Shedding the Bits: Pushing the Boundaries of Quantization with Minifloats on FPGAs

Shivam Aggarwal*†, Hans Jakob Damsgaard‡†, Alessandro Pappalardo†, Giuseppe Franco§, Thomas B. Preußer†, Michaela Blott†, Tulika Mitra*
*School of Computing, National University of Singapore, Singapore, †AMD Research and Advanced Development, Dublin, Ireland, ‡Electrical Engineering Unit, Tampere University, Finland, §AMD, Germany

***Abstract*—Post-training quantization (PTQ) is a powerful technique for model compression, reducing the numerical precision in neural networks without additional training overhead. Recent works have investigated adopting 8-bit floating-point formats (`FP8`) in the context of PTQ for model inference. However, floating-point formats smaller than 8 bits and their relative comparison in terms of accuracy-hardware cost with integers remains unexplored on FPGAs. In this work, we present minifloats, which are reduced-precision floating-point formats capable of further reducing the memory footprint, latency, and energy cost of a model while approaching full-precision model accuracy. We implement a custom FPGA-based multiply-accumulate operator library and explore the vast design space, comparing minifloat and integer representations across 3 to 8 bits for both weights and activations. We also examine the applicability of various integer-based quantization techniques to minifloats. Our experiments show that minifloats offer a promising alternative for emerging workloads such as vision transformers.**

***Index Terms*—minifloats, multiply-accumulate, quantization**

I. Introduction

With the increasing demand for deploying machine learning models on resource-constrained devices, post-training quantization (PTQ) [1]–[3] has been the prevailing choice for model compression, offering high model accuracy with reduced compute and memory usage. PTQ enables low-precision arithmetic without any model re-training and with minimal fine-tuning or calibration using only a small unlabeled dataset.

Multiply-accumulate (MAC) operations are ubiquitous in deep learning models. While GPUs with their high concentration of MACs remain the primary computing platform for these workloads, their support for different numerical precisions is inherently limited. In contrast, FPGAs offer unparalleled flexibility by supporting arbitrary formats at the bit level, presenting a unique advantage. Lately, several quantization works [4]–[6] have explored the idea of utilizing 8-bit floating-point (`FP8`) representations in lieu of integers. Integer formats are often preferred over floating-point alternatives due to their simpler hardware implementation. However, as the bit-width of floating-point formats decreases, so do their resource footprints. This leads us to hypothesize floating-point MACs with size and throughput akin to their integer counterparts.

For instance, prior research by Xilinx demonstrated a 7-bit floating-point representation that reduces resource consumption significantly while maintaining comparable or even superior accuracy in comparison to the `INT8` format [7]. This observation makes reduced-precision floating-point arithmetic on FPGAs a practical and valuable choice, especially for models like transformers [8], which frequently feature numerous outliers in their activation distribution [9].

Building upon these advancements, our study delves into the realm of *minifloat quantization* and explores the hardware efficiency of representations with fewer than 8 bits on FPGAs. Such minifloats with low bit-widths can exhibit better model performance compared to their integer counterparts, mostly due to their superior dynamic range with more values close to zero. Moreover, the flexibility to adjust both the number of exponent and mantissa bits allows us to mitigate the loss in model accuracy that often accompanies reduced precision.

Unfortunately, existing literature predominantly focuses on reduced-precision integer quantization, with limited exploration of minifloats in terms of accuracy-hardware tradeoffs. To address this gap, we undertake a comprehensive investigation of the design space encompassing minifloats and integer quantization. Our investigation spans precisions ranging from 3 to 8 bits, applied to both weights and activations across a set of deep learning vision workloads for the ImageNet classification task [10].

Furthermore, we implement custom bit-width MAC units for both minifloats and integers to analyze the impact of the number of exponent and mantissa bits within different minifloat formats on FPGAs. Finally, we investigate the effect of existing post-training optimization techniques, such as SmoothQuant [9], gradient-based learned rounding [2], and GPTQ [3] for minifloats. Our main contributions are:

- We propose a novel PTQ quantization framework for low-precision minifloats, ranging from 3 to 8 bits.
- We implement an operator library to realize custom bit-width integer and minifloat MACs on FPGAs.
- We thoroughly explore the accuracy-hardware trade-offs, providing an in-depth analysis of three prominent vision models – ResNet-18, MobileNetV2, and ViT-B-32 – based on our custom FPGA-based operator library.

Our experiments indicate that minifloat quantization typically outperforms integer quantization for bit-widths of four or more, both for weights and activations. However, when compared against our FPGA hardware cost model, integer quantization often retains its Pareto optimality due to its slightly smaller hardware footprint than minifloats at a given precision.

Work by Shivam Aggarwal and Hans Jakob Damsgaard was carried out during internships with AMD Research.

II. Related Work

Recent research proposes the adoption of FP8 formats for efficient model inference. Micikevicius *et al.* [4] introduce two FP8 formats to represent the weights and activations of a neural network. Kuzmin *et al.* [5] and Noune *et al.* [6] study the effects of the FP8 format, varying the number of exponent and mantissa bits, and the exponent bias. They observe significant performance improvements by searching for the best exponent bias term instead of setting the bias as per the IEEE 754 standard [11]. More complex group-wise quantization techniques [12] have also been proposed for floating-point formats smaller than 8 bits, targeting specifically large language models (LLMs). However, these methods maintain actual computation in the full-precision format without considering any hardware constraints.

Numerous studies [13]–[15] explore optimized MAC designs for deep learning on FPGAs. Most of these works focus on 8-bit fixed-point quantization [15], [16], with a few considering binary quantization that converts MACs into pop-counts [17], [18]. A few works propose accurate floating-point MACs for FP32 or FP64 operands [14], [19], [20]. Other studies investigate quantization for low-precision accumulation [21], [22]. Finally, FloPoCo offers a flexible library for a wide range of arithmetic units for FPGAs, including floating-point MACs, but limits designs to have heterogeneous operand formats [23]. In contrast to existing works, we systematically analyze various configuration settings for reduced-precision integer and minifloat quantization on FPGAs using custom bit-width MACs. Our study provides a detailed perspective on the considerations involved in selecting the appropriate precision formats with the target neural network, model accuracy, and various hardware constraints in mind.

III. Background: Integer Quantization

Table I outlines the notations used in this work. For integer (INT) quantization, given a tensor $\mathbf{X}$, associated scaling factor $\mathtt{s}$, and zero-point $\mathtt{z}$, the quantization Eq. (1) and dequantization Eq. (2) operations are defined as follows:

$$\mathbf{X}_q = \text{quantize}(\mathbf{X}; \mathtt{s}, \mathtt{z}) = \text{clip}\left(\left\lfloor \frac{\mathbf{X}}{\mathtt{s}} \right\rceil + \mathtt{z};\ q_{\text{min}}^{(\text{INT})}, q_{\text{max}}^{(\text{INT})}\right) \quad (1)$$

$$\text{dequantize}(\mathbf{X}_q; \mathtt{s}, \mathtt{z}) = \mathtt{s} \cdot (\mathbf{X}_q - \mathtt{z}) \quad (2)$$

where $\lfloor \cdot \rceil$ is the round-to-nearest operator and $\mathtt{s}$ is defined as:

$$\mathtt{s} = \frac{t}{q_{\text{max}}^{(\text{INT})}} \quad (3)$$

over the entire tensor **(per-tensor)** or for each output channel in the tensor **(per-channel)**. For weights, we define $t = \max(|\mathbf{X}|)$ or $t_j = \max(|\mathbf{X}_j|)$, $j = 1, 2, \ldots, C_o$, where j refers to the j-th channel and C_o corresponds to the total number of output channels. For activations, t is a per-tensor value determined through a calibration procedure. In general, $[q_{\text{min}}^{(\text{INT})}, q_{\text{max}}^{(\text{INT})}]$ is the quantization range controlled by the target bit-width r. For signed integers, $q_{\text{min}}^{(\text{INT})} = -2^{r-1}$ and $q_{\text{max}}^{(\text{INT})} = 2^{r-1} - 1$. For unsigned integers, $q_{\text{min}}^{(\text{INT})} = 0$ and $q_{\text{max}}^{(\text{INT})} = 2^r - 1$. For both weights and activations, we keep $z = 0$.

TABLE I: Notation used in this work.

Name	Notation
tensors (general, weights, activations)	$\mathbf{X}$, $\mathbf{W}$, $\mathbf{Y}$
tensor value	$\mathbf{x}$
scaled tensor value	$\overline{\mathbf{x}}$
quantized tensor	$\mathbf{X_q}$
scaling factor	$\mathtt{s}$
zero-point	$\mathtt{z}$
maximum value of tensor	t
quantization range	$[q_{\text{min}}, q_{\text{max}}]$
FP format fields (sign, exponent, mantissa)	S, E, M
FP mantissa bit-width	m
FP exponent bit-width	e
total bit-width	r
exponent bias	b
exponent value	u
fine-grained, internal scale	ss

Fig. 1a describes the complete integer quantization flow for the INT4 format.

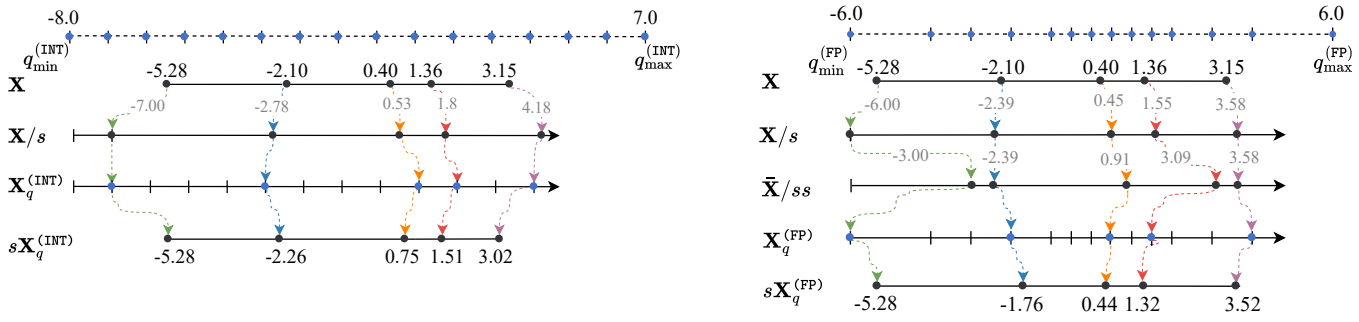
IV. Minifloat Quantization

A standard IEEE floating-point (FP) format consists of three key components: a sign bit $S \in \{0, 1\}$, an m-bit mantissa M, and an e-bit exponent E [11]. These elements sum up to the total bit-width of the format r. Given the exponent bias b as the final parameter of the format, a normalized floating-point representation is interpreted by:

$$x^{(\text{FP})} = (-1)^S \times 2^{u-b} \times \left(1 + \sum_{i=1}^{m} M_i \times 2^{-i}\right) \quad (4)$$

where $0 \leq u < 2^e$ represents the range of exponent values and $M_i \in \{0, 1\}$ denotes the i^{th} bit of the m-bit mantissa. The mantissa combined with the implicit digit of the floating-point number constitutes its *significand*. The IEEE standard FP32 format [11] features $e = 8$ and $m = 23$, with an exponent bias defined as $b = 2^{e-1} - 1$, which equals 127. The number of mantissa bits determines the precision of values within a given range, the number of exponent bits governs the dynamic range of representable values, and the exponent bias controls the position of the range on the real number line. In general, $[q_{\text{min}}^{(\text{FP})}, q_{\text{max}}^{(\text{FP})}]$ is the quantization range determined by the number of exponent and mantissa bits and the exponent bias. Here, $q_{\text{min}}^{(\text{FP})} = -(2 - 2^{-m}) \cdot 2^{2^e - b - 1}$ and $q_{\text{max}}^{(\text{FP})} = (2 - 2^{-m}) \cdot 2^{2^e - b - 1}$.

In this study, we investigate *minifloats*, which are reduced-precision floating-point representations ranging from 3 to 8 bits. Later, we denote FP formats with $e = x$ and $m = y$ by ExMy. We adhere to IEEE standards for the exponent bias and set it to $b = 2^{e-1} - 1$. However, we deviate from IEEE conventions on the treatment of inf and NaN and represent neither. We choose not to represent inf as we assume that the values outside the representation range of a minifloat are saturated. Once inf is excluded, NaN cannot be generated when performing only multiplications and additions. Our design space exploration for various floating-point representations is guided by the constraints: $e \in [1, r-1)$ and $m = r-1-e$. We do take advantage of subnormal numbers to maintain precision for values near 0. Subnormal numbers have an exponent value



(a) INT4 Quantization.

(b) E2M1 Quantization.

Fig. 1: PTQ process with INT4 representation and E2M1 representation.

u set to 1 and are interpreted to have a leading significand digit of zero rather than the one shown in Eq. (4). This allows for extra dynamic range and a graceful degradation of significand precision by introducing leading zeros.

We present a novel minifloat quantization method with two-level scaling, as illustrated in Fig. 1b. Several recent works emphasize the careful selection of appropriate scaling factors and exponent biases for FP8 formats [5], [24]. We take a more straightforward approach aligned with existing integer quantization techniques. We introduce a coarse-grained scaling factor s maintained **per-tensor** or **per-channel** to normalize the original tensor. Formally, for tensor $\mathbf{X}$ and scaling factor $\mathtt{s}$:

$$\mathtt{s} = \frac{t}{q_{\text{max}}^{(\text{FP})}} \qquad (5) \qquad\qquad \overline{\mathbf{X}} = \frac{\mathbf{X}}{\mathtt{s}} \qquad (6)$$

In accordance with the integer quantization process, we also define for weights $t = \max(|\mathbf{X}|)$ or $t_j = \max(|\mathbf{X}_j|)$, j = 1, 2, . . . , C_o, where j refers to the j-th channel and C_o corresponds to the total number of output channels. For activations, t is a per-tensor value determined through the calibration procedure.

We maintain a fine-grained, internal scaling factor $\mathtt{ss}$ for each element $\overline{\mathtt{x}}$ in the scaled tensor $\overline{\mathbf{X}}$ as in [5]. This scaling factor depends on the number of mantissa bits m and the specific range the element $\overline{\mathtt{x}}$ falls into. The scaling factor $\mathtt{ss}$ maps each element $\overline{\mathtt{x}}$ onto its nearest neighbor within the minifloat quantization grid. It is maintained as a power-of-two and can be stored with minimal overhead. The expression 2^{1-b-m} serves as the threshold for the smallest value representable by a given minifloat format. Consequently, any value p that falls below this threshold is subjected to a clipping operation and subsequently assigned a value of $1-b-m$. Given the specified values of $\overline{\mathtt{x}}$, m, and b, we define this condition as follows:

$$\log_2 \mathtt{ss} = p = \max(\lfloor \log_2 |\overline{\mathtt{x}}| \rfloor - m, 1 - b - m) \qquad (7)$$

Finally, we define the minifloat quantization operation Eq. (8) for $\overline{\mathtt{x}}$ and its associated dequantization operation Eq. (9) as:

$$\begin{aligned} \mathbf{x}_q &= \text{quantize}(\overline{\mathtt{x}}; e, m, b, \mathtt{s}, \mathtt{ss}) \\ &= \text{clip}\left(\mathtt{ss}\left\lfloor \frac{\overline{\mathtt{x}}}{\mathtt{ss}} \right\rceil;\ q_{\text{min}}^{(\text{FP})}, q_{\text{max}}^{(\text{FP})}\right) \end{aligned} \qquad (8)$$

$$\text{dequantize}(\mathbf{x}_q; \mathtt{s}) = \mathtt{s} \cdot \mathbf{x}_q \qquad (9)$$

where $\lfloor \cdot \rceil$ is the round-to-nearest operator.

A. PTQ Optimization for Minifloats

In this section, we discuss several *integer* PTQ methods and their applicability to minifloat quantization.

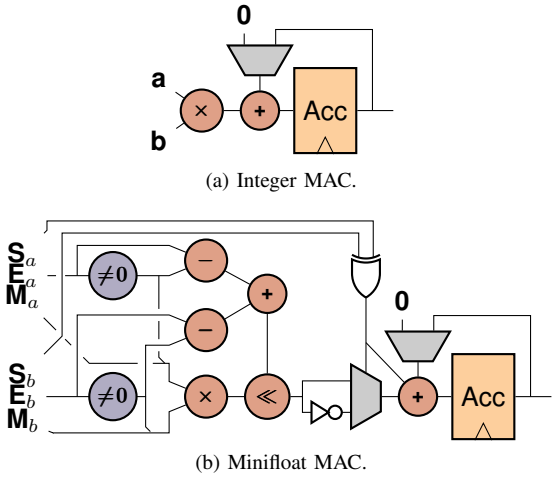
a) SmoothQuant: Activation quantization can be challenging, especially because of outliers, which are difficult to capture with static per-tensor scaling factors. SmoothQuant [9] addresses this challenge by introducing per-channel smoothing factors, thus distributing the burden of quantization between input activations and the following weight matrix. This integration, facilitated by scaling before quantization, seamlessly aligns with the proposed minifloat quantization flow.

b) Bias Correction: Another major challenge in quantization is the introduction of biased errors within the output distribution of a layer. These biases primarily stem from the errors introduced by both weights and activations in the preceding layer, ultimately leading performance degradation, as highlighted in [1], [25], [26]. In this work, we study the impact of *empirical bias correction* for minifloat quantization by analyzing the layer-wise quantization errors using a small calibration dataset. By subtracting the product of these errors and the mean value of the input activations from the biased output, we aim to alleviate the detrimental effects of biased errors within subsequent layers.

c) Gradient-based Learned Rounding: The integer and minifloat quantization processes rely on the round-to-nearest operator $\lfloor \cdot \rceil$, as shown in Eq. (1) and Eq. (8), to project full-precision FP32 values onto their nearest neighbor in the quantization grid. However, previous works [2], [27] underscore the sub-optimality of this operation in the context of post-training *integer* quantization. Building upon these insights, we observe a similar scenario with the minifloat quantization, where the round-to-nearest operation may be sub-optimal under certain circumstances. To address this, we advocate adopting a gradient-based learned rounding approach akin to the principles expounded in the literature. Specifically, we adapt Eq. (8) for the scaled weight values $\overline{\mathbf{w}}$ as follows:

$$\begin{aligned} \mathbf{w}_q^{(\text{FP})} &= \text{quantize}(\overline{\mathbf{w}}; e, m, b, \mathtt{s}, \mathtt{ss}) \\ &= \text{clip}\left(\mathtt{ss}\left\lfloor \frac{\overline{\mathbf{w}}}{\mathtt{ss}} \right\rfloor + h(\mathbf{V});\ q_{\text{min}}^{(\text{FP})}, q_{\text{max}}^{(\text{FP})}\right) \end{aligned} \qquad (10)$$

Here, $h(\mathbf{V})$ is a rectified sigmoid function [28] using the



(a) Integer MAC.

(b) Minifloat MAC.

Fig. 2: Simplified illustrations of the considered integer and minifloat MACs with two operands, **a** and **b**.

learnable variable $\mathbf{V}$, and hyperparameters ζ $(=1.1)$ and γ $(=-0.1)$. To minimize the overall objective, the regularization function encourages the term $h(\mathbf{V})$ to converge on 0 or 1, forcing the round-to-nearest operator to approximate to the floor or ceil function. All other hyperparameters remain consistent with those employed in integer-based learned rounding.

d) GPTQ: Lastly, we investigate applying GPTQ [3] to minifloat quantization. GPTQ leverages tricks such as unordered quantization, local updates, and Cholesky decomposition to efficiently quantize large-scale models at very low precision. Remarkably, the minifloat quantization formulation seamlessly integrates within the existing GPTQ framework, requiring no additional hyperparameter fine-tuning.

V. FPGA Operator Library

To explore our hypothesis on minifloat hardware efficiency, we implement MACs for integer and minifloat representations on a recent AMD Versal™ device. Our integer MAC shown in Fig. 2a is trivial, whereas our minifloat MAC is more complex and involves a long fixed-point accumulator [29]. As Fig. 2b shows, the two minifloat operands undergo several preliminary operations – primarily, the extraction and subsequent multiplication and shifting of their significands – before they can be added to the accumulator. The sign-inversion of the mantissa products, shown with an inverter and a multiplexer, is merged with the subsequent addition. This architecture is modeled after the well-established *long adder and shift* design from [30, Section 8.4.1] and [31, Section 2.2]. Unlike recent work that applies segmentation to the accumulator for improved latency [30, Sections 8.4.2-4] [31], we maintain the long adder as it maps well to the hardened carry primitives available in the AMD FPGAs [32].

Both MACs feature parameterizable operand formats, which we adapt to align with our experiments. Additionally, we set their pipeline depth to a fixed value of two, ensuring reasonable operating frequencies. Notably, we omit considerations for the hardware costs associated with converting the accumulated values in the minifloat MAC back into the floating-point format, as this operation can typically be merged into the subsequent activation function as by implementations based on thresholding [33]. Note that our model accuracy assessments are conducted with accumulators in `FP32`. Given the limited precision of the minifloat formats under consideration, this choice adequately ensures the validity of our results.

As the integer MAC accumulates products in a straightforward manner, its accumulator width is $\mathrm{r}_a + \mathrm{r}_b + \lceil \log_2 n \rceil + 1$, where r_a and r_b are the operand widths, and n is the maximum number of supported products [21]. However, as the minifloat MAC necessitates shifting the significand product before accumulation, its accumulator width grows exponentially with the exponent bit-widths of the operand formats and is computed as $2^{e_a} + m_a + 2^{e_b} + m_b + \lceil \log_2 n \rceil - 1$, where e_a and e_b are the exponent bit-widths and m_a and m_b are the mantissa bit-widths [31]. This leads us to anticipate that minifloat MACs for operands with wide exponents may be uneconomical. For our experiments, we choose n as the maximum dot product size found in ResNet-18[1], 4608.

VI. Experimental Setup

We focus our experiments on three deep learning-based vision models: ResNet-18 [34], MobileNetV2 [35], and ViT-B-32 [36] on the ImageNet-1K dataset [10]. Our code is implemented in PyTorch, utilizing the Brevitas library for post-training integer and minifloat quantization [37]. We restrict our experiments to signed representations with z set to 0, a choice widely adopted to mitigate computational overhead during the inference process [38]. We quantize the first and last layers in the `INT8` and `E3M4` formats for integer and minifloat quantization, respectively, since these layers are sensitive to quantization [39], [40]. In line with previous work [41], we use a small calibration dataset of 1000 images for activation calibration, bias correction, learned rounding, and GPTQ.

We consider data types spanning 3 to 8-bit formats for both weights and activations in convolutional, linear, and general matrix multiplication layers. We keep the same number of mantissa and exponent bits across all network layers. We conduct a thorough *design-space exploration*, experimenting with all combinations of mantissa and exponent bits for a given target bit-width, and report the configurations corresponding to the best model accuracy. Our experiments encompass both per-tensor and per-channel weight scaling with scaling factors preserved in `FP32` format. For a given experiment, we apply either GPTQ or learned rounding but not both, while all other PTQ methods can be enabled concurrently.

We investigate the accuracy-hardware trade-offs of our quantized models using two hardware cost metrics. The first metric is the **memory footprint**, defined with respect to the weight bit-width in a quantized model to compare the relative costs of different precision configurations. The second metric is **look-up table (LUT) utilization** of a MAC on the FPGA for a particular set of weight and activation formats and dot

[1]MobileNetV2 and ViT-B-32 feature maximum dot product sizes of 1280 and 3072, meaning they require one or two fewer bits in their accumulators. These differences have negligible impact on the LUT utilization of our MACs.

TABLE II: Top-1 model accuracy (%) and hardware utilization (#LUTs) results for different floating-point (FP) and integer (INT) representations compared to the full-precision FP32 model in ascending order of the weights and activation bit-widths. Weight and activation bit-widths are denoted by W×A. The best model accuracies are highlighted in **Green**, while those closely trailing the best model, with a difference of less than 0.05, are highlighted in Olive Green.

#config W×A	#LUTs	ResNet-18					MobileNetV2					ViT-B-32				
		Acc (%)		FP config			Acc (%)		FP config			Acc (%)		FP config		
	INT	INT	FP	W	A	#LUTs	INT	FP	W	A	#LUTs	INT	FP	W	A	#LUTs
FP32	–	69.76		E8M23		–	71.90		E8M23		–	75.91		E8M23		–
3×3	25	**48.22**	46.0	E1M1	E1M1	27	0.21	0.17	E1M1	E1M1	27	0.11	0.11	E1M1	E1M1	27
3×4	37	62.36	**62.50**	E1M1	E2M1	42	5.13	3.93	E1M1	E2M1	42	**33.45**	0.11	E1M1	E1M1	27
3×5	38	**65.95**	65.48	E1M1	E2M2	44	25.38	**25.57**	E1M1	E2M2	44	67.16	**69.29**	E1M1	E3M1	54
4×4	40	64.71	**66.48**	E2M1	E2M1	43	17.50	**21.83**	E1M2	E2M1	45	41.47	**62.07**	E2M1	E2M1	43
3×6	44	**67.08**	66.14	E1M1	E2M3	57	**48.38**	41.28	E1M1	E2M3	57	71.98	**72.06**	E1M1	E3M2	75
4×5	50	67.67	**68.60**	E2M1	E2M2	57	50.13	**54.30**	E1M2	E2M2	58	68.62	**74.01**	E2M1	E3M1	78
3×7	51	**67.49**	66.30	E1M1	E3M3	74	**56.59**	45.63	E1M1	E2M4	60	**73.45**	72.46	E1M1	E3M3	74
3×8	57	**67.69**	66.51	E1M1	E4M3	107	**59.45**	47.63	E1M1	E3M4	86	**74.04**	72.56	E1M1	E3M4	86
4×6	56	68.79	**69.13**	E2M1	E2M3	64	62.44	**64.09**	E2M1	E2M3	64	74.08	**75.27**	E2M1	E3M2	83
5×5	55	68.02	**69.02**	E2M2	E2M2	65	56.42	**60.30**	E2M2	E2M2	65	69.44	**74.64**	E2M2	E3M1	83
4×7	64	69.05	**69.25**	E2M1	E2M4	78	66.04	**66.81**	E2M1	E2M4	78	74.91	**75.49**	E2M1	E3M3	111
5×6	63	69.20	**69.47**	E1M3	E2M3	82	66.64	**67.96**	E2M2	E2M3	89	74.39	**75.54**	E2M2	E3M2	95
4×8	67	69.24	**69.37**	E2M1	E2M5	78	**68.05**	67.93	E2M1	E2M5	78	75.45	**75.58**	E2M1	E3M4	110
5×7	71	69.47	**69.56**	E1M3	E2M4	103	68.91	**70.16**	E2M2	E2M4	87	75.29	**75.75**	E2M2	E3M3	118
6×6	72	69.18	**69.55**	E2M3	E2M3	89	67.80	**68.30**	E2M3	E2M3	89	74.55	**75.64**	E3M2	E3M2	110
5×8	78	**69.66**	69.60	E1M3	E2M5	86	70.26	**70.63**	E2M2	E2M5	101	75.74	**75.83**	E2M2	E3M4	154
6×7	78	69.54	**69.60**	E2M3	E3M3	122	69.98	**70.62**	E2M3	E2M4	119	75.26	**75.84**	E2M3	E3M3	122
6×8	87	69.63	**69.67**	E3M2	E3M4	134	71.06	**71.24**	E2M3	E2M5	102	75.70	**75.89**	E1M4	E3M4	110
7×7	92	**69.65**	69.63	E4M2	E2M4	174	70.31	**70.77**	E1M5	E2M4	107	75.30	**75.79**	E2M4	E3M3	131
7×8	110	69.65	**69.69**	E2M4	E3M4	130	71.30	**71.38**	E2M4	E2M5	121	75.72	**75.90**	E4M2	E3M4	183
8×8	116	**69.71**	69.68	E3M4	E2M5	147	71.36	**71.52**	E2M5	E2M5	133	75.79	**75.89**	E4M3	E4M3	191

product size. Prior research efforts employ similar metrics for memory and arithmetic density [21], [42]. We choose to focus solely on LUTs rather than hardened digital signal processing slices (DSPs) for two reasons: firstly, DSPs are relatively rare in Versal™ FPGAs (roughly, one per 200-400 LUTs [32]) and their lack of native support for minifloat formats would give integer designs an unintended upper hand; and secondly, using one metric simplifies comparisons. All included results are post-implementation utilization numbers produced with default settings in Vivado™ 2023.1.

VII. Evaluation & Discussion

A. Impact on Model Accuracy

We first discuss the Top-1 model accuracy (%) for various precision configurations and present the best-case results across all post-training optimization techniques in Table II. We observe that integer quantization outperforms minifloat quantization with 3-bit weights and 3- to 4-bit activations across all three models. However, this performance gap diminishes as the weight precision increases to 4 bits and beyond. For configurations where weights and activations are set to 4 and 5 bits, respectively, and above, minifloat quantization outperforms integer quantization by up to 6% for models such as ViT-B-32. Notably, with 4-bit weights and 8-bit activations, both integer and minifloat representations closely approach the accuracy of the full-precision model. Upon individual model analysis, we conclude that while traditional convolutional models like ResNet-18 perform equally well with integer and minifloat quantization, more complex models such as MobileNetV2 and ViT-B-32, characterized by the presence of outliers in their weight and activation distributions, benefit from minifloat quantization due to its larger dynamic range.

B. Optimal Minifloat Formats

Table II showcases the minifloat formats that achieve the best model accuracy at each weight and activation bit-width configuration. Given that these exponent and mantissa bit-width choices directly influence hardware resource utilization costs, a closer examination of their values is essential. As anticipated, most configurations comprise at least two exponent bits when considering weights. Conversely, a consistent trend emerges for activations, with most models requiring exponents of two or more bits. We note a substantial disparity in the necessary representations of the weights and activations across all three models. Such design requirements can only be accommodated on re-programmable fabric such as FPGAs.

C. Impact on Memory Footprint

We illustrate the trade-off between model accuracy and memory footprint for 3- to 8-bit formats through Pareto curves in Fig. 3. The black dotted lines represent the accuracy of the three models in FP32. Both integer and minifloat formats exhibit sub-optimal performance at 3 bits, particularly for MobileNetV2 and ViT-B-32, where 3-bit integers marginally outperform minifloats. For ResNet-18, both formats exhibit similar performance at different weight bit-widths. For complex networks such as MobileNetV2 and ViT-B-32, minifloats dominate integers at higher bit-widths since they can easily accommodate outliers. For instance, for ViT-B-32, minifloat quantization can attain a remarkable accuracy of 75.89% with

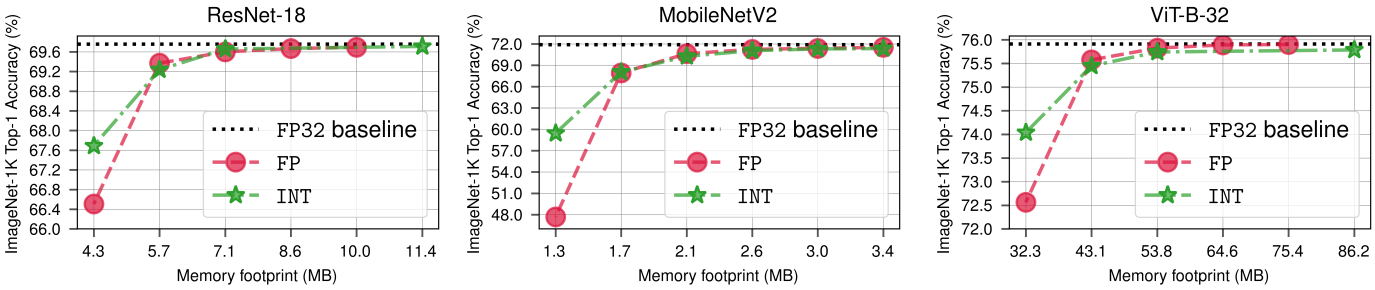


Fig. 3: Trade-off analysis between model accuracy (%) and memory footprint for integers (INT) and minifloats (FP).

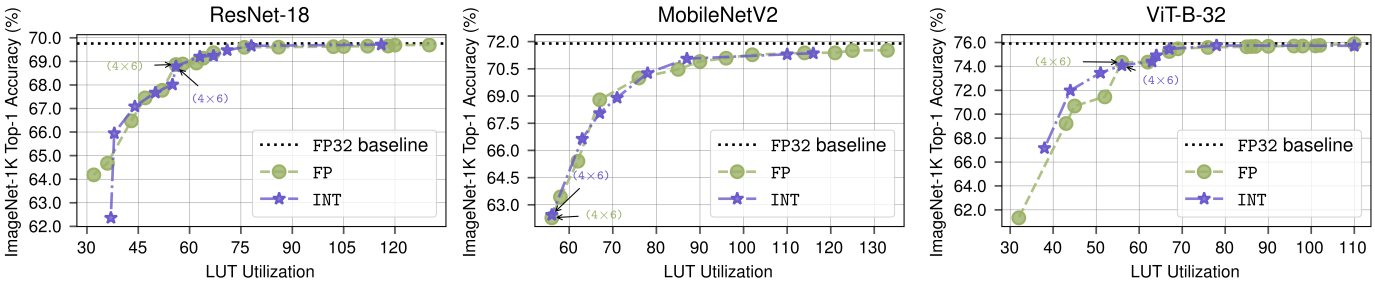


Fig. 4: Trade-off analysis between model accuracy (%) and #LUT utilization for integers (INT) and minifloats (FP). Points where the gap between the two formats converge are labeled with their corresponding bit-width configurations as (W×A).

an impressive 8× compression rate compared to the full-precision model (calculated as 32/4).

D. Impact on FPGA Resource Utilization

Unlike our previous analysis, which focused on determining the optimal model accuracy at varying memory budgets, this section shifts focus to identifying the best model accuracy with respect to the number of LUTs. For the integer MAC, a lower-precision configuration generally utilizes fewer LUTs. However, the resource utilization of a minifloat MAC depends on its distribution of exponent and mantissa bits, especially due to its shifter and, possibly, long accumulator. Observant readers will notice that, in some cases, increasing the bit-width of minifloat formats can lead to reduced LUT utilization. For example, Table II shows that the 3×7 configuration outperforms 3×6 for ViT-B-32. We attribute this to some formats fitting the underlying fabric better, e.g., the seven-bit E3M3 format [7], and to some degree of randomness in the optimizations performed by the synthesis tool.

From the curves in Fig. 4, we observe that integer quantization tends to outperform minifloat quantization when the LUT utilization is very limited, albeit with significant accuracy degradation as compared to the FP32 baseline. However, as the resource budget increases, both integer and minifloat accuracies show an upward trend and eventually converge, as indicated by the labeled bit-width configurations. In the case of ResNet-18, we notice an accuracy difference of <1% when the resource budget is around 45 LUTs. This gap drops significantly as the resource budget surpasses 50 LUTs. Similarly, for MobileNetV2, minifloat quantization performs on par with integer representations, where the margin is less than 0.1% for resource budgets exceeding 64 LUTs. Results for ViT-B-32 show a trend similar to ResNet-18, with the gap narrowing as the number of LUTs exceeds 60.

VIII. Conclusion

In this study, we present a minifloat quantization approach for bit-widths below 8 bits. We conduct a thorough examination of the impact of various PTQ techniques on minifloats and integers with reduced precision spanning from 3 to 8 bits, for both weights and activations. Our research centers on assessing trade-offs between accuracy and hardware resource utilization for ResNet-18, MobileNetV2, and ViT-B-32 models and two bespoke MAC designs implemented on an FPGA. Our experiments demonstrate the relevance of low-precision minifloat quantization, especially in terms of memory footprint at 4 bits or above, while conceding that integer quantization still dominates along the MAC resource utilization Pareto frontier. Our experiments primarily focus on vision models and assume a uniform choice of data formats for all weights and activations in a given model, which limits the design space of configurations that can be explored. We plan to address these limitations in our future work.

IX. Acknowledgments

We thank the anonymous reviewers, Ian Colbert (AMD), and Dan Wu (NUS) for their valuable feedback. Shivam Aggarwal's work is partially supported by the National Research Foundation, Singapore, under its Competitive Research Programme Award NRF-CRP23-2019-0003 and Singapore Ministry of Education Academic Research Fund T1 251RES1905. Hans Jakob Damsgaard acknowledges funding from European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska Curie grant agreement No. 956090 (APROPOS: Approximate Computing for Power and Energy Optimisation, http://www.apropos-itn.eu/).

REFERENCES

[1] M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling, "Data-Free Quantization Through Weight Equalization and Bias Correction," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1325–1334, 2019.

[2] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort, "Up or Down? Adaptive Rounding for Post-Training Quantization," in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML'20. JMLR.org, 2020.

[3] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate Post-training Compression for Generative Pretrained Transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[4] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, N. Mellempudi, S. Oberman, M. Shoeybi, M. Siu, and H. Wu, "FP8 Formats for Deep Learning," 2022.

[5] A. Kuzmin, M. van Baalen, Y. Ren, M. Nagel, J. W. T. Peters, and T. Blankevoort, "FP8 Quantization: The Power of the Exponent," *ArXiv*, vol. abs/2208.09225, 2022.

[6] B. Noune, P. Jones, D. Justus, D. Masters, and C. Luschi, "8-bit Numerical Formats for Deep Neural Networks," 2022.

[7] AMD-Xilinx, "Higher Performance Neural Networks with Small Floating Point," *White Paper*, 2021.

[8] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov, "Transformer-XL: Attentive Language Models beyond a Fixed-Length Context," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 2978–2988.

[9] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A Large-Scale Hierarchical Image Database," in *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009, pp. 248–255.

[11] "IEEE Standard for Floating-Point Arithmetic," *IEEE Std 754-2008*, pp. 1–70, 2008.

[12] T. Dettmers and L. Zettlemoyer, "The Case for 4-bit Precision: k-bit Inference Scaling Laws," 2023.

[13] M. Véstias, R. P. Duarte, J. T. de Sousa, and H. Neto, "Efficient Design of Low Bitwidth Convolutional Neural Networks on FPGA with Optimized Dot Product Units," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 1, dec 2022.

[14] S. Sun and J. Zambreno, "A Floating-point Accumulator for FPGA-based High Performance Computing Applications," in *2009 International Conference on Field-Programmable Technology*, 2009, pp. 493–499.

[15] M. P. Véstias, R. P. Duarte, J. T. de Sousa, and H. C. Neto, "Hybrid Dot-Product Calculation for Convolutional Neural Networks in FPGA," *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 350–353, 2019.

[16] Y. Yu, T. Zhao, K. Wang, and L. He, "Light-OPU: An FPGA-based Overlay Processor for Lightweight Convolutional Neural Networks," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 122–132.

[17] P. Guo, H. Ma, R. Chen, P. Li, S. Xie, and D. Wang, "FBNA: A Fully Binarized Neural Network Accelerator," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 51–513.

[18] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 65–74.

[19] D. Wilson and G. Stitt, "The Unified Accumulator Architecture: A Configurable, Portable, and Extensible Floating-Point Accumulator," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 9, no. 3, pp. 1–23, 2016.

[20] E. Kadric, P. Gurniak, and A. DeHon, "Accurate Parallel Floating-Point Accumulation," *IEEE Transactions on Computers*, vol. 65, no. 11, pp. 3224–3238, 2016.

[21] I. Colbert, A. Pappalardo, and J. Petri-Koenig, "A2Q: Accumulator-Aware Quantization with Guaranteed Overflow Avoidance," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 16 989–16 998.

[22] A. Jain, P. Goel, S. Aggarwal, A. Fell, and S. Anand, "Symmetric k-Means for Deep Neural Network Compression and Hardware Acceleration on FPGAs," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 737–749, 2020.

[23] F. de Dinechin, B. Pasca, O. Cret, and R. Tudoran, "An FPGA-Specific Approach to Floating-Point Accumulation and Sum-of-Products," in *2008 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2008, pp. 33–40.

[24] S. P. Perez, Y. Zhang, J. Briggs, C. Blake, J. Levy-Kramer, P. Balanca, C. Luschi, S. Barlow, and A. W. Fitzgibbon, "Training and Inference of Large Language Models using 8-bit Floating Point," 2023.

[25] E. Meller, A. Finkelstein, U. Almog, and M. Grobman, "Same, Same But Different: Recovering Neural Network Quantization Error Through Weight Factorization," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 4486–4495.

[26] A. Finkelstein, U. Almog, and M. Grobman, "Fighting Quantization Bias with Bias," *ArXiv*, vol. abs/1906.03193, 2019.

[27] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, "BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction," *ArXiv*, vol. abs/2102.05426, 2021.

[28] C. Louizos, M. Welling, and D. P. Kingma, "Learning Sparse Neural Networks through L_0 Regularization," in *6th International Conference on Learning Representations, ICLR 2018*. OpenReview.net, 2018.

[29] U. Kulisch and V. Snyder, "The Exact Dot Product as Basic Tool for Long Interval Arithmetic," *Computing*, vol. 91, no. 3, pp. 307–313, 2011.

[30] U. Kulisch, *Computer Arithmetic and Validity: Theory, Implementation, and Applications*. Walter de Gruyter, 2008.

[31] Y. Uguen and F. de Dinechin, "Design-Space Exploration for the Kulisch Accumulator," HAL Inria, Tech. Rep. hal-01488916v2, 2017.

[32] *Versal Architecture and Product Data Sheet: Overview*, 9 2023.

[33] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[35] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[36] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *ICLR*, 2021.

[37] A. Pappalardo, "Xilinx/brevitas," 2023.

[38] S. R. Jain, A. Gural, M. Wu, and C. Dick, "Trained Uniform Quantization for Accurate and Efficient Neural Network Inference on Fixed-Point Hardware," *ArXiv*, vol. abs/1903.08066, 2019.

[39] R. Zhao, Y. Hu, J. Dotzel, C. D. Sa, and Z. Zhang, "Improving Neural Network Quantization without Retraining using Outlier Channel Splitting," *ArXiv*, vol. abs/1901.09504, 2019.

[40] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized Clipping Activation for Quantized Neural Networks," *ArXiv*, vol. abs/1805.06085, 2018.

[41] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, "Accurate Post Training Quantization with Small Calibration Sets," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 139. PMLR, 2021, pp. 4466–4475.

[42] B. Darvish Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner, A. Forin, H. Zhu, T. Na, P. Patel, S. Che, L. Chand Koppaka, X. SONG, S. Som, K. Das, S. T, S. Reinhardt, S. Lanka, E. Chung, and D. Burger, "Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 10 271–10 281.