April: Accuracy-Improved Floating-Point Approximation For Neural Network Accelerators

Yonghao Chen[1], Jiaxiang Zou[2,†], Xinyu Chen[1,*]

[1]The Hong Kong University of Science and Technology (Guangzhou)
[2]University of Electronic Science and Technology of China

***Abstract*—Neural Networks (NNs) have achieved breakthroughs in computer vision and natural language processing. However, modern models are computationally expensive, with floating-point operations posing a major bottleneck. Floating-point approximation, such as Mitchell's logarithm, enables floating-point multiplication using simpler integer additions, thereby improving hardware efficiency. However, its practical adoption is hindered by challenges such as precision degradation, efficient hardware integrations, and management of trade-offs between accuracy and resource efficiency.**

In this paper, we propose a hardware-efficient down-sampling-based compensation method to mitigate precision loss and a flexible bias mechanism to accommodate diverse data distributions in NN models. Building on this foundation, we design configurable systolic arrays optimized for NN accelerators. To further support practical adoption, we introduce April, a co-design framework that balances the accuracy and resource usage of generated synthesizable systolic arrays. Our FPGA-based evaluations demonstrate that April-generated systolic arrays reduce root mean square error (RMSE) by up to 96% and achieve 34%-52% area reduction even compared to INT8-based implementations while maintaining comparable or improved model accuracy. Our design is open-sourced at https://github.com/CLabGit/April.

***Index Terms*—Floating-point approximation, FPGAs**

I. Introduction

Neural Networks (NNs), including convolutional neural networks [1]–[3], vision transformers [4]–[6], and large language models [7]–[9], have achieved state-of-the-art performance across various applications, such as computer vision and natural language processing. However, their exceptional performance comes at the cost of high computational complexity, with floating-point multiplications emerging as a major bottleneck. While floating-point arithmetic ensures numerical precision, its implementation demands significant chip area.

Recent advancements in floating-point approximation have demonstrated great potential to reduce the hardware costs of multiplication operations. Based on Mitchell's logarithm approximation [10] that a floating-point number can be interpreted as a number in a logarithmic number system, recent works [11]–[13] theoretically prove that floating-point multiplication can be performed with integer additions. This suggests that *expensive floating-point multiplication units can be replaced with simpler integer addition units*, unlocking substantial resource-saving opportunities for heavy computations of NNs.

However, a significant gap remains in applying the aforementioned floating-point multiplication approximation (FPMA) to real hardware. First, the transition from a floating-point number to a logarithmic number inevitably introduces precision degradation. As a result, an effective and hardware-efficient compensation solution is essential for maintaining the model's accuracy. Second, comprehensive hardware designs for NNs that seamlessly integrate both approximation and compensation methods are necessary to quantitatively evaluate the benefits of FPMA in real-world settings. Third, since compensation methods inherently involve trade-offs between precision and hardware costs, a flexible framework is needed to guide these design decisions based on application-specific requirements. In this paper, we address these challenges and make the following contributions:

- We propose a hardware-efficient down-sampling-based compensation method, together with a flexible bias mechanism to FPMA to improve model accuracy.
- We develop configurable systolic arrays that integrate proposed accuracy-improved FPMA.
- We introduce April, a framework that balances model accuracy and hardware cost for systolic array generation.
- Extensive FPGA evaluations demonstrate that April generated systolic array (FP8) reduces RMSE by up to 96% and saves 34%-52% hardware resources compared to INT8-based systolic arrays while offering better model accuracy.

II. Background and Motivation

A. Approximating FP Multiplication with Integer Addition

According to the IEEE 754-2019 Standard [14], a normalized FP (*Floating-Point*) number x can be represented as,

$$x = (-1)^{S_x} \times 2^{E_x - b} \times (1 + M_x),\ 0 \le M_x < 1, \quad (1)$$

with S_x as a sign bit, E_x as an W_E-bit exponent, $b = 2^{W_E-1} - 1$ as a bias, and M_x as a W_M-bit mantissa without the leading one. Although FP provides a wide dynamic range, operations like multiplication are hardware-intensive due to the complexity of managing the sign, exponent, mantissa, and additional steps like rounding and normalization.

While integer-based quantization has been widely used to reduce computation overhead by replacing floating-point

*Corresponding author: Xinyu Chen
†Work conducted during internship at HKUST(GZ)

multiplication with simpler integer multiplication [15], [16], Gustafsson et al. [11] further simplifies FP multiplication by approximating it as an integer addition. Specifically, based on Mitchell's logarithmic approximation [10], a floating-point number x can be mapped to the logarithmic domain as:

$$\log_2(x) \approx (E_x - b) + M_x = X - B, \tag{2}$$

where $X = \{E_x, M_x\}$ represents the binary form of x, and $B = b \ll W_M$ is the shifted bias. While approximation from $\log_2(1 + M_x)$ to M_x introduces some errors, it enables the Floating-Point Multiplication Approximation (FPMA). The binary representation of the multiplication result R can be expressed as:

$$R \approx log2(x \times y) + B = X + Y - B, \tag{3}$$

where X and Y denote the binary representations of the two operands. This method simplifies the complexity of floating-point multiplication by reducing it to integer addition.

B. Challenges of Applying FPMA to NN Accelerators

Approximation techniques, while promising for improving hardware efficiency, often result in precision loss [17], [18], which can severely degrade model accuracy; therefore, we first study the impact of FPMA on model accuracy. We compared the accuracy of MobileNetV2 and other models under the original FP8 data type (golden accuracy) and various approximated FP8 formats. Table I summarizes the results. The results reveal that naively applying FPMA can lead to unacceptable accuracy drops. For example, MobileNetV2's accuracy decreases from 66.28% (golden) to 58.37% under FP8(E4M3), representing an 8% reduction. Similarly, FastViT-t12 suffers a drastic decline from 76.59% to 30.47% under FP8(E3M4). While some configurations, such as FP8(E2M5) with a mantissa width of 2, introduce no error, others exhibit significant accuracy degradation, particularly with wider mantissa bit-widths. These findings highlight two critical challenges in applying FPMA to hardware accelerators for neural networks.

TABLE I: Top-1 accuracy comparison of MobileNetV2, FastViT, DeiT-T and MobileOne on the ImageNet-1k dataset, with original FP8 or FPMA.

NN Model	Accuracy	E2M5	E3M4	E4M3	E5M2
MobileNetV2 [19]	‡Golden	71.08%	70.30%	66.28%	48.11%
	FPMA	64.65%	62.76%	58.37%	†48.11%
FastViT-t12 [20]	‡Golden	65.38%	76.59%	74.04%	56.30%
	FPMA	0.64%	30.47%	67.02%	†56.30%
DeiT-T [21]	‡Golden	68.40%	69.10%	68.02%	64.54%
	FPMA	0.08%	0.27%	66.75%	†64.54%
MobileOne [22]	‡Golden	67.74%	67.84%	41.63%	0.46%
	FPMA	3.52%	16.94%	14.58%	†0.46%

† FPMA will not introduce error for M2 (mantissa width of 2) [23].
‡ Golden accuracy on FP8 obtained with post-training quantization [24].

Challenge 1: Hardware efficient error correction for high model accuracy. The accuracy loss introduced by FPMA, as shown in Table I, highlights the need for practical and hardware-efficient error correction methods. Existing works [11], [23] explore fine-grained or single-value compensation methods to mitigate precision loss. However, they either suffer from high hardware costs or insufficient accuracy. Balancing effective error correction with minimal hardware overhead remains a key challenge for deploying FPMA in neural network accelerators.

Challenge 2: Efficient NN hardware design with FPMA. To our knowledge, no existing hardware accelerators incorporate FPMA for NNs, leaving a significant gap in architecture design. On the other hand, error correction usually introduces trade-offs between computational efficiency and model accuracy. Consequently, hardware design with FPMA introduces new architectural considerations, such as selecting the right error correction granularity based on application requirements, making hardware support of FPMA nontrivial.

C. Our Solutions Toward Practical FPMA for NNs

Solution 1: Improved Precision with hardware-efficient compensation and flexible bias. To address the precision loss introduced by FPMA, we propose a hardware-efficient compensation method combined with flexible bias. The compensation mechanism uses localized averaging through a downsampling approach, reducing hardware costs while maintaining accuracy. This method dynamically adjusts the error compensation window size to balance precision and efficiency. Additionally, a flexible bias mechanism allows adaptive bias settings for activations and weights to accommodate their distinct distribution patterns. This ensures a wider representable range and significantly enhances model accuracy, overcoming the limitations of fixed-bias FPMA designs.

Solution 2: A model accuracy-aware framework for FPMA-enabled systolic array generation. We present April, a co-design framework for generating systolic arrays with accuracy-enhanced FPMA. This framework integrates error correction and format conversion, enabling optimized compensation window sizes and seamless data format adjustments. It also supports customizable systolic array designs tailored to user-defined accuracy and hardware cost requirements. By leveraging April, this solution effectively optimizes the trade-offs between hardware efficiency and computational accuracy, offering a practical FPMA integration to NN accelerators.

III. Hardware-Efficient Error Correction

A. Error Distribution Analysis

To design an effective error correction strategy, we first analyze the error characteristics in detail. Figure 1 presents the error distribution of $X \times Y$ across approximated FP8 formats (E4M3, E3M4, and E2M5). Since the error arises solely from the mantissa (as approximated by $log_2(1+M_x) \approx M_x$), we use the mantissa values of X and Y (denoted as X_{mant} and Y_{mant}) as coordinates of the heatmap figures. For instance, in Figure 1a, E4M3 has three mantissa bits, so X_{mant} and Y_{mant} range from 0 to 7. Each heatmap enumerates all possible combinations of X_{mant} and Y_{mant} for a given data format, ensuring comprehensive coverage of potential errors. The heatmap values reflect the difference between the golden and approximate results at these specific mantissa values and are

measured in units of last place (ulp), a widely adopted metric for error analysis [11], [25].

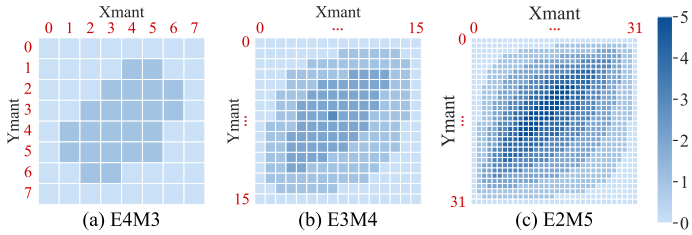


Fig. 1: Visualization of error distribution of $X \times Y$ with various approximated FP8 formats. Values measured in units of the last place (ulp) given the mantissa values of X and Y.

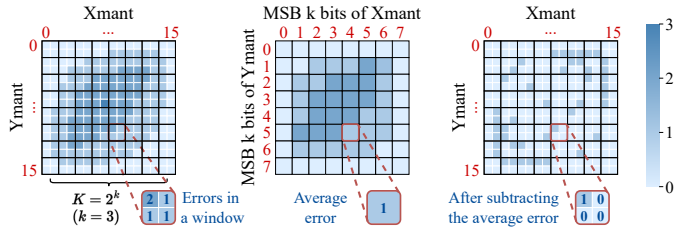
The analysis reveals three key findings. First, the errors are nonuniformly distributed, making it difficult to establish a simple arithmetic relationship between errors and mantissa values for direct compensation [23]. Second, error values vary significantly in heatmaps. As a result, applying a constant compensation value for each X_{mant}-Y_{mant} combination (i.e., a point (X_{mant}, Y_{mant}) in the heatmap) yields poor accuracy [11]. Third, as the number of mantissa bits increases (from E4M3 to E2M5), the error distribution transitions from sparse to dense. This leads to fine-grained compensation (i.e., each X_{mant}-Y_{mant} combination has an individual compensation value) impractical due to the high on-chip storage cost required [23].

Despite these challenges, we identify an opportunity in the strong locality of the error distribution. Adjacent error data points (X_{mant}, Y_{mant}) often exhibit similar values within small regions of the heatmap. This suggests that errors in these regions can be compensated with a single representative value, significantly reducing hardware complexity and storage requirements. Leveraging this observation, we propose a down-sampling-based compensation method that exploits the locality of the error distribution, providing a hardware-efficient solution for FPMA error correction.

B. Down-Sampling-Based Compensation Method

We propose a novel down-sampling-based compensation method that balances numerical accuracy and hardware cost. *The core idea is to average the errors of mantissa combinations (X_{mant},Y_{mant}) within a defined window and apply the averaged value as compensation.* Smaller window sizes provide higher compensation accuracy, while larger windows trade some accuracy for better hardware efficiency. Furthermore, our approach is versatile and can be effectively applied to any FP format such as BF16, FP16, and FP32, as it is determined by the data format and is independent of the input data distribution.

Figure 2 provides a step-by-step visual representation of how the original error heatmap of E3M4 is transformed into a compensated version through down-sampling. First, the original error map is divided into $K \times K$ windows, where $K = 2^k$ and k is the down-sampling factor. A larger k corresponds to smaller window sizes, allowing for finer granularity. In the example shown in Figure 2a, $k = 3$ results in $2^3 \times 2^3$ windows, each of size 2×2. Second, the errors within each window are averaged to generate a down-sampled error heatmap, as illustrated in Figure 2b, where the size is reduced to $K \times K$. Third, each X_{mant}-Y_{mant} combination uses the most significant k bits of X_{mant} and Y_{mant} to lookup the averaged value for compensation. Figure 2c shows error distribution after applying the down-sampling-based compensation method, demonstrating significantly reduced errors.



(a) Divide the original error heatmap into $K \times K$ sub-block. (b) Each sub-block is down-sampled. (c) After compensation. Error has been redued significantly.

Fig. 2: Transformation of the original error heatmap of E3M4 using the down-sampling-based compensation method.

This approach only requires storing the compact down-sampled heatmap, minimizing hardware demands while maintaining sufficient compensation accuracy. The trade-off between hardware efficiency and numerical precision can be tuned by adjusting the k, which also served as a compensation factor. The compensation factor k is usually chosen within the range $[3, W_M]$, where W_M is the total bit width of the mantissa. When $W_M = 2$, no error is introduced by FPMA [23]. For practical implementations, k is typically set to $k \geq 3$ to effectively capture the error distribution while minimizing the index size. On current FPGA architectures, with $k = 3$, the down-sampled heatmap with the size of 8×8 can be stored in one LUT6 [26] that has 2^6 entries. If k is increased, additional LUT6 units are required to accommodate the larger input space.

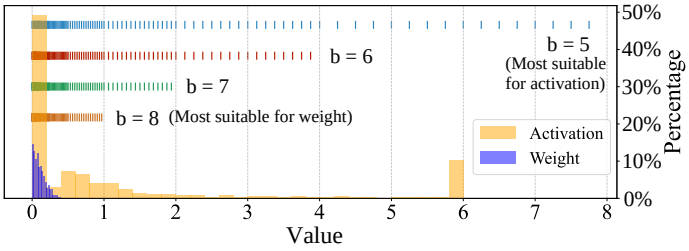


Fig. 3: Distribution of activation and weight values of MobileNetV2 network and numerical representation space of E3M4 under different biases (b).

C. Enabling Flexible Bias to FPMA

For FP formats, the bias (b, as shown in Equation 1) not only simplifies the representation of exponents but also enables flexibility to adjust the representation range of FP numbers. The original FPMA [11] uses a uniform bias, providing the same range of representable numbers for inputs and outputs (i.e., X, Y, and R). However, the distributions of activations and weights of NNs can vary significantly. Figure 3 illustrates the distribution of activations (orange) and weights (purple) in

a linear layer of MobileNetV2 and the numerical space of the E3M4 FP8 format under different bias values (b = 5, 6, 7, 8), all in absolute value. The horizontal axis represents the numerical range of FP numbers determined by the bias, while the vertical axis shows the percentage of activations and weights within specific ranges. The results show that bias b = 5 is most suitable for activations, as their distribution aligns closely with the representable range, whereas b = 8 best accommodates weights for the same reason. Selecting an appropriate bias for one often leads to suboptimal representation for the other, potentially degrading model accuracy. This motivates the need for flexible or adaptive bias strategies.

In this paper, we propose a flexible bias design for FPMA and provide theoretical proof of its correctness. By scaling the original input x with a factor of $2^b/2^{b_1}$, Equation 2 adjusts to:

$$\begin{aligned} log_2(x\times\frac{2^b}{2^{b_1}}) &= log_2(2^{E_x-b+b-b_1}\times(1+M_x)) \\ &= (E_x-b_1)+log_2(1+M_x)\approx(E_x-b_1)+M_x \\ &= X-B_1, \end{aligned} \tag{4}$$

where $B_1 = b_1 << W_M$. This adjustment enables the binary representation X to represent a custom value space scaled by $2^b/2^{b_1}$. Similarly, for scaling y by $2^b/2^{b_2}$, we derive:

$$log_2(y\times\frac{2^b}{2^{b_2}})\approx Y-B_2. \tag{5}$$

When the two scaled input operands are multiplied, their product can be expressed as:

$$\begin{aligned} &log_2(x\times\frac{2^b}{2^{b_1}}\times y\times\frac{2^b}{2^{b_2}}) = log_2(x\times y\times\frac{2^{2b}}{2^{b_1+b_2}}) \\ &= log_2(x\times y\times\frac{2^b}{2^{b_1+b_2-b}})\approx R-B_3, \end{aligned} \tag{6}$$

where $B_3 = (b_1+b_2-b) << W_M = B_1+B_2-B$ is used for the replacement of B in the FPMA calculation. This flexible bias adjustment ensures that the computation remains accurate and faithful to the scaled value space, enabling FPMA to adapt seamlessly to diverse data range distributions in NNs.

IV. Accuracy-aware Systolic Array Generation

A systolic array is a hardware architecture commonly used for accelerating matrix operations, particularly in the linear layers of neural networks. By integrating accuracy-improved FPMA into a systolic array, our design largely reduces hardware complexity while maintaining high model accuracy. Furthermore, we propose April, a comprehensive framework that determines the optimal window size for error compensation, performs data format conversion, and generates a synthesizable systolic array tailored to the user's model accuracy and hardware cost requirements.

A. Integrating Accuracy-Improved FPMA to Systolic Arrays

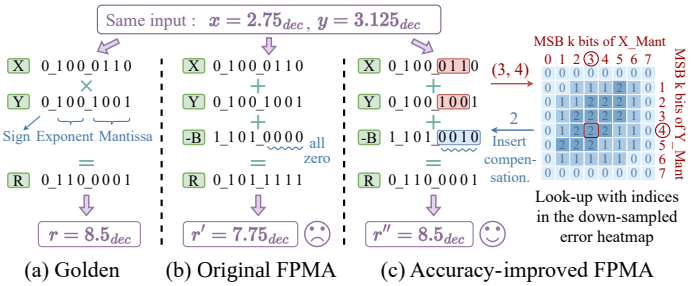
Figure 4 illustrates the multiplication process of $x\times y$ using three methods: standard floating-point multiplication (Golden), original FPMA, and the proposed accuracy-improved FPMA, demonstrated in the FP8(E3M4) format. While the original FPMA performs multiplication using only two integer additions, it introduces a noticeable error (8.5_{golden} vs. 7.75). In contrast, the accuracy-improved FPMA compensates for this error through an additional table lookup operation. Specifically, the most significant three bits (MSBs) of the mantissa of X_{mant} and Y_{mant} are extracted and used as indices to query the precomputed error compensation table. The retrieved compensation value is then added to the result, correcting the error and producing an accurate output ($r'' = 8.5$). This demonstrates that our approach effectively reduces errors with minimal additional computational and hardware overhead.

Figure 5 illustrates the proposed configurable systolic array architecture integrated with accuracy-improved FPMA for efficient linear layer computations. It utilizes an output-stationary dataflow strategy. The systolic array (a) consists of interconnected MAC units that process inputs (Iact, Wght, CascO) and output results (Oact, PassO), enabling pipelined matrix operations. Each MAC unit (b) incorporates an optimized FPMA module that performs lightweight floating-point multiplication with error compensation and flexible bias. The FPMA module (c) decomposes inputs (Iact, Wght) into sign, exponent, and mantissa components, extracts the most significant k bits of the mantissa, and uses them to query precomputed compensation values stored in LUT6 units. The retrieved values are directly appended to the lower vacant bits of the flexible bias (Flexible.B) and then added with Iact and Wght through an area-efficient ternary adder [27].

B. A Software-Hardware Co-designed Framework - April

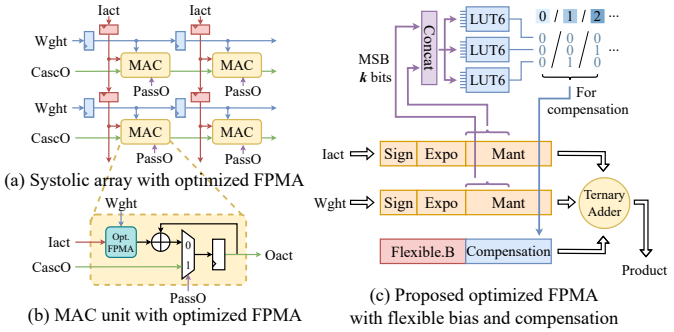
The April framework is a hardware-software co-design solution designed to balance the trade-offs between accuracy and resource consumption for FPMA-based systolic arrays. Essentially, given the model accuracy and hardware cost constraints from users, the framework could determine the most suitable compensation factor k and biases for each layer and generate the synthesizable hardware code for systolic arrays. Figure 6 outlines its workflow.

Step ① involves **Flexible Bias Selection**, where the framework selects the optimal bias for each tensor. This is achieved by analyzing the distribution of activation ($Iact$) and weight ($Wght$) values and aligning them with the numerical space of the floating-point (FP) format. This step ensures accurate and efficient representation through flexible bias adjustment.



(a) Golden (b) Original FPMA (c) Accuracy-improved FPMA

Fig. 4: Comparison of multiplication methods in FP8(E3M4) format: (a) Standard floating-point multiplication (Golden), (b) Original FPMA, and (c) Accuracy-improved FPMA.



(a) Systolic array with optimized FPMA

(b) MAC unit with optimized FPMA

(c) Proposed optimized FPMA with flexible bias and compensation

Fig. 5: Design of the proposed configurable systolic array with accuracy-improved FPMA for linear layers.

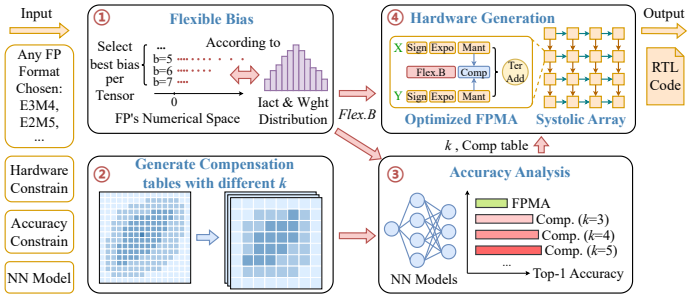


Fig. 6: The proposed April framework.

Step ② performs **Compensation Table Generation** by generating error compensation tables for various compensation factors (k) using down-sampling techniques. These precomputed tables correct FPMA errors while maintaining hardware efficiency.

Step ③ conducts **Accuracy Analysis**, where neural network models are simulated with FPMA to evaluate the impact of different compensation configurations ($k = 3, 4, 5$, etc.) on accuracy. This step identifies the optimal trade-off between accuracy and resource usage, allowing developers to choose parameters based on their optimization priorities, such as resource efficiency or accuracy maximization.

Step ④ involves **Hardware Generation**. The optimized compensation table and flexible bias settings are integrated into the systolic array design. The hardware generator produces RTL code for systolic arrays, incorporating components such as the ternary adder, flexible bias logic, and compensation modules to enable efficient FPMA computation.

The proposed April framework seamlessly integrates model-level accuracy tuning with hardware design space exploration, enabling resource-efficient and numerically accurate systolic arrays for neural network hardware accelerators.

V. Evaluation

A. Experimental Setup

For accuracy evaluation, experiments were conducted on NVIDIA GeForce RTX 3090 GPU, Torch 2.4.1, and CUDA 12.6, leveraging an open-source FP8 quantization tool from Qualcomm [24]. For resource evaluation, designs are implemented on an AMD Alveo U250 Accelerator Card with 1,728K LUTs and 3,456K registers, using Vivado 2022.2.

B. Evaluation on Down-Sampling based Compensation

1) Error Analysis: Table II evaluates the error reduction capability of the proposed compensation method for FPMA across formats such as FP8, BF16, and FP16, using metrics such as Mean Absolute Error (MeanAbsErr) and Maximum Absolute Error (MaxAbsErr) in units of last place (ulp). The results demonstrate significant error reductions across all formats, with MeanAbsErr reductions ranging from 74.1% (FP8_E3M4) to 81.1% (FP16_E5M10) and MaxAbsErr reductions ranging from 60.0% (FP8_E2M5) to 77.2% (BF16_E8M7). Errors were fully eliminated for FP8_E4M3 (100% reduction). These results highlight the method's effectiveness in minimizing errors for improved model accuracy.

TABLE II: Error analysis of the proposed compensation method across different floating-point formats.

FPMA	MeanAbsErr (ulp)			MaxAbsErr (ulp)		
Formats	Base.	Comp.	Reduction	Base.	Comp.	Reduction
FP8_E5M2	0	0	/	0	0	/
FP8_E4M3	0.36	0	**↓100%**	1	0	**↓100%**
FP8_E3M4	0.85	0.22	↓74.1%	3	1	↓66.6%
FP8_E2M5	1.79	0.48	↓73.2%	5	2	↓60.0%
FP12_E5M6	3.62	0.77	↓78.7%	11	3	↓72.7%
BF16_E8M7	7.27	1.44	↓80.2%	22	5	**↓77.2%**
FP16_E5M10	58.22	10.98	**↓81.1%**	175	52	↓70.3%

2) Area Analysis: Figure 7 compares area consumption (LUTs) of original FPMA, the proposed accuracy-optimized FPMA, and FPMA with fine-grained compensation [23] across various floating-point (FP) formats. While the overhead of fine-grained compensation method expands dramatically as the mantissa width increases, the proposed accuracy-optimized FPMA exhibits only a linear increase in LUT consumption relative to the original FPMA baseline, demonstrating efficient scalability even with higher bit-widths. The additional area required for compensation remains minimal, ensuring the design's area efficiency while maintaining improved computational accuracy.

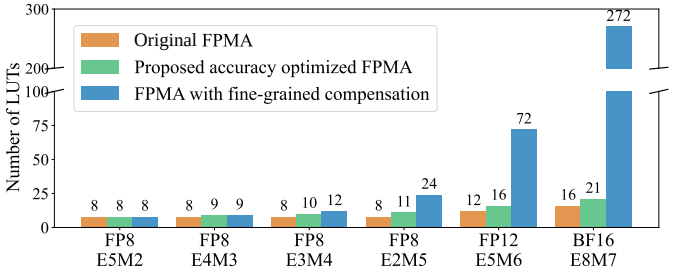


Fig. 7: Area (LUTs) consumption of a single multiplication unit implemented with different methods. The compensation factor k is set to 3 for our accuracy-optimized FPMA.

C. Evaluation on April-generated Systolic Arrays

1) Error Analysis: Figure 8 evaluates the accuracy of April-generated systolic arrays. Random matrices with values uniformly distributed in the range [1, 2) were represented in various formats and multiplied on systolic arrays with different mantissa widths (M) and matrix sizes ($S = 32, 64, 128$). Results were compared to systolic arrays with original FPMA

(FPMA) using Root Mean Square Error (RMSE). First, April-generated systolic arrays demonstrate significant error reductions over baselines (FPMA) across all settings. Second, smaller mantissas (e.g., $M3$, $M4$) showed RMSE reductions of up to 100%, while larger mantissas (e.g., $M7$, $M8$) achieved reductions of up to 96% with larger k.

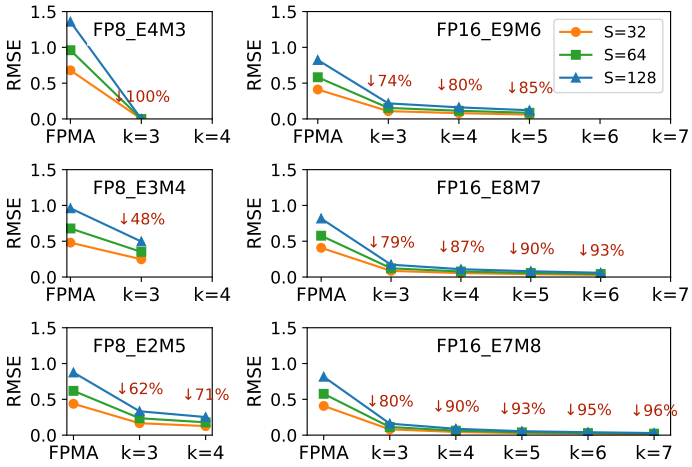


Fig. 8: RMSE analysis of matrix multiplication on April-generated systolic array with various mantissa widths (M), matrix sizes (S), and compensation factors (k). The baseline is the systolic array with original FPMA (FPMA).

2) Area Analysis: Table III compares area consumption of April-generated systolic arrays with INT8 and original FPMA-based systolic arrays. All configurations are matched in terms of parallelism and pipeline depth, with DSP usage intentionally avoided, making LUT utilization the primary metric. The results show that April-generated systolic arrays deliver significant LUT area reductions across all configurations, with smaller formats like E5M2 achieving up to 52% reduction consistently across matrix sizes. Formats with larger mantissas (e.g., E2M5) result in slightly smaller reductions of 34%-35%, reflecting a trade-off between hardware cost and precision.

TABLE III: Area (LUT) cost of April-generated (April) and original FPMA (FPMA) based systolic arrays compared to INT8-based (INT8) systolic arrays. Vivado strategy is set to Flow_AreaOptimized_high, with the frequency of 250MHz.

Formats	Mult	16×16	32×32	64×64
INT8	Exact	23808	95303	380291
E5M2	FPMA	11630	46183	184126
	April	†11630 (↓51%)	†46183 (↓52%)	†184126 (↓52%)
E4M3	FPMA	13446	52974	210956
	April	13774 (↓42%)	53629 (↓44%)	214169 (↓44%)
E3M4	FPMA	13724	55429	221075
	April	14241 (↓40%)	57786 (↓39%)	229274 (↓40%)
E2M5	FPMA	14887	59145	236195
	April	15662 (↓34%)	62145 (↓35%)	248628 (↓35%)

† FPMA does not introduce errors when mantissa equals 2.

D. End-to-end Model Accuracy Evaluation

The experiment in Figure 9 evaluates the end-to-end top-1 accuracy of four neural networks on the ImageNet-1k dataset under different configurations, including Golden INT8 (post-training quantization with INT8), Golden FP8 (post-training quantization with FP8), original FPMA (FPMA), FPMA with the proposed compensation method (FPMA + Comp), and the April (i.e., FPMA + Comp + flexible bias) with varying compensation factor ($k = 3, 4, 5$).

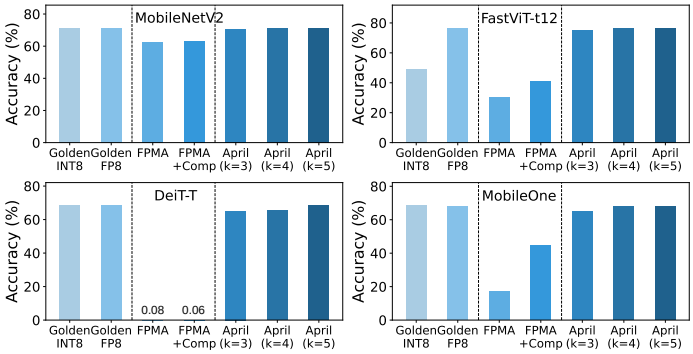


Fig. 9: Comparison of top-1 accuracy for neural networks on the ImageNet-1k dataset under different configurations.

The results show that original FPMA significantly reduces accuracy compared to both Golden FP8 and Golden INT8, highlighting the need for effective error correction. FPMA with compensation recovers most of the accuracy loss, and the April framework further improves accuracy, often matching the Golden FP8 baseline and achieving comparable or better results than Golden INT8, particularly with larger compensation factors (e.g., $k = 5$). Different models exhibit varying sensitivity to FPMA, with lightweight models like MobileOne benefiting substantially from the April framework. While earlier experiments showed significant resource reduction from accuracy-improved FPMA, these findings confirm its ability to maintain end-to-end model performance, making it practical for neural network accelerators.

VI. Conclusion

This paper addressed the challenges in adopting FPMA to neural network accelerators. We proposed accuracy-improved FPMA with down-sampling-based compensation and flexible bias methods to mitigate precision loss in FPMA. We introduced April, a co-design framework that balances accuracy and resource efficiency for systolic array generation. April-generated systolic arrays demonstrated up to 52% area reduction compared to INT8-based ones while maintaining better accuracy, and achieved up to 96% RMSE reduction compared to the original FPMA, making it a practical and efficient solution for deploying FPMA in neural network accelerators. This work bridges the gap between floating-point approximation and practical hardware implementation.

Acknowledgment

This work is supported by the Guangzhou-HKUST(GZ) Joint Funding Program (No.2025A03J3568). We also thank the AMD Heterogeneous Accelerated Compute Cluster (HACC) Program [28] for providing access to hardware resources.

REFERENCES

[1] Z. Liu *et al.*, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 976–11 986.

[2] X. Ding *et al.*, "Repvgg: Making vgg-style convnets great again," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13 728–13 737, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:231572790

[3] M. Tan *et al.*, "Efficientnetv2: Smaller models and faster training," in *International conference on machine learning*. PMLR, 2021, pp. 10 096–10 106.

[4] A. Vaswani *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[5] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *ArXiv*, vol. abs/2010.11929, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:225039882

[6] Z. Liu *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9992–10 002, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:232352874

[7] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[8] H. Touvron *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *ArXiv*, vol. abs/2307.09288, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259950998

[9] A. Liu *et al.*, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," *arXiv preprint arXiv:2405.04434*, 2024.

[10] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[11] O. Gustafsson *et al.*, "Approximate floating-point operations with integer units by processing in the logarithmic domain," in *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2021, pp. 45–52.

[12] A. Kosson *et al.*, "Multiplication-free transformer training via piecewise affine operations," *Advances in Neural Information Processing Systems*, vol. 36, pp. 8208–8223, 2023.

[13] T. Mogami, "Deep neural network training without multiplications," 2020. [Online]. Available: https://arxiv.org/abs/2012.03458

[14] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[15] A. Gholami *et al.*, "A survey of quantization methods for efficient neural network inference," *ArXiv*, vol. abs/2103.13630, 2021.

[16] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[17] M. Imani *et al.*, "Approxlp: Approximate multiplication with linearization and iterative error control," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19, 2019.

[18] C. Chen *et al.*, "Pam: A piecewise-linearly-approximated floating-point multiplier with unbiasedness and configurability," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2473–2486, 2022.

[19] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[20] P. K. A. Vasu *et al.*, "Fastvit: A fast hybrid vision transformer using structural reparameterization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 5785–5795.

[21] H. Touvron *et al.*, "Training data-efficient image transformers & distillation through attention," in *International conference on machine learning*. PMLR, 2021, pp. 10 347–10 357.

[22] P. K. A. Vasu *et al.*, "Mobileone: An improved one millisecond mobile backbone," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 7907–7917.

[23] T. Lindberg *et al.*, "On approximate 8-bit floating-point operations using integer operations," *arXiv preprint arXiv:2406.18441*, 2024.

[24] A. Kuzmin *et al.*, "Fp8 quantization: The power of the exponent," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 651–14 662, 2022.

[25] J.-M. Muller, "On the definition of ulp(x)," 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:264519989

[26] "Amd fpga advantages over competing legacy lut4 architectures," AMD, Tech. Rep., 2024. [Online]. Available: https://docs.amd.com/v/u/en-US/wp558-amd-lut6

[27] M. Kumm *et al.*, "Multiple constant multiplication with ternary adders," in *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE, 2013, pp. 1–8.

[28] (2025) Heterogeneous accelerated compute cluster (HACC) at NUS. Https://xacchead.d2.comp.nus.edu.sg/.