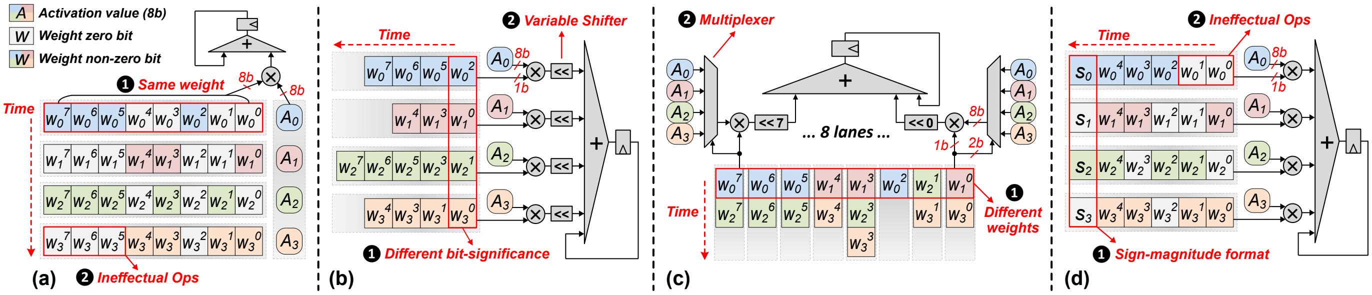

Fig. 2: High-level computation flow of (a) bit-parallel PE, (b) Pragmatic [1], (c) Bitlet [26], (d) BitWave [39].

Through extensive evaluation on seven representative DNN benchmarks, including both vision and language models, we demonstrate that *BitVert* achieves up to 3.03× speedup and 2.44× energy saving compared to prior DNN accelerators, while having negligible accuracy loss ($<0.5\%$ on average) together with the preserved statistical characteristics of the uncompressed model.

II. Background and Related Works

A. Sparse Bit-serial Accelerators

We first describe the computation flow of bit-parallel processing and recent sparse bit-serial accelerators [1], [26], [39] using a 4-way dot product example between 8-bit operands. We focus on weight sparsity in our discussion. In Fig. 2(a), a bit-parallel PE exploits bit-level parallelism by performing the multiplication between an 8-bit activation and all bits of the same weight, but leading to many ineffectual bit operations. Since zero bits do not contribute to the final result, it is desirable to skip as many zero bits as possible for improved performance and efficiency.

Pragmatic [1] processes only non-zero bits of every weight as shown in Fig. 2(b). However, since different bit-significance can be processed simultaneously, Pragmatic requires a variable shifter after every bit-serial multiplier to synchronize the significance of essential bits. Bitlet [26] leverages the sparsity parallelism, motivated by the observation that every bit significance shows similar sparsity among a group of weights. As shown in Fig. 2(c), Bitlet digests multiple weights and activations, and computes every bit-significance independently. However, since every bit lane can absorb the essential bit from an arbitrary weight, Bitlet requires a large multiplexer (*e.g.*, 64:1) to select the correct activation in every lane, leading to non-trivial hardware overhead (35.9% of the PE area as revealed by Bitlet's breakdown report).

Both Pragmatic and Bitlet suffer from load imbalance issues, where the latency of Pragmatic is dominated by the weight with the highest number of one bits, and the latency of Bitlet is dominated by the bit significance with the highest number of one bits. To address this, BitWave [39] attempts to skip zero bits at the coarse bit-column granularity, as illustrated in Fig. 2(d). Because most weight values are typically small in a DNN, BitWave relies on sign-magnitude format which inherently generates many zero bit columns. The bit column sparsity offers balanced workload, but inevitably leads to many ineffectual bit operations since only a bit column with all zero bits can be skipped during computation. On the top of these three design philosophies, our proposed *BitVert* tries to balance the bit-serial workload while skipping as many sparse bits as possible. By extending bit sparsity to BBS, *BitVert* skips zero bits when a bit column contains many zeros, while it switches to skip one bits when a bit column contains less zero bits. Section III details our BBS methodology.

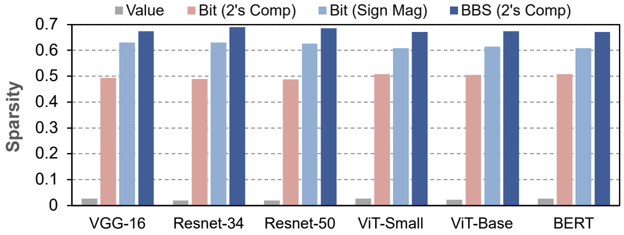

Fig. 3: Comparison of inherent weight value sparsity, bit sparsity and BBS (with a bit-vector size of 8) in INT8 DNNs.

B. Rethinking Bit-level Sparsity

While recent advances in PTQ can compress DNNs to 8-bit with little or no accuracy loss [5], [25], [32], [44], [45], the resulting weight tensor exhibits extremely low value sparsity. As shown in Fig. 3, the value-based weight sparsity is less than 5% in a series of popular 8-bit quantized DNNs. This is because that a well-designed PTQ algorithm tries to utilize all available quantization levels to minimize the quantization MSE compared to original floating-point models. On the other hand, the bit-level sparsity is inherently more abundant and can achieve around 50% in 2's complement format. Owing to the facts that DNN weight tensors usually exhibit Gaussian-like distribution and most values tend to be small [16], [34], [46], the sign-magnitude binary representation yields even higher bit sparsity [2], [39] due to abundant zero bits at higher bit significance. However, adopting sign-magnitude arithmetic for bit-serial computing still has two challenges. First, every bit-serial multiplier requires a 2's complementer for partial sum generation, resulting in large area overhead [18]. Second, the irregular distribution of zero bits remains, leading to load imbalance and synchronization overhead. Whereas our proposed BBS maintains the 2's complement binary representation, and treats zero or one that has a higher occurrence as sparse bits. Hence, BBS ensures that any bit-vector exhibits at least 50%

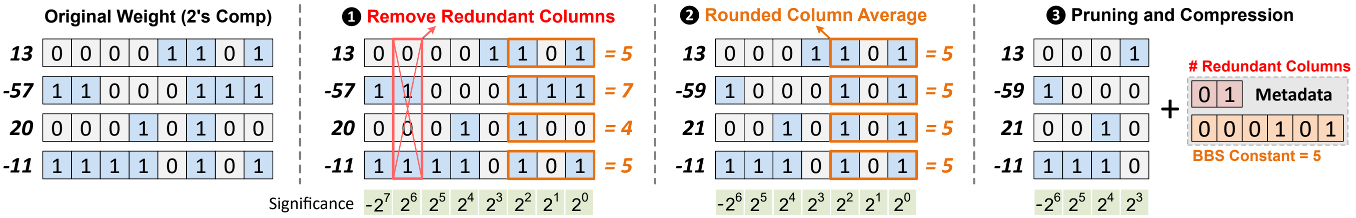

Fig. 4: Example of bit-level binary pruning with rounded column averaging to generate 4 sparse bit columns.

bit sparsity, resulting in higher total bit sparsity than sign-magnitude format while achieving balanced workload across different PEs.

III. BBS: Bi-directional Bit-level Sparsity

In this section, we first introduce the concept of BBS based on 2's complement binary representation. Next, we present *binary pruning*, a technique that modifies the original weight tensor to generate more structured BBS, together with a new encoding scheme that provides an extra opportunity for model compression. Finally, we propose a hardware-aware strategy to compress different weight channels of a DNN model based on the global awareness of pruning sensitivity, which can achieve favorable accuracy-compression trade-offs.

A. BBS Theorem

Without loss of generality, we describe BBS using a dot product operation that multiplies a group of N weights (W) and activations (A) in p-bit precision, where N is referred to as the *group size*. In the rest of this paper, we use the term "*group*" to refer to multiple weights or activations that contribute to the same dot product output. The dot product operation can be formally written as:

$$\sum_{i=0}^{N-1} W_i \times A_i = \sum_{b=0}^{p-1} 2^b \times \sum_{i=0}^{N-1} W_i^b \times A_i \tag{1}$$

where W_i^b is the b^{th} bit of W_i. Since any bit of W can only be one or zero, the second partial sum on the right-hand side of Eq. 1 can be re-organized as:

$$\begin{aligned} \sum_{i=0}^{N-1} W_i^b \times A_i &= \sum_{\forall\, i:\, W_i^b=1} A_i \qquad (2) \\ &= \sum_{j=0}^{N-1} A_j - \sum_{\forall\, i:\, W_i^b=0} A_i \qquad (3) \end{aligned}$$

From Eq. 2 and 3, we can infer that instead of adding the effectual activations associated with non-zero weight bits, the same result can be obtained by subtracting the activations indicated by zero weight bits from the sum of all activations, which is a constant for a given group. Since more zero-bits in a vector implies less one-bits, Eq. 2 and Eq. 3 always process no more than half of the bits—when there are more than 50% zero-bits in a bit-vector, the computation can skip them as in conventional bit-serial accelerators. But if there is less than 50% bit sparsity, the bit-vector can be *inverted* so that the original one-bits become sparse, and subtract the bit-serial dot product from $\sum_{i=0}^{N-1} A_i$. Since both zero and one can become sparse bits, we call this *bi-directional bit sparsity (BBS)*.

The idea of BBS can effectively improve the load balance of bit-serial computing. Although there is ∼50% zero bit sparsity in 2's complement format and more than 50% zero bit sparsity in sign-magnitude format (Fig. 3), the sparsity within a bit-vector is unpredictable. Moreover, because bit-serial computing relies on strongly increased parallelism to simultaneously process many bit-vectors from different weight groups, any bit-vector with low zero bit sparsity will hamper the performance of the whole PE array. On the other hand, BBS ensures at least 50% sparsity in a bit-vector of arbitrary length, achieving balanced workload during parallel execution while skipping as many ineffectual bit operations as possible.

B. Bit-level Binary Pruning

In addition to balanced bit sparsity, BBS offers a new opportunity for model compression through *binary pruning*—which can prune a bit column that contains all zero-bits or all one-bits within a weight group. Specifically, Eq. 2 implies that if all weight bits at a bit significance are zero, then the bit-serial dot product at that significance is simply zero. Similarly, Eq. 3 implies that if all weight bits at a significance are one, then the bit-serial dot product at that significance is the sum of activations in the group. As a result, a bi-directional sparse bit column can be compressed to just one bit that indicates whether its bit-serial dot product produces zero or sum of activations. Based on this observation, we propose two BBS-enhancing strategies to generate more bi-directional sparse bit columns in the original weight group, which can be effectively pruned through a new encoding scheme.

BBS with Rounded Averaging Fig. 4 describes the procedure of the first BBS-enhancing strategy, *rounded averaging*, using a group of 4 weights. Given the target number of sparse bit columns (4 in this example), Step ❶ identifies if there are *redundant* bit columns that immediately follow the most-significant column with the same content (e.g., the second bit column). Removing the redundant columns does not affect the original weight values as long as the remaining bits are interpreted as 2's complement format. For instance, the decimal number -57 in 8-bit 2's complement format is 11000111_b, where the most-significant bit is multiplied by -2^7. Removing the second bit leads to a 7-bit number 1000111_b, which is still equal to -57 if the most-significant bit is multiplied by -2^6. After pruning the redundant column, the required number

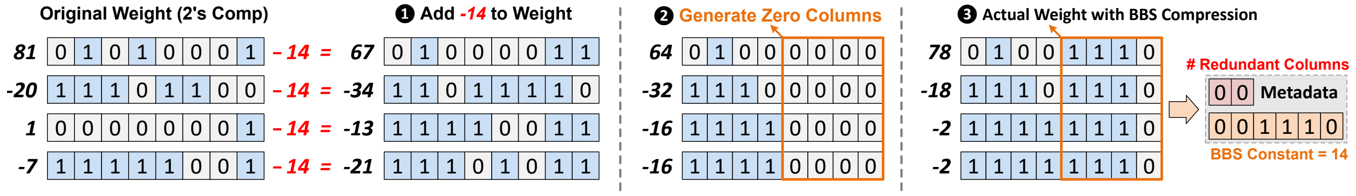

Fig. 5: An example of bit-level binary pruning with zero-point shifting to generate 4 sparse bit columns.

of bi-directional sparse columns to be generated is 3. These sparse columns are always generated from the lower significant bits, since modifying higher bit significance will increase the MSE exponentially. In Step ❷, this is achieved by calculating the rounded average of the values represented by the 3 lower significant bits of original weights. Essentially, this is replacing the 3 lower significant bits of all weights with a 3-bit constant while minimizing the MSE. Finally, Step ❸ compresses the original weight group by storing only the remaining 4 bit columns and an 8-bit encoding metadata.

BBS Compression Encoding The encoding metadata contains 2 bits to specify the number of redundant columns, which can vary from 0 to 3, and 6 bits to store the BBS constant. The size of the metadata is chosen empirically. First, although there may be more than 3 redundant columns in a group, we find that this probability is extremely low for a large group size (*e.g.*, 32) which amortizes the cost of metadata. If there are more than 3 redundant columns, we simply prune the first 3 and average additional lower significant columns instead. Second, using more than 6 bits to store the constant is also unnecessary since pruning 7 columns of an 8-bit tensor leaves only one effective bit, while pruning 8 columns means replacing all weights with the same 8-bit constant. Both situations can lead to unacceptable accuracy loss.

BBS with Zero-point Shifting The rounded column averaging strategy is particularly suitable for pruning a small number of bit columns, where the lower significant bits within a group are likely to have similar values. However, for more eager compression, *i.e.*, pruning many columns, simply taking the rounded average over many lower significant bits of a group may lead to large MSE. Here is a simple example: assume we want to average only the least significant bit within a group of weights, then some weights will have no error after rounded averaging. On the other hand, if we average 4 lower significant bits, then all weights may produce error since any weight can have a different value in the 4 lower significant bits.

To address this, we propose a second BBS-enhancing strategy called *zero-point shifting*. The idea is to add an optimal constant to the original weight group (*i.e.*, shifting its zero-point), which in turn facilitates the generation of sparse bit columns in the new weight group while minimizing the MSE. Fig. 5 exemplifies this procedure for generating 4 sparse bit columns. In Step ❶, assume a constant -14 is added to the original weight, which changes the binary content of all numbers. Fortunately, the change of binary content makes it easier to generate zero columns in lower significant bits. As shown in Step ❷, to minimize the MSE when pruning the 4 lower significant bit columns, a number can either directly zero out the 4 lower bits (e.g., the first number changes from 67 to 64), or round up to the higher bit significance (e.g., the second number changes from -34 to -32). Finally, Step ❸ shows the actual values after binary pruning and stores the new zero-point in the encoding metadata.

Algorithm 1: Finding the optimal constant for zero-point shifting.

```
   Input  : Weight group: $W$, BBS constant precision: $p$,
            target number of sparse bit columns: $N$
   Output : Compressed weight: $W_C$, metadata: $D$
1  def Compress($W$, $N$, $p$):
2      bestMSE = $\infty$ ;
3      for constant = $-2^{p-1}$ to $2^{p-1} - 1$ do
4          $W_{tmp}$ = Clip ($W$ + constant )
           // Get number of redundant columns
5          numRedunCol = GetNumRedunCol($W_{tmp}$)
6          $W_{tmp}$ = RemoveRedunCol($W_{tmp}$ , numRedunCol )
           // Generate zero sparse columns
7          numSparseCol = $N -$ numRedunCol
8          $W_{tmp}$ = GenSparseCol($W_{tmp}$ , numSparseCol )
9          newMSE = $|W_{tmp} - W|^2$
10         if newMSE $<$ bestMSE then
11             bestMSE = newMSE
12             $W_C$ = $W_{tmp}$
13             $D$ = { numRedunCol , constant }
14     return $W_C$, $D$
```

Algo. 1 details the algorithm to find the optimal BBS constant for a weight group. Given the precision of the constant (6-bit in our proposed BBS encoding), the algorithm iterates through all possible constants (Line 3). In every iteration, it adds the current constant to the original weight group, followed by clipping to avoid overflow (Line 4). Next, similar to *rounded averaging*, we calculate the number of redundant columns, and generate required number of sparse columns while minimizing MSE (Line 5 – 7). Since the best constant will be stored in the BBS constant region of the metadata, we only generate zero sparse bit columns (Line 8) so that no extra encoding information is needed. Lastly, the algorithm checks whether the current constant results in lower MSE and updates the weight group and metadata accordingly (Line 9 – 13).

Although Algo. 1 describes the procedure using a single weight group, the whole algorithm can be vectorized to find the optimal constant of all groups within a DNN layer simultaneously. During real implementation, the algorithm takes several milliseconds to several seconds per layer (totally $\sim$15s to

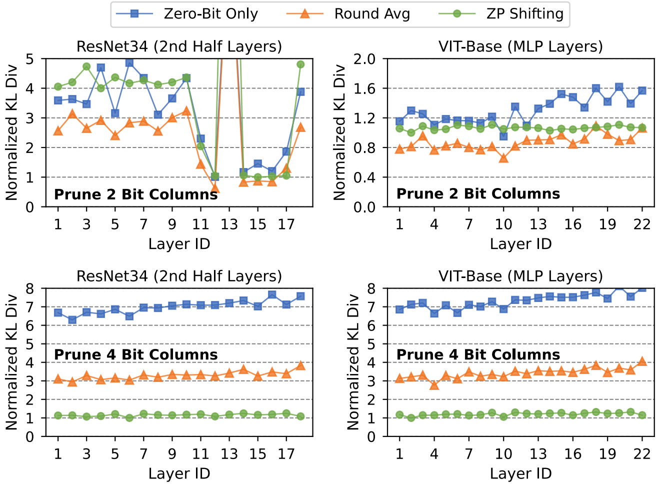

Fig. 6: Normalized KL divergence (lower is better) of different bit-level pruning techniques with a weight group size of 32.

compress the whole ResNet50) on a single Nvidia RTX 3090 GPU. Hence, the proposed bit-level binary pruning method exhibits high efficiency and fast compression compared to prior quantization-oriented algorithms [5], [45], [46].

Rationality of Binary Pruning To demonstrate the rationality of the proposed two binary pruning strategies compared to previous zero-bit-only pruning [23], [35], [39], we apply the three techniques to compress the quantized 8-bit ResNet-34 and ViT-Base. Fig. 6 shows the resulting KL divergence of different methods after pruning 2 and 4 bit columns with a weight group size of 32. The KL divergence is a common metric to quantify the difference between two distributions [11], [17]. A lower KL divergence indicates that the compressed weight tensor can better preserve the information of the original 8-bit weight, thus achieving better inference accuracy (evaluated in Section V-B).

Specifically, Fig. 6 shows that when pruning 2 bit columns, *rounded averaging* consistently outperforms other approaches. The reason is that different weights within a group are likely to have similar values in the lower significant bits. On the other hand, *zero-point shifting* yields much lower KL divergence when pruning 4 bit columns. This is because it can better exploit the binary characteristics of a weight group to find the optimal zero point that facilitates the generation of more sparse bit columns. Furthermore, the proposed binary pruning permits the existence of both zero and one in any bit significance after compression, thus are able to preserve all quantization levels of the original 8-bit weights as opposed to zero-bit-only pruning. As a result, both of our strategies show significant improvements when applied to a large number of bit columns.

C. Hardware-aware Global Binary Pruning

So far, we have described binary pruning at the group level. In order to fully exploit the structured BBS sparsity induced by binary pruning while mitigating the accuracy loss for the whole DNN, we propose a hardware-aware global binary pruning approach at the *per-channel* granularity. Specifically, we find that the pruning sensitivity of different weight channels can be effectively quantified through magnitude-based proxies. For example, in convolutional neural networks, the sensitive filters (*i.e.,* weight channels) usually contain many outliers with large magnitude. More specifically, in per-channel quantized DNNs, the sensitive channels of a weight tensor will have large scaling factors to accommodate these outliers [27], [44]. The per-channel weight quantization has been widely adopted to achieve high accuracy in state-of-the-art DNN accelerators [3], [16] and acceleration frameworks such as TensorRT [33]. Therefore, we consider per-channel quantized 8-bit DNNs as the baseline for global binary pruning [1].

Algorithm 2: Global binary pruning.

```
   Input  : Model: $M$, per-channel scaling factors: $S$
            threshold: $\beta$, hardware parameter: $C_H$
   Output : Pruned model: $M_P$
1  def GlobalPrune($M$, $S$, $\beta$, $C_H$):
       // Global channel sorting
2      channelSorted = SortChannel($M.channel$, $S$)
3      sensChannel = channelSorted[$1 : \beta \times Length(S)$]
4      for $L$ in $M.layers$ do
           // Ensure every layer has a multiple
              of $C_H$ sensitive channels
5          layerChannel = SortChannel($L.channel$, $S[L]$)
6          numSens = Count( layerChannel $\cap$ sensChannel )
7          numSens = Ceiling( numSens / $C_H$ ) $\times$ $C_H$
           // Get sensitive channels of layer $L$
8          topChannel = layerChannel[1 : numSens]
9          sensChannel = sensChannel $\cup$ topChannel
10     normalChannel = $M.channel -$ sensChannel
11     if eagerCompression then
12         $M_P$ = RoundedAveraging(normalChannel)
13     else
14         $M_P$ = ZeroPointShifting(normalChannel)
15     return $M_P$
```

To apply global binary pruning, we define a hyperparameter β to specify the minimum percentage of sensitive weight channels. Also, we define a hardware-aware parameter C_H, which specifies the number of weight channels processed in parallel during hardware acceleration (*e.g.,* $C_H = 32$ in our *BitVert* accelerator). Algo. 2 details the procedure of global binary pruning. The algorithm starts with global channel sorting to identify β sensitive channels based on the scaling factors (Line 1 – 2). For every layer, we force the number of sensitive channels to be a multiple of C_H (Line 4 – 9). For example, in the convolution layer, if the number of sensitive filters is less than C_H after global channel sorting, then we simply select C_H filters with the highest scaling factors as new sensitive channels. Finally, we apply binary pruning to the remaining channels (Line 10 – 14), which can either prune a different number of bit columns for different layers [39] or prune the same number of bit columns for all layers.

[1] For 8-bit DNNs that do not use per-channel quantization, other channel importance proxies such as the standard deviation of a weight channel can also be used to identify sensitive channels.

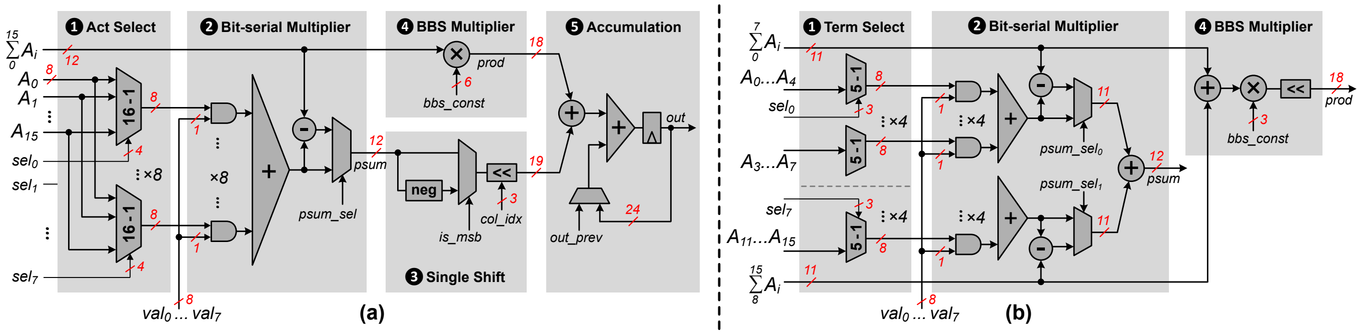

Fig. 7: BitVert PE: (a) baseline design, (b) modified design.

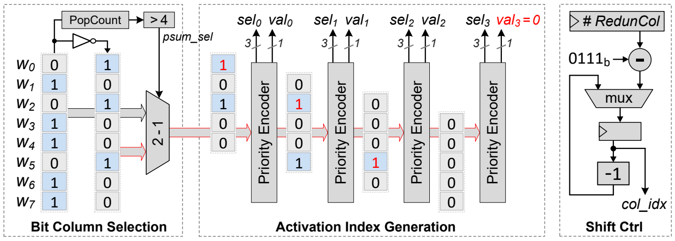

Fig. 8: BitVert scheduler.

The identification of sensitive channels further reduces the MSE and KL divergence while eliminating the need for resource-intensive and time-consuming retraining. In most of our DNN benchmarks (Section V-A), we are able to set $\beta = 10\%$ or 20% while pruning a large number of bit columns in the remaining channels. However, since the locations of sensitive channels are random within a layer, two challenges arise for efficient hardware acceleration. First, identifying the location of sensitive channels requires significant indexing overhead. Second, different precision will cause unaligned memory access to the weight tensor in DRAM. The proposed *BitVert* accelerator addresses these challenges through a channel-reordering mechanism as will be discussed shortly.

IV. BitVert Hardware Architecture

To fully exploit the potential of BBS and binary pruning, we design a bit-serial accelerator, named *BitVert*, which includes an efficient PE and scheduler to support BBS with compression, along with the channel reordering mechanism for hardware-aware global binary pruning.

A. BitVert Processing Element

The *BitVert* PE performs bit-serial multiplication between a group of 16 weights and activations, where weights are processed bit-serially. Fig. 7(a) shows a baseline *BitVert* PE that performs the computation in 5 steps. Step ❶ receives 16 activations $A_0, ..., A_{15}$ and selects 8 of them based on $sel_0, ..., sel_7$ that indicates the position of effectual bits in the weight bit-vector. Step ❷ performs bit-serial multiplication using valid signals $val_0, ..., val_7$ in case there are less than 8 effectual bits (*i.e.*, more than 50% sparsity in the weight bit-column). A subtractor subtracts the adder tree result from the sum of activations (Eq. 2), followed by a mux to select the partial sum. Step ❸ then shifts the partial sum based on the column index col_idx that specifies the significance of current weight bits. The col_idx can vary across different groups according to the number of redundant columns during binary pruning (Section III-B). Recall that BBS compression stores a constant, whose "0" bit indicates a bit-column of all zero-bits and "1" bit indicates a bit-column of all one-bits. Hence, Step ❹ multiples this constant with the sum of activations. Finally, the product and bit-serial partial sum are accumulated in Step ❺. The activations are reused for multiple clock cycles until all bit-columns belonging to the same weight group are processed. The control signals such as sel, val, and col_idx are updated by the *BitVert* scheduler in every cycle (described in Section IV-B).

Due to the random distribution of effectual bits within a weight bit-column, the baseline PE accounts for the worst case by using a 16:1 mux for every activation term. Since BBS guarantees at least 50% sparsity in a bit-vector of arbitrary length, it is possible to reduce the mux cost with a smaller group size. Based on this observation, we propose a modified PE that computes bit-serial multiplication within a smaller *sub-group* as shown in Fig. 7(b). The sub-group size is a design parameter that offers a trade-off between area and power. A smaller sub-group can reduce the mux cost but requires more subtractors. Therefore, we conduct a PE design space exploration (Section V-E) and choose a sub-group size of 8 in our design. Furthermore, because the PE supports 50% bit sparsity, at most 4 activations will be selected within a sub-group. In the worst case, the selected activations within the sub-group $\{A_0, ..., A_7\}$ will be $\{A_4, A_5, A_6, A_7\}$. Hence, we only need four 5:1 muxes to locate all effectual activations, where the first mux selects among $\{A_0, ..., A_4\}$, the second mux selects among $\{A_1, ..., A_5\}$, and so on. Using 5:1 muxes further reduces the PE area compared to 8:1 muxes.

It is also possible to reduce the cost of the BBS multiplier in Step ❹. Since BBS can prune a maximum of 6 bit columns in a weight group (Section III-B), it requires at least 2 cycles to process the remaining columns when the weight precision is 8 bits. This allows time-multiplexing the BBS multiplier by multiplying 3 bits per cycle, followed by a shifter to align the significance. Section V-E evaluates the reduction in PE area overhead achieved by the proposed optimization.

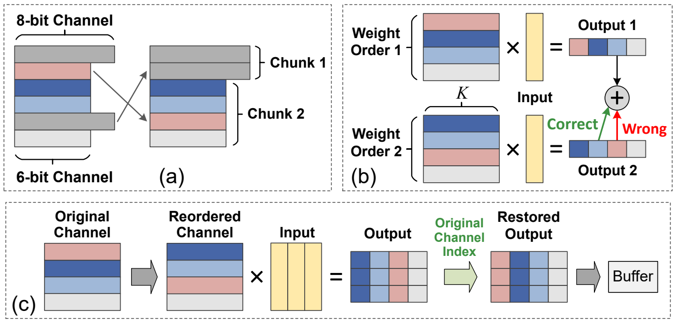

Fig. 9: Channel reordering: (a) Store channels with the same precision in the same memory chunk. (b) Two weight tensors in a residual block with different channel orders can lead to the wrong result when processing the same input. (c) Unshuffle the output to restore the original channel order.

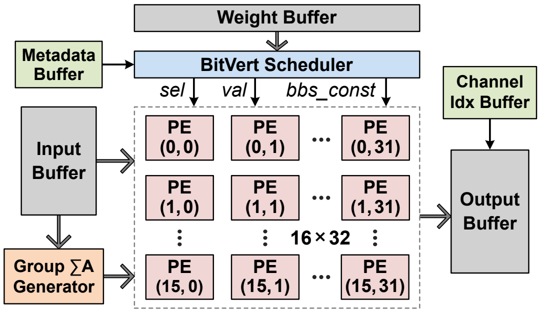

Fig. 10: BitVert accelerator.

B. BitVert Scheduler

BitVert adopts a low-cost scheduler to control the operation within a PE, as illustrated in Fig. 8. To control the bit-serial dot product, the scheduler first identifies whether there are more zero bits in a bit column. It then sends the original or inverted bit column to a series of 4 priority encoders. Every priority encoder receives 5 consecutive bits from the weight bit column. For example, the first priority encoder receives $\{w_0, .., w_4\}$, the second receives $\{w_1, .., w_5\}$, and so on. The encoder detects the location of the first “1” bit in the received 5-bit vector. If exists, it will mask the detected “1” bit and sends the remaining bits to the next encoder. On the other hand, if the received 5-bit vector contains all zero-bits, the encoder will signal $val = 0$ to disable the corresponding bit-serial multiplier in the PE.

The scheduler also generates the col_idx signal to control the shifting of bit-serial multiplier in every PE. When a new dot product begins, the scheduler receives the BBS metadata which contains the number of redundant columns, $\#RedunCol$, in a weight group. The highest bit significance of the compressed weight group indicates the initial col_idx and is obtained by subtracting the number of redundant columns from 7 (i.e., the highest bit significance of uncompressed weight). The col_idx is updated in every cycle by subtracting one until the bit-serial bot product completes.

C. Channel Reordering

With per-channel global binary pruning, the sensitive and normal channels will have different precision, resulting in unaligned memory layout. To address this issue, we adopt a channel reordering mechanism as shown in Fig. 9(a). There are 6 weight channels in this example, and channels with the same precision are grouped together and stored in a memory chunk to avoid unaligned access. Recall from Section III-C that the proposed global binary pruning is hardware-aware, which forces the number of sensitive channels in every layer to be a multiple of the number of channels processed in parallel. Therefore, the grouped channels can be efficiently accessed by *BitVert* to ensure full hardware utilization.

The channel reordering mechanism has also been explored in SparTen’s greedy balancing [13]. However, the reordering criteria is completely different. SparTen is a value-based sparse DNN accelerator that reorders weight channels based on their sparsity, while *BitVert* groups channels based on their sensitivity to binary pruning. Furthermore, SparTen statically unshuffles the next layer’s weights in software, which may not guarantee the correctness when different weight tensors need to process the same input. Consider the example shown in Fig. 9(b), where two weight tensors multiply the same input and generate two output tensors that require element-wise addition (*e.g.*, as in the residual block of ResNet). SparTen statically unshuffles the two weight tensors along the K-dimension to align with the channel order of the previous layer, but the different channel order between the two weight tensors remains, which produce two output tensors with different orders. In this example, the second element of output 2 is supposed to be added with the third element of output 1, while a conventional design like SparTen will add the same position of two output tensors, leading to the wrong result.

To solve the above issue, we propose to unshuffle the output tensor when writing back to memory. As shown in Fig. 9(c), after completing the whole dot product between the input tensor and reordered weight, the outputs are directly restored to the original channel order. This restoring only needs to know the original index of every weight channel to calculate the corresponding memory address for storing the final outputs. Fortunately, since a weight channel usually contains hundreds to thousands of values, the overhead of storing one index per channel is trivial. Moreover, because the same weight channel can process many inputs (3 in this example) to compute many outputs simultaneously, these outputs can be unshuffled together to amortize the cost of channel reordering.

D. BitVert Accelerator

Fig. 10 shows the overall architecture of the *BitVert* accelerator. The 16 × 32 PE array adopts an output-stationary dataflow, and exploits both weight-sharing and input-sharing by processing 32 weight channels and 16 input windows in parallel. The weight and input buffers are banked to provide adequate bandwidth for the access from PEs. Outputs are read out of the PE array and written to the output buffer, one column at a time. Additionally, *BitVert* incorporates a metadata buffer to store BBS compression metadata, and a channel index buffer to store the original index of weight channels being processed. The ΣA generator calculates the sum of input activations for

BBS-based bit-serial multiplication inside the PE. Since the same input group is multiplied by 32 weight channels, the ΣA generator incurs practically no overhead.

V. Evaluation

A. Experimental Methodology

DNN Benchmarks We evaluate seven representative DNN models, including CNNs and transformer networks as summarized in Table I. For CNNs, we evaluate VGG-16, ResNet-34 and ResNet-50 on the ImageNet-1K dataset. For transformers, we choose two vision transformers, ViT-Small and ViT-Base, as well as BERT on MRPC and SST2 tasks from the GLUE dataset [41]. We obtain pre-trained CNNs and transformers from PyTorch Library and HuggingFace, respectively. We then conduct post-training per-channel quantization to obtain the baseline 8-bit models, which shows negligible accuracy loss compared to FP32 models. The 8-bit models are used to evaluate the proposed binary pruning technique and *BitVert* accelerator. For every model, we apply two levels of binary pruning, *conservative* (cons) and *moderate* (mod), with a weight group size of 32. For conservative pruning, 10% sensitive channels are maintained at 8 bits and the remaining channels have 2 bit-columns pruned using the rounded averaging strategy. For moderate pruning, 20% sensitive channels are maintained at 8 bits and the remaining channels have 4 bit-columns pruned using the zero-point shifting strategy.

Accelerator Baselines We compare *BitVert* against six DNN accelerators, including four bit-serial accelerators: Stripes [19], Pragmatic [1], Bitlet [26], BitWave [39], and two value-based accelerators: SparTen [13], ANT [16]. Stripes is an early bit-serial accelerator that exploits reduced precision for DNN computation, yet it mainly relies on 16-bit models and does not consider below-8-bit compression. Therefore, we treat Stripes as a dense bit-serial accelerator and use our baseline 8-bit models to evaluate its performance. Pragmatic and Bitlet target zero-bit skipping during on-chip computation only, while BitWave enhances structured bit-column sparsity to save both computation and memory access. SparTen exploits two-sided value sparsity for DNN acceleration. ANT combines different datatypes in a unified manner for low-bit DNN acceleration. We use 6-bit precision to evaluate ANT, a configuration demonstrated by ANT to maintain acceptable accuracy without the need of retraining.

Implementation We implement the proposed binary pruning algotirhm in Pytorch. We design the *BitVert* accelerator at RTL-level using SystemVerilog and synthesize it with Synopsys Design Compiler in TSMC 28nm technology to find area. We use Synopsys VCS to generate data-driven activity factors at 800 MHz for power estimation. The area and power of on-chip SRAM buffer are modelled with CACTI [4]. To estimate the DRAM power, we use the DDR3 model from DRAMSim3 [22]. For the end-to-end performance evaluation of *BitVert* and other baseline accelerators, we develop cycle-accurate simulators to model the execution time. To ensure a fair comparison, all accelerators are scaled to contain the same number of multipliers, where an 8-bit multiplier is equivalent to eight bit-serial multipliers. For on-chip SRAM, we equip ANT and all bit-serial accelerators with 256 KB activation buffer and 256 KB weight buffer. For SparTen, we reduce the size of its on-chip buffer due to the existence of the local buffer inside every PE.

Type	CNN		Transformer		
Model	VGG-16	ResNet-34 / 50	ViT-S / B	BERT	
Dataset	ImageNet			MRPC	SST2
FP32 Acc %	73.36	73.31 / 76.13	80.16 / 84.54	90.7	91.8
INT8 Acc %	73.35	73.39 / 76.17	80.05 / 84.52	90.4	91.63

TABLE I. Summary of evaluated models and datasets.

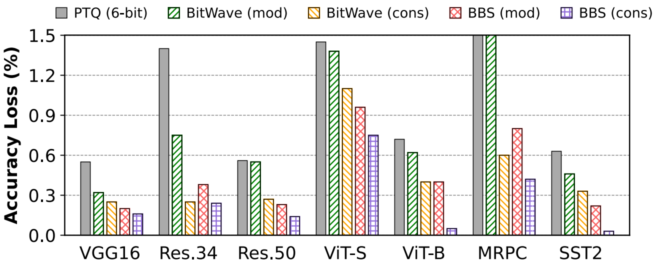

Fig. 11: Comparison of accuracy loss between PTQ, BitWave and BBS under conservative (cons) and moderate (mod) compression.

B. Accuracy Comparison

We first evaluate the accuracy impact of BBS binary pruning compared to naive PTQ and BitWave's bit-flip strategy [39] for compression below 8-bit. When using PTQ for compression, we follow the widely-used calibration [10] by calibrating the quantization parameters based on a subset (1024 images) of the ImageNet dataset. In particular, conventional PTQ relies on the calibration dataset to ensure the optimized quantization parameters and accuracy, while the naive data-free quantization leads to significant accuracy degradation ($> 10\%$). On the contrary, the proposed BBS compresses the model to lower precision **without** any calibration dataset. For both PTQ and BitWave, we use the same setting as BBS by maintaining 20% and 10% sensitive channels for moderate and conservative pruning, respectively. This ensures that our accuracy benefits purely come from the proposed binary pruning.

Fig. 11 shows the accuracy impact of applying different approaches on the baseline DNNs. On average, the conservative and moderate binary pruning can compress the memory footprint of the baseline 8-bit DNNs by $1.29\times$ and $1.66\times$, while incurring an accuracy loss of only 0.25% and 0.45%, respectively. Both BitWave and BBS with moderate pruning can attain higher accuracy than PTQ. These accuracy improvements stem from their ability to exploit fine-grained bit-level redundancy, thereby preserving more information from the original 8-bit models. Additionally, the proposed binary pruning consistently outperforms BitWave. This is because BBS allows any bit significance to be zero or one, thus retaining all quantization levels of the 8-bit precision.

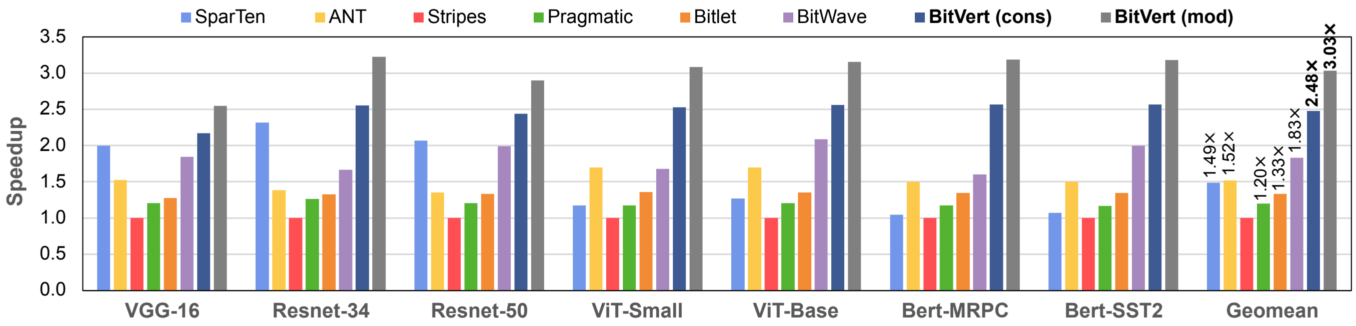

Fig. 12: Speedup results normalized to Stripes (higher is better).

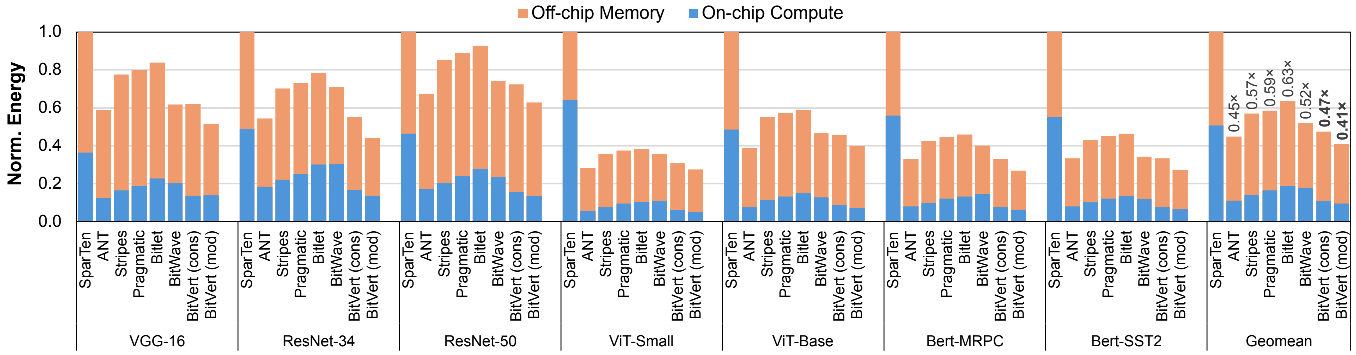

Fig. 13: Energy consumption breakdown normalized to SparTen (lower is better).

Comparison against ANT We compare the accuracy between moderate binary pruning and ANT [16]. As shown in Table II, BBS outperforms ANT in terms of both accuracy and effective weight bit width. While ANT uses adaptive datatypes for low-bit quantization, it cannot take the advantage of inherent bit-level redundancy. On the other hand, the binary pruning fully exploits the bit-level sparsity to best preserve the original 8-bit weight distribution, resulting in minimal accuracy degradation.

Comparison against PTQ Works We compare the accuracy loss between BBS and state-of-the-art PTQ works, including Microscaling [36] and NoisyQuant [24], on vision transformers. We apply 6-bit weight quantization using the two PTQ methods while maintaining activation to 8-bit. Table III shows that the moderate binary pruning outperforms NoisyQuant with lower memory footprint. Moreover, the conservative binary pruning has much better accuracy than Microscaling at similar bit width. Miscroscaling also has an 8-bit metadata, which represents the shared exponent for a group of 32 weights. However, the exponent is determined by the largest value in every group, which forces small values to become zero due to insufficient operand precision to store the aligned mantissa. On the other hand, BBS exploits bit-level redundancy to better preserve the statistical characteristics of uncompressed weight, thereby achieving higher accuracy.

C. Accelerator Performance and Energy

Performance Fig. 12 presents the accelerator performance normalized to that of Stripes. On average, *BitVert* with conservative and moderate binary pruning achieves 2.48× and 3.03× speedup compared to Stripes, respectively. These speedups are attributed to exploiting both balanced BBS and binary pruning for abundant bit skipping and reduced memory access. Despite leveraging two-sided value sparsity, SparTen demonstrates limited performance on transformer-based models due to the lack of weight value sparsity in 8-bit models and nearly-dense activations from non-ReLU functions. ANT only explores reduced value precision but not fine-grained bit-level sparsity, leading to 1.63× and 1.97× lower speedup than *BitVert* at conservative and moderate pruning, respectively. While Pragmatic and Bitlet utilize variable degrees of bit-level sparsity, they suffer from workload imbalance and lack of exploration in further compressing DNNs below 8-bit. This explains why *BitVert* outperforms Pragmatic and Bitlet by 1.86 – 2.53× across all benchmarks. Although BitWave exploits structured

Model	**BBS (mod)**	ANT [16]
VGG-16	**0.2% (4.32 bits)**	0.68% (6 bits)
ResNet-50	**0.23% (4.79 bits)**	0.89% (6 bits)

TABLE II. Comparison of accuracy loss and weight bit width between BBS and 6-bit ANT without fine-tuning.

	ViT-Small		ViT-Base	
	Δ Acc ↓	Bits	Δ Acc ↓	Bits
Microscaling [36]	2.49%	6.25	0.33%	6.25
NoisyQuant [24]	2.08%	6	0.64%	6
BBS (cons)	0.75%	6.33	0.05%	6.25
BBS (mod)	0.96%	5.19	0.39%	5.07

TABLE III. Comparison of accuracy loss and weight bit width between BBS, Microscaling and NoisyQuant.

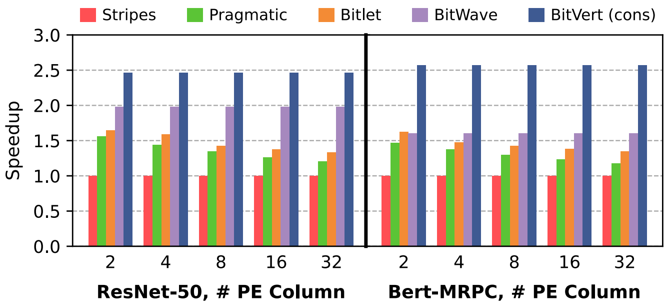

Fig. 14: Normalized speedup on ResNet-50 and Bert-MRPC with increasing number of PE columns (*i.e.*, processing more weight groups in parallel).

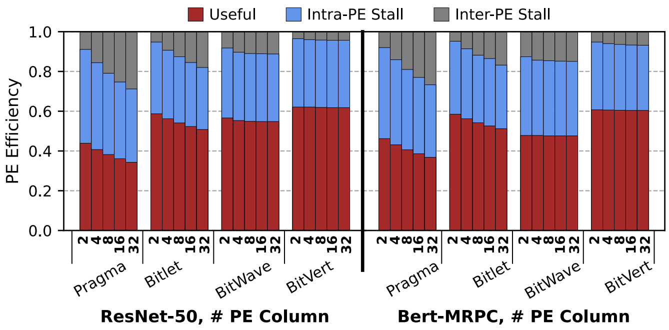

Fig. 15: Breakdown of execution cycles w.r.t. the number of PE columns.

bit-column pruning to achieve better performance, its moderate pruning results in unacceptable accuracy loss ($> 1\%$) on many DNNs such as ViT-small and Bert-MRPC. Therefore, it has to reduce the degree of pruning for improved accuracy while sacrificing performance. Overall, *BitVert* provides the best accuracy-performance trade-offs, with up to $1.98\times$ speedup over BitWave.

Energy Consumption Fig. 13 presents the normalized energy breakdown of different accelerators. where the on-chip compute energy includes both buffer and core energy. SparTen demonstrates the poorest energy efficiency primarily due to its substantial overhead from the sparse bitmask encoding (12.5% at 8-bit precision) and the expensive hardware required to exploit sparsity. This overhead is particularly pronounced in 8-bit DNNs, where value sparsity is inherently scarce. As a result, SparTen consumes $2.13\times$ and $2.44\times$ higher energy than *BitVert* with conservative and moderate pruning, respectively. Although ANT is able to quantize both activations and weights, it dissipates higher energy than *BitVert* with moderate pruning due to the complicated hardware to support custom data types. Owing to the balanced BBS-skipping and substantial reduction in model size, *BitVert* with moderate pruning achieves an average energy reduction of $1.39\times$, $1.43\times$, $1.54\times$, and $1.27\times$ over Stripes, Pragmatic, Bitlet, and BitWave, respectively.

D. Analysis of Load Imbalance

BitVert can leverage the structured BBS for improved load balance. Fig. 14 demonstrates this with the performance on ResNet-50 and Bert-MRPC with respect to different number of PE columns, where every PE column processes a different weight group. When there are more PE columns, Pragmatic and Bitlet exhibit a noticeable drop in speedup over Stripes that does not exploit bit sparsity. For instance, when the number of PE columns increases from 2 to 32, the speedup of Bitlet on Bert-MRPC drops from $1.63\times$ to $1.35\times$. This is because that processing more weight groups in parallel exacerbates the load imbalance across PE columns, and the performance is bottlenecked by the weight group with the lowest bit sparsity. In contrast, the structured bit sparsity allow BitWave and *BitVert* to efficiently scale the performance, thus maintaining nearly constant speedup over Stripes. Moreover, *BitVert* always achieves the highest performance thanks to the binary pruning that can induce higher BBS with negligible accuracy loss.

Fig. 15 further details the breakdown of execution time with respect to the number of PE columns to highlight its impact on load balance. Since one PE contains many bit-serial multipliers, intra-PE stall can be caused by a multiplier that needs to process more effectual bits. On the other hand, the inter-PE stall arises from variance in bit sparsity across different weight groups. As the number of PE columns increases, Pragmatic and Bitlet experience higher intra-PE and inter-PE loss, which explains their lower resulting speedup. BitWave only exploits coarse-grained bit-column sparsity that has much lower occurrence than fine-grained BBS. Therefore, it shows lower PE utilization than *BitVert*. Furthermore, *BitVert* has minimal inter-PE stall due to the more balanced distribution of BBS across different weight groups, thereby achieving superior performance over other bit-serial accelerators.

E. PE Design Space Exploration

Recall from Section IV-A that the sub-group size within the *BitVert* PE offers a trade-off between area and power. A smaller sub-group has lower mux cost, but increases the number of subtractors. Furthermore, by exploiting the structured nature of BBS and its encoding scheme, we are able to further reduce the PE area by using compact mux and a smaller BBS multiplier. Hence, we conduct a PE design space exploration to evaluate the optimal group size and the proposed optimizations. As shown in Table IV, a sub-group size of 16 without optimization incurs a significant area overhead of 38.2% compared to the optimized design. In the end, a sub-group size of 8 with the proposed PE optimization offers the best trade-off between area and power, which is therefore adopted in our *BitVert* accelerator.

Sub-group	Without Optimization		With Optimization	
Size	Area (um^2)	Power (mW)	Area (um^2)	Power (mW)
16	1342.3	0.61	971.5	0.53
8	896.6	0.49	739.6	0.45
4	878.7	0.51	786.5	0.47

TABLE IV. PE area and power of *BitVert* with different sub-group sizes before and after applying our circuit optimizations.

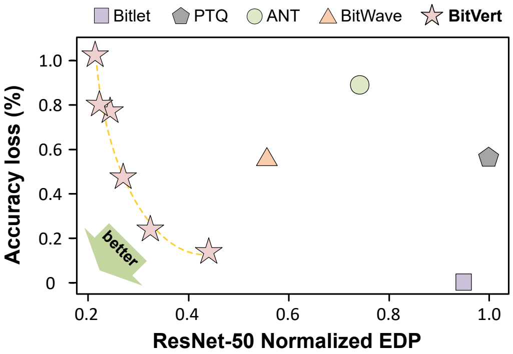

Fig. 16: EDP-acccuracy loss pareto frontier for ResNet50.

F. PE Area and Power Comparison

The *BitVert* accelerator adopts an area- and energy-efficient PE with low overhead to support BBS. We compare the PE design of *BitVert* and other bit-serial accelerators, with all PEs containing 8 bit-serial multipliers at 800 MHz target frequency. Table V summarizes the area and power of different PEs. Bitlet experiences the highest area and power consumption due to significant overhead (*e.g.*, a 64-1 mux before every bit-serial multiplier) for zero bit skipping. Pragmatic needs a variable shifter to align the bit significance, leading to a larger bit-serial multiplier and non-trivial overhead. BitWave requires 2's complementer to support sign-magnitude arithmetic, resulting in 1.32× larger area and 1.4× power than Stripes. Moreover, since BitWave can only leverage coarse-grained bit-column sparsity, the potential performance improvement is limited. The proposed *BitVert* enjoys the optimal trade-off between performance and hardware cost. Its PE occupies 1.39× area and consumes 1.22× power compared to Stripes, yet is able to exploit 50% balanced BBS and binary pruning for efficient bit skipping and model compression, respectively. Since BBS naturally exists in a bit-vector with arbitrary length and does not depend on the operand precision, it provides a promising solution for future bit-serial computing paradigm.

G. Accuracy-Efficiency Trade-offs

The proposed binary pruning and *BitVert* can offer good trade-offs between accuracy and efficiency. To demonstrate this, we conduct design-space exploration on ResNet-50 with different pruning ratios. We compare the relationship between energy-delay product (EDP) and accuracy loss of *BitVert* and previous works, including Bitlet, BitWave, ANT and conventional PTQ. As shown in Fig. 16, the lower left region indicates a good trade-off between accuracy and EDP. Although BitWave and ANT propose different algorithm-hardware co-design approaches for DNN compression and acceleration, they fail to preserve the original value distribution of the baseline model and do not efficiently leverage the balanced bit sparsity that inherently appears in DNNs. In contrast, binary pruning is able to preserve all quantization levels of the original DNN. Combining with BBS and efficient hardware design, *BitVert* is able to always sit on the Pareto frontier.

Accelerator	PE Area (um^2)				PE Power
	Multiplier	Others	Total	Ratio	(mW)
Stripes [19]	286.3	246.5	532.8	1×	0.37
Pragmatic [1]	319.2	603.9	923.1	1.73×	0.51
Bitlet [26]	223.2	1442.4	1665.6	3.13×	0.57
BitWave [39]	286.3	416.1	702.4	1.32×	0.49
BitVert (ours)	**332.4**	**407.2**	**739.6**	**1.39×**	**0.45**

TABLE V. PE area and power of BitVert and prior bit-serial accelerators under 28 nm technology and 800 MHz frequency.

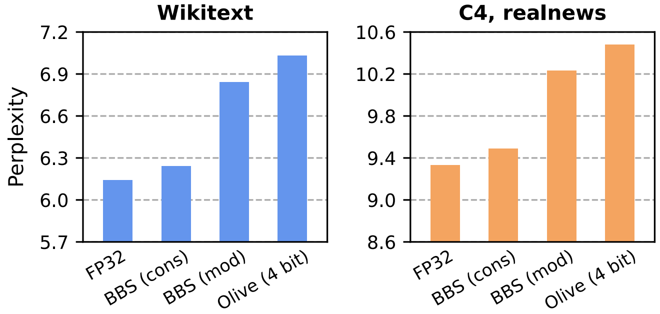

Fig. 17: Comparison between BBS and Olive on compressing Llama-3-8B weights. The accuracy metric is perplexity, **lower is better**.

Accelerator	Area (um^2)	Power (mW)	Norm. Perf	Norm. Perf / Area
Olive [15]	291.6	0.18	1×	1×
BitVert (mod)	739.6	0.45	4×	1.58×

TABLE VI. Comparison between Olive and *BitVert* PEs.

H. Applicability to Large Language Models

Large language models (LLMs) have achieved great success in generative tasks [40], [47]. We compare BBS with a recent PTQ work Olive [15] for LLM weight compression. We evaluate a state-of-the-art LLM, Llama-3-8B [29] on Wikitext [28] and C4 [8] datasets. For BBS, we apply conservative and moderate binary pruning to *all* weight channels with a group size of 32, resulting in an effective weight precision of 6.25 and 4.25 bits, respectively. Fig. 17 shows the accuracy impact of different compression methods. The moderate BBS pruning achieves better perplexity than Olive with a similar memory footprint (4.25 vs. 4 bits), while the conservative BBS pruning has little perplexity loss compared to the FP32 baseline. To compare the hardware efficiency, we synthesize the Olive PE for 4-bit weight and 8-bit activation. Table VI shows that the proposed *BitVert* PE with moderate binary pruning can achieve 1.58× better performance per area compared to Olive. The benefits of *BitVert* are twofold. First, Olive adopts separate datatypes for normal and outlier values, where the latter has a much wider numerical range. Therefore, the Olive PE requires a larger multiplier than fixed-point PE to accommodate outliers. Second, the *BitVert* PE exploits BBS to efficiently compute 16 multiplications in 4 cycles under moderate pruning, while the Olive PE does not leverage bit sparsity and only computes one multiplication per cycle.

VI. Conclusion

In this paper, we introduce BBS, a new concept to exploit bit-level sparsity in a symmetrical way to prune either zero-bits or one-bits. BBS pushes the limit of post-training DNN compression to a new state-of-the-art through binary pruning, a data-free optimization that generates bi-directional sparse bit columns inside DNN weights while maximally preserving the statistical characteristics of the original uncompressed model. As a result, the proposed binary pruning technique achieves much higher accuracy compared to previous bit-sparsity-aware pruning methods. On top of the algorithmic innovation, we design a bit-serial accelerators named *BitVert* with an area- and power-efficient PE to fully mine the potential of BBS. Compared to prior DNN accelerators, *BitVert* achieves up to 3.03× speedup and 2.44× energy saving, while having negligible accuracy degradation on both vision and language models with large-scale benchmark datasets.

Acknowledgment

This project is supported in part by Intel Corporation and the Center for the Co-Design of Cognitive Systems (CoCoSys) in JUMP 2.0, an SRC Program sponsored by DARPA. We would like to thank Mahesh Iyer, Grace Zgheib, Sergey Gribok, Ahmed AbouElhamayed, Zhewen Yu, Marta Andronic, and the anonymous reviewers for their constructive feedback. We also thank Man Shi for the helpful discussion about BitWave. The code for BBS binary pruning can be found at `https://github.com/yc2367/BBS-MICRO.git`.

References

[1] J. Albericio, A. Delmas, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-Pragmatic deep neural network computing," *50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[2] H. An, Y. Chen, Z. Fan, Q. Zhang, P. Abillama, H.-S. Kim, D. Blaauw, and D. Sylvester, "An 8.09tops/w neural engine leveraging bit-sparsified sign-magnitude multiplications and dual adder trees," *IEEE International Solid- State Circuits Conference (ISSCC)*, pp. 422–424, 2023.

[3] T. Andrulis, J. S. Emer, and V. Sze, "RAELLA: Reforming the arithmetic for efficient, low-resolution, and low-loss analog pim: No retraining required!" *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA)*, 2023.

[4] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, June 2017.

[5] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "Zeroq: A novel zero shot quantization framework," *arXiv preprint arXiv:2001.00281*, 2020.

[6] C. Deng, Y. Sui, S. Liao, X. Qian, and B. Yuan, "GoSPA: An energy-efficient high-performance globally optimized sparse convolutional neural network accelerator," *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *North American Chapter of the Association for Computational Linguistics*, 2019.

[8] J. Dodge, A. Marasovic, G. Ilharco, D. Groeneveld, M. Mitchell, and M. Gardner, "Documenting large webtext corpora: A case study on the colossal clean crawled corpus," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint abs/2010.11929*, 2020.

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

[11] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv preprint arXiv:2103.13630*, 2021.

[12] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "AI and memory wall," *IEEE Micro*, 2024.

[13] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. N. Vijaykumar, "SparTen: A sparse tensor accelerator for convolutional neural networks," *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[14] A. Gondimalla, M. Thottethodi, and T. N. Vijaykumar, "Eureka: Efficient tensor cores for one-sided unstructured sparsity in dnn inference," *2023 56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023.

[15] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y.-B. Liu, M. Guo, and Y. Zhu, "OliVe: Accelerating large language models via hardware-friendly outlier-victim pair quantization," *50th ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2023.

[16] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y.-B. Liu, M. Guo, and Y. Zhu, "ANT: Exploiting adaptive numerical data type for low-bit deep neural network quantization," *55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022.

[17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[18] D. Im, G. Park, Z. Li, J. Ryu, and H.-J. Yoo, "Sibia: Signed bit-slice architecture for dense dnn acceleration with slice-level sparsity exploitation," *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[19] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[20] A. D. Lascorz, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, K. Siu, and A. Moshovos, "Bit-Tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks," *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[21] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," *arXiv preprint arXiv:1810.02340*, 2019.

[22] S.-J. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "DRAMsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, pp. 106–109, 2020.

[23] F. Liu, W. Zhao, Z. He, Z. Wang, Y. Zhao, Y. Chen, and L. Jiang, "Bit-Transformer: Transforming bit-level sparsity into higher preformance in reram-based accelerator," *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021.

[24] Y. Liu, H. Yang, Z. Dong, K. Keutzer, L. Du, and S. Zhang, "NoisyQuant: Noisy bias-enhanced post-training activation quantization for vision transformers," *arXiv preprint arXiv:2211.16056*, 2023.

[25] Z. Liu, Y. Wang, K. Han, S. Ma, and W. Gao, "Post-training quantization for vision transformer," *arXiv preprint arXiv:2106.14156*, 2021.

[26] H. Lu, L. Chang, C. Li, Z. Zhu, S. Lu, Y. Liu, and M. Zhang, "Distilling bit-level sparsity parallelism for general purpose deep learning acceleration," *54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021.

[27] E. Meller, A. Finkelstein, U. Almog, and M. Grobman, "Same, same but different - recovering neural network quantization error through weight factorization," *arXiv preprint arXiv:1902.01917*, 2019.

[28] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[29] Meta, "Meta llama 3." [Online]. Available: https://github.com/meta-llama/llama3

[30] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius, "Accelerating sparse deep neural networks," *arXiv preprint arXiv:2104.08378*, 2021.

[31] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient transfer learning,” *International Conference on Learning Representations,*, 2017.

[32] M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling, “Data-free quantization through weight equalization and bias correction,” *arXiv preprint arXiv:1906.04721*, 2019.

[33] NVIDIA, “Tensorrt: A c++ library for high performance inference on nvidia gpus and deep learning accelerators.” [Online]. Available: https://github.com/NVIDIA/TensorRT

[34] E. Park, D. Kim, and S. Yoo, “Energy-efficient neural network accelerator based on outlier-aware low-precision computation,” *ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018.

[35] S. Qu, B. Li, Y. Wang, and L. Zhang, “ASBP: Automatic structured bit-pruning for rram-based nn accelerator,” *58th ACM/IEEE Design Automation Conference (DAC)*, 2021.

[36] B. D. Rouhani, R. Zhao, V. Elango, R. Shafipour, M. Hall, M. Mesmakhosroshahi, A. More, L. Melnick, M. Golub, G. Varatkar, L. Shao, G. Kolhe, D. Melts, J. Klar, R. L’Heureux, M. Perry, D. Burger, E. S. Chung, Z. Deng, S. Naghshineh, J. Park, and M. Naumov, “With shared microexponents, a little shifting goes a long way,” *ACM/IEEE 50th Annual International Symposium on Computer Architecture (ISCA)*, 2023.

[37] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos, “Laconic deep learning inference acceleration,” *ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, 2019.

[38] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, “Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network,” in *45th ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2018.

[39] M. Shi, V. Jain, A. Joseph, M. Meijer, and M. Verhelst, “BitWave: Exploiting column-based bit-level sparsity for deep learning acceleration,” *Proceedings of the 30th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024.

[40] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, “LLaMA: Open and efficient foundation language models,” *arXiv preprint arXiv:2302.13971*, 2023.

[41] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, “GLUE: A multi-task benchmark and analysis platform for natural language understanding,” *arXiv preprint arXiv:1804.07461*, 2018.

[42] Y. Wang, C. Zhang, Z. Xie, C. Guo, Y. Liu, and J. Leng, “Dual-side sparse tensor core,” *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[43] Y. N. Wu, P.-A. Tsai, S. Muralidharan, A. Parashar, V. Sze, and J. S. Emer, “HighLight: Efficient and flexible dnn acceleration with hierarchical structured sparsity,” *56th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023.

[44] G. Xiao, J. Lin, M. Seznec, J. Demouth, and S. Han, “SmoothQuant: Accurate and efficient post-training quantization for large language models,” *arXiv preprint arXiv:2211.10438*, 2022.

[45] Z. Yuan, C. Xue, Y. Chen, Q. Wu, and G. Sun, “PTQ4ViT: Post-training quantization framework for vision transformers with twin uniform quantization,” *arXiv preprint arXiv:2111.12293*, 2022.

[46] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, “GOBO: Quantizing attention-based nlp models for low latency and energy efficient inference,” *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020.

[47] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. T. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, “OPT: Open pre-trained transformer language models,” *arXiv preprint arXiv:2205.01068*, 2022.