RESEARCH-ARTICLE

SmartBlock: Adaptive Block Floating Point Quantization for Efficient DNN Acceleration

XIN JU, National University of Defense Technology China, Changsha, Hunan, China

JINGKUI YANG, National University of Defense Technology China, Changsha, Hunan, China

MEI WEN, National University of Defense Technology China, Changsha, Hunan, China

JUN HE, National University of Defense Technology China, Changsha, Hunan, China

JING FENG, National University of Defense Technology China, Changsha, Hunan, China

MINJIN TANG, National University of Defense Technology China, Changsha, Hunan, China

View all

Published: 08 September 2025

ICPP '25: 54th International Conference on Parallel Processing
September 8 - 11, 2025
CA, San Diego, USA

ICPP '25: Proceedings of the 54th International Conference on Parallel Processing (September 2025)
https://doi.org/10.1145/3754598.3754660
ISBN: 9798400720741

SmartBlock: Adaptive Block Floating Point Quantization for Efficient DNN Acceleration

Xin Ju
National University of Defense Technology
Changsha, China
jx@nudt.edu.cn

Jingkui Yang
National University of Defense Technology
Changsha, China
yangjingkui2001@nudt.edu.cn

Mei Wen*
National University of Defense Technology
Changsha, China
meiwen@nudt.edu.cn

Jun He
National University of Defense Technology
Changsha, China
hejun19@nudt.edu.cn

Jing Feng
National University of Defense Technology
Changsha, China
fengjing22@nudt.edu.cn

Minjin Tang
National University of Defense Technology
Changsha, China
tangminjin@nudt.edu.cn

Zhaoyun Chen
National University of Defense Technology
Changsha, China
chenzhaoyun@nudt.edu.cn

Yang Shi
National University of Defense Technology
Changsha, China
shiyang14@nudt.edu.cn

Abstract

Deep Neural Networks (DNNs) have achieved remarkable success as model sizes continue to grow, driving the need for optimizations in both computational and energy efficiency. Block Floating Point (BFP) quantization has emerged as an effective model compression technique, offering a favorable trade-off between model accuracy and hardware cost. However, the frequent use of floating-point (FP) accumulation across BFP blocks remains a significant bottleneck, limiting further improvements in energy efficiency. State-of-the-art (SotA) accelerators mitigate this issue by introducing low-overhead accumulators with a narrower dynamic range ahead of the FP accumulator to handle a small range of values. While this approach reduces the activation of power-hungry alignment and format conversion units, it increases the complexity of the processing elements (PEs), thereby limiting the overall energy savings.

To address these limitations, we propose *SmartBlock*, a novel algorithm-architecture co-designed scheme that employs dynamic block size quantization combined with a unified PE architecture. *SmartBlock* achieves high energy efficiency and low area overhead while maintaining superior model accuracy. Our key insight is that although DNN data exhibit a wide dynamic range, most values are densely distributed, with only a small fraction being outliers. This observation allows the use of larger BFP block sizes in outlier-free regions, effectively reducing reliance on FP accumulation and thereby improving overall efficiency.

Experimental results on six CNN- and transformer-based models across multiple representative datasets demonstrate that *SmartBlock* achieves up to 1.39× and 1.65× speedup, and reduces energy consumption by 32.04% and 42.76% compared to the SotA architectures, WinAcc and Bucket, respectively.

CCS Concepts

• **Computer systems organization → Systolic arrays**.

Keywords

Block Floating Point, Quantization, Energy Efficiency, Deep Neural Network.

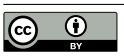
ACM Reference Format:
Xin Ju, Jingkui Yang, Mei Wen, Jun He, Jing Feng, Minjin Tang, Zhaoyun Chen, and Yang Shi. 2025. SmartBlock: Adaptive Block Floating Point Quantization for Efficient DNN Acceleration. In *54th International Conference on Parallel Processing (ICPP '25), September 08–11, 2025, San Diego, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3754598.3754660

1 Introduction

Deep neural networks (DNNs) have achieved remarkable success across a wide range of application domains, including computer vision, natural language processing, and recommendation systems [23, 30]. However, the rapid increase in model complexity has led to substantial growth in both parameter sizes and computational demands [21, 36], posing significant challenges for efficient deployment. One promising approach to address these challenges is the use of low-bit data representations through quantization [12, 20, 22, 39, 40], which can significantly reduce memory footprint and computational overhead. Among them, Block Floating Point (BFP) [46] has emerged as a particularly attractive solution.

BFP is a efficient number format in which each block (e.g., 16 values) is encoded as individual sign-magnitude integers sharing a

*Corresponding Author



ICPP '25, San Diego, CA, USA

ACM ISBN 979-8-4007-2074-1/25/09
https://doi.org/10.1145/3754598.3754660

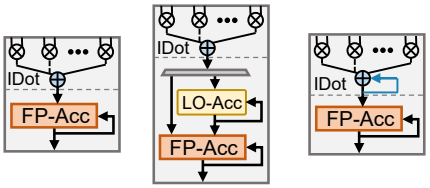
common exponent. This structure replaces power-hungry floating-point (FP) operations with more energy-efficient integer (INT) operations within each block, and has been widely adopted in both academic research and industrial designs [27, 39, 40, 49]. While BFP effectively converts FP multiplications and additions into INT operations, the need for FP accumulation across blocks remains a significant bottleneck, hindering further gains in energy efficiency and silicon area reduction.

To mitigate this issue, prior BFP-based approaches have explored enhancements to the accumulation architecture. As summarized in Figure 1, FAST [49] adopts a basic BFP-PE design that triggers the FP accumulator (FP-Acc) for every inter-block operation. Bucket [28] partitions accumulation terms into six groups based on their exponent values, using low-overhead accumulator (LO-Acc) for intra-group accumulation and resorting to FP-Acc only for values with exponents falling outside these predefined groups. This strategy reduces the frequency of costly FP operations. Similarly, WinAcc [19] introduces a configurable LO-Acc with an adjustable dynamic range, effectively capturing the majority of densely distributed values while delegating only extreme outliers to the FP-Acc. Although both Bucket and WinAcc reduce dependence on FP-Accs by incorporating LO-Accs, their hardware implementations remain relatively complex due to the additional control logic and specialized accumulators. As a result, FP accumulation continues to be a bottleneck in energy and area efficiency for state-of-the-art (SotA) BFP-based accelerators.

In this work, we propose *SmartBlock*, a software-hardware co-designed scheme for efficient DNN acceleration, leveraging low-bit BFP formats with dynamic block sizes. On the software side, *SmartBlock* performs vector-wise dynamic block size quantization (DBSQ) using BFP. Specifically, small block sizes are applied in tensor regions with high concentrations of outliers as fine-grained quantization to preserve accuracy. In contrast, large block sizes are utilized in outlier-free regions as coarse-grained quantization to improve energy efficiency. To further reduce memory and decoding overhead, block size parameters are encoded directly into the BFP mantissa during the rounding process. On the hardware side, we adopt an iterative computing paradigm and design a unified PE architecture that supports various BFP block sizes, achieving a balance between computational performance and hardware cost.

Our key contributions are as follows:

- We propose *SmartBlock*, a software-hardware co-designed solution with DBSQ that adaptively adjusts block sizes based on value distributions.
- We develop a hardware-friendly block size encoding scheme that embeds block size information into the BFP mantissa during the rounding stage, which eliminates the need for additional storage and decoder logic, with negligible impact on accuracy.
- We design a unified PE architecture capable of executing BFP operations with variable block sizes. For large blocks, BFP data are automatically decomposed and accumulated over multiple cycles, enabling an effective trade-off between performance and energy efficiency.

	FAST [49]	Bucket [28] & WinAcc [19] †	***SmartBlock***
Block Size	16	16	**Dynamic**
PE Architecture			
FP-Acc Usage	High	**Low**	**Low**
Energy Cost	High	Medium	**Low**
HW Complexity	**Low**	High	**Low**

† The architecture of LO-Acc is different.

Figure 1: Comparative Analysis of SotA BFP-Based PEs. *SmartBlock* adaptively configures block size based on data distribution, with over 50% of blocks exceeding 16. This enables more values to be accumulated using IDot, significantly reducing FP accumulator usage and thereby lowering energy cost.

We evaluate *SmartBlock* on six CNN- and transformer-based models across multiple representative datasets. On average, *SmartBlock* achieves speedups of 1.39×, 1.65×, and 1.18×, and energy reductions of 32.04%, 42.76%, and 53.34% compared to WinAcc, Bucket, and FAST, respectively.

2 Background and Motivation

2.1 Numerical Formats for DNNs

The number format employed in DNNs plays a critical role in determining both model accuracy and hardware efficiency. As a result, extensive research has focused on developing efficient numerical representations for DNN computation, as illustrated in Figure 2. The standard 32-bit IEEE-754 FP format (FP32) [1] provides a wide dynamic range and high precision but results in substantial hardware and energy overhead. Thus low-bit formats have been widely adopted, to significantly reduce memory footprint and computational complexity with a careful trade-off between dynamic range and hardware complexity. ❶ Fixed-point formats (e.g., INT4, INT8) are highly hardware-efficient due to their simple arithmetic logic. However, their limited dynamic range often causes significant accuracy degradation, particularly in large-scale and high-variance DNNs [39]. ❷ Low-bit FP formats such as BrainFloat [20] and TensorFloat [22], adopted by Google and NVIDIA respectively, reduce the bit-width of mantissas while retaining 8-bit exponents (e.g., 1s7m8e and 1s10m8e). While these formats reduce hardware costs compared to FP32, their computational overhead remains nontrivial. ❸ BFP formats offer a middle ground between fixed-point and FP representations. In BFP, each block of values shares a common exponent while maintaining individual mantissas. For instance, Microsoft's MX-INT4 [40] (also referred to as MSFP-12) uses a single 8-bit exponent shared across sixteen 4-bit sign-magnitude mantissas. This design enables power-efficient INT dot products within blocks while supporting a wider dynamic range via inter-block FP

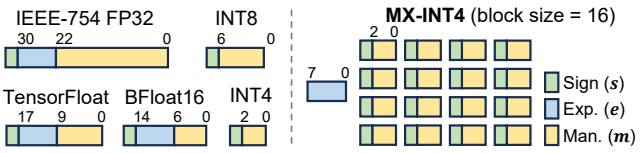
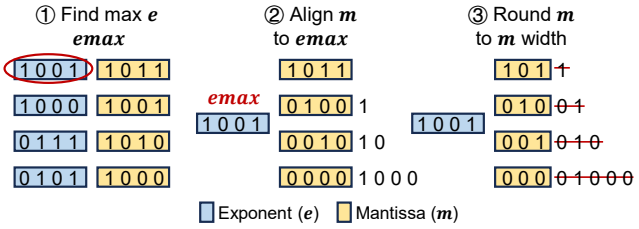
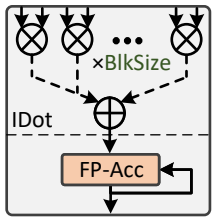


Figure 2: Number formats in DNN training and inference



(a) A case for BFP quantization process

```
for m in 0..M:
  for n in 0..N:
    for k in 0..K:
      if (k + 1) % BlkSize:
        conduct INT Dot Product
      else:
        conduct FP Accumulate
```



(b) BFP matrix multiplication dataflow

Figure 3: BFP quantization and matrix multiplication.

accumulation, making BFP a compelling solution for efficient DNN acceleration.

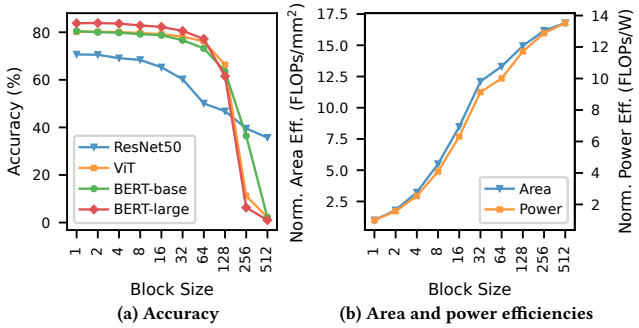
Despite its advantages, BFP still suffers from a key limitation: *it requires frequent FP accumulation for small block sizes (e.g., 16), leading to substantial energy consumption.* This emerging bottleneck limits further efficiency gains and motivates the exploration of alternative strategies.

2.2 BFP Quantization Challenges: Block Size Matters

We demonstrate the importance of the BFP block size on quantization accuracy and multiplication cost. The quantization process of BFP is illustrated in Figure 3(a), where sign bits are omitted for simplicity. For each block of values, the maximum exponent e_{max} is selected as the shared exponent. Each individual mantissa m is then right-shifted by the difference between e_{max} and its private exponent e_{prt}, followed by a rounding operation (e.g., truncated rounding). The resulting quantization error can be expressed as:

$$2^{e_{prt}-bias} \times \frac{m[e_{max} - e_{prt} + 1 : 0]}{2^{mantissaBitWidth}} \quad (1)$$

This formulation reveals that a narrow variation among values within a block (achievable by reducing the block size) leads to fewer bits being truncated during quantization, thereby reducing



(a) Accuracy **(b) Area and power efficiencies**

Figure 4: Evaluation impact of block size on (a) model accuracy and (b) hardware efficiency.

quantization error. This effect is especially pronounced in blocks containing outliers, where smaller values may be completely lost due to exponent alignment. Figure 4(a) demonstrates this effect empirically: *the smaller the block size, the higher the model accuracy.*

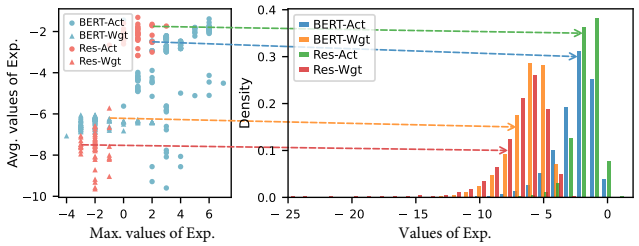
The BFP-based matrix multiplication is illustrated in Figure 3(b), where input matrices of size $(M, K) \times (K, N)$ are processed using a block size denoted as *BlkSize*. Within each block, dot products are computed using INT dot product (IDot) units, taking advantage of the efficiency of fixed-point operations. However, accumulations across blocks still be performed using FP arithmetic, which remains a dominant contributor to hardware area and power overhead. Larger blocks allow more values to share the same exponent, thus aggregating more computations within the INT domain before invoking the expensive FP-Acc. This reduction in FP usage leads to measurable improvements in both area and energy efficiency, as shown in Figure 4(b).

Hence, there exists a fundamental conflict between quantization accuracy and hardware efficiency: larger blocks improve hardware efficiency but may degrade model accuracy, whereas smaller blocks enhance accuracy at the cost of hardware benefits, which is a critical issue that must be addressed.

2.3 Motivation: Towards Dynamic Block Size Quantization

As we discussed in Section 2.2, small block sizes improve model accuracy by mitigating quantization error, while large block sizes enhance hardware efficiency by reducing FP accumulation frequency . However, existing BFP-based accelerators typically adopt a fixed block size (e.g., 16) to strike a conservative compromise [4, 39, 40, 49], limiting the potential for further optimization. There remains a critical need for a more effective algorithm–hardware co-design that better balances model accuracy and hardware overhead. In response to this demand, this work seeks to fundamentally reduce the reliance on FP operations, thereby lowering the associated hardware cost.

Fortunately, prior researches have shown that both weights and activations in DNNs typically follow a Laplace-like distribution [12, 26]. Building upon this insight, we further observe that the exponent values of weights and activations exhibit similar distribution patterns. As a consequence, we profile the tensor-wise



(a) Exponnet profiling (b) Exponent distribution

Figure 5: The tensor-wise exponent profiling and distribution in DNNs.

exponent distributions of activations and weights in ResNet [13] and BERT [8], as shown in Figure 5. Although different tensors demonstrate distinct exponent ranges (Figure 5(a)), we find that a Laplace distribution approximates most tensors well (Figure 5(b)). This implies that exponent values are densely concentrated, with only a small fraction of outliers within each tensor.

Leveraging this observation, we propose DBSQ that adapts to the presence of outliers. Specifically, we assign larger block sizes (e.g., 256 or 512) to regions without outliers to minimize hardware overhead, and smaller block sizes (e.g., 8) around outliers to reduce quantization error. This approach enables a fine-grained trade-off between accuracy and efficiency across the model, overcoming the limitations of fixed-block strategies. To the best of our knowledge, this is the first work to incorporate dynamic block sizing into BFP-based quantization.

3 Quantization ALGORITHMIC IMPLEMENTATION

In this section, we introduce the DBSQ algorithm and the hardware-friendly block size parameter encoding scheme of *SmartBlock*. We first detail how to adaptively quantize the input tensor based on its numerical distribution (Section 3.1), followed by an explanation of the block size parameter encoding mechanism (Section 3.2).

3.1 Dynamic Block Size Quantization

As discussed in Section 2.3, the exponents within a tensor tend to concentrate within a limited value range. This observation allows us to use larger block sizes to represent the majority of values with negligible accuracy loss. The key challenge lies in configuring different block sizes for different tensor regions, based on the distribution of outliers, while keeping the blocks as large as possible to maximize hardware efficiency without compromising accuracy.

In response to this demand, we propose DBSQ based on the MSFP framework [39] to optimize quantization granularity. The algorithm operates through an iterative refinement process as shown in Algorithm 1. ❶ Quantization Criterion Calculation: We begin by computing the mean squared error (MSE) for a baseline block size of 16 (MSE_{16}), which serves as the quantization error threshold. ❷ High-Error Block Detection: We then identify the blocks whose MSE exceeding MSE_{16} under the current block size. If no such blocks exist or the current block size has reached the minimum allowed value, the quantized tensor AQ is returned. ❸ Iterative Block

Algorithm 1: DBSQ algorithm

input : Original tensor: A; Max, Min block size: $BlkSizeMax$, $BlkSizeMin$
output: Quantized tensor: AQ

```
def DynBlkQuant(A, BlkSizeMax, BlkSizeMin):
    MSE16 = MSEAvg(MsfpQ(A, 16), A)
    def RecurQ(A, BlkSize):
        AQ = MsfpQ(A, BlkSize)
        MSEIdx = MSEBlk(AQ, A) > MSE16
        if MSEIdx == None or BlkSize == BlkSizeMin
         then
            return AQ
        return AQ[~ MSEIdx] + RecurQ
         (A[MSEIdx], BlkSize/2)
    return RecurQ(A, BlkSizeMax)
```

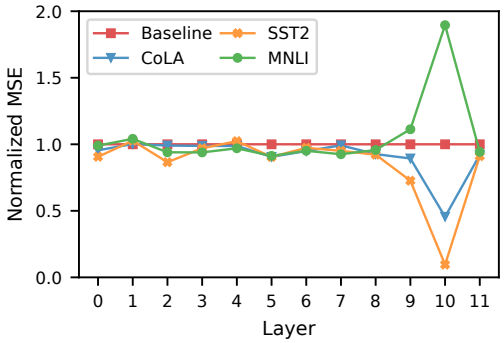


Figure 6: Normalized layer-wise MSE of the BERT-base model on GLUE tasks (CoLA, SST-2, MNLI), with fixed block size of 16 as baseline

Refinement: For blocks with high quantization error, the tensor is partitioned into two subsets: one that meets the error threshold and another that does not. The latter subset is further divided by halving the block size until the MSE criterion is satisfied. The final quantized tensor is obtained by concatenating all refined blocks. Note that the DBSQ algorithm is executed only a few times per tensor, as the distribution of tensor values remains relatively consistent—a property that has also been leveraged by many previous works [12, 19, 24].

To evaluate the effectiveness of our approach, we analyze the layer-wise MSE of BERT-base on the GLUE benchmark, with results normalized against a fixed block size of 16. As shown in Figure 6, the dynamically blocked tensor achieves significantly lower MSE, highlighting the effectiveness of our adaptive blocking strategy.

3.2 Hardware-Friendly Block Size Encoding

Following DBSQ, we obtain three types of data: shared exponents, sign-magnitude mantissas, and block size parameters, as illustrated in Figure 8(a). The block size parameters are additional metadata introduced by DBSQ, in contrast to prior approaches that assume a fixed block size of 16 [4, 39, 40]. To minimize the associated

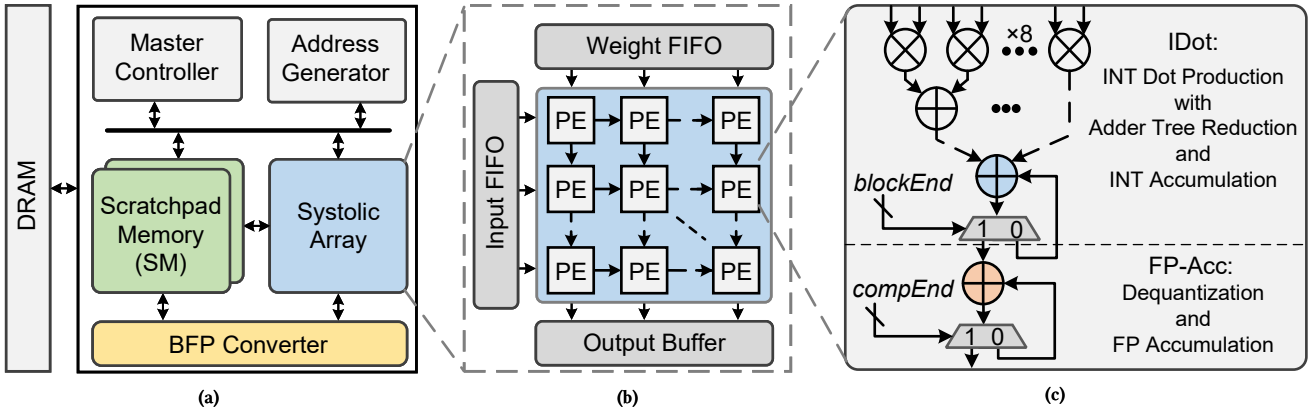


Figure 7: *SmartBlock* architecture. (a) High-level architecture of *SmartBlock*. (b) Systolic array with FIFOs attached for skewing data. (c) The unified PE with processing stage.

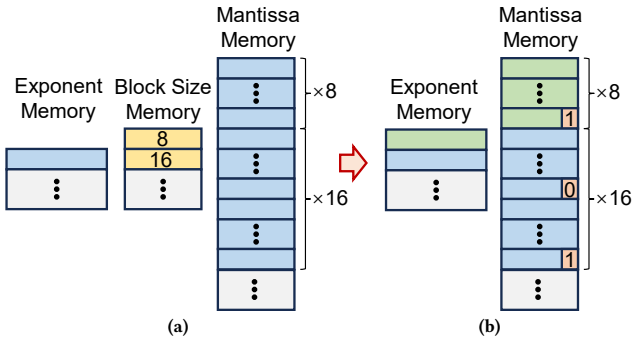


Figure 8: Encode block size parameters into the least significant bit of mantissas.

overhead and simplify decoding, we propose a hardware-friendly encoding strategy that embeds block size information into the least significant bit (LSB) of the mantissa of the last element in the smallest block (e.g., the LSB of the eighth element's mantissa), as shown in Figure 8(b). Specifically, we designate this LSB as a block-end flag: a value of 1 indicates the end of a block, while a value of 0 indicates continuation. This approach eliminates the need to separately decode block size information. Moreover, the encoding is seamlessly integrated into the rounding logic, ensuring that it introduces no additional hardware cost at the encoding stage.

Regarding the impact on accuracy, assuming an equal probability distribution of 0s and 1s in the mantissa's LSB, the probability that a bit needs to be modified is $\frac{1}{8} \times \frac{1}{2} = 6.25\%$. This minimal modification has a negligible effect on numerical accuracy, and we validate this experimentally in Section 5.2.

4 *SmartBlock* Architecture

4.1 Overview

we propose the *SmartBlock* hardware architecture tailed for DBSQ. The major components of SmartBlock are illustrated in Figure 7(a). The scratchpad memory (SM) stores and reuses the input and weight tensors, while the master controller orchestrates data movement between DRAM and SM, and issues addresses to the SM to fetch data into the systolic array for computation. The address generator computes the data addresses for both DRAM and SM. A conventional systolic array design [2, 18] is adopted as the matrix multiplication engine, as shown in Figure 7(b), with a streamlined hardware configuration to enhance efficiency. The systolic array consists of a 2D mesh of PEs (Section 4.2) with attached FIFOs for skewing inputs and weights. It operates in the output-stationary mode [41], where partial sums are accumulated within each PE. The final outputs, stored in FP32 format in the output buffer, are passed to the BFP converter (Section 4.3) for rescaling back to the BFP format.

4.2 Unified Processing Element

To support BFP formats with varying block sizes, we design a unified PE architecture, as shown in Figure 7(c). Each PE comprises two main components: an IDot unit with an adder tree and INT accumulator, and a FP-Acc. The IDot performs parallel BFP mantissa dot production across 8 lanes, reduces the results via an adder tree, and accumulates them when the block size exceeds 8. The FP-Acc then dequantizes the IDot output by combining it with the shared exponent, normalizes the values to FP32, and accumulates the final result in the FP domain.

The signal *blockEnd* acts as a block termination flag, corresponding to the least significant bit of the mantissa of the last element in the smallest block. The *compEnd* signal indicates whether the entire computation is complete. Based on the values of *blockEnd* and *compEnd*, the PE operates in one of three compute modes, as illustrated in Figure 9. After each 8-lane IDot computation, the control logic determines the dataflow path: (a) If $blockEnd = 0$, the IDot result is accumulated in the INT domain. If $blockEnd = 1$, the result is forwarded to the FP-acc. (b) If $compEnd = 0$, FP accumulation continues. (c) If $compEnd = 1$, the final output is produced.

For data with a block size of 8, the PE achieves optimal performance. For larger block sizes, partial sums are efficiently accumulated in the INT domain without incurring significant energy overhead. This architecture strikes a balanced trade-off between computational latency and energy efficiency.

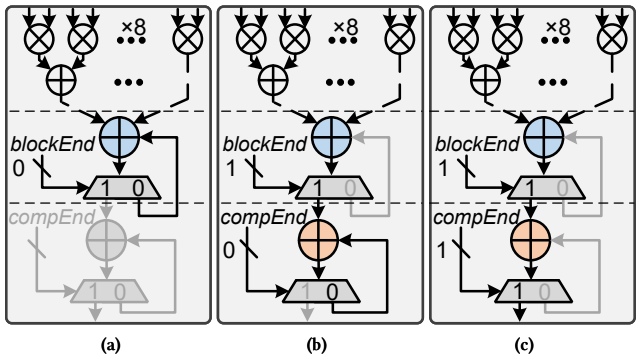


Figure 9: Three PE compute modes. (a) Accumulate mantissa dot product within block in INT domain. (b) Accumulate partial sum across blocks in FP-Acc. (c) Produce the final FP result.

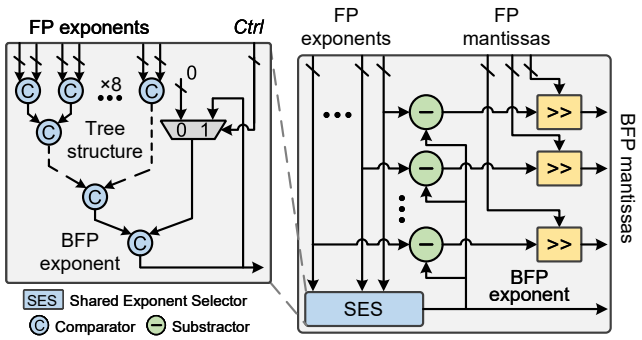


Figure 10: The design of BFP converter.

4.3 BFP Converter

The BFP Converter transforms a block of FP values into BFP format, following the procedure illustrated in Figure 3(a). As shown in Figure 10, the converter comprises two main components: a shared exponent selector (SES) and eight groups of subtractors and right shifters. The SES operates at a minimum granularity of 8 and identifies the maximum exponent within a block of FP inputs. It is implemented as a tree structure with 15 two-input comparators, where each comparator selects the larger of two exponents and forwards it to the next level. To support block sizes larger than 8, the SES integrates an additional two-input comparator and a 2-to-1 multiplexer. When the block size is 8, the control signal *Ctrl* is set to 0, and the SES outputs the maximum exponent among the current 8 inputs as the shared exponent. When the block size exceeds 8, *Ctrl* is set to 1. In this mode, the maximum exponent from each group of 8 inputs is compared recursively with that of the next group until the entire block is processed. The final result is used as the BFP shared exponent. Once the shared exponent is selected, eight subtractors compute the difference between the shared exponent and each value's private exponent. These differences are then used by barrel shifters [35] to right-shift the mantissas of the FP values, completing the conversion to BFP format.

5 Evaluations

We evaluate *SmartBlock* in the aspect of model accuracy, performance, energy consumption, and area overhead in this section.

5.1 Experimental Methodology

Software Implementation. We implement the DBSQ algorithm in PyTorch [33] and evaluate it on two representative groups of models: CNN-based and transformer-based, covering both computer vision and natural language processing tasks, as summarized in Table 1. For CNN-based models, we evaluate VGG-19, ResNet-50, and ResNet-152 on the ImageNet dataset [5], using pre-trained models from the PyTorch Model Zoo to validate accuracy. The network structures are based on those provided in the torchvision library. For transformer-based models, we evaluate BERT-Base and BERT-Large on eight datasets from the GLUE benchmark [45], as well as on the summarization tasks SQuAD1.1 [38] and SQuAD_v2 [37]. We also include ViT (Vision Transformer), a transformer-based model that has achieved excellent performance on vision tasks.

Our DBSQ algorithm, along with other baselines (except OliVe [11]), supports quantization-aware training (QAT) to improve accuracy. For a fair comparison, we use consistent hyperparameters across all methods, including the number of fine-tuning epochs and learning rate. All computations simulate quantization effects using FP32 arithmetic [16]. Trainable quantization parameters for weights and activations are inserted into the computation graph to enable fine-tuning of the quantized models.

Quantization Baselines. We compare the performance of our quantization scheme, DBSQ (with and without block size encoding), against a range of post-training quantization (PTQ) and QAT methods. Specifically, we benchmark against MX [40], ANT [12], and OliVe [11]. MX is a BFP-based format introduced by Microsoft, which uses a fixed block size of 16. ANT proposes an adaptive approach that selects different data types for different tensors based on their dynamic value ranges. OliVe builds upon the data format proposed in ANT and introduces outlier-victim pair encoding, which sacrifices the normal value next to the outlier to preserve the important outlier value.

Hardware Implementation. We implement the *SmartBlock* architecture in Verilog and verify the functionality of each component through RTL simulation. Area and static/dynamic power estimates are obtained by synthesizing the components using a 28 *nm* CMOS process at a clock frequency of 500 MHz. For memory structures, we use CACTI [31] to estimate area, latency, and power consumption. Additionally, to evaluate the end-to-end performance of *SmartBlock* and other baselines, we develop a cycle-accurate simulator based on DNNWeaver [42].

Accelerator Baselines. We compare the performance and energy consumption between *SmartBlock* and SotA BFP-based accelerators:

- FAST [49] implements a systolic array with PEs support dot product computations under multiple BFP precisions (e.g., MX-INT2 and MX-INT4), and applying stochastic rounding to weight gradient updates.

Table 1: Accuracy comparison between different quantization scheme without fine-tuning.

	Model	Dataset	FP32	MX	ANT	OliVe	DBSQ-wo†	DBSQ-w‡
CNN	ResNet50	ImageNet	75.984	65.276	**65.344**	60.618	63.396 (-1.88)	63.470 (-0.07)
	ResNet152		78.252	67.328	64.036	62.572	**67.556** (+0.23)	67.526 (+0.03)
	VGG19		74.172	68.072	62.408	57.494	68.864 (+0.79)	**68.866** (-0.00)
Transformer	ViT	ImageNet	80.970	**79.196**	71.908	78.522	79.180 (-0.02)	79.088 (+0.09)
	BERT-base	CoLA	59.604	56.766	38.481	59.300	**59.616** (+2.85)	58.145 (+1.47)
		SST2	93.349	92.316	89.678	92.430	**92.546** (+0.23)	91.858 (+0.69)
		MNLI	84.941	84.075	69.179	**84.100**	83.668 (-0.41)	83.973 (-0.31)
		SQuAD1.1	80.823	**78.742**	33.680	78.160	78.354 (-0.39)	78.392 (-0.04)
		SQuADv2	73.604	72.652	51.099	72.080	**72.770** (+0.12)	72.585 (+0.18)
	BERT-large	CoLA	63.102	63.819	52.778	63.990	**64.204** (+0.39)	64.204 (+0.00)
		SST2	93.578	93.348	82.454	92.890	**93.807** (+0.46)	93.807 (+0.00)
		MNLI	85.420	**85.492**	48.191	84.890	84.799 (-0.69)	85.094 (-0.29)
		SQuAD1.1	84.465	82.280	53.604	81.002	**82.516** (+0.24)	82.129 (+0.39)
		SQuADv2	76.425	74.101	51.293	73.149	**74.842** (+0.74)	74.632 (+0.21)

† DBSQ without block size parameter encoding. Parentheses indicate the accuracy improvement of DBSQ-w over MX.
‡ DBSQ with block size parameter encoding. Parentheses indicate the accuracy drop of DBSQ-w compared to DBSQ-wo, negative values represent accuracy improvements.

- Bucket [28] propose a bucket-based accumulation unit prior to FP-Acc, which uses multiple small accumulators (i.e., LO-Acc) that are responsible for a small range of exponent values where intermediate results are distributed accordingly.
- WinAcc [19] introduce a LO-Acc with narrow shifters and adders preceding FP-acc, and encode the data into a customized intermediate format, enabling the LC-Acc to offload most of the loads.

For an iso-area comparison, we synthesize the PEs and accumulators of each accelerator and configure the number of PEs accordingly. For consistency in evaluation, we simplify FAST by excluding variable precision and stochastic rounding techniques, and all architectures adopt the MX-INT4 data format. Also, we set the same memory bandwidth and on-chip buffer size for all accelerators, which are large enough to fully utilize the compute core.

5.2 Quantization Accuracy

PTQ Performance We evaluate the proposed DBSQ method both with (DBSQ-w) and without (DBSQ-wo) the encode block size option, against MX, ANT, and OliVe on various CNN-based models on ImageNet dataset, as well as transformer-based models on eight GLUE dataset suit and summarization tasks, for PTQ accuracy. Due to space limitations, we report results for only three representative GLUE tasks (CoLA, SST-2 and MNLI) in Table 1, as other datasets yield similar trends. For the MX method, we adopt MX-INT4 with a fixed block size of 16. Both ANT and OliVe quantize activations and weights to 4 bits. In CNN-based models and ViT, where the dynamic range is relatively narrow, all methods achieve comparable accuracy. However, on BERT models, which contain significant outliers, ANT suffers from substantial accuracy degradation (averaging a 21.3% drop). In contrast, OliVe, which addresses outliers through outlier-victim pair encoding, maintains high accuracy, with an average accuracy drop of less than 1.4%.

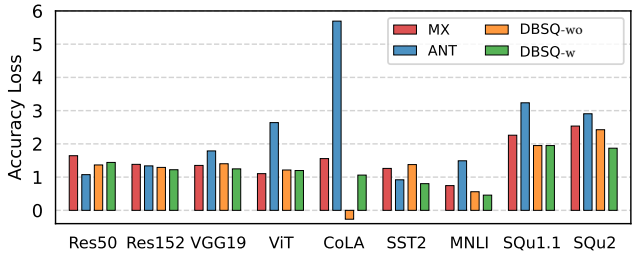


Figure 11: The accuracy loss with fine-tuning on CNNs, ViT and BERT-base.

DBSQ dynamically groups tensors based on data distribution, substantially reducing quantization errors and achieving optimal accuracy across several benchmarks. On BERT, the average accuracy drop for DBSQ-wo is only 0.8%. DBSQ-w, which encodes the grouping information within the mantissa, introduces a slight error but shows no significant difference from DBSQ-wo (as marked ‡ in Table 1), with an average accuracy drop of only 0.17%.

Fine Tuning To further improve accuracy, we evaluate the QAT performance of DBSQ (with and without block size encoding) on ResNet-50, ResNet-152, VGG-19, and ViT using the imageNet dataset, as well as BERT-Base on the GLUE benchmark and SQuAD datasets. We compare the results against prior methods MX and ANT. As shown in Figure 11, both MX and ANT experience more significant accuracy degradation, particularly on CoLA and SQuAD datasets, where the presence of outliers and wide dynamic ranges amplify quantization errors. In contrast, DBSQ consistently achieves superior accuracy retention across all benchmarks.

Overall, the results demonstrate that DBSQ outperforms existing quantization methods across a wide range of models and tasks, maintaining high accuracy with minimal degradation. Its dynamic bloking mechanism effectively accommodates both CNN and transformer architectures, making it a versatile solution for PTQ and

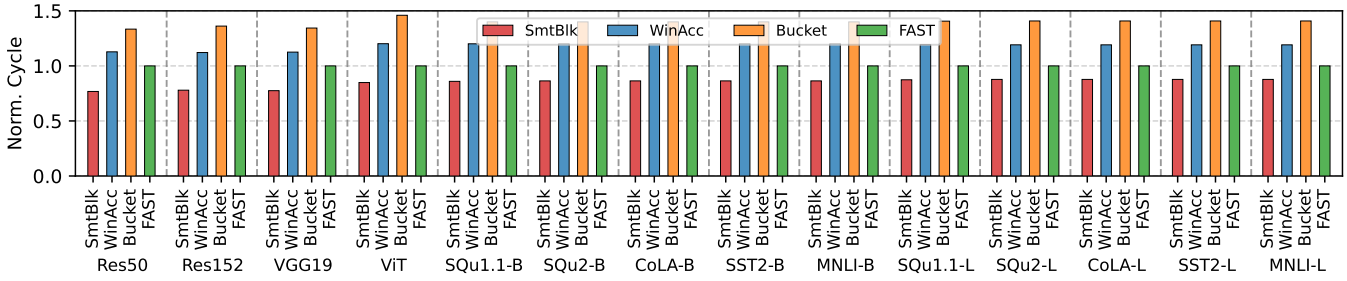


Figure 12: Comparison of the normalized energy in different designs.

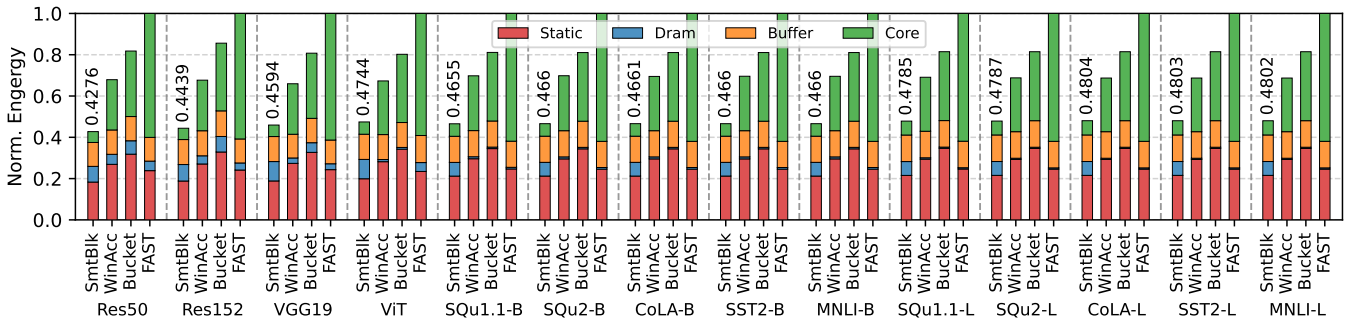


Figure 13: Comparison of the normalized latency in different designs.

QAT scenarios. The encoded variant (DBSQ-w) offers hardware advantages with negligible impact on model performance, further reinforcing DBSQ's practical value.

5.3 Performance, Energy and Area

Performance. Figure 12 shows the normalized total execution cycles for six neural networks across different accelerator designs. All results are normalized to FAST, which serves as the baseline architecture. Both WinAcc and Bucket suffer from performance degradation due to their more complex PE architectures. Specifically, WinAcc incurs a performance drop of 12% to 20% relative to FAST, while Bucket experiences a reduction of 33%to 46%. In contrast, *SmartBlock* delivers significant performance improvements, particularly on CNN-based models. It achieves speedups of up to 1.47×, 1.75×, and 1.30× compared to WinAcc, Bucket, and FAST, respectively. These gains underscore the benefits of leveraging larger block sizes in convolutional networks, where regular computation patterns align well with coarse-grained quantization. On average, *SmartBlock* achieves 1.39×, 1.65×, and 1.18× speedups over WinAcc, Bucket, and FAST, respectively. These improvements are primarily driven by *SmartBlock*'s lightweight PE design and the efficiency of the DBSQ algorithm.

Energy. Figure 13 compares the normalized energy consumption between *SmartBlock* with WinAcc, Bucket, and FAST, considering both static and dynamic energy (including DRAM, on-chip buffer, and core consumption). The complex PE architectures of WinAcc and Bucket result in higher static power consumption but reduce the energy-intensive FP-acc calls, leading to lower core energy consumption. Specifically, core energy consumption is reduced by 57.97% and 46.28% compared to FAST, respectively. *SmartBlock* demonstrates the lowest energy overhead across all evaluated models, highlighting its superior efficiency in diverse network architectures. For CNN-based models such as ResNet-50, ResNet-152, and VGG19, *SmartBlock* achieves over 54% energy reduction compared to FAST. This significant improvement suggests that these models are more amenable to larger quantization blocks, benefiting from the reduced control overhead and simplified hardware operations. In contrast, transformer-based models exhibit slightly lower energy savings. Notably, BERT-Large on the CoLA dataset shows only a 51.96% reduction in energy consumption. This relatively modest gain indicates the presence of local outliers in the data distribution, which demand finer-grained quantization to maintain model accuracy. On average, *SmartBlock* reduces energy consumption by 32.04%, 42.76%, and 53.34% compared to WinAcc, Bucket, and FAST, respectively.

Area. Table 2 compares the PE configuration and area breakdown between *SmartBlock* and three architectures under 28 *nm* process. Despite all designs being constrained to a similar area budget (i.e., $7.681 \sim 7.941 mm^2$), *SmartBlock* achieves the highest PE density, integrating 1024 PEs within $7.693 mm^2$, significantly outperforming WinAcc (380 PEs), Bucket (324 PEs), and FAST (462 PEs). This improvement stems from *SmartBlock*'s lightweight 8-way IDot unit and simplified INT/FP accumulation units, resulting in a compact PE footprint and enabling more parallelism under the same area constraint. In contrast, WinAcc, Bucket and FAST adopt more area-costly 16-way IDot engines, furthermore, WinAcc and Bucket introduce more complexed multi-stage accumulation units, leading to significantly larger PE area.

Table 2: The PE configuration and area breakdown of *SmartBlock* and other works under 28 *nm* process

Architecture	MX-INT4 PE	Number	Area (mm^2)
SmartBlock	8-way IDot (2063.676 μm^2) INT-Acc (683.027 μm^2) FP-Acc (4530.554 μm^2)	1024	7.693
WinAcc	16-way IDot (8662.406 μm^2) LO-Acc (4642.611 μm^2) FP-Acc (6055.056 μm^2)	380	7.941
Bucket	16-way IDot (8723.994 μm^2) LO-Acc (7908.705 μm^2) FP-Acc (5914.502 μm^2)	324	7.825
FAST	16-way IDot (7807.041 μm^2) FP-Acc (8203.630 μm^2)	462	7.681

Block Size Ratio Figure 14 illustrates the distribution of block sizes used by the proposed outlier-aware DBSQ scheme across various DNN models, covering both vision and language tasks. Since larger block sizes are more hardware-efficient—reducing the usage of FP-acc and thereby lowering power consumption—this distribution serves as a direct indicator of the quantization strategy's hardware friendliness. For vision models such as ResNet-50 and ResNet-152, the block size distribution is relatively uniform, with the exception of the largest block size of 512, which appears infrequently. In contrast, BERT-base exhibits a highly consistent and hardware-friendly block size profile across all workloads. Notably, large block sizes such as 512 and 256 comprise up to 33.95% of the quantized data, demonstrating BERT-base's suitability for coarse-grained quantization under our DBSQ scheme. BERT-large, however, presents a more dispersed block size distribution. Less than 20% of its quantized data on the GLUE benchmark utilizes block sizes larger than 64, while a significant portion is assigned to finer-grained blocks (e.g., 16 and 8). This reflects the presence of local outliers and greater internal variability, which necessitate finer quantization granularity to preserve accuracy. Despite the differences in block size usage among these models, *SmartBlock* consistently delivers substantial energy savings, as shown in earlier evaluations. These results confirm the adaptability of our DBSQ approach: it effectively balances quantization granularity based on the presence of outliers, applying small blocks only when needed to protect accuracy, and favoring large blocks elsewhere to maximize hardware efficiency.

6 Related Works

DNN Acceleration. To efficiently accelerate DNNs, a broad range of domain-specific accelerators have been proposed to exploit the computation and memory access characteristics of DNN workloads [9, 9, 10, 14, 34, 50, 52]. These accelerators typically adopt specialized processing elements and optimized dataflows to improve parallelism, increase data reuse, and minimize control overhead. For instance, systolic arrays [2, 18] and other spatial architectures [6, 17] enable local data reuse across processing elements, significantly reducing data movement and energy consumption. In addition, modern general-purpose GPUs integrate dedicated matrix multiplication engines like Tensor Cores [3], which are optimized for

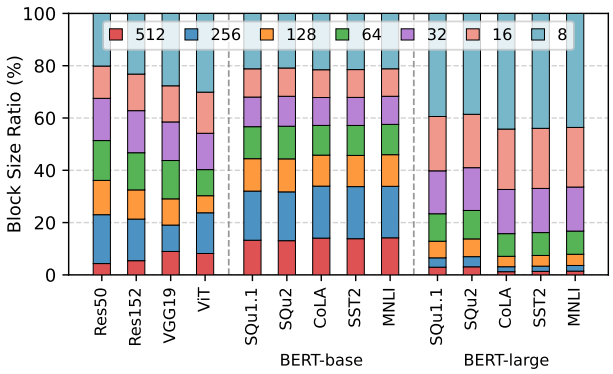


Figure 14: Block size ratio across various DNN models under DBSQ.

SIMD operations and widely used for accelerating DNN workloads. Some works further push the frontier by introducing near-memory computing [15, 29] to mitigate the memory bottleneck in DNN inference and training.

DNN Quantization. Quantization is an efficient technique for reducing the size and computational overhead of DNN models by using lower bit-width representations, thereby decreasing memory and compute requirements [7, 24, 47, 51]. To maintain model accuracy, some approaches employ mixed-precision quantization strategies. BitFusion [43], for example, combines low-bit PEs to support various bit-widths. OLAccel [32] selects different bit-width MACs (multiply accumulators) for different layers, while DRQ [44] adjusts precision based on data sensitivity. GOBO [48] applies higher precision to quantize outlier weights. SPARK [25] introduces a variable-length encoding scheme that exploits the sparsity of high-significance bits based on the distribution range of parameters. These approaches typically require multi-precision MACs and often result in misaligned memory accesses.

Other works fix the bit-width but dynamically adapt the data type. ANT [12] proposes an adaptive numeric format that considers tensor value distributions. OliVe [11] builds on ANT's numeric type and introduces special encoding for outliers. However, these methods rely on FP scale factors and involve complex encoder-decoder designs.

In summary, existing methods may not fully integrate data representation, compression, and hardware efficiency in a cohesive manner. To bridge this gap, we propose *SmartBlock*, a solution that leverages the sparse distribution of outliers. Through efficient quantization and encoding strategies, *SmartBlock* achieves an excellent balance between model accuracy and hardware cost.

7 Conclusion

In this work, we present *SmartBlock*, an algorithm–hardware co-designed solution for efficient DNN acceleration, featuring an outlier-aware DBSQ scheme based on BFP. The key insight is that outliers are both rare and sparsely distributed. By leveraging this property, smaller blocks are applied in regions containing outliers to preserve accuracy, while larger blocks are used in outlier-free

regions to reduce hardware overhead. To the best of our knowledge, *SmartBlock* is the first work to leverage variable block sizes in BFP-based number formats, achieving an exceptional balance between accuracy and hardware overhead. Additionally, block size information is seamlessly encoded into the mantissa during the quantization rounding process, eliminating the need for extra storage or decoding overhead. On the hardware side, *SmartBlock* introduces a unified PE architecture that efficiently supports BFP data with variable block sizes. As a result, *SmartBlock* outperforms SotA BFP-based accelerators, WinAcc and Bucket, achieving up to 1.39× and 1.65× speedup, and 32.04% and 42.76% energy reduction, respectively, with a superior model accuracy.

Acknowledgments

This work was supported by Natural Science Foundation of Hunan Province (No. 2024JJ6470, 2023JJ40679).

References

[1] 2008. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008* (2008), 1–70.

[2] Yu-Hsin Chen, Tien-Ju Yang, Joel S. Emer, et al. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE J. Emerg. Sel. Topics Circuits Syst.* 9, 2, 292–308. doi:10.1109/JETCAS.2019.2910232

[3] Jack Choquette, Wishwesh Gandhi, Olivier Giroux, et al. 2021. NVIDIA A100 Tensor Core GPU: Performance and Innovation. *IEEE Micro* 41, 2 (2021), 29–35.

[4] Steve Dai, Rangharajan Venkatesan, Mark Ren, et al. 2021. VS-Quant: Per-vector Scaled Quantization for Accurate Low-Precision Neural Network Inference. In *Proceedings of the Fourth Conference on Machine Learning and Systems, MLSys*. mlsys.org.

[5] Jia Deng, Wei Dong, Richard Socher, et al. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 248–255.

[6] Jinyi Deng, Xinru Tang, Jiahao Zhang, et al. 2023. Towards Efficient Control Flow Handling in Spatial Architecture via Architecting the Control Flow Plane. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*. ACM, 1395–1408.

[7] Tim Dettmers, Mike Lewis, Younes Belkada, et al. 2022. GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.).

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, et al. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186.

[9] Cheng-Yan Du, Chieh-Fu Tsai, Wen-Ching Chen, et al. 2023. A 28nm 11.2TOPS/W Hardware-Utilization-Aware Neural-Network Accelerator with Dynamic Dataflow. In *IEEE International Solid- State Circuits Conference, ISSCC*. IEEE, 332–333.

[10] Dionysios Filippas, Christodoulos Peltekis, Giorgos Dimitrakopoulos, et al. 2023. Reduced-Precision Floating-Point Arithmetic in Systolic Arrays with Skewed Pipelines. In *5th IEEE International Conference on Artificial Intelligence Circuits and Systems, AICAS*. IEEE, 1–5.

[11] Cong Guo, Jiaming Tang, Weiming Hu, et al. 2023. OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA*, Yan Solihin and Mark A. Heinrich (Eds.). ACM, 3:1–3:15.

[12] Cong Guo, Chen Zhang, Jingwen Leng, et al. 2022. ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network Quantization. In *55th IEEE/ACM International Symposium on Microarchitecture, MICRO*. IEEE, 1414–1433.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE Computer Society, 770–778.

[14] Kartik Hegde, Jiyong Yu, Rohit Agrawal, et al. 2018. UCNN: Exploiting Computational Reuse in Deep Neural Networks via Weight Repetition. In *45th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA*, Murali Annavaram, Timothy Mark Pinkston, and Babak Falsafi (Eds.). IEEE Computer Society, 674–687.

[15] Pouya Houshmand, Giuseppe Maria Sarda, Vikram Jain, et al. 2023. DIANA: An End-to-End Hybrid DIgital and ANAlog Neural Network SoC for the Edge. *IEEE J. Solid State Circuits* 58, 1 (2023), 203–215.

[16] Benoit Jacob, Skirmantas Kligys, Bo Chen, et al. 2018. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. Computer Vision Foundation / IEEE Computer Society, 2704–2713.

[17] Norman P. Jouppi, George Kurian, Sheng Li, et al. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA*, Yan Solihin and Mark A. Heinrich (Eds.). ACM, 82:1–82:14.

[18] Norman P. Jouppi, Cliff Young, Nishant Patil, et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*. ACM, 1–12.

[19] Xin Ju, Jun He, Mei Wen, et al. 2025. WinAcc: Window-based Acceleration of Neural Networks Using Block Floating Point. In *Design, Automation & Test in Europe Conference, DATE*. IEEE, 1–7.

[20] Dhiraj D. Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, et al. 2019. A Study of BFLOAT16 for Deep Learning Training. *CoRR* abs/1905.12322 (2019).

[21] Foroozan Karimzadeh, Mohsen Imani, Bahar Asgari, et al. 2023. Memory-Based Computing for Energy-Efficient AI: Grand Challenges. In *2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration (VLSI-SoC)*. 1–8.

[22] Paresh Kharya. 2020. TensorFloat-32 in the A100 GPU Accelerates AI Training, HPC up to 20x. *NVIDIA Corporation, Tech. Rep* (2020).

[23] Sabuj Laskar, Pranati Majhi, Sungkeun Kim, et al. 2024. Enhancing Collective Communication in MCM Accelerators for Deep Learning Training. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 1–16.

[24] Ji Lin, Jiaming Tang, Haotian Tang, et al. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys*. mlsys.org.

[25] Fangxin Liu, Ning Yang, Haomin Li, et al. 2024. SPARK: Scalable and Precision-Aware Acceleration of Neural Networks via Efficient Encoding. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA*. IEEE, 1029–1042.

[26] Lian Liu, Zhaohui Xu, Yintao He, et al. 2024. Drift: Leveraging Distribution-based Dynamic Precision Quantization for Efficient Deep Neural Network Acceleration. In *Proceedings of the 61st ACM/IEEE Design Automation Conference, DAC*. ACM, 140:1–140:6.

[27] Yun-Chen Lo, Tse-Kuang Lee, and Ren-Shuo Liu. 2023. Block and Subword-Scaling Floating-Point (BSFP) : An Efficient Non-Uniform Quantization For Low Precision Inference. In *The Eleventh International Conference on Learning Representations, ICLR*. OpenReview.net.

[28] Yun-Chen Lo and Ren-Shuo Liu. 2023. Bucket Getter: A Bucket-based Processing Engine for Low-bit Block Floating Point (BFP) DNNs. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*. ACM, 1002–1015.

[29] Yun-Chen Lo and Ren-Shuo Liu. 2023. Morphable CIM: Improving Operation Intensity and Depthwise Capability for SRAM-CIM Architecture. In *60th ACM/IEEE Design Automation Conference, DAC*. IEEE, 1–6.

[30] Sparsh Mittal. 2020. A survey on modeling and improving reliability of DNN algorithms and accelerators. *Journal of Systems Architecture* 104 (2020), 101689.

[31] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. 2009. CACTI 6.0: A Tool to Model Large Caches.

[32] Eunhyeok Park, Dongyoung Kim, and Sungjoo Yoo. 2018. Energy-Efficient Neural Network Accelerator Based on Outlier-Aware Low-Precision Computation. In *45th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA*, Murali Annavaram, Timothy Mark Pinkston, and Babak Falsafi (Eds.). IEEE Computer Society, 688–698.

[33] Adam Paszke, Sam Gross, Francisco Massa, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 8024–8035.

[34] Christodoulos Peltekis, Dionysios Filippas, Giorgos Dimitrakopoulos, et al. 2023. ArrayFlex: A Systolic Array Architecture with Configurable Transparent Pipelining. In *Design, Automation & Test in Europe Conference & Exhibition, DATE*. IEEE, 1–6.

[35] Matthew R. Pillmeier, Michael J. Schulte, and Eugene George Walters III. 2002. Design alternatives for barrel shifters. In *Advanced Signal Processing Algorithms, Architectures, and Implementations XII*, Franklin T. Luk (Ed.), Vol. 4791. International Society for Optics and Photonics, SPIE, 436 – 447.

[36] Raghu Prabhakar, Ram Sivaramakrishnan, Darshan Gandhi, et al. 2024. SambaNova SN40L: Scaling the AI Memory Wall with Dataflow and Composition of

Experts. In *57th IEEE/ACM International Symposium on Microarchitecture, MICRO*. IEEE, 1353–1366.

[37] Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know What You Don't Know: Unanswerable Questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, 784–789.

[38] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, et al. 2016. SQuAD: 100, 000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP*, Jian Su, Xavier Carreras, and Kevin Duh (Eds.). The Association for Computational Linguistics, 2383–2392.

[39] Bita Darvish Rouhani, Daniel Lo, Ritchie Zhao, et al. 2020. Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

[40] Bita Darvish Rouhani, Ritchie Zhao, Venmugil Elango, et al. 2023. With Shared Microexponents, A Little Shifting Goes a Long Way. In *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA*. ACM, 83:1–83:13.

[41] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, et al. 2020. A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim. In *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS*. IEEE, 58–68.

[42] Hardik Sharma, Jongse Park, Divya Mahajan, et al. 2016. From high-level deep neural models to FPGAs. In *49th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*. IEEE Computer Society, 17:1–17:12.

[43] Hardik Sharma, Jongse Park, Naveen Suda, et al. 2018. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. In *45th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA*, Murali Annavaram, Timothy Mark Pinkston, and Babak Falsafi (Eds.). IEEE Computer Society, 764–775.

[44] Zhuoran Song, Bangqi Fu, Feiyang Wu, et al. 2020. DRQ: Dynamic Region-based Quantization for Deep Neural Network Acceleration. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 1010–1021.

[45] Alex Wang, Amanpreet Singh, Julian Michael, et al. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP*, Tal Linzen, Grzegorz Chrupala, and Afra Alishahi (Eds.). Association for Computational Linguistics, 353–355.

[46] James Hardy Wilkinson. 1959. Rounding errors in algebraic processes. In *Information Processing, Proceedings of the 1st International Conference on Information Processing, UNESCO*. UNESCO (Paris), 44–53.

[47] Guangxuan Xiao, Ji Lin, Mickaël Seznec, et al. 2023. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In *International Conference on Machine Learning, ICML (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 38087–38099.

[48] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, et al. 2020. GOBO: Quantizing Attention-Based NLP Models for Low Latency and Energy Efficient Inference. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*. IEEE, 811–824.

[49] Qian Zhang, Sai, McDanel, et al. 2022. FAST: DNN Training Under Variable Precision Block Floating Point with Stochastic Rounding. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 846–860.

[50] Sai Qian Zhang, Thierry Tambe, Nestor Cuevas, et al. 2024. CAMEL: Co-Designing AI Models and eDRAMs for Efficient On-Device Learning. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA*. IEEE, 861–875.

[51] Yilong Zhao, Chien-Yu Lin, Kan Zhu, et al. 2024. Atom: Low-Bit Quantization for Efficient and Accurate LLM Serving. In *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys*, Phillip B. Gibbons, Gennady Pekhimenko, and Christopher De Sa (Eds.). mlsys.org.

[52] Zeyu Zhu, Fanrong Li, Gang Li, et al. 2024. MEGA: A Memory-Efficient GNN Accelerator Exploiting Degree-Aware Mixed-Precision Quantization. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA*. IEEE, 124–138.