MERSIT: A Hardware-Efficient 8-bit Data Format with Enhanced Post-Training Quantization DNN Accuracy

Nguyen-Dong Ho, Gyujun Jeong, Cheol-Min Kang, Seungkyu Choi and Ik-Joon Chang
Kyung Hee University

ABSTRACT

Post-training quantization (PTQ) models utilizing conventional 8-bit Integer or floating-point formats still exhibit significant accuracy drops in modern deep neural networks (DNNs), rendering them unreliable. This paper presents MERSIT, a novel 8-bit PTQ data format designed for various DNNs. While leveraging the dynamic configuration of exponent and fraction bits derived from Posit data format, MERSIT demonstrates enhanced hardware efficiency through the proposed merged decoding scheme. Our evaluation indicates that MERSIT yields more reliable 8-bit PTQ models, exhibiting superior accuracy across various DNNs compared to conventional floating-point formats. Furthermore, the proposed processing unit saves 26.6% in area and 22.2% in power consumption compared to the Posit-based unit, while maintaining comparable efficiency to the floating-point-based unit.

1 INTRODUCTION

Quantization has effectively addressed the issue of excessive data movement between computational units and primary memory in processing deep neural networks (DNNs). However, this approach can adversely affect DNN accuracy due to precision reduction. To counter this, two primary categories of techniques have been developed: post-training quantization (PTQ) and quantization-aware training (QAT) [4]. QAT necessitates intensive training processes using massive datasets, often referred to as fine-tuning, while PTQ runs the calibration of pre-trained models using only a fraction of the dataset. Clearly, QAT allows for more aggressive quantization, providing superior accuracy compared to its PTQ counterpart. However, the practical deployment of QAT is considerably constrained by the resource-intensive demands of fine-tuning.

Numerous studies [10, 14] have demonstrated that for many DNN models initially trained in full precision (FP32), PTQ can effectively convert weights and activations to 8-bit integers (INT8s) with minimal accuracy loss. However, modern DNN models, including compact vision models [12] and large language models (LLMs) [3], often suffer significant accuracy degradation when quantized to INT8 through PTQ [13], mainly due to the limited dynamic range. To mitigate this, several methodologies [10, 14] have been developed, focusing on calibrating quantized weights and activations to match DNN parameter distributions. However, these methods are not universally applicable and are often model-specific. Moreover, the extensive calibration process required can reduce the overall utility and efficiency of PTQ models, posing a challenge in scenarios requiring rapid deployment or low-overhead optimization.

These challenges have increased interest in 8-bit floating-point (FP8) format for its wider dynamic range. A recent industry report [13], evaluating the performance of FP8, presented extensive research on this matter. The findings from this study indicate that a significant improvement in the accuracy of DNNs is obtained when employing FP8 PTQ models, as compared to INT8 PTQ ones. However, this progress is tempered by a noticeable drop in accuracy for some DNNs. The study also noted that the optimal number of exponent bits for PTQ accuracy in FP8 varies by DNN model. Consequently, industrial accelerators often support multiple FP8 configurations [1], each with varying exponent bits, at the cost of hardware efficiency.

In contrast to traditional data formats, Posit, introduced in 2017 [6], incorporates regime bits, enabling a significantly broader dynamic range. This format dynamically adjusts the precision of its fraction according to the range of the data it represents. Certain studies have shown that the inherent characteristics of Posit enable it to maintain commendable accuracy levels across diverse DNN architectures when applied within PTQ frameworks [2, 8]. However, there is a critical issue arises when adopting this data format, as it demands a high-cost computing unit due to its intricate decoding logic. For example, our research has found that an 8-bit Posit multiplier incurs substantial penalties, with an 80% increase in area and a 46% rise in power consumption compared to its FP8 equivalent. This overhead imposes practical limitations on the widespread deployment of Posit.

Motivated by these observations, we present a novel 8-bit data format, named MERSIT. This format retains key features of Posit, such as the use of regime information and a dynamic fraction arrangement. However, it diverges by expressing regime information through groups with multiple bits, effectively minimizing the decoding overhead. Our MERSIT model demonstrates the capability to achieve high accuracies across diverse DNNs using a singular configuration, similar to Posit. Concurrently, it ensures significantly better hardware efficiency than Posit.

We further summarize the contribution of this work as follows.

- We propose the 8-bit MERSIT, characterized by its adjustable number of the merge level of exponent bits. Specifically, merge levels of two and three (*i.e.* (8,2), (8,3)) are examined and compared with various FP8 and Posit configurations.
- The comparative 8-bit PTQ analysis shows that MERSIT attains accuracies that are on par with those of 8-bit Posit, significantly exceeding PTQ accuracies of INT8 and FP8 over most inspected DNNs.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06…$15.00
https://doi.org/10.1145/3649329.3655907

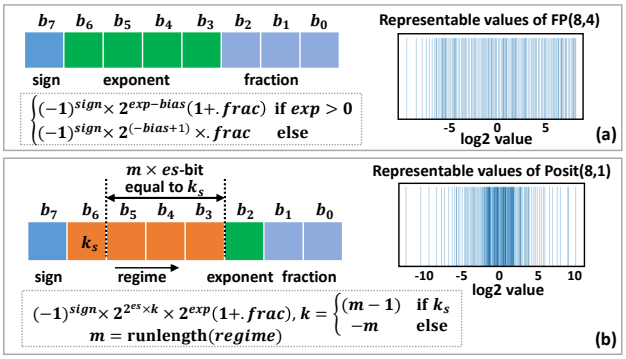


Figure 1: The detailed expression of (a) FP8 and (b) Posit8

- We compare the hardware costs of multiply-and-accumulate (MAC) units specifically designed for FP8, 8-bit Posit, our MERSIT. It demonstrates that our MERSIT leads to 26.6% area reduction and 22.2% power saving compared to 8-bit Posit, while maintaining comparable efficiency to FP8.

2 BACKGROUND

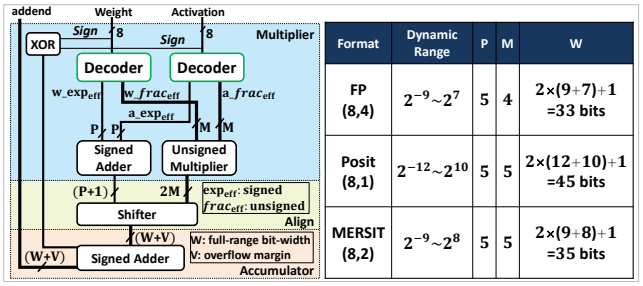
2.1 Existing 8-bit Data Formats

This section provides a concise overview of sub-8-bit data formats previously established, including FP8 and Posit [6].

Floating-Point (FP): Fig. 1a illustrates the structure of the FP8 data format, comprising sign, exponent, and fraction components. In contrast to full- and half-precision formats, known as FP32 and FP16, respectively, FP8 does not conform to an IEEE standard and is therefore not rigidly defined. This flexibility allows users to adjust the number of bits in the exponent. In this paper, the FP8 configuration characterized by E exponent bits is designated as FP(8,E). FP8 offers a wider exponent range using subnormal representation when the exponent equals zero.

Posit: Fig. 1b illustrates an 8-bit Posit example with a single exponent bit, herein referred to as Posit(8,1), where the first and second numbers express the number of data bits and exponent bits. In this Posit format, the two most significant bits — b_7 and b_6 bits as shown in Fig. 1b — encode the signs of the data and the regime, respectively. The data sign bit operates identically to that in floating-point data formats, with '0' denoting a positive value and '1' indicating a negative. Whereas, the regime sign bit, denoted as 'k' in Fig. 1b, is determined to be non-negative if the b_6 bit is '1', and negative otherwise.

Apart from these two bits, the remainder of the sequence is allocated for the regime, exponent, and fraction. Notably, in Posit formats, the regime, exponent, and fraction bits are not statically positioned. The regime is established by locating the first '1' (for a non-negative regime) or '0' (for a negative regime). The bits immediately following the initial '1' or '0' define the exponent and the fraction. The data format prescribes the size of exponent bits (referred to as es in Fig. 1b); for example, Posit(8,1) is an 8-bit Posit configuration to include a single exponent bit, where es = 1, positioned right after the leading bit that signifies the regime. Then, the remaining other bits are assigned to the fraction. Distinct from floating-point formats, the fraction length in Posit varies according to the regime value. As the regime's absolute value increases, the allocation for fraction bits correspondingly decreases.



Format	Dynamic Range	P	M	W
FP (8,4)	2^{-9}~2^{7}	5	4	2×(9+7)+1 =33 bits
Posit (8,1)	2^{-12}~2^{10}	5	5	2×(12+10)+1 =45 bits
MERSIT (8,2)	2^{-9}~2^{8}	5	5	2×(9+8)+1 =35 bits

Figure 2: Multiply-and-accumulate (MAC) architecture

Other Formats: [11] proposes 'AdaptivFloat', which is a simplified floating-point format without subnormal range and utilizes a variable exponent bias to function as scaling factors for specific groups of data. [15] presents an 8-bit block floating-point (BFP)-based adaptive data format that utilizes a shared exponent value, which also functions as a scaling parameter for grouped data. Meanwhile, our experiments to validate accuracy also employ the channel- and layer-level scaling parameters for all the data formats. Therefore, we can presume that these data formats align with FP8, eliminating the need for a separate comparison.

Besides the introduced data formats, various studies have been working on achieving effective precision lower than 8 bits [5, 9]. However, these works typically require high-cost hardware resources due to mixed-precision or complex data reformation. Moreover, they still yield unreliable results in certain models or incorporate a fine-tuning stage to mitigate accuracy degradation aligning with the principles of QAT. It is important to note that these studies fall outside the scope of our comparative review, as our exclusive focus is on PTQ settings with unified precision.

2.2 MAC Unit Design with Kulisch Accumulator

It is a well-established fact that the majority of computations in DNNs consist of MAC operations. The MAC units for exponent-encoded data formats such as floating-point are known to be both area- and power-expensive [13]. Yet, the complexity can be notably reduced when handling 8-bit encoded formats, given their limited dynamic ranges in comparison to higher precision ones. For example, the complexity of the multiplier can be prominently reduced due to the shortened width of the represented fraction bits. Furthermore, owing to the moderate dynamic range they offer, the accumulator can be substituted with a fixed-point adder covering the entire possible range of the multiplied results. This type of accumulator is referred to as the Kulisch accumulator [7]. It enhances the hardware efficiency by simplifying the adder logic and eliminating the reconstructing effort between multiplication and accumulation. We also leverage this existing MAC design to provide an optimized implementation for low-bit encoded data that covers the dynamic range. The schematic of the design is shown in Fig. 2, which is implemented with a multiplier, an alignment logic, and a fixed-point adder. Such a MAC scheme is compatible with all the data formats under consideration in this paper: FP8, Posit8, and MERSIT8.

The multiplier part is broken down into three integral parts: a decoder, a signed adder, and an unsigned multiplier. The decoder extracts the effective exponent and fraction values from the input data. These values are represented as exp_{eff} and $frac_{eff}$ respectively,

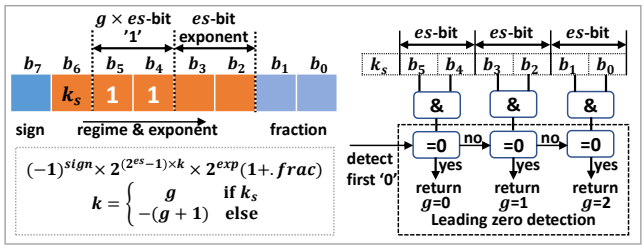


Figure 3: Our proposed MERSIT format

as illustrated in Fig. 2. Following the decoding process, the signed adder sums the exp_{eff} values of the inputs while the unsigned multiplier multiplies the $frac_{eff}$ values of the inputs. Utilizing the summed exponent values from the multiplier output, the alignment logic determines the correct bit position for addition. Then, the multiplied result is fed into the fixed-point adder for accumulation. Referring to [13], the bit-width of the fixed-point adder is given by 'W+V', where 'W' denotes the bit-width of the full range of the product, and 'V' denotes additional bits to prevent overflow.

It should be noted that the value of 'W' varies based on the dynamic range of each data format. For example, in the case of FP(8,4), Posit(8,1), and our MERSIT(8,2), the 'W' values are 33, 45, and 35, respectively. These values correspond to twice the dynamic range of their respective data formats. This relationship highlights a trade-off: data formats with wider dynamic ranges necessitate MAC units with larger 'W' values, potentially impacting hardware efficiency. Therefore, when selecting a data format for DNNs, it is important to balance the need for a wider dynamic range with the implications it has on hardware efficiency.

3 THE PROPOSED DATA FORMAT

3.1 MERSIT Data Format

Figure 3 describes the MERSIT data format proposed in our study. A MERSIT word is structured with a sign bit, a regime sign indicator (k_S in Fig. 3), and multiple exponent candidates (ECs). In the MERSIT, the EC that incorporates a leading zero is designated as the exponent. The location of this exponent within the word determines the regime value. Let us express that $sign$: sign value, es:exponent size, k: regime, exp: exponent value, and $frac$: fraction value. Then, the representative value of a MERSIT word can be expressed as follows.

$$(-1)^{sign} \times 2^{(2^{es}-1)\times k} \times 2^{exp} \times (1 + .frac). \quad (1)$$

The regime information, k, is obtained by

$$k = \begin{cases} g & \text{if } k_s \\ -(g+1) & \text{else.} \end{cases} \quad (2)$$

The MERSIT data format also allows for numerous configurations depending on es. Let us express a certain MERSIT configuration as MERSIT(N,E), where N represents the total number of bits in a MERSIT word and E specifies es. This work is focused on 8-bit representations for PTQ, and therefore, we exclusively examine instances where N equals 8.

We show the decoding principles of MERSIT(8,2) on the right side of Fig. 3. In MERSIT formats, the two highest-order bits (b_7 and b_6 in Fig. 3) respectively express a sign bit and a regime sign indicator, analogous to the Posit format. The regime sign follows the same representation as in Posit: '0' signifies a negative k, while '1' indicates a non-negative k. The value of k is determined by the position of the exponent, the EC to contain a leading zero as mentioned above. To detect the position of the exponent, all bits within each EC are concurrently AND-gated. Then, we find the EC to produce the first zero among the AND-gating outputs. The ECs following the exponent contain fraction information. Similar to Posit, the quantity of fraction bits is contingent upon the value of the regime. However, the MERSIT format differs from Posit in several key aspects. Firstly, MERSIT utilizes ECs consisting of multiple bits for conveying regime information, which results in more efficient decoding hardware compared to Posit. Secondly, in our MERSIT, exponent and regime bits are not separated, so-called merged exponent and regime. This unique design fundamentally alters the representative value equation of a MERSIT word, distinguishing it from that of Posit. Given these differences, we have named our proposed data format MERSIT, an acronym that signifies a "Posit-like data format with merged exponent and regime".

Table 1: The 8-bit MERSIT(8,2) Representation

$b_6b_5b_4b_3b_2b_1b_0$	k	exp	$(2^{es}-1)\times k + exp$	$\#of Fraction Bits$
$\mathbf{0111111_2}$		zero		
0111100_2	-3	$00_2 = 0$	-9	0
0111101_2		$01_2 = 1$	-8	
0111110_2		$10_2 = 2$	-7	
$01100xx_2$	-2	$00_2 = 0$	-6	2
$01101xx_2$		$01_2 = 1$	-5	
$01110xx_2$		$10_2 = 2$	-4	
$000xxxx_2$	-1	$00_2 = 0$	-3	4
$001xxxx_2$		$01_2 = 1$	-2	
$010xxxx_2$		$10_2 = 2$	-1	
$100xxxx_2$	0	$00_2 = 0$	0	4
$101xxxx_2$		$01_2 = 1$	1	
$110xxxx_2$		$10_2 = 2$	2	
$11100xx_2$	1	$00_2 = 0$	3	2
$11101xx_2$		$01_2 = 1$	4	
$11110xx_2$		$10_2 = 2$	5	
1111100_2	2	$00_2 = 0$	6	0
1111101_2		$01_2 = 1$	7	
1111110_2		$10_2 = 2$	8	
$\mathbf{1111111_2}$		$\pm\infty$		

(x: *fraction bits*, es = 2)

Table 1 displays the decoding outcomes for MERSIT(8,2). In this context, the term "$(2^{es}-1)\times k + exp$" can be interpreted as the effective exponent value in a MERSIT word. For the MERSIT(8,2) format, this value ranges between −9 and 8. In our MERSIT format, the number of fraction bits is determined by the value of k. As the absolute value of k decreases, the number of fraction bits increases, reaching a maximum of 4 in MERSIT(8,2). Conversely, when k equals −3 or 2, MERSIT(8,2) does not allocate any fraction bits.

3.2 Dynamic Range and Precision Comparison

In this section, we graphically represent the dynamic ranges and fraction precisions of three data formats: FP8, Posit8, and our MERSIT8. For each of these formats, we analyze various configurations to illustrate their capabilities. These configurations include FP8 in forms FP(8,2), FP(8,3), FP(8,4), FP(8,5); Posit in Posit(8,0), Posit(8,1), Posit(8,2); and MERSIT in MERSIT(8,2) and MERSIT(8,3). It is important to note that while the fraction precision in FP8 configurations is static within their standard representations, the effective precision varies in their subnormal representations. This variation and its

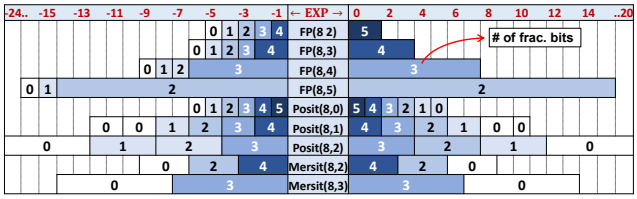


Figure 4: Range and precision of various 8-bit data formats

implications are detailed in Fig. 4, where we provide a comparative visual analysis of these formats under different configurations.

Previous research has indicated that in PTQ, configurations such as FP(8,3), FP(8,4), and Posit(8,1) generally yield high accuracies across various DNN models [11]. When comparing our MERSIT(8,2) to these configurations, our MERSIT(8,2) offers a dynamic range that is broader than those of FP(8,3) and FP(8,4) but narrower than that of Posit(8,1). Notably, the maximum precision in FP(8,4) is limited to 3 bits, while Posit(8,1) can provide up to a 4-bit precision and has the most extensive dynamic range among these four configurations. Despite MERSIT(8,2) having a smaller overall dynamic range compared to Posit(8,1), it is important to highlight that the range within which MERSIT(8,2) can maintain a 4-bit precision is broader than that of Posit(8,1). This distinction underlines the unique balance of precision and dynamic range that MERSIT(8,2) brings to the table.

3.3 Our MERSIT Decoding Scheme

In the MAC scheme of Fig. 2, all the components except the decoder are implemented as widely used circuits, such as adder, multiplier, and shifter. Although their sizes vary according to the used data format, their implementations adhere to well-established design principles. In contrast, the decoder design fully relies on the characteristics of the data format, not sufficiently formulated.

In this work, we propose a decoding scheme for our MERSIT format. Inherent to the concept of the merged regime and exponent, MERSIT(8,E) allows decoding the data in groups of 'E' bits for extracting the effective exponent and fraction values. Fig. 5a illustrates the design of the decoder for the MERSIT(8,2) configuration as a specific example. In contrast to Posit, which requires 1-bit resolution decoding, the bits in MERSIT are grouped into twos to serve as inputs for a dynamic shifter and a Leading-Zero-Detection (LZD) unit. The output of these units is then utilized to derive the effective values. In the overall design process, the MERSIT decoder centers around two particularly challenging components: the 3-bit LZD unit and the computing unit for the operation "$k \times (2^{es} - 1)$". The 3-bit LZD unit identifies the position of the first zero bit, which plays a significant role in determining the value of k. The "$k \times (2^{es} - 1)$" computing unit is responsible for calculating the effective exponent value, a critical step in the decoding process. We implement these units using a minimal number of logical gates, shown in Fig. 5b. As a result, with the logic optimization of the grouped decoding scheme, the proposed decoder can achieve notable savings in both area and power consumption compared to the Posit decoder.

Our research findings indicate that the area and power consumption of the decoder are significant components of the overall MAC unit, whose detail is discussed in Section 4.3. This observation holds true for MAC units designed for both FP8 and Posit data formats. A key advantage of our MERSIT format is its ability to facilitate a more efficient implementation of the decoder, which is translated into enhanced area and power efficiency of the MAC unit.

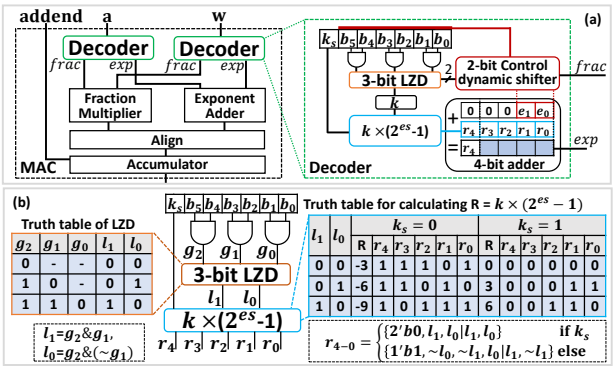


Figure 5: The proposed MERSIT(8,2) decoding scheme

4 EVALUATION

4.1 Experimental Setup

PTQ Accuracy Comparison: We conduct a comprehensive evaluation of PTQ model accuracies using a variety of data formats and configurations. These include FP8 formats of (8,3), (8,4) and (8,5)), Posit formats of (8,0), (8,1), (8,2) and (8,3), and our MERSIT formats of (8,2) and (8,3). Utilizing the PyTorch framework, we source pre-trained models from "Model Zoo" and "hugging face transformer" for our experiments. We focus on two major tasks: Image classification on ImageNet and GLUE benchmark. For image classification, we utilize diverse DNN models, including VGG16, ResNet18, ResNet50, ResNet101, MobileNet_v2, MobileNet_v3, EfficientNet_b0, and EfficientNet_v2. For the GLUE benchmark, our experiments were conducted with the BERT-base model.

The methodology of our PTQ process is as follows. We utilize a small subset of training data to identify the maximum values for weights and activations per channel and layer, respectively. These values are then implemented as scaling parameters in our PTQ models. Specifically, we randomly choose 1000 images from ImageNet and 5% of the data inputs from the GLUE benchmark. This approach of using a limited dataset for scaling parameter determination is a common practice in PTQ, as it allows for a quicker and more efficient quantization process while still aiming to capture the essential statistical characteristics of the full dataset. Crucially, our study deliberately avoids advanced PTQ techniques like PD-Quant and QDrop in [10, 14]. We aim to ensure a fair and unbiased comparison across different data formats, and advanced techniques, potentially favoring certain formats, could skew our findings. This approach allows us to attribute any performance differences to the data formats themselves rather than the optimization techniques. Additionally, we seek to demonstrate that our MERSIT format can achieve satisfactory PTQ accuracies even in basic settings, highlighting the format's inherent strengths and potential utility in scenarios where advanced PTQ techniques are impractical.

Hardware-Efficiency Comparison: We compare the hardware efficiency of FP8, Posit8 and our MERSIT8 data formats. Based on the results of our PTQ accuracy experiments, we chose a single configuration for each data format that demonstrated the best performance. Subsequently, we developed three distinct MAC units,

Table 2: PTQ Accuracy Results

Format→	FP32	INT8	FP (8,2)	FP (8,3)	FP (8,4)	FP (8,5)	Posit (8,0)	Posit (8,1)	Posit (8,2)	Posit (8,3)	MERSIT (8,2)	MERSIT (8,3)
Models	Classification task on ImageNet											
VGG16	73.38	73.27	72.38	73.33	73.25	72.80	73.29	73.37	73.35	72.86	73.33	73.31
ResNet18	69.76	69.60	69.07	69.71	69.52	68.88	69.66	69.67	69.46	68.89	69.70	69.49
ResNet50	80.84	80.69	79.86	80.71	79.90	77.67	80.60	80.69	79.96	77.87	80.77	79.93
ResNet101	81.89	81.71	81.23	81.68	81.31	80.48	81.62	81.75	81.38	80.47	81.67	81.32
MobileNet_v2	72.15	71.79	70.73	70.78	66.30	41.33	71.52	70.92	66.35	41.29	71.12	66.32
MobileNet_v3	75.26	70.55	0.15	73.84	72.72	50.38	47.74	74.43	72.68	50.34	74.53	72.63
EfficientNet_b0	77.68	50.25	0.02	72.20	75.56	63.13	0.12	76.89	75.51	63.13	76.82	75.54
EfficientNet_v2	84.23	25.30	0.02	82.36	83.87	82.48	0.02	84.24	83.82	82.33	84.12	83.79
Datasets	GLUE benchmark with BERT-base Model											
CoLA	83.51	75.32	64.24	80.92	83.13	82.96	69.13	83.13	83.60	83.03	83.43	83.17
MNLI-mm	84.24	82.94	35.05	83.96	84.41	84.08	31.93	84.29	84.46	84.16	84.27	84.44
MRPC	85.29	83.33	31.62	85.05	85.29	84.56	31.62	85.78	85.05	85.05	85.54	85.78
SST-2	92.22	91.51	49.08	92.20	92.32	92.55	64.68	92.43	92.55	92.20	92.22	92.25

each optimized for these selected configurations, utilizing the MAC scheme described in Section 2.2. It is important to note that standardized decoder design methodologies for each data format have not yet been established. To ensure a fair comparison, we applied a similar approach to optimize the area and power efficiency of the FP8 and Posit8 decoders as we did for our MERSIT8 decoder, as detailed in Section 3.3.

Their RTL designs are fully implemented in Verilog and synthesized with Synopsys Design Compiler with a 45nm library. The design is synthesized at a 100 MHz clock frequency. The power consumption is extracted using PrimeTime PX with the average value obtained from actual DNN data. Note that we set a moderate frequency to exclude any considerations related to timing and delay issues in the implementation, despite our decoder having a shorter critical path than the Posit one. The analysis is aimed to fairly compare only the logic overhead between the designs.

4.2 Comparison Study of PTQ Accuracies

Table 2 in our paper provides a detailed overview of the results from our PTQ accuracy experiments. A notable finding from these experiments is the consistent and high performance of Posit(8,1) and our MERSIT(8,2) configurations. These configurations demonstrate the ability to achieve accuracies nearly equivalent to baseline levels across all testing scenarios. This level of consistency emphasizes the adaptability and robustness of Posit(8,1) and MERSIT(8,2), showcasing their potential for versatile applications in different DNN models and tasks.

In our experiments, Posit(8,3) and FP(8,5) demonstrate comparable performance across all test cases, experiencing significant accuracy declines in MobileNet_v2, MobileNet_v3, and EfficientNet_b0. A common trait shared by these configurations is their wide dynamic range, coupled with a limited maximum fraction precision of only two bits, as depicted in Fig. 4. This observation suggests that, in certain scenarios, fraction precision plays a critical role in affecting PTQ accuracies.

In contrast, Posit(8,0) and FP(8,2) offer up to 5-bit fraction precision but possess the smallest dynamic range among the configurations we evaluated. These formats exhibit severe accuracy drops

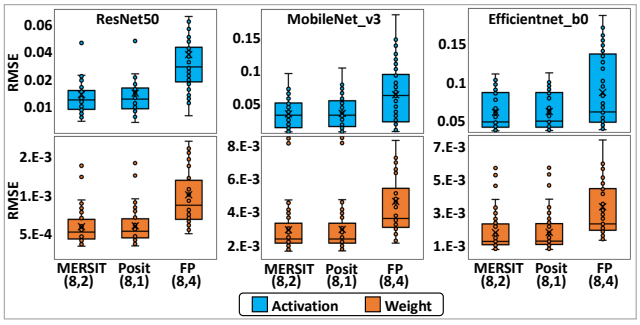


Figure 6: Root-Mean-Square-Error (RMSE) Comparison

in MobileNet_v3, EfficientNet_b0, EfficientNet_v2, and the GLUE benchmarks. The results of Posit(8,0), Posit(8,3), FP(8,2), and FP(8,5) highlight the delicate balance required between fraction precision and dynamic range in data formats for PTQ. Ensuring that both attributes are adequately represented is crucial for maintaining high accuracy levels across various DNNs, especially in advanced and diverse model architectures. INT8, while excelling in specific scenarios, exhibits significant drops in accuracy in advanced image classification tasks involving MobileNet_v3, EfficientNet_b0, and EfficientNet_v2, as well as in the natural language processing tasks with the CoLA dataset from the GLUE benchmark. While FP(8,3) and FP(8,4) configurations generally outperform INT8, they still show a notable decrease in accuracy, particularly in the MobileNet_v2 model.

Among the FP8 configurations, FP(8,4) exhibited the most consistent performance across a variety of experimental scenarios. Consequently, we conduct a more detailed analysis and comparison of three specific configurations: FP(8,4), Posit(8,1), and our MERSIT(8,2). We calculate the root-mean-square error (RMSE) for these configurations as applied to ResNet50, MobileNet_v3, and EfficientNet_b0. The results of this analysis are illustrated in Fig. 6.

The data reveals that the RMSE for our MERSIT(8,2) configuration is slightly better or at least comparable to that of Posit(8,1), and notably lower than that of FP(8,4). This finding is particularly significant as it helps to explain why the PTQ accuracies for both Posit(8,1) and our MERSIT(8,2) are consistently better than those

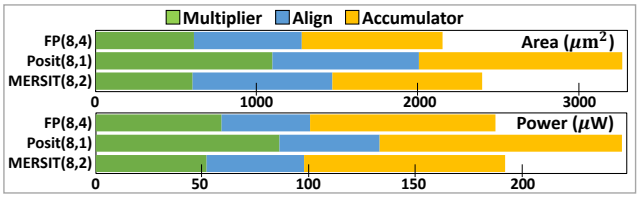


Figure 7: Area and power estimation for the MAC designs

Table 3: Multiplier Breakdown Analysis

Format	FP(8,4)	Posit(8,1)	MERSIT(8,2)
		Area (μm^2)	
Decoder	434	830	338
Exponent-Adder	46	54	54
Fraction-Multiplier	128	216	216
Total	608	1100	607
		Power (μW)	
Decoder	41.73	63.52	33.95
Exponent-Adder	6.57	3.78	6.25
Fraction-Multiplier	12.60	19.50	11.00
Total	60.90	86.8	51.20

of FP(8,4) in the aforementioned DNN models. Our analysis thus underscores the effectiveness of MERSIT(8,2) in maintaining high accuracy while undergoing the PTQ process, making it a promising configuration for practical applications in DNN processing.

4.3 Comparison Study of Hardware Efficiency

Based on our comprehensive accuracy analysis, we choose the optimal configuration of each data format: FP(8,4), Posit(8,1) and MERSIT(8,2). We compare the area and power of the MAC units tailored to these configurations. Fig. 7 shows the area and power comparison of the synthesized MAC units. As depicted in the graph, FP(8,4) and MERSIT(8,2) units occupy significantly less area and consume less power than the Posit unit.

It is noteworthy that the MERSIT(8,2) configuration demonstrates PTQ accuracy nearly equivalent to that of Posit(8,1), signifying that MERSIT(8,2) achieves a significant reduction in hardware resources – specifically, a 26.6% decrease in area and a 22.2% reduction in power consumption compared to Posit(8,1). While the area required for MERSIT(8,2) is 11% larger than that for FP(8,4), this increase can be justified by the enhanced PTQ accuracy offered by MERSIT(8,2). Despite its larger area, the power consumption of MERSIT(8,2) remains almost on par with FP(8,4). This efficiency in MERSIT(8,2) is attributed to its unique configuration, where a considerable portion of weights have zero-length fractions. As illustrated in Fig. 4, the dynamic range of values with fractions in MERSIT(8,2) is more restricted, spanning from 2^{-6} to 2^{5}, as opposed to the broader range of 2^{-8} to 2^{7} found in Posit(8,1) or FP(8,4). Such a characteristic of MERSIT(8,2) leads to lower switching power compared to FP(8,4).

Our MERSIT(8,2) configuration, owing to its wider dynamic range, occupies a larger area than FP(8,4) in components like the aligner and accumulator. Intriguingly, despite having more fraction bits, MERSIT(8,2) exhibits area and power consumption for the multiplier that are nearly equivalent to those of FP(8,4). To delve into the reasons behind this seemingly paradoxical situation, we analyze the area and the power of the multiplier component, described in Table 3.

The results clearly show that the area and power of our MERSIT(8,2) decoder are achieved with the least value. It outperforms Posit(8,1), saving 59.2% of the area thanks to our simplified decoding logic. FP(8,4) also occupies a non-negligible area for the decoder, larger than MERSIT(8,2), as it deals with subnormal numbers and exponent biasing. However, with a shorter fraction bit-width, FP(8,4) utilizes less area for the fraction multiplier, resulting in a similar area occupation for the multiplier compared to MERSIT(8,2). On the other hand, the power of MERSIT(8,2) shows a notably lower value due to the data format characteristic mentioned above.

5 CONCLUSION

This paper presents MERSIT, a novel 8-bit PTQ data format that leverages the dynamic range with the proposed merged decoding scheme. Deep learning acceleration using MERSIT demonstrates superior hardware efficiency compared to Posit while achieving PTQ accuracy results comparable to the Posit. Compared to FP8, MERSIT demonstrates similar hardware efficiency while exhibiting significantly more reliable accuracy than the FP8 model. Ultimately, the MERSIT data format has been successfully designed with the aim of winning both accuracy and hardware efficiency in processing DNN models, including various vision models and LLM models.

ACKNOWLEDGMENTS

This work was partly supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) under Grant 2021-0-00105, Grant 2021-0-00106 and Grant RS-2022-00155911 (Artificial Intelligence Convergence Innovation Human Resources Development (Kyung Hee University)).

REFERENCES

[1] Michael Andersch et al. 2022. *NVIDIA Hopper Architecture In-Depth.* Technical Report. NVIDIA Corporation.
[2] Zachariah Carmichael et al. 2019. Deep Positron: A Deep Neural Network Using the Posit Number System. In *DATE '19.*
[3] Jacob Devlin et al. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]
[4] Amir Gholami et al. 2021. A Survey of Quantization Methods for Efficient Neural Network Inference. arXiv:2103.13630 [cs.CV]
[5] Cong Guo et al. 2022. ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network Quantization. In *MICRO '22.*
[6] Gustafson and Yonemoto. 2017. Beating Floating Point at Its Own Game: Posit Arithmetic. *Supercomput. Front. Innov.: Int. J.* (2017).
[7] Ulrich Kulisch. 2012. *Computer Arithmetic and Validity: Theory, Implementation, and Applications.* De Gruyter, Berlin, Boston.
[8] Hamed F. Langroudi et al. 2020. Adaptive Posit: Parameter aware numerical format for deep learning inference on the edge. In *CVPR '20.*
[9] Ji Lin et al. 2023. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. arXiv:2306.00978 [cs.CL]
[10] Jiawei Liu et al. 2023. PD-Quant: Post-Training Quantization Based on Prediction Difference Metric. In *CVPR '23.*
[11] Thierry Tambe et al. 2020. Algorithm-Hardware Co-Design of Adaptive Floating-Point Encodings for Resilient Deep Learning Inference. In *DAC '20.*
[12] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *ICML '19.*
[13] Mart van Baalen et al. 2023. FP8 versus INT8 for efficient deep learning inference. arXiv:2303.17951 [cs.LG]
[14] Xiuying Wei et al. 2022. QDrop: Randomly Dropping Quantization for Extremely Low-bit Post-Training Quantization. In *ICLR '22.*
[15] Thomas Yeh et al. 2022. Be Like Water: Adaptive Floating Point for Machine Learning. In *ICML '22.*