FP2: A 2-bit Floating-Point Format for Edge-AI Inference and Fine-Tuning

Qiwei Dang, Chengyu Ma, Haiduo Huang, Gelin Fu, *Member, IEEE*, Zhiwang Huo, Guoming Yang, Pengchen Zong, Tian Xia, *Member, IEEE*, Wenzhe Zhao, *Member, IEEE*, and Pengju Ren, *Member, IEEE*

***Abstract*—The increasing scale of Deep Neural Networks (DNNs) has made 2-bit quantization crucial for mitigating memory bottlenecks on edge devices. Low-bitwidth floating-point formats, offering larger dynamic ranges and avoiding quantization steps, have emerged as promising alternatives to fixed-point quantization. However, constructing viable floating-point representations with fewer than 3 bits remains challenging, as conventional formats require at least one sign bit, one exponent bit, and one mantissa bit. We address this challenge by introducing a novel data compression method that uses a 4-bit encoding space to represent two floating-point values, achieving an effective storage density of 2 bits per value. Depending on the bit width of the exponent and mantissa, we propose two different 2-bit floating-point encodings: FP2-E1M0 and FP2-E0M1. Based on FP2, we introduce two computing architectures that simplify floating-point multiply-accumulate (MAC) operations into bitwise addition and logic operations, reducing floating-point computation by factors of 2× and 4×. As a result, FP2 offers a practical solution for efficient inference using floating-point arithmetic on resource-constrained edge devices. Moreover, we analyze the error characteristics of the FP2 data format from three perspectives. To validate the effectiveness of the FP2 format, we conduct experiments on ResNet18/50 and ConvNeXt-Tiny using the CIFAR-10 and ImageNet-1K datasets. Compared to FP4, our approach reduces model size by 47%, with accuracy loss is less than 2 percentage points. Notably, on CIFAR-10, some results are close to those of FP32. In contrast, when evaluated under 2-bit GPTQ, FP2 demonstrates significant advantages over the baseline method on the LLAMA model. For hardware evaluation, we implement our design at the RTL level and evaluate it on both FPGA and ASIC platforms. Compared to computation architectures based on FP4, our FP4×FP2 processing element (PE) array reduces area by 15% and power consumption by 8%. Furthermore, our FP2×FP2 PE array achieves a remarkable 78% reduction in both area and power consumption.**

***Index Terms*—Deep neural network, low-precision floating-point number, hardware accelerator.**

I. Introduction

In RECENT years, the increasing size of models has become a development trend in deep neural networks (DNNs), especially with the rise of transformer-based networks [1], [2]. The enormous computational cost associated with models having billions of parameters has exacerbated the gap between computing power demand and supply [3], [4], [5]. Therefore, an effective approach to the efficient processing of DNNs is to represent numbers with fewer bits while maintaining negligible loss in model accuracy. Using lower-bit-width data formats not only enables high-quality DNN inference and fine-tuning results, but also significantly reduces the processing cost of DNNs. DNNs have always been constrained by large amounts of data and computation. By using appropriate numerical representations, data can be encoded with fewer bits, which helps DNN hardware accelerators process the networks with lower memory usage and computational complexity. To address these challenges, the research community has been actively exploring the use of low-bit-width fixed-point data formats during model inference as an alternative to the traditional FP32 format. Recently, the representation of numbers in DNN models has become more diversified, with some works attempting to use low-precision floating-point formats during both model fine-tuning and inference. Notable examples are shown in Figure1(a) include FP16, BF16, FP8, and FP4 [6], [7], [8], [9].

In recent years, these formats have been widely used in neural network computing [10], [11], [12], [13] and neural

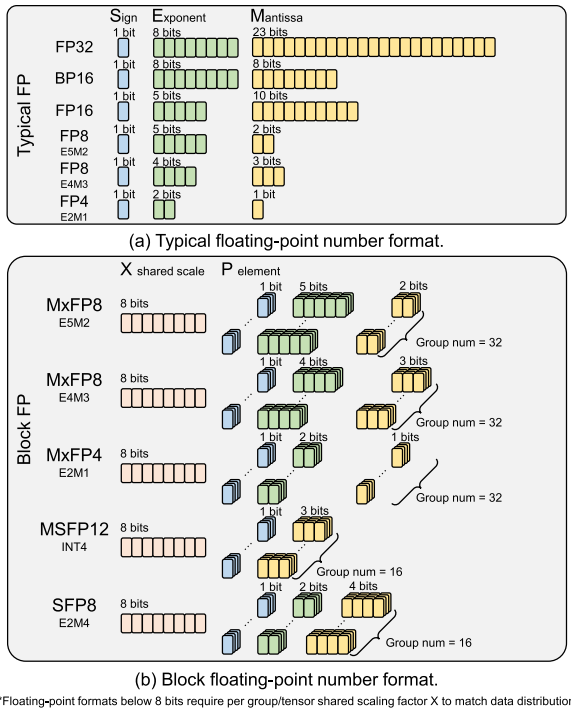


Fig. 1. (a) The typical floating-point number format. (b) Block floating-point number format.

Received 30 September 2025; revised 13 November 2025; accepted 25 November 2025. Date of publication 11 December 2025; date of current version 29 May 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62088102, Grant 62302381, and Grant 52441602. This article was recommended by Associate Editor H. Cai. *(Corresponding author: Wenzhe Zhao.)*

The authors are with the National Key Laboratory of Human–Machine Hybrid Augmented Intelligence, the National Engineering Research Center of Visual Information and Applications, and the Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China (e-mail: wenzhe@xjtu.edu.cn).

Digital Object Identifier 10.1109/TCSI.2025.3638831

network accelerator design. Zhang et al. found that when the bit-width is reduced to 8 bits, the hardware costs of low-bit integer and floating-point MACs become very similar, while the floating-point format greatly enhances the quantization of LLMs [14]; Yan et al. applied floating-point numbers of different bit-widths to FFT processor design [15]; Niu et al. designed a logarithmic multiplier supporting FP8 [16]; Park et al. developed a lightweight floating-point computing unit for RISC-V processors [17]; Fang et al. proposed a reconfigurable floating-point division and square root (FDSR) architecture to achieve high-precision acceleration of the Softmax operator [18].

Furthermore, as the scale of deep neural networks (DNNs) continues to grow, memory bottlenecks have become a key constraint for edge AI accelerators. Computing architectures relying on emerging non-volatile memory (NVM) technologies—such as spin-transfer torque magnetic random access memory (STT-MRAM) [19], [20], [21] and resistive random access memory (RRAM) [22]—have shown great potential in alleviating this issue. These emerging NVMs are highly favored for their low standby power consumption and high storage density, which aligns well with the advantage of low-bit-width data formats in compact parameter representation. By reducing the amount of data read from or written to NVMs, the memory power consumption of edge AI accelerators can be further reduced. For example, Chang et al. implemented a DNN accelerator using block floating-point data formats and compute-in-memory (CIM) [23].

Typically, two approaches are used to construct low-precision floating-point formats. The simplest method is to adjust the lengths of the exponent and mantissa, with a typical example being BF16 [24]. It has an 8-bit exponent and a 7-bit mantissa, and it was proposed for efficient DNN processing. Currently, it is widely applied. However, when compressing models to around 8 bits using this method, excessively low exponent width can impair model accuracy. The second method constructs low-precision floating-point numbers by sharing the exponent within a block [25], [26]. A typical example is FP8, where the maximum exponent is shared within a data block containing 32 data points. As a result, each data element typically requires only 4 bits for the exponent and 3 bits for the mantissa. Industry leaders, such as NVIDIA, have not only begun to establish standards for these formats but have also incorporated support for FP8 in their cutting-edge hardware (e.g., the H100 GPU) [27]. By sharing the exponent, the minimum floating-point data format of 4 bits, FP4, can be constructed. However, some research indicates that computation circuits with bit-widths lower than 3 bits can significantly enhance hardware efficiency [28], [29], [30]. Therefore, we aim to establish a floating-point standard with bit-widths lower than 3 bits.

However, when the bit-width is reduced to below 3 bits, constructing feasible floating-point structures based on the traditional sign, exponent, and mantissa components becomes highly challenging due to the severe limitations on available bits. This paper addresses this issue by introducing a new data compression method. Specifically, this data format is built on a shared exponent block floating-point format, with finer granularity for sharing the exponent and mantissa. We use a 4-bit encoding space to represent two floating-point values, such that each floating-point number has an effective bit-width of 2 bits. Since the parameters of low-precision models are often sparse, it is necessary to retain the ability to represent the zero state for each of the two floating-point values. We reserve 2 bits for flag bits (C) within the 4-bit encoding space to convey the zero-state information for each value. A 1-bit sign bit is used to store the sign of the data. Thus, only 1 bit remains in the encoding space to store the exponent or mantissa of the floating-point number. The proposed compression method generates two distinct 2-bit floating-point encodings: FP2-E1M0 and FP2-E0M1.

Binary weight networks (BWN) [28], ternary weight networks (TWN) [29], APoT [31], BitNet [32] and BitNet 1.58b [33] have all explored 2-bit ultra-low bit-width data formats. Notably, under the 2-bit constraint, floating-point numbers degenerate into fixed-point representations, while both uniform and non-uniform quantization schemes reduce to ternary schemes. This inherent limitation highlights the fundamental expressive ceiling of traditional 2-bit quantization methods. Considering binary-based storage hardware, from an information-theoretic perspective, storing ternary data in 2 bits restricts the effective storage density to $\left(\frac{3}{4}\right)^{b/2}$ (where b is the total bit-width). As data volume increases, this results in significant representational capacity wastage. For example, representing two adjacent ternary values using 4 bits achieves only 9/16 utilization of the expressive capacity, implying that compressed storage of adjacent values could theoretically double the representation range compared to ternary data.

This makes floating-point implementations and both uniform and non-uniform quantization feasible under 2-bit constraints. In Equation 1, we present the quantization levels of ternary data versus FP2 formats. Each FP2 encoding provides two additional quantization levels compared to ternary format. Moreover, FP2-E1M0 exhibits uniformly distributed levels while FP2-E0M1 has non-uniformly distributed levels. Simultaneously, the compression/decompression overhead remains bounded since only pairwise value processing is required.

$$\begin{aligned} &\text{Ternary:} && scale \times \{0, \pm 1\} \\ &\text{FP2-E1M0:} && scale \times \{0, \pm 0.5, \pm 1\} \\ &\text{FP2-E0M1:} && scale \times \{0, \pm 1, \pm 1.5\} \end{aligned} \tag{1}$$

To validate the effectiveness of the FP2 format, we conduct a comprehensive series of experiments on the CIFAR-10 [34] and ImageNet [35] datasets using ResNet18/50 [36] and ConvNeXt-Tiny [37]. Our results indicate that when activations are retained in the FP4 format to preserve more image information, compressing weights to FP2 and fine-tuning on the high-resolution ImageNet dataset results in an accuracy drop of approximately 1–2 percentage points compared to FP4. This drop diminishes as the network model size increases. On the simpler CIFAR-10 dataset, fine-tuning with FP4×FP2(A4W2) achieves accuracy comparable to both FP4 and FP32. Furthermore, when activations are also compressed to 2-bit and fine-tuned using FP2×FP2(A2W2), the accuracy loss compared to FP32 remains around 2 percentage points. We also conducted a comparative analysis of the zero-shot performance for the proposed FP2 format based on the LLAMA [38] model across three benchmark datasets (namely Winogrande, Winograd, and Hellaswag [39], [40], [41]). The results demonstrate that in a comparative evaluation under 2-bit PTQ, FP2 exhibits significant advantages over baseline methods.

In terms of model size, using the FP2 format instead of FP4 reduces model weight storage overhead by 47%. Taking FP32 as the baseline, FP2 achieves a 93% reduction in model weight storage overhead. Additionally, we evaluate our design on FPGA and ASIC platforms. Compared to computing

architectures based on the FP4 data format, our proposed FP4×FP2 processing element (PE) array reduces area by 15% and power consumption by 8%. Furthermore, our FP2×FP2 PE array achieves a remarkable 78% reduction in both area and power consumption. We believe FP2 compression is grounded in two key observations. First, weight redundancy: our 2-bit compression essentially merges pruning and quantization, so model redundancy is a prerequisite for effective compression. Second, since models rely heavily on multiply-accumulate (MAC) operations, outputs undergo temporal accumulation. Consequently, pruning small values or compressing adjacent data causes minimal fluctuations that do not significantly alter overall trends after accumulation. Thus, this design is well-suited for AI applications. The main contributions of this paper are as follows:

- We propose a 2-bit floating-point encoding standard for deep neural network fine-tuning and inference, which reduces storage overhead by 47% compared to FP4 and by 93% compared to FP32.
- We introduce two multiply-accumulate (MAC) computation schemes based on the FP2 format, FP4×FP2 and FP2×FP2, maximizing system efficiency.
- We propose a multiply circuit design compatible with two different encodings of the FP2 format, effectively reusing logic circuits to maximize hardware resource utilization.
- We present the evaluation results of the proposed design, demonstrating that, compared to existing designs, the proposed architecture significantly reduces resource and energy consumption while maintaining acceptable model accuracy.

II. Background

A. Floating-Point Formats

In deep neural networks, the commonly used data format is FP32. In order to further reduce the computational and storage costs of models, many researches have started to focus on using low-bit-width data formats instead of FP32, such as FP16, BF16, FP8, and FP4. The specific structures of these formats are illustrated in Figure 1(a). Similar to the definition of FP32 in the IEEE-754 standard, the binary representation of the above floating-point data formats consists of three components: sign, exponent, and mantissa. The decimal value of these floating-point numbers is represented as:

$$V_{dec} = (-1)^S \times 2^{E-B} \times 1.M \tag{2}$$

where V_{dec} is the decimal value and S, E, B and M are unsigned numbers denoting the sign, exponent, exponent bias and mantissa, respectively. In this paper we will use *EaMb* to represent different bit-width combinations of the exponent and the mantissa, where a and b indicate the bit-width of E and M. The IEEE-754 standard includes three special definitions. The first is subnormal numbers, which can be calculated as follows:

$$V_{dec} = (-1)^S \times 2^{1-B} \times 0.M \tag{3}$$

Since subnormal numbers are distributed close to zero, all subnormal numbers are set to zero in our work to simplify the calculations. The other two special definitions are *Infinity (Inf)* and *Not a Number (NaN)*. Our work encodes these special values by encoding the shared scale X as all ones.

B. Block Floating-Point Formats

To mitigate the high computational overhead of floating-point operations in deep neural network (DNN) inference, the Block Floating-Point (BFP) data representation format has gained widespread attention due to its hardware-friendly characteristics. As shown in Figure 1(b), a BFP block consists of multiple data elements P and a shared scaling factor X. All elements P within a block adopt the same data format—for example, fixed-point in MSFP and floating-point in MXFP. By sharing a single scaling factor X within a BFP block, traditional wide-bit-width floating-point multiply-accumulate (MAC) operations are simplified into decoupled computations of element-level operations and shared factor operations. Since elements can be quantized to narrower bit-widths, the BFP format achieves a better trade-off between arithmetic logic unit (ALU) area overhead and numerical representation accuracy. The setting of group num for BFP blocks is primarily balanced by two factors: model accuracy loss and hardware design complexity. For instance, Group num=16 in MSFP12 to reduce accuracy loss caused by the element format (INT4); Group num=16 in SFP8 to leverage FPGA DSP48s resources for designing accumulation chains (supporting up to 16 cascades); and Group num=32 in MXFP, a widely recognized trade-off point between model accuracy loss and hardware design complexity.

With NVIDIA's support for low-bit-width floating-point data formats in its latest GPU architectures, the FP8/4 formats have been more widely adopted in the LLM field. It is worth noting that since floating-point formats below 8 bits require a shared scaling factor X per group to mitigate model accuracy loss in practical applications, FP8/4 in real-world use typically refers to the element format of MXFP8/4—as it does in this paper. However, existing research on BFP representation remains focused on 8-bit or 4-bit configurations, leaving a significant research gap in the theory and application of ultra-low-bit-width BFP formats (e.g., below 4 bits).

C. Extreme Quantization (below 4 bits)

Extreme quantization of deep neural networks (DNNs), typically referring to bit-widths below 4 bits, aims to maximize hardware efficiency while maintaining model accuracy. Courbariaux et al. [42], [43] were the first to demonstrate that neural networks could operate at extremely low precision, subsequently introducing binary neural networks (BNN) to achieve 32× memory compression. However, BNNs face significant accuracy degradation in complex vision tasks.

Rastegari et al. [28] introduced XNOR-Net, which achieved a 58× speedup on CPUs via XNOR-based bit counting operations; Meanwhile, binary weight networks (BWN) [28] and ternary weight networks (TWN) [29] explored intermediate quantization levels: TWN employed ternary weights {-α, 0, +α} to achieve a 16× compression, whereas BWN demonstrated that when 1-bit weights are combined with a scaling factor and paired with 32-bit activations, CNNs can achieve acceptable accuracy on ImageNet. Zhou et al. [44] proposed an end-to-end quantization framework called DoReFa-Net, which supports the joint optimization of ultra-low bit-widths.

Research on Adaptive Power-of-Two (APoT) [31] quantizations explores non-uniform quantization methods based on fixed-point numbers constructed from powers of two, revealing that powers of two are superior for quantizing Gaussian-distributed model parameters. Notably, APoT degenerate into a ternary distribution (i.e., parameter values

are restricted to $\{-a, 0, a\}$) under a 2-bit bit-width constraint. However, by employing specific tricks such as learning the upper/lower bounds of quantization clipping ranges during training and applying weight normalization, models based on ternary distributions can achieve accuracy comparable to full-precision (floating-point) models. With the advancement of Large Language Models (LLMs), the concepts of Binary Weight Networks (BWN) and Ternary Weight Networks (TWN) have been applied to LLMs like BitNet [32] and BitNet 1.58b [33]. Using tricks like large learning rates and straight-through estimators, successful training of LLMs with binary or ternary parameters has been achieved on mass-scale text corpora, reaching performance levels comparable to the original full-precision models.

It is evident that under the 2-bit bit-width limitation, extreme quantization efforts invariably converge on ternarized parameter distributions. The aforementioned research also demonstrates that ternary parameters, coupled with specialized fine-tuning tricks, can achieve capabilities comparable to full-precision models, and this generalizability holds across various model architectures. However, ternary parameters face significant limitations from a hardware storage perspective. Binary storage hardware inherently requires ternary data to occupy a 2-bit storage width. From an information-theoretic standpoint, the effective storage density is constrained to $\left(\frac{3}{4}\right)^{b/2}$ (where b denotes the total bit-width). As data volume increases, this leads to significant wastage of representational capacity.

Recently, ultra-low-bit quantization via Vector Post-Training Quantization (VPTQ) [45] has gained significant traction in the Large Language Model (LLM) space. Vector quantization (VQ), similar to lookup table quantization, maps high-dimensional vectors to a set of predefined low-dimensional vectors, which are pre-stored in a codebook (lookup table). During encoding, each data point is represented by the index of its corresponding codebook vector; during decoding, these indices approximate the original data. This reduces storage requirements and enables fast vector reconstruction via simple indexing, thus achieving equivalent 2-bit weight compression. However, during inference, it not only requires looking up and reconstructing weight vectors from the codebook but also demands high precision for both activations and reconstructed weights—resulting in continued complexity for hardware computation circuit design. Notably, vector quantization methods like VPTQ are not in the same category as our scalar-wise approach.

III. The Format of 2-Bits Floating-Point

We propose a generic 2-bit floating point format (FP2) that can further improve the hardware efficiency of deep learning, including storage and computational efficiency.

Typically, floating-point formats smaller than 3 bits are considered infeasible due to the basic structure limitation. A floating-point number consists of three components: sign, exponent, and mantissa, each requiring at least 1 bit. Eliminating the exponent bit would effectively convert the floating-point format into a fixed-point representation, thus it no longer makes sense as a floating-point. Conversely, removing the mantissa bit would leave the format capable of representing only 0 and *Inf*. Since floating-point numbers inherently represent both +0 and -0, even when using the shared scale X to encode *Inf*, such a floating-point number can only represent three values: -2, 0, and 2. We propose a data format where two adjacent numbers share a set of sign bits, exponent bits or mantissa bits, along with an additional 1 bit specifically allocated to indicate whether a floating-point number is zero. The proposed data format not only represents a wider range of numbers but is also symmetric about zero. Given that most neural network parameters follow a Gaussian distribution, this format is better suited for representing the weights of neural networks.

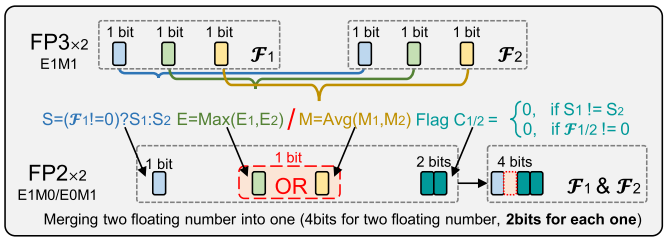


Fig. 2. The definition of the FP2 format. Two adjacent numbers in each block are grouped, and each number in the group shares 1 bit of sign and 1 bit of exponent or mantissa. FP2 is classified as FP2-E0M1 and FP2-E1M0 depending on whether the preserved 1 bit is used for the exponent or mantissa.

Figure 2 illustrates the definition of the FP2 format. Specifically, 32 original floating-point numbers are grouped into a block, with each block sharing an 8-bit shared scale X. Within each block, two adjacent floating-point numbers form a pair, and each floating-point number within the pair shares 1 bit for the sign, 1 bit for either the exponent or the mantissa, and 2 bits for a flag C. The flag C indicates whether the original number is zero, enabling the 1-bit exponent or mantissa to represent valid values and enhance efficiency. Depending on whether the retained bit is for the exponent or the mantissa, the FP2 format is classified into FP2-E0M1 and FP2-E1M0. Since 2 adjacent floating-point numbers share a 4-bit encoding space, the equivalent encoding of an average floating-point number is 2 bits. Our data format reduces storage overhead by 92.97% compared to FP32 and by 47.06% compared to FP4.

The detailed encoding scheme is summarized in Table I.

The encoding formulas for FP2-E1M0 and FP2-E0M1 are given as Equation 4 and 5, where V represents the decimal value, E the exponent, M the mantissa, and X the shared scale.

$$V_{FP2-E1M0} = (-1)^S \times 2^{(X-E)} \tag{4}$$

$$V_{FP2-E0M1} = (-1)^S \times 1.M \times 2^X \tag{5}$$

Furthermore, regarding irregular numbers such as NaN/Inf, we encode the entire block as NaN/Inf on a block-wise basis. The specific encoding is shown in the equation:

$$\begin{aligned} V_{NaN} &: \{X : [1111\ 1111]\ \{P_i\}_{i=1}^{n} : [Not\ Zero]\} \\ V_{Inf} &: \{X : [1111\ 1111]\ \{P_i\}_{i=1}^{n} : [Zero]\} \end{aligned} \tag{6}$$

Figure 3(a) shows the numerical range of FP2. Compared to integer data formats, FP2 supports the representation of both integers and fractional values. By leveraging the shared scale X, FP2 exhibits superior expression capability compared to integer formats. Furthermore, FP2-E1M0 and FP2-E0M1 exhibit distinct distributions. FP2-E1M0 follows an exponential distribution, while FP2-E0M1 performs interpolation within powers of 2, enabling it to represent the original data more accurately. Figure 3(b) illustrates the range of FP2 values within a block. Since each block shares the same scale X, each FP2 encoding can represent five distinct values. Since neural network weights typically follow the normal distribution, most values are concentrated near zero. As a result, FP2-E1M0 demonstrates strong representational capability for weights. In contrast, the FP2-E0M1 format has a lower exponent truncation threshold, thus Dropouting more parameters to

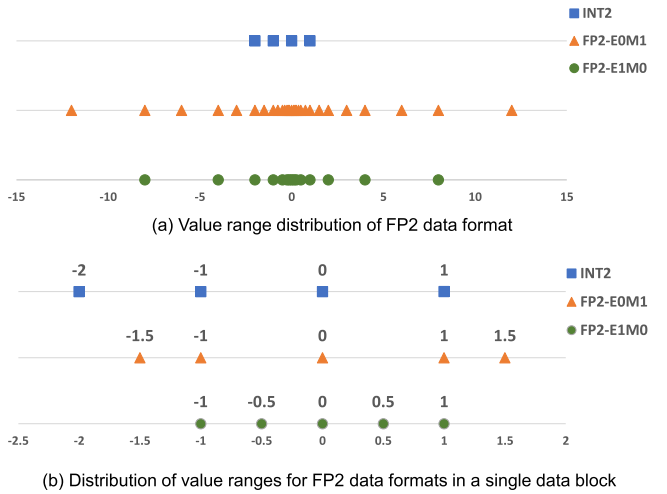


Fig. 3. The data range of FP2 format.

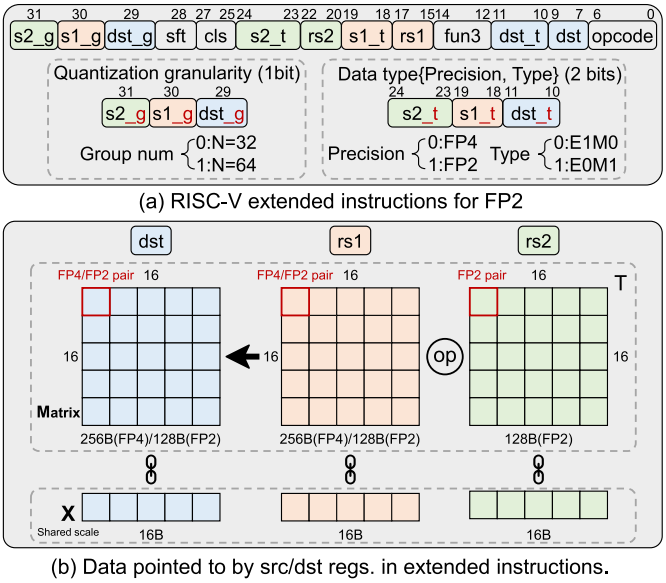


Fig. 4. RISC-V extended instructions for FP2.

zero—resulting in sparser pruned data while enabling more accurate representation of outliers. Typically, this introduces more errors, leading to degraded model performance, but it provides an additional quantization option for representing irregularly distributed data in specific network layers.

Moreover, as we pointed out in Section II-C, traditional fixed-point schemes (e.g., APoT, BitNet b1.58) under a 2-bit bit-width constraint exhibit a representable range of $\{0, \pm a\}$ (where a denotes the quantization interval boundary). Since the scaling factor *scale* can be stored independently, this representation is fundamentally equivalent to $scale \times \{0, \pm 1\}$ (ternary representation).

From a mathematical perspective, our proposed FP2 format provides the following quantization steps at 2-bit storage density: FP2-E1M0($scale \times \{0, \pm 0.5, \pm 1\}$), FP2-E0M1($scale \times \{0, \pm 1, \pm 1.5\}$).

By fully utilizing the encoding space, our FP2 data format mathematically constitutes a superset of ternary representation. Therefore, under identical conditions, FP2 theoretically possesses superior representational capacity compared to ternary networks.

IV. FP2-Based DNN Computing Architecture

DNN accelerators need to handle various operations across different layers, with computation—especially matrix multiplication—being the most critical for performance. This chapter introduces a neural network compute unit designed specifically for FP2 data. By analyzing the multiplication operation of FP2 data, we propose the following optimizations to enhance the compute architecture:

- Separate the storage of the shared scale X from the data itself, implementing an independent compute channel for X in hardware. This improves both data locality and compute efficiency.
- Optimize the multiplication-accumulation (MAC) process for FP4 and FP2 formats by replacing multiplication operations with bit-wise additions.
- Propose a method to directly compute the FP2 format by using the flag C to decode the multiplication factor.
- Reuse components to support computations for both FP2 formats (E0M1 and E1M0), to further reducing hardware overhead.

Subsequent sections will detail these optimizations.

A. RISC-V Extended Instructions for FP2

To achieve compatibility between the FP2 format and RISC-V processors, we have extended the RISC-V instruction set for the FP2 format. The key to compatibility between the FP2 format and the RISC-V instruction set is enabling the computing unit to perceive the data format of inputs and outputs. We transmit data precision (FP4/FP2) and encoding methods (FP2-E0M1/E1M0) to the computing unit through additional register fields.

In traditional RISC-V instruction sets, data pointed to by sources and destinations are 1Byte, 2Byte, 4Byte, and 8Byte. In contrast, the FP2 structure adopted in this paper contains a data pair of only 4 bits; thus, we do not use the SISD representation. To improve instruction efficiency, our instructions adopt a SIMD style. Source and destination registers represent a matrix with 16 rows and 16 columns of FP2 pairs. For an FP2 matrix, its size is 128B, and X is 16B. This requires the computing microarchitecture to read 128+16Bytes of feature maps and weight data of the same size per cycle when accessing memory. When the input feature map uses the FP4 format, the required data bandwidth is 256B. The data specified by sources and destinations during instruction execution of matrix multiplication is shown in Figure 4(b).

The encoding format of the extended RISC-V instructions is shown in Figure 4(a). To adapt to our proposed FP2 data format and related hardware architecture, we appended two additional fields, t and g, after each input/output register specification in the instruction. The field t has a 2-bit encoding space, representing the data precision (4-bit/2-bit) and encoding type (FP2-E0M1/E1M0) of the input/output data, respectively. The field g can be used to adjust the quantization granularity of input/output data; common group numbers are 32 and 64, and we use 32 by default in our work. Since our data types can be configured to hardware along with instructions, we can achieve flexible data configuration at the instruction granularity.

B. The Neural Network Accelerator

Figure 5 presents the architecture of a standard neural network accelerator. The Processing Element (PE) array features a 16×16 grid structure, with each PE containing 16

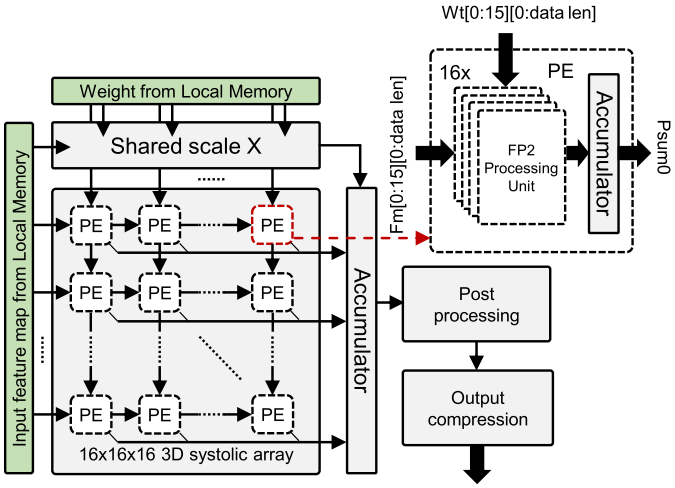


Fig. 5. The architecture of DNN accelerator using FP2 format.

FP2 computing units that handle partial sum(Psum) calculations for multiply-accumulate (MAC) operations. This forms a 16 × 16 × 16.3D systolic computing array, enabling the processing of convolutional Psums for 256 input feature map blocks and 256 weight blocks per cycle.

One block comprises 32 raw data points, each with a shared scaling factor X—the maximum exponent in the block. We set 32 data points per block, drawing on the MXFP data format proposed by Microsoft and NVIDIA [27], which strikes a good balance between model accuracy and hardware resources. We decouple scaling factor computation from inter-data calculations: activation and weight data are processed in the MAC array, whereas the scaling factor is computed via a dedicated bypass. The processing granularity of the 16 PE arrays is fully aligned with the block structure, meaning multiple X values (each tied to a data block) coexist in the MAC array. These are managed systematically via the 'block-PE binding' mechanism, detailed below:

For data transmission, the PE sub-array—comprising 16 FP2 Processing Units per cycle—takes inputs of Fm[0:15][0:data len] and Wt[0:15][0:data len]. Here, [0:15] corresponds to the 16 FP2 Processing Units, and aligns with the structure where 32 data points in each FP2 data block undergo 2-to-1 compression into 16 pieces of 4-bit data. [0:data len] denotes the input data bit-width of each Processing Unit: it ranges from [0:3] for the FP2 format, whereas for feature map data in FP4 format, it ranges from [0:7].

The entire PE array processes 16 × 16 input feature map blocks and 16 × 16 weight blocks per cycle. Since each FP2 block maps to one X, 256 blocks accordingly map to 256 X values—requiring 256 Xs to be processed per cycle. For Psums within an activation block and a weight block, activation-weight multiplication occurs first. Since X is a shared exponent, the X values of the activation and weight must be summed. During intra-block data accumulation, the shared exponent X obviates the need to normalize accumulated data. Consequently, for one feature map block and one weight block, their corresponding X values only require addition during intra-block multiply-accumulate (MAC) operations—hardware-wise, this equates to 16 FP2 PE sub-arrays tied to a single 8-bit fixed-point addition bypass. Thus, X introduces minimal extra overhead in the spatial MAC section and no additional latency.

For storage, as X is block-level shared data (one 8-bit X bound to every 32 data points), the proportion of its additional data volume is far lower than that of the main activations and weights data. Specifically, the proportion of X in the FP2 data format is quantified as: $8/(2 \times 32 + 8) \approx 11.1\%$. Since weight compression can be executed offline and FP4×FP2 computation results are stored in FP4 format without compression, only FP2×FP2 computation results require online compression before the final results are sent back to on-chip SRAM.

C. The Temporal Accumulation Module

The Temporal Accumulation Module is a critical component in DNN accelerators, responsible for accumulating Psums generated by the spatial multiply-accumulate (MAC) module cycle-by-cycle to yield the final convolution results. Its core design addresses the issue that the three steps of traditional floating-point addition—exponent alignment, fixed-point mantissa addition, and exponent normalization—cannot be completed in a single cycle, causing accumulation pipeline stalls, while ensuring no precision loss during accumulation. Drawing on our prior work [26], our design fully reuses FPGA DSP48E2 resources. Leveraging the DSP's inherent features—17-bit fixed right-shift output circuitry, configurable arbitrary-bit left-shift for the 27 × 18 multiplier, and single-cycle 48-bit accumulation—it enables parallel processing of the fractional and exponent parts of floating-point data. Notably, the 'exponent' in the FP2 format temporal accumulation process refers to the Psum's scaling factor X. The module operates in three steps:

- Initialization: The module initializes the fractional part and scaling factor X of the internally buffered accumulated value (inner) to either 0 or Bias.
- Dynamic alignment: Calculate the difference S between inner.X and the input Psum.X. Based on S's magnitude and sign, determine whether to invoke the DSP's fixed 17-bit right-shift circuitry. For example, if S is between 0 and 17, keep inner.X unchanged and only shift the Psum's fractional part for alignment, avoiding complex bidirectional adjustments.
- Non-blocking accumulation: Since inner.X can be computed in a single cycle without waiting for the previous pipeline stage's result, the module achieves non-blocking back-to-back accumulation. This effectively breaks the efficiency bottleneck of traditional floating-point accumulation, ensuring the continuity and high efficiency of the accelerator's overall computation.

Subsequently, these results are sent to the post-processing module for further operations such as activation and pooling.

D. Design of FP4×FP2 Processing Unit

One common application scenario for the FP2 format is to compress weight parameters using the FP2 format while employing the standard FP4 format (which consists of 2 bits for the exponent and 1 bit for the mantissa) for activations [27]. This approach offers two main advantages: first, the use of the FP2 data format can greatly alleviate storage constraints during model deployment, and second, all data compression can be performed offline. However, the FP4×FP2 computation unit requires the additional decoding of the FP2 format. Considering that adjacent weights share storage in the FP2 encoding, we optimized the multiplication computation process between FP4 and FP2 at the granularity of adjacent data by referring to the Mitchell approximation principle [46].

Taking four inputs $\mathcal{F}_1$, $\mathcal{F}_2$, W_1 and W_2 as an example, where $\mathcal{F}_1$ and $\mathcal{F}_2$ are in FP4 format, and W_1 and W_2 in FP2 format

share the same exponent or mantissa, ensuring $|W_1| = |W_2|$. By decoding the sign S and flag C of W into the original weight's sign, the multiply-accumulate operation simplifies to:

$$\begin{aligned} P_{sum} &= \mathcal{F}_1 \times W_1 + \mathcal{F}_2 \times W_2 \\ &= (\mathcal{F}_1 \pm \mathcal{F}_2) \times W \end{aligned} \tag{7}$$

For weights encoded in the FP2-E1M0 format, the multiplication simplifies to bit-wise addition as Equation 8. For weights in the FP2-E0M1 format, depending on the mantissa value M, the multiplication further simplifies as Equation 9

$$\mathcal{F} \times W = \mathcal{F} \times 2^E = \mathcal{F} + 2'bE0 \tag{8}$$

$$\mathcal{F} \times W = \mathcal{F} \times 1.M = \mathcal{F} + 2'b0M \tag{9}$$

Therefore multiplication can be replaced by bit-wise addition:

$$P_{sum} = BitAdd((\mathcal{F}_1 \pm \mathcal{F}_2), W) \tag{10}$$

Through the aforementioned simplifications, the computational workload can be reduced by approximately half.

To correct the error in the approximate implementation of FP4×FP2 multiplication via bitwise addition, we traversed all multiplication results and their corresponding addition results. The findings show that a discrepancy between the multiplication result and the addition result only occurs when the mantissas (M) of both FP4 and FP2 are 1. When M=1, the fractional part is 1.5. In this case, the actual multiplication result is 2.25, while the result approximated by bitwise addition is 2. Thus, we correct this approximation error by introducing an additional bit ($M1$&$M2$).

E. Design of FP2×FP2 *Processing Unit*

For FP2 multiplication, our approach enables computation directly in the FP2 format by decoding the flag bit C of $\mathcal{F}$ and W into a multiplication factor K. For the multiplication between input activation $\mathcal{F}$ and weight W, the flag bit C is first used to decode $\mathcal{F}$ and W into their original floating-point representations $\mathcal{F}_1, \mathcal{F}_2, W_1, W_2$. The four values are then used to compute their multiply-accumulate operations.

As shown in Table II, the decoded floating-point values based on C have only three possible states: 0, $\mathcal{F}$, and $-\mathcal{F}$. From all possible decompression cases, the resulting product-sum operation for a set of FP2 data can be expressed in the form of $K \times (\mathcal{F} \times W)$, where K is referred to as the multiplication factor.

- When $P_{\text{sum}} = 0$, the multiplication factor K is set to 0, and no further multiplication is required.
- When $P_{\text{sum}} = -(\mathcal{F} \times W)$, K is set to -1, and the sign bit of the multiplication result is inverted.
- When $P_{\text{sum}} = 2 \times (\mathcal{F} \times W)$, K is set to 2, and the multiplication result is shifted left by one bit.
- For all other cases, K is set to 1, and the multiplication result directly serves as the final output.

By converting the multiply-accumulate operations in floating-point format into multiplication factor and operations in the FP2 format, we optimized the FP2 computation units, reducing the computational workload by approximately one-quarter. Compared to traditional data compression methods, the FP2 format eliminates the need for decompression, reducing on-chip storage space by half. Additionally, it saves hardware area by avoiding the introduction of complex decompression logic.

TABLE I

THE ENCODING METHOD OF THE FP2 FORMAT

Number		**S**	**E**	**M**	**C**
$\mathcal{F}_1$		S_1	E_1	M_1	None
$\mathcal{F}_2$		S_2	E_2	M_2	None
Condition of judgment		S	E	M	C
FP2-E0M1 ($S_1 = S_2$)	$\mathcal{F}_1 \neq 0$ & $\mathcal{F}_2 \neq 0$	S_1	None	M_{avg}	11
	$\mathcal{F}_1 \neq 0$ & $\mathcal{F}_2 = 0$	S_1	None	M_1	10
	$\mathcal{F}_1 = 0$ & $\mathcal{F}_2 \neq 0$	S_2	None	M_2	01
FP2-E1M0 ($S_1 = S_2$)	$\mathcal{F}_1 \neq 0$ & $\mathcal{F}_2 \neq 0$	S_1	E_{max}	None	11
	$\mathcal{F}_1 \neq 0$ & $\mathcal{F}_2 = 0$	S_1	E_1	None	10
	$\mathcal{F}_1 = 0$ & $\mathcal{F}_2 \neq 0$	S_2	E_2	None	01
$S_1 \neq S_2$, $\mathcal{F}_1 \neq 0$ & $\mathcal{F}_2 \neq 0$		Same as above.			00
$\mathcal{F}_1 = 0$ & $\mathcal{F}_2 = 0$		0	0	0	00

TABLE II

DECODING THE MULTIPLICATION FACTOR

K	$\mathcal{F}$	W	Psum	Out
	$C_{\mathcal{F}}$	C_W	$\mathcal{F}_1 \times W_1 + \mathcal{F}_2 \times W_2$	
0	00	11	$\mathcal{F} \times W + (-\mathcal{F}) \times W$	0
	01	10	$0 \times W + \mathcal{F} \times 0$	
	10	01	$\mathcal{F} \times 0 + 0 \times W$	
	11	00	$\mathcal{F} \times W + \mathcal{F} \times (-W)$	
-1	00	01	$\mathcal{F} \times 0 + (-\mathcal{F}) \times W$	$-\mathcal{F} \times W$
	01	00	$0 \times W + \mathcal{F} \times (-W)$	
2	00	00	$\mathcal{F} \times W + \mathcal{F} \times W$	$2 \times \mathcal{F} \times W$
	11	11	$\mathcal{F} \times W + (-\mathcal{F}) \times (-W)$	
1	Other cases		$\mathcal{F} \times W$	$\mathcal{F} \times W$

TABLE III

THE TRUTH TABLE OF FP2 MULTIPLICATION

	Input		Calculate	Out			
	$M_{\mathcal{F}}$	M_W		C	M_1	M_2	M_3
FP2 E0M1	0	0	$1.0 \times 1.0 = 01.00$	0	1	0	0
	0	1	$1.0 \times 1.1 = 01.10$	0	1	1	0
	1	0	$1.1 \times 1.0 = 01.10$	0	1	1	0
	1	1	$1.1 \times 1.1 = 10.01$	1	0	0	1
	$E_{\mathcal{F}}$	E_W			M_1	M_2	M_3
FP2 E1M0	0	0	$1.0 \times 1.0 = 1.00$		1	0	0
	0	1	$1.0 \times 0.1 = 0.10$		0	1	0
	1	0	$0.1 \times 1.0 = 0.10$		0	1	0
	1	1	$0.1 \times 0.1 = 0.01$		0	0	1

F. Design of FP2×FP2 *Multiplication Circuit*

To accommodate different data encoding formats in various layers of neural networks, our design ensures hardware compatibility with both two FP2 formats. For the multiplication between FP2-E1M0 and FP2-E0M1, the hardware implementation is very simple, requiring only one 1-bit right shift of the FP2-E1M0 data.For FP2-E0M1 multiplications, the maximum output width is 4 bits, consisting of 1 carry bit and 3 mantissa bits, with the decimal point between M_1 and M_2. For FP2-E1M0, the output width is 3 bits, entirely in the mantissa. As shown in Table III, we derive the input-output relationship for both formats based on these properties. According to Table III, we express the direct relationship between the output and input of the two FP2 formats under 2 bits by Equation 11 and 12.

$$FP2_{E0M1} \begin{cases} S = S_{\mathcal{F}} \oplus S_W \\ C = M_{\mathcal{F}} \wedge M_W \\ M_1 = \neg(M_{\mathcal{F}} \wedge M_W) \\ M_2 = M_{\mathcal{F}} \oplus M_W \\ M_3 = M_{\mathcal{F}} \wedge M_W \end{cases} \tag{11}$$

$$FP2_{E1M0} \begin{cases} S = S_{\mathcal{F}} \oplus S_W \\ M_1 = \neg(E_{\mathcal{F}} \parallel E_W) \\ M_2 = E_{\mathcal{F}} \oplus E_W \\ M_3 = E_{\mathcal{F}} \wedge E_W \end{cases} \tag{12}$$

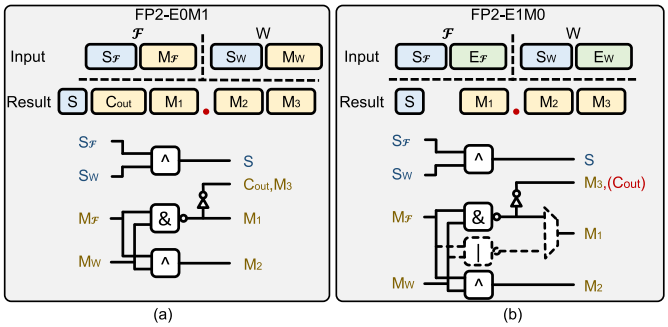


Fig. 6. The multiplication logical circuit of FP2.

TABLE IV
THE STATISTICS OF THE E-DIFF OF THE ACTIVATION AND WEIGHT IN RESNET18

E-diff	Pretrain Model		Fine-tuning Model	
	Weight	Activation	Weight	Activation
0∼1(FP2-E0M1)	76%	95%	78%	95%
0∼2(FP2-E1M0)	87%	97%	89%	97%
>2(Dropout)	13%	2%	11%	2%

In the FP2-E0M1 encoding, the logic circuits for C and M_3 are identical and opposite to that of M_1. Therefore, as shown in Figure 6(a), we implement the logic circuits for C, M_3, and M_1 by reusing the NAND gates. In the FP2-E1M0 encoding, the logic circuits for outputs S, M_2, and M_3 are exactly the same as the corresponding circuits in the FP2-E0M1 encoding. As a result, by building upon the multiplication circuit of the FP2-E0M1 encoding, the multiplication for the FP2-E1M0 encoding can be achieved with the addition of just one NAND gate and a two-way MUX, as illustrated in Figure 6(b). Compared to the implementation of separate circuits for different formats, our approach achieves compatibility for both FP2 encodings by reusing the components, thus reducing hardware resource overhead.

V. ERROR ANALYSIS

In deep learning applications, the FP2 data format enhances hardware efficiency through an innovative encoding scheme, yet it inevitably introduces errors. These errors arise from diverse sources and have a complex impact on model accuracy and performance. The following provides an in-depth analysis of the rounding error, quantization error, and mean-value error.

A. Rounding Error

In the FP2 format, when processing two adjacent nonzero floating-point numbers, $\mathcal{F}_1$ and $\mathcal{F}_2$, the larger exponent, $E_{\max}$, is chosen as the shared exponent. This means that the number with the smaller exponent must be adjusted during representation, which introduces a rounding error. As shown in Table IV, we evaluate this error by statistically analyzing the exponent difference (E-diff) between adjacent floating-point numbers in ResNet-18 model parameters, with the experimental setup of 2-bit activations, 2-bit weights and the CIFAR-10 dataset.

Due to the different expressive capabilities of the FP2 encoding schemes, when E-diff exceeds 1, the number with the smaller exponent is completely truncated and becomes unrepresentable under the FP2-E0M1 encoding. In contrast, under the FP2-E1M0 encoding, the number with the smaller exponent is entirely truncated when E-diff exceeds 2.

As illustrated in Table IV, in the pre-trained ResNet18 model, approximately 76% of the weight parameters and 95% of the activation values are representable using the FP2-E0M1 encoding, while about 87% of the weight parameters and 97% of the activations are representable using the FP2-E1M0 encoding. We also analyzed the parameter representation after fine-tuning. Compared to the pre-trained model, the fine-tuned model exhibits an approximately 2% increase in the number of representable weight parameters. Moreover, some studies have demonstrated that the accuracy loss caused by pruning 25% of the weight parameters in DNN models can be recovered through fine-tuning [47].

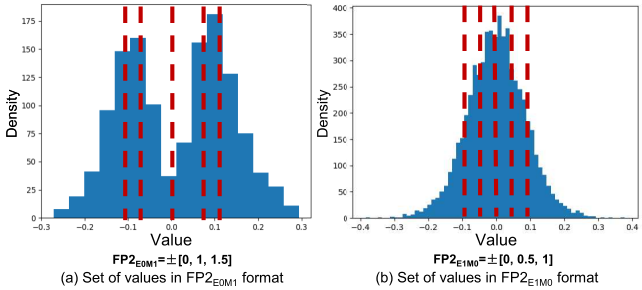


Fig. 7. Adaptation relationship between different floating-point data distributions and FP2 encoding: Showing the scenario of FP2-E0M1 for representing high outliers in bimodal distributions (scenario without truncation when E-diff ≤ 1), and the low quantization error characteristic of FP2-E1M0 for bell-shaped distributions (no truncation when E-diff ≤ 2); different encoding methods match the E-diff truncation rules and representability requirements under different distributions.

B. Quantization Error

Quantization error is primarily introduced by differences in numerical range and precision. In contrast to the commonly used full-precision format, FP32, whose binary representation comprises 1 sign bit, 8 exponent bits, and 23 mantissa bits—thus offering an extremely wide numerical range and high precision—the FP2 format uses only 2 bits to represent a floating-point number, resulting in a very limited representable range and lower precision. Some studies have shown that quantization error is affected by both the distribution of the original data and that of the quantized data [48], [49]. When comparing the distributions of full-precision data and FP2-quantized data, the full-precision distribution accurately reflects the characteristics of the original data, whereas the limited quantization levels in FP2 lead to changes in the distribution. Specifically, an originally continuous data distribution may become aggregated or discretized after quantization.

When representing the weights of a neural network, full-precision weights often exhibit a smooth distribution that reflects the model's varied focus on different features. However, after quantization to the FP2 format, some weight values may be merged into the same quantization level, resulting in a discrete distribution that fails to accurately capture the model's ability to differentiate between features.

We statistically analyzed the distribution of weight parameters, and common parameter distributions conform to either a bell-shaped (Gaussian) distribution or a bimodal distribution, as shown in Figure 7. For example, as illustrated in Figure 7(a), our proposed FP2-E1M0 data format, which exhibits a uniform distribution, results in a smaller quantization error for bell-shaped distributions centered around zero, whereas for certain distributions—such as bimodal distributions—the quantization error is larger. Conversely, as shown in Figure 7(b), the FP2-E0M1 encoding exhibits a smaller quantization error for bimodal distributions, but incurs a larger quantization error

for bell-shaped distributions. To mitigate quantization error, we employ different FP2 encoding schemes based on the data distributions of various layers in the DNN.

C. Mean-Value Error

The FP2-E0M1 encoding format additionally introduces the averaging of the mantissa part M. Replacing each individual mantissa with the average of the two mantissas for computation obviously introduces calculation errors. Therefore, we have quantitatively analyzed this error. Taking $\mathcal{F}_1$, $\mathcal{F}_2$, W_1, and W_2 as examples, the error introduced by performing multiplication using the averaged mantissas can be expressed by the following Equation 13.

$$\begin{aligned} Err &= (\mathcal{F}_1 W_1 + \mathcal{F}_2 W_2) - 2\left(\frac{\mathcal{F}_1 + \mathcal{F}_2}{2}\right)\left(\frac{W_1 + W_2}{2}\right) \\ &= \frac{(\mathcal{F}_1 - \mathcal{F}_2)(W_1 - W_2)}{2} \end{aligned} \tag{13}$$

The maximum relative error can be expressed by the following Equation 14.

$$Err = \frac{(\mathcal{F}_1 - \mathcal{F}_2)(W_1 - W_2)}{2(\mathcal{F}_1 W_1 + \mathcal{F}_2 W_2)} \tag{14}$$

Since $\mathcal{F}_1, \mathcal{F}_2, W_1, W_2$ are compressed using the FP2-E0M1 format, we have $\mathcal{F}_1 = M_{\mathcal{F}_1}$, $\mathcal{F}_2 = M_{\mathcal{F}_2}$, $W_1 = M_{W_1}$ and $W_2 = M_{W_2}$. Therefore, the above expression can be rewritten as Equation 15.

$$Err = \frac{(M_{\mathcal{F}_1} - M_{\mathcal{F}_2})(M_{W_1} - M_{W_2})}{2(M_{\mathcal{F}_1} M_{W_1} + M_{\mathcal{F}_2} M_{W_2})} \tag{15}$$

Since $M \in [1, 2)$, the above expression attains a maximum relative error of 10% when $M_{\mathcal{F}_1} \approx 2$, $M_{\mathcal{F}_2} = 1$, $M_{W_1} \approx 2$, and $M_{W_2} = 1$. Because the maximum value of M is given by $2 - \frac{1}{2^{b_M}}$, where b_M denotes the bit-width of the mantissa, the achievable maximum relative error is related to the bit-width of M. Specifically, when M is represented with 0, 1, 2, and 3 bits, the maximum relative errors are 3.8%, 7%, 8.5%, and 9.2%, respectively, and as the bit-width of M increases, the error approaches 10%. Considering that reducing the bit-width of the fractional part leads to a smaller error from mantissa averaging, this is highly favorable for our application scenarios involving extremely low-bit-width floating-point numbers.

VI. Evaluation

A. Model Evaluation

We compared FP2 format against FP4, FP8 [6], ternary weight networks (TWN) [29], binary weight networks (BWN) [28], and full-precision networks (FWN) on classification tasks using the CIFAR-10 and ImageNet-1K datasets.The encoding scheme for the FP8 format is E4M3, and the FP4 data format uses an encoding comprising 2 exponent bits and 1 mantissa bit, with 32 numbers grouped together to share a common scaling factor.

The work on APoT [31] demonstrates that, by employing additional fine-tuning optimization tricks such as learning the upper/lower bounds of quantization clipping ranges during fine-tuning and applying weight normalization, ternary quantized models can achieve accuracy comparable to full-precision models. Since our proposed FP2 scheme is a superset of quantization for ternary neural networks, these fine-tuning tricks could theoretically be transferred to FP2. However, they were not incorporated into our work. Instead, we chose TWN (Ternary Weight Networks) as the evaluation baseline because our research primarily focuses on the data format itself, rather than the design of fine-tuning tricks.

TABLE V
ACCURACY(%) OF CIFAR10-RESNET18/50

	ResNet18	ResNet50
FWN($FP32$)	95.00	95.10
BWN($A32W1$)	91.73	/
TWN($A32W2$)	92.56	/
FP4×FP4($A4W4$)	94.32	94.74
FP4×FP2($A4W2$)	**94.50**	**94.46**
FP2×FP2($A2W2$)	**92.71**	**92.30**

[1] Fine-Tuning epochs: 10, Learning rate: 10^{-3}.
[2] The first and last layers of the network were not compressed.

For the model fine-tuning process, we followed the most recent MXFP work published by Microsoft [27]. Prior to the input of each convolutional layer during the forward pass of neural network fine-tuning, both input activations and weights are converted to the corresponding formats. In that work, to avoid gradient overflow and underflow, the backpropagation process employs the BF16 data format and adopts a straight-through strategy, thereby not propagating gradients from the data compression process [24]. We also use the BF16 data format during the backpropagation phase of model fine-tuning. Our learning rate was set to 10^{-3}, and fine-tuning was conducted for approximately 10 epochs.

For simplified datasets such as CIFAR-10, we evaluated fine-tuning using 2-bit activations and weights. Numerous studies have shown that in deep learning, the activation bit-width needs to be greater than that of the weights [44]. This is because excessive compression of high-resolution image inputs (e.g., ImageNet) leads to overly sparse activations and significant loss of image information. Therefore, in this benchmark, we employed FP4 for activations and FP2 for weights (A4W2). In addition, in line with many works in the quantization domain, we did not compress the first and last layers of the network. Compressing the first layer typically results in overly sparse representations that severely degrade network performance, while compressing the last layer, which contains relatively few parameters, often leads to unacceptable accuracy loss.

Table V presents the classification results on the CIFAR-10 dataset, where we evaluate ResNet18 and ResNet50 models. The baseline for comparison is the Full-Precision Weight Network (FWN). Binary Weight Networks (BWN) are also tested, which use 32-bit full-precision activations while restricting weights to 1-bit values represented by a sign function (+1 or -1). Additionally, we test Ternary Weight Networks (TWN), which also employ 32-bit full-precision activations but represent weights using 2-bit values to encode +1, -1, and 0.

When using FP2 to compress both activations and weights, the results show that FP2 outperforms BWN and TWN. The performance gap between FP2 and TWN is smaller than the gap between FP2 and BWN. However, FP2 demonstrate a noticeable accuracy gap compared to FWN. We attribute this discrepancy to the significant loss of activation information when compressed to 2 bits. To address this limitation, we modify the activations to FP4 while keeping weights in FP2 format for benchmarking. Experimental results indicate that with FP4×FP2, the FP2 model achieves performance comparable to the full-precision model.

TABLE VI

ACCURACY(%) OF IMAGENET-1K RESNET18/50

	ResNet18	ResNet50	ConvNeXt-Tiny
FWN($FP32$)	69.76	77.40	82.52
BWN($A32W1$)	60.80	/	/
TWN($A32W2$)	61.80	/	/
FP4×FP4($A4W4$)	67.19	74.86	79.87
FP4×FP2($A4W2$)	**65.58**	**74.28**	**78.32**

[1] Fine-Tuning epochs: 10, Learning rate: 10^{-3}.
[2] The first and last layers of the network were not compressed.

TABLE VII

ZERO-SHOT RESULTS FOR FP2 FORMAT AND THE BASELINES

Models	Methods	Wbits	Param	Benchmark Avg↑
LLaMA 1B	BF16	16	1x	47.3
	GPTQ	2	8x	40.5
	FP2(A4W2)	2	8x	**44.4**
LLaMA 3B	BF16	16	1x	54.1
	GPTQ	2	8x	44.8
	FP2(A4W2)	2	8x	**47.5**

[1] Fine-Tuning epochs: 10, Learning rate: 10^{-3}.
[2] The first and last layers of the network were not compressed.

Table VI summarizes the classification results on the ImageNet-1K benchmark. We fine-tune inference using the FP2 format for 10 epochs on a full-precision model without applying any additional optimization techniques. Compared to CIFAR-10, the accuracy loss of low-precision data formats is more pronounced on large-scale image datasets. However, the performance gap between FP2 and the full-precision model is significantly smaller than that observed with Ternary Weight Networks (TWN) and Binary Weight Networks (BWN). Furthermore, when we replaced ResNet18 with larger-scale networks such as ResNet50 and ConvNeXt-Tiny, the accuracy loss of the FP2 format compared to full-precision models was reduced to approximately 3–4 percentage points.

Table VII provides a detailed comparative analysis of the zero-shot performance of our proposed FP2 format against various baseline methods on three benchmark datasets (i.e., Winogrande, Winograd, and Hellaswag). The current work in this paper focuses on model fine-tuning and inference. For fair comparison, the selected baselines are the original LLAMA model (BF16) and a model quantized via the most common PTQ scheme, GPTQ-2bit. Results show that FP2 demonstrates significant advantages over baseline methods in the comparative evaluation of 2-bit PTQ, with the reported values representing the average performance across the three benchmark datasets.

Experimental results demonstrate that under the same storage density, compared to ternary quantized model parameters, FP2 exhibits superior parameter representational capacity. Specifically, FP2 provides more quantization levels and achieves higher accuracy through pairwise data compression. Regarding CNNs, as the model scale increases, the accuracy gap between FP2 and full-precision networks narrows. This suggests that as neural network models expand, the redundancy in their parameters also increases. By fine-tuning models using the low-bit-width floating-point FP2 format, redundant information can be compressed during fine-tuning, significantly reducing model parameter size while incurring minimal performance loss.

B. Hardware Evaluation

Table IX presents the storage and computational benefits of our proposed architectures compared to various floating-point formats. The proposed architectures compress weights to FP2, achieving an offline storage reduction of 93% compared to FP32 and 47% compared to FP4. In terms of computational efficiency, our architectures reduce computation by a factor of 2× through optimized multiplication in FP4×FP2 and by a factor of 4× through further optimizations in FP2×FP2. The proposed design is modeled in RTL Verilog and synthesized using Xilinx Vivado, with implementation on the Xilinx ZU9 FPGA [50]. The MAC units for FP32, BWN, TWN, and FP8 formats utilize floating-point multipliers and adders based on Xilinx-provided IP cores. Given that FP4 is the most advanced low-bit-width floating-point format in the industry and serves as the most competitive baseline for FP2, we implement a PE Array based on the FP4 format as a baseline to evaluate the hardware resource overhead of our proposed architectures. Table VIII presents the hardware resource overhead of our proposed computational architectures compared to the baseline. As shown, the FP4×FP2 architecture reduces resource usage by 25.3% compared to the FP4 baseline, while the FP2×FP2 architecture achieves a significant 78% reduction in resource usage.

For our ASIC evaluation, we use Synopsys Design Compiler to synthesize the design at a 1GHz frequency under a 28nm

TABLE VIII

FPGA RESOURCE COMPARISON

	PE Array (LUTs)	Res. Comp. (PE Array)	Per PE (LUTs)	Res. Comp. (Per PE)
FWN(A32W32)	8701.4K	43.30×	33.99K	45.62×
BWN(A32W1) TWN(A32W2)	3909K	19.45×	15.27K	20.50×
FP8(A8W8)	176.7K	8.79×	7.17K	9.62×
FP4(A4W4)	201K	1×	745	1×
FP2(A4W2)	150K	0.75×	562	0.75×
FP2(A2W2)	43.8K	0.22×	155	0.21×

[1] Device: Xilinx ZU9 FPGA; Frequency: 200 MHz.
[2] Modeling based on RTL Verilog; synthesis with Xilinx Vivado, referenced to Xilinx ZU9 FPGA technical documentation [50].
[3] 'Res. Comp.' in the table refers to 'Resource Compression Ratio', with FP4(A4W4) as the baseline (set to 1×).

TABLE IX

OVERHEAD OF THE PROPOSED FP2 ARCHITECTURE

	Storage Saving	Operations Used in DNN	Computation Saving
FWN(A32W32)	1	×, +	1
BWN(A32W1)	32×	Nxor, +	∼2×
TWN(A32W2)	16×		
FP8(A8W8)	3.88×	×, +	1
FP4(A4W4)	7.53×	×, +	1
FP2(A4W2)	14.22×	Bit-Add, +	∼2×
FP2(A2W2)	14.22×	Log. Ops, +	∼4×

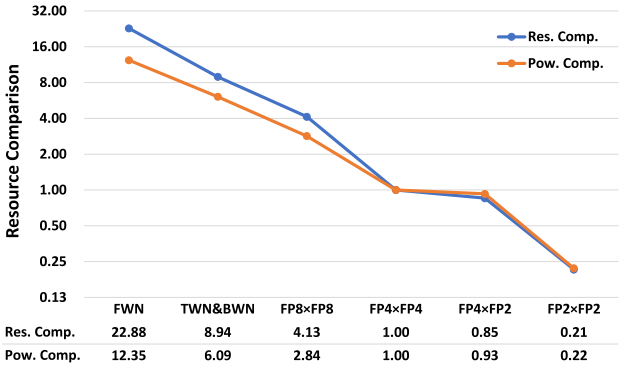


Fig. 8. Comparison of PE area and power for various floating point format architectures.

TABLE X
ASIC RESOURCE COMPARISON

		FWN (A32W32)	BWN(A32W1) TWN(A32W2)	FP8 (A8W8)	FP4 (A4W4)	FP2 (A4W2)	FP2 (A2W2)
Area(μm^2)	PE Array	39.76M	15.53M	7.18M	1.74M	1.48M	0.37M
	Per PE	155.3K	60.3K	28.3K	6.6K	5.7K	1.2K
	MUL	3113	83	493	173	183	7
	ADD	1246	1175	271	48	33	27
	Fixed2FP	NAN	NAN	NAN	189	156	156
	Others	17.1K	21.2K	4157	3073	2238	723
	Res. Comp.	22.88×	8.94×	4.13×	1×	0.85×	0.21×
	TOps/mm^2	0.41	1.05	2.28	9.42	11.07	44.28
Power(W)	PE Array	20.08	9.89	4.63	1.63	1.51	0.36
	Pow. Comp.	12.35×	6.09×	2.84×	1×	0.93×	0.22×
	TOps/W	0.82	1.66	3.54	10.05	10.85	45.51

[1] Process: 28nm CMOS; Frequency: 1GHz; Synthesis Tool: Synopsys Design Compiler; Synthesis Library: DesignWare [51]; Voltage: 0.81V; Temperature:$-40°C$.
[2] The Area and Power consumption data only count the PE array, excluding the control unit and RAM.
[3] 'Res. Comp.' and 'Pow. Comp.' refer to 'Area Compression Ratio' and 'Power Compression Ratio' respectively, both with FP4 (A4W4) as the baseline (set to 1×).

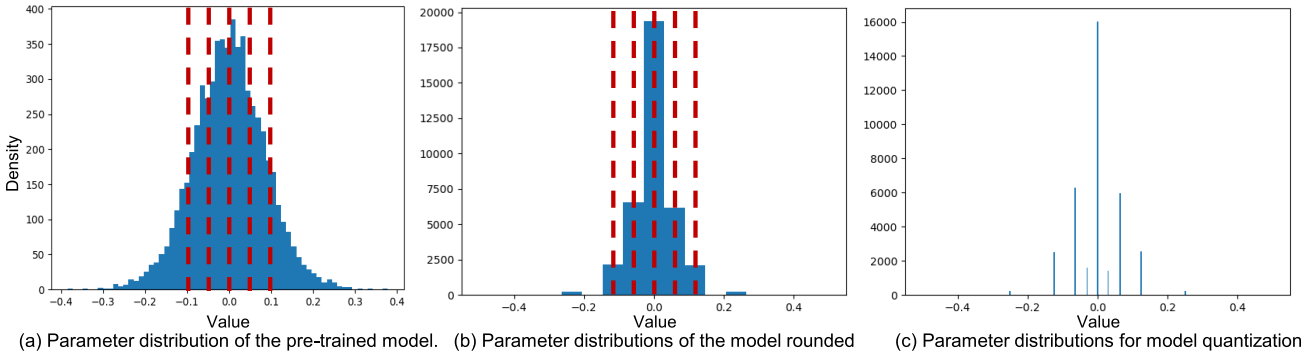
CMOS process, estimating both area and power consumption. The 28nm process represents the best available technology node for our implementation. Additionally, we synthesize computational circuits for FWN, BWN, TWN, FP8, and FP4 as baselines for comparison. The floating-point MAC units in the comparison designs are based on DesignWare [51]. Figure 8 and Table X present the processing element (PE) area across different data formats. Table X details the area breakdown of various components within a single PE unit, which consists of 32 multipliers, 31 adders, and additional logic. A complete PE array contains 16 × 16 PE units. Figure 8 compares the area and power consumption of a full PE array across different data bit-widths, using the FP32 format as the baseline.

For the PE Arrays of BWN, TWN, FP8, and FP4, we use the FP32 PE Array as the baseline. The TWN and BWN formats reduce floating-point multiplication to XOR operations by compressing weight parameters to either two or three discrete values, leading to a 97% reduction in multiplier area. However, since activations remain in the FP32 format, the accumulation circuitry and other control logic do not benefit from the same reduction, resulting in an overall PE area reduction of 61% compared to the baseline. The FP8 format employs a 4-bit exponent and a 3-bit mantissa, leading to an overall bit-width reduction of 4× compared to FP32, with the mantissa width reduced by 6× and the exponent width reduced by 2×. As a result, the multiplier area is reduced by 84%, the adder area by 78%, and the control logic by 76%, leading to a total area reduction of 80%. The compression ratio of the FP8 PE Array area relative to FP32 closely aligns with the bit-width compression ratio of FP8 compared to FP32.

Since the DesignWare floating-point computation library does not support the FP4 format, we implemented the FP4-based MAC unit ourselves. The FP4 format uses a 2-bit exponent and a 1-bit mantissa. Compared to FP32, the FP4 format achieves a 16× compression in the mantissa, a 4× compression in the exponent, and an overall 8× compression in data representation. As shown in Figure 8, the FP4-based multiplier unit achieves a 94% area reduction compared to the FP32 baseline, while the adder unit achieves a 96% reduction. The greater resource reduction in the adder compared to the multiplier is attributed to the narrow exponent width of the FP4 format and the fact that the scaling factor is shared within a PE. This allows floating-point numbers to be expanded into fixed-point values for accumulation within a PE, with exponent normalization only performed at the final stage. Consequently, each floating-point adder is optimized into a fixed-point adder, at the cost of introducing a fixed-to-floating-point conversion unit within each PE. Through these hardware optimizations, the FP4-based PE unit achieves a 95.7% area reduction compared to the FP32 baseline.

Since the PE Array area based on FP32 differs significantly from that of other formats and FP32 is not suitable for efficient model inference, we use FP4 as the baseline for a better evaluation of our proposed FP2-based PE Array. In our proposed FP4×FP2 scheme, compared to the FP4×FP4 scheme, the slightly larger multiplier unit in FP4×FP2 is due to the need to support two different FP2 encoding formats, which requires decoding within the multiplication unit. Despite this, the PE area of the FP4×FP2 architecture is reduced by 15% compared to FP4×FP4. In our proposed FP2×FP2 scheme, we eliminate the need for FP2 decoding by leveraging accumulation coefficients and reuse circuitry to support both FP2 formats. As a result, the optimized FP2 multiplier achieves a 96% area reduction compared to FP4×FP4, while the adder area is reduced by 43%, leading to an overall 82% reduction in PE area. In terms of power consumption, our FP4×FP2 and FP2×FP2 architectures achieve reductions of 8% and 78%, respectively, compared to the FP4 baseline.

By evaluating our proposed design in terms of model accuracy, storage efficiency, PE area, and power consumption, we compare it against the state-of-the-art FP4×FP4 scheme. We believe that our design achieves an optimal balance between model accuracy and hardware efficiency across different scenarios. For complex benchmarks such as ImageNet, our proposed FP4×FP2 scheme achieves a 15% reduction in PE area, an 8% reduction in power consumption, and a 47% reduction in storage requirements, while maintaining model accuracy loss within 1–2 percentage points. For simpler benchmarks such as CIFAR-10, our proposed FP2×FP2 scheme achieves a 78% reduction in both PE area and power consumption, along with a 47% reduction in storage requirements, while keeping model accuracy loss within 1–2 percentage points.



(a) Parameter distribution of the pre-trained model. (b) Parameter distributions of the model rounded according to the E-diff. (c) Parameter distributions for model quantization to FP2-E1M0 with fine-tuning.

Fig. 9. Changes in pre-trained model parameter distributions with error introduction and fine-tuning: (a) Bell-shaped parameter distribution of the pre-trained model (unaffected by E-diff truncation); (b) After introducing FP2-E1M0 rounding error, i.e., truncating small exponent values with E-diff > 2, 87% of the weights can be represented; (c) Structured parameter distribution after fine-tuning, where parameters are constrained to the FP2-E1M0 quantization steps, and 2% of the parameter loss caused by quantization is recovered after fine-tuning.

TABLE XI
IMPACTS OF ERROR TYPES ON RESNET18'S INFERENCE & FINE-TUNING ACCURACY

Gradually introduce Errors		Individual Error		Fine-tuning
Error Type	Inference	Error Type	Inference	
E1	47.84	E1	47.84	69.77
E1+E2(E1M0)	28.02	E2(E1M0)	40.16	65.61
E1+E2(E0M1)	0.31	E2(E0M1)	0.22	59.44
E1+E2+E3	0.17	E3(E0M1)	54.78	58.69

[1] E1: Rounding Error; E2: Quantization Error; E3: Mean-Value Error.
[2] Activation: FP4; Weight: FP2; Model: ResNet18; Benchmark: ImageNet.

C. Error Evaluation

We evaluated the impact of the proposed format errors on model accuracy. Table XI presents the evaluation of three types of errors on model performance, where E1, E2, and E3 in Table XI represent rounding error, quantization error, and mean error, respectively. Additionally, E1M0 and E0M1 denote models quantized to FP2-E1M0 and FP2-E0M1, respectively. The left column of Table XI records the inference accuracy of a pre-trained ResNet18 model (originally in FP32 format) when progressively introducing these errors without fine-tuning. The middle column quantifies the accuracy loss caused by each individual error. To isolate the effect of each error, the model is fine-tuned after introducing a single error before proceeding to introduce additional errors. The right column in Table XI reports the accuracy after stepwise fine-tuning. From Table XI, we observe that rounding error and mean error can be effectively recovered within a few epochs of fine-tuning. The primary factor causing significant accuracy degradation is the loss of information when weights are quantized to extremely low-bit representations. In the following sections, we provide a detailed discussion of the impact of each type of error on model accuracy.

First, we analyze the rounding error. In the pre-trained model, when the exponent difference between two adjacent weight parameters exceeds 1, we discard the smaller parameter while retaining the precision of the remaining weight at 32 bits. This operation effectively acts as an unstructured pruning process with a pruning rate of approximately 25%, leading to an accuracy drop of 22 percentage points. However, after fine-tuning the model with the rounding error introduced, experimental results show that the accuracy loss can be recovered within a few epochs of fine-tuning. As illustrated in Figure 9(a)(b), after fine-tuning, the parameter distribution of the model undergoes a transformation from the bell-shaped distribution in Figure 9(a) to the structured distribution in Figure 9(b). The results indicate that the parameter values are constrained into several distinct quantization levels.

Next, we introduce quantization error on top of the rounding error. We quantize the model weights to the FP2-E1M0 format, retaining only a 1-bit exponent. In this case, direct inference from the full-precision original model results in a drastic accuracy loss of approximately 40 percentage points. When performing inference from the model fine-tuned after introducing rounding error, the accuracy loss is reduced to around 28 percentage points. Fine-tuning the model after quantization yields a parameter distribution as shown in Figure 9(c), and the model accuracy recovers to 65.58. Since the FP2-E1M0 format does not include a mantissa, there is no need for mean-based adjustments, and thus no mean error is introduced. However, when we quantize the model weights to the FP2-E0M1 format, which retains only a 1-bit mantissa, a significant mismatch occurs between the distribution of weight values and the FP2-E0M1 data format, leading to severe quantization error. In this case, the accuracy drops to an unusable level. After fine-tuning, the model accuracy recovers to 59.32.

Finally, we consider mean error. We compute the mean of the mantissa values of two adjacent weights in the FP2-E0M1 format and replace both weight values with this mean. Additionally, Table XI presents the accuracy of direct inference using the FP2-E0M1 format. The results in Table XI indicate that the accumulation of these three types of errors renders direct inference from an untrained quantized model infeasible. When performing direct inference from the model fine-tuned with FP2-E0M1, accuracy drops to 54.78. However, further fine-tuning restores accuracy to 58.69, which is within one percentage point of the quantized model's accuracy of 59.44.

VII. CONCLUSION AND FUTURE WORK

In this paper, we propose an innovative encoding scheme, FP2, which compresses floating-point representation to below 3 bits. Compared to FP4, our approach reduces the size of deep neural network (DNN) models by 47%. We analyze the error characteristics of this data format from three perspectives and introduce two DNN accelerator architectures based on FP2. To address the trade-off between accuracy and hardware efficiency in complex tasks, we optimize the multiply-accumulate (MAC) operation into bitwise addition and logic operations, reducing computational requirements by approximately 2× and 4×. We evaluate our proposed data format on CIFAR-10 and Ima-

geNet benchmarks using ResNet18/50 and ConvNeXt-Tiny models. On ResNet50 and ConvNeXt-Tiny, our FP2 format maintains an accuracy loss of approximately 3% compared to full-precision (FP32) models. Furthermore, we conduct a comprehensive evaluation on FPGA and ASIC platforms. Compared to FP4, our FP2-based architectures achieve up to a 78% reduction in both resource utilization and power consumption across different platforms. This design enables the efficient deployment of advanced neural networks on edge devices. We also conducted a comparative analysis of the zero-shot performance for the proposed FP2 format based on the LLAMA model across three benchmark datasets. The results demonstrate that in a comparative evaluation under 2-bit PTQ, FP2 exhibits significant advantages over baseline methods.

REFERENCES

[1] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2025, pp. 5998–6008.

[2] A. Dosovitskiy et al., "An image is worth 16×16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.

[3] C. Jiang, A. Jayarajan, H. Lu, and G. Pekhimenko, "Arbitor: A numerically accurate hardware emulation tool for DNN accelerators," in *Proc. USENIX Annu. Tech. Conf.*, Jan. 2023, pp. 1–12.

[4] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1474–1479.

[5] N. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Architect.*, 2017, pp. 1–12.

[6] P. Micikevicius et al., "FP8 formats for deep learning," 2022, *arXiv:2209.05433*.

[7] A. N. Kuzmin, M. V. Baalen, Y. Ren, M. Nagel, J. W. T. Peters, and T. Blankevoort, "FP8 quantization: The power of the exponent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 14651–14662.

[8] B. D. Rouhani et al., "Pushing the limits of narrow precision inferencing at cloud scale with Microsoft floating point," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 10271–10281.

[9] A. Tirumala and R. Wong, "NVIDIA Blackwell platform: Advancing generative AI and accelerated computing," in *Proc. IEEE Hot Chips Symp. (HCS)*, Aug. 2024, pp. 1–33.

[10] S.-Y. Liu, Z. Liu, X. Huang, P. Dong, and K.-T. Cheng, "LLM-FP4: 4-bit floating-point quantized transformers," 2023, *arXiv:2310.16836*.

[11] H. Peng et al., "FP8-LM: Training FP8 large language models," 2023, *arXiv:2310.18313*.

[12] H. Touvron et al., "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

[13] J. Wang, H. Liu, D. Feng, J. Ding, and B. Ding, "FP4-quantization: Lossless 4bit quantization for large language models," in *Proc. IEEE Int. Conf. Joint Cloud Comput. (JCC)*, Jul. 2024, pp. 61–67.

[14] Y. Zhang et al., "Integer or floating point? New outlooks for low-bit quantization on large language models," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2024, pp. 1–6.

[15] C. Yan, X. Zhao, T. Zhang, J. Ge, C. Wang, and W. Liu, "Design of high hardware efficiency approximate floating-point FFT processor," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 11, pp. 4283–4294, Nov. 2023.

[16] Z. Niu, T. Zhang, H. Jiang, B. F. Cockburn, L. Liu, and J. Han, "Hardware-efficient logarithmic floating-point multipliers for error-tolerant applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 71, no. 1, pp. 209–222, Jan. 2024.

[17] J. Park et al., "Designing low-power RISC-V multicore processors with a shared lightweight floating point unit for IoT endnodes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 71, no. 9, pp. 4106–4119, Sep. 2024.

[18] X. Fang, Y. Wang, L. Chen, and F. An, "A reconfigurable floating-point division and square root architecture for high-precision softmax," in *Proc. IEEE Trans. Circuits Syst. I, Reg. Papers*, Jun. 2025, pp. 4737–4750.

[19] Y. Hou et al., "A 40nm 4Mb high-reliability STT-MRAM achieving 18ns write-time and 94.9% Wafer-level-die-yield across -55°C-to-125°C," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2025, pp. 1–3.

[20] P.-H. Lee et al., "A 16 nm 32Mb embedded STT-MRAM with a 6ns read-access time, a 1M-cycle write endurance, 20-year retention at 150°C and MTJ-OTP solutions for magnetic immunity," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2023, pp. 494–496.

[21] H. Cai, Y. Hou, M. Zhang, B. Liu, and L. A. D. B. Naviner, "Dependable STT-MRAM with emerging approximation and speculation paradigms," *IEEE Design Test*, vol. 40, no. 3, pp. 17–25, Jun. 2023.

[22] Y.-C. Huang et al., "A 32Mb RRAM in a 12 nm FinFet technology with a $0.0249\mu m^2$ bit-cell, a 3.2 GB/S read throughput, a 10 Kcycle write endurance and a 10-year retention at 105 °C," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2024, pp. 288–290.

[23] C.-Y. Chang, C.-T. Huang, Y.-C. Chuang, K.-C. Chou, and A.-Y. Wu, "BFP-CIM: Runtime energy-accuracy scalable computing-in-memory-based DNN accelerator using dynamic block-floating-point arithmetic," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 71, no. 5, pp. 2079–2092, May 2024.

[24] S. Wang and P. Kanwar, "BFloat16: The secret to high performance on cloud TPUs," *Google Cloud Blog*, vol. 4, no. 1, Jan. 2020.

[25] B. Darvish Rouhani et al., "With shared microexponents, a little shifting goes a long way," in *Proc. 50th Annu. Int. Symp. Comput. Archit.*, Jun. 2023, pp. 1–13.

[26] W. Zhao, Q. Dang, T. Xia, J. Zhang, N. Zheng, and P. Ren, "Optimizing FPGA-based DNN accelerator with shared exponential floating-point format," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 11, pp. 4478–4491, Nov. 2023.

[27] B. Darvish Rouhani et al., "Microscaling data formats for deep learning," 2023, *arXiv:2310.10537*.

[28] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 525–542.

[29] F. Li, B. Liu, X. Wang, B. Zhang, and J. Yan, "Ternary weight networks," 2016, *arXiv:1605.04711*.

[30] X. Geng, S. Liu, J. Jiang, K. Jiang, and H. Jiang, "Compact powers-of-two: An efficient non-uniform quantization for deep neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2024, pp. 1–6.

[31] Y. Li, X. Dong, and W. Wang, "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks," 2019, *arXiv:1909.13144*.

[32] H. Wang et al., "BitNet: Scaling 1-bit transformers for large language models," 2023, *arXiv:2310.11453*.

[33] S. Ma et al., "The era of 1-bit LLMs: All large language models are in 1.58 bits," 2024, *arXiv:2402.17764*.

[34] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2009.

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[37] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 11976–11986.

[38] A. Grattafiori et al., "The Llama 3 herd of models," 2024, *arXiv:2407.21783*.

[39] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "HellaSwag: Can a machine really finish your sentence?," 2019, *arXiv:1905.07830*.

[40] H. J. Levesque, E. Davis, and L. Morgenstern, "The Winograd schema challenge," in *Proc. KR*, 2011, pp. 552–561.

[41] K. Sakaguchi, R. Le Bras, C. Bhagavatula, and Y. Choi, "WinoGrande: An adversarial Winograd schema challenge at scale," *Commun. ACM*, vol. 34, no. 5, pp. 8732–8740, Apr. 2020.

[42] M. Courbariaux, Y. Bengio, and J. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

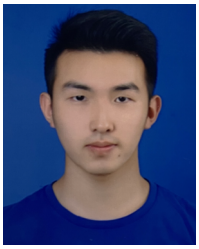
[43] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to + *or* -," 2016, *arXiv:1602.02830*.

[44] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*.

[45] Y. Liu et al., "VPTQ: Extreme low-bit vector post-training quantization for large language models," 2024, *arXiv:2409.17066*.

[46] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Trans. Electron. Comput.*, vols. EC–11, no. 4, pp. 512–517, Aug. 1962.
[47] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," 2018, *arXiv:1810.05270*.
[48] Z. Tu, X. Chen, P. Ren, and Y. Wang, "AdaBin: Improving binary neural networks with adaptive binary sets," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2022, pp. 379–395.
[49] Z. Tu, J. Ma, T. Xia, W. Zhao, P. Ren, and N. Zheng, "CAQ: Context-aware quantization via reinforcement learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–8.
[50] *Zynq Dpu V3.2 Production Guide (PG338)*, Xilinx, 2020.
[51] (2019). *Designware Library-Datapath and Building Block IP, P-2019.03- DWBB 201903.0.* [Online]. Available: https://www.synopsys.com/dw/buildingblock.php

Qiwei Dang received the bachelor's degree in information engineering from Xi'an Jiaotong University in 2020, where he is currently pursuing the Ph.D. degree with the Department of Artificial Intelligence, Institute of Artificial Intelligence and Robotic. His research interests include computing architecture and domain-specific accelerator design.



Chengyu Ma received the bachelor's degree from Northwestern Polytechnical University, Xi'an, China, in 2020. He is currently pursuing the Ph.D. degree with the Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University. His current research interests include DNN-based accelerator and model compression.

Haiduo Huang received the bachelor's degree from Northwest University in 2018 and the master's degree from Xi'an Jiaotong University in 2021, where he currently pursuing the Ph.D. degree, with a specializing in model compression and acceleration.

Gelin Fu (Member, IEEE) received the bachelor's degree in control science and engineering from Chongqing University, Chongqing, China, in 2019. He is currently pursuing the Ph.D. degree with the College of Artificial Intelligence, Xi'an Jiaotong University, Xi'an, China. His research interests include system modeling and optimization and computer architecture.

Zhiwang Huo received the bachelor's degree from Shenzhen University, Shenzhen, China, in 2017, and the master's degree from Xi'an University of Technology, Xi'an, China, in 2021. He is currently pursuing the Ph.D. degree with the College of Artificial Intelligence, Xi'an Jiaotong University, Xi'an. His current research interests include DNN-based accelerator computer architecture.

Guoming Yang received the bachelor's degree from Northeastern University, Shenyang, China, in 2019, and the master's degree from Xi'an Jiaotong University, Xi'an, China, in 2022, where he is currently pursuing the Ph.D. degree with the Department of Artificial Intelligence, Institute of Artificial Intelligence and Robotics. His current research direction is deep neural network (DNN)-based computing architecture and VLSI design.

Pengchen Zong received the bachelor's degree in engineering from Northwestern Polytechnical University and the master's degree in electronic science and technology from Xi'an Jiaotong University, where he is currently pursuing the Ph.D. degree, under the Supervision of Prof. Pengju Ren and Prof. Tian Xia. He is also engaging in the research on high performance computing and computing architecture design.

Tian Xia (Member, IEEE) received the Ph.D. degree from the National Institute of Applied Sciences (INSA), France, in 2016. He has been an Associate Professor with the College of Artificial Intelligence, Xi'an Jiaotong University, since 2019. His current research domains include the cloud-computing virtualization, and innovative parallel computation architecture. He is also interested in reinforcement learning algorithms and applications.

Wenzhe Zhao (Member, IEEE) received the bachelor's, master's, and Ph.D. degrees from Xi'an Jiaotong University in 2005, 2008, and 2014, respectively. From 2011 to 2013, he was a Visiting Student with the Rensselaer Polytechnic Institute (RPI). He has been an Assistant Professor with the College of Artificial Intelligence, Xi'an Jiaotong University, since 2020. His current research direction is DNN based computing architecture and VLSI design.

Pengju Ren (Member, IEEE) received the Ph.D. degree from Xi'an Jiaotong University in 2012. From 2009 to 2011, he was a Visiting Ph.D. Student with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology (MIT). He has been a Professor with the College of Artificial Intelligence, Xi'an Jiaotong University, since 2015. His current research interests include on-chip network and computer architecture.