SERA-Float: A Soft Error Resilient Approximate Floating-Point Computing Format

Vishesh Mishra, Marcello Traiola, Angeliki Kritikakou, Olivier Sentieys, Urbi Chatterjee

SERA-Float: A Soft Error Resilient Approximate Floating-Point Computing Format

Vishesh Mishra*, Marcello Traiola†, Angeliki Kritikakou†, Olivier Sentieys†, Urbi Chatterjee*
*IIT Kanpur, India - 208016, †Univ Rennes, CNRS, Inria, IRISA - UMR 6074, F-35000 Rennes, France
*{vishesh, urbic}@cse.iitk.ac.in, †{marcello.traiola, angeliki.kritikakou, olivier.sentieys}@inria.fr

***Abstract*—Approximate computing (AxC) reduces power consumption with minimal accuracy loss, benefiting error-tolerant, compute-intensive tasks such as machine learning, deep learning, and image processing. However, existing AxC methods often ignore the vulnerability to soft errors. Such errors can interact with approximation-induced errors, causing system failures or unexpected exceptions. To our knowledge, no work has addressed both soft error resilience and exception avoidance in approximate floating-point computing. This gap is particularly critical in deep neural network (DNN) inference, where soft error-induced errors or exceptions can significantly affect the stability and accuracy of computations.**

In this paper, we introduce *SERA-Float*, an approximate floating-point format resilient to soft errors. Specifically, it is designed to protect floating-point computations from soft error-induced errors and exception-triggering bit-flips. Unlike prior floating-point formats, *SERA-Float* protects the sign and exponent bits using error-correcting codes and relies on storing 8 valid bits of mantissa rather than performing coarse truncation. Additionally, by tracking critical bits in the floating-point representation, *SERA-Float* prevents *overflow, underflow,* and *NaN* exceptions. Our evaluation demonstrates that *SERA-Float* improves the reliability of floating-point operations during DNN inference by significantly reducing exceptions and ensuring the stability of computations. Moreover, it enables energy-efficient arithmetic by leveraging narrower arithmetic units, yielding up to 80.3% energy savings per multiplication with a 0.9% reduction in DNN inference accuracy.

***Index Terms*—Approximate Computing, Floating-point Format, Energy Efficiency, Fault-Tolerant Computing, Neural Networks.**

I. Introduction

Deep Neural Networks (DNNs) have become fundamental to many modern machine learning applications, including computer vision, natural language processing, and autonomous systems [1]. However, DNNs are computationally intensive, requiring significant processing power, memory, and energy, even during inference. To deploy and accelerate DNNs in resource-constrained environments, such as mobile devices and edge computing platforms, low-precision and approximate computing (AxC) techniques are used [2], [3]. More specifically, approximate arithmetic units, such as adders and multipliers, minimize *power consumption*, *resource usage*, and *latency* at the cost of bounded loss in precision [4], [5]. While AxC offers significant benefits, it is *vulnerable to soft errors (SEs)* [6]. Soft errors are typically induced by environmental factors, including cosmic radiation, high-energy particles, and alpha radiation from packaging materials, all of which can cause transient bit-flips in memory and computational units [7]. Although these errors are non-permanent, they can severely affect computations, especially when they occur in critical locations, such as the most significant bits of integer numbers [8], [9], or the sign, exponent, or mantissa bits of floating-point numbers.

This work was supported by the Information Security Education and Awareness (ISEA), an initiative of the Ministry of Electronics and Information Technology (MeitY), Government of India, by C3i (cybersecurity and cybersecurity for Cyber-Physical Systems) Innovation Hub, IIT Kanpur, by ANR FASY (ANR-21-CE25-0008-01) and ANR RE-TRUSTING (ANR-21-CE24-0015-02). Vishesh Mishra was supported by Prime Minister's Research Fellowship (PMRF) from the Ministry of Education, Government of India.

Recent studies have highlighted the practical relevance of soft errors. For instance, authors in [7] show that a typical 6T-SRAM cell in 45-nm technology has a soft error rate of approximately 1095 errors per million cells per billion hours at sea level, with this rate increasing nearly 500 times at altitudes of 40,000 feet. In general, around 1-4% of the total computations can suffer from 1-bit flips, with 0.02-0.08% experiencing 2-bit flips [10]. In the absence of error detection or correction mechanisms, these bit-flips can cause *Silent Data Corruptions (SDCs)*, leading to erroneous results, application crashes, or system failures [11]. For example, SDCs propagate through integer memory hierarchies, compromising DNN inference accuracy by up to 33.7% when affecting integer operands [8]. Floating-point corruptions, however, present amplified risks: perturbations to mantissa or exponent fields induce non-linear error propagation, particularly in iterative algorithms. Modern computing systems exacerbate this vulnerability through widespread reliance on IEEE-754-compliant FPUs for mission-critical applications, including autonomous systems and scientific simulations. Prior work explores custom floating-point formats such as FP16 [12], BrainFloat16 (BF16) [13], FP8 [14]. for power and area efficiency. Also, the error resilience of the Posit format [15] has been performed. However, no systematic analysis addresses soft-error resilience in floating-point arithmetic units.

In addition to soft error-induced erroneous results, approximate floating-point computing faces another challenge: the occurrence of *unexpected exceptions*, such as *overflow, underflow, and NaN* (Not-a-Number), triggered by bit-flips in critical components of floating-point representations (e.g., the sign or exponent bits). In fact, the presence of soft errors can result in up to *2.54× more exception triggering* compared to fault-free operations (see Section 2). These exceptions can disrupt computation and induce undefined behaviour, which is particularly detrimental in systems that rely on precise floating-point calculations. For instance, in DNNs, soft error-induced errors or exceptions can lead to significant computational instability, affecting the reliability of predictions and halting inference processes due to *NaN* or *overflow* exceptions.

Existing floating-point formats, such as IEEE 754 FP32 and FP64 [16], prioritize precision to provide accurate numerical results in a wide range of applications. Other formats, such as *Binary16* or FP16 [12], *Brainfloat16* or BF16 [13], *DLfloat16* [17], FP8 [14], and INT8 [18] focus on energy efficiency by narrowing the precision and range, making them suitable for deploying DNNs in resource-constrained environments. However, none of these formats is designed to specifically address the *issue of soft error resilience*. In this paper, *for the first time*, we propose a soft error-resilient approximate floating-point format, *SERA-Float*, that addresses both *soft error resilience* and the elimination of *unexpected exceptions* in approximate floating-point computations. *SERA-Float* protects the sign and exponent bits using error-correcting codes and relies on storing valid bits of mantissa rather than performing coarse truncation, i.e., truncating least significant bits of mantissa. Additionally, *SERA-Float* prescribes rules to prevent unexpected *overflow, underflow, and NaN* exceptions.

Furthermore, we use *narrow precision arithmetic units* for mantissa computations to improve energy efficiency without sacrificing application accuracy. To summarize, the main contributions of this work are:

- *SERA-Float*, a 32-bit soft error-resilient format that protects sign and exponent bits via error-correcting codes and prevents unexpected exceptions in floating-point computations. More specifically, the format facilitates 2-bit error detection and correction for exponent bits, protection of the sign bit, and 1-bit error detection for mantissa bits.
- A mechanism to compress the 23-bit mantissa of FP32 to the 8-bit compressed mantissa of *SERA-Float* without losing the data integrity of mantissa bits. Further, we *track and save* the number type of the floating-point representation using a 2-bit identifier. This ensures that soft errors occurring in *SERA-Float* do not trigger a false exception.
- We demonstrate the *energy efficiency* of *SERA-Float*, illustrating that the use of narrow (8-bit) arithmetic units can lead to up to 80.3% energy savings in floating-point multiplications.
- To validate the effectiveness of *SERA-Float*, we conduct a comprehensive evaluation across five applications: Image Sharpening, Gaussian Smoothing, K-Nearest Neighbors (KNN), Deep Neural Networks (DNN), and Support Vector Machines (SVM). The results show that *SERA-Float* provides superior resilience to soft errors, making it a robust and reliable choice for resource-constrained and fault-prone environments.

Paper Organization. The structure of the paper is as follows: Section II provides the motivation for this work. Section III introduces the *SERA-Float* format, while Section IV presents the experimental evaluation. Finally. Section V concludes the paper.

II. Motivation

In the context of floating-point arithmetic, soft errors can lead to two distinct outcomes: (a) erroneous computations and (b) exception triggering. The specific impact of these errors depends on the location of the bit-flip within the floating-point representation. For instance, a bit-flip in critical fields, such as the exponent or sign bit, may cause significant deviations in the computed value or even trigger exceptions, such as NaN or Infinity. Understanding these potential outcomes is essential for designing floating-point formats that are resilient to soft errors and prevent computational failures.

Table I quantifies soft error impacts across *INT8, FP8 (E3M4), FP16, BF16, and FP32* formats. Bit-flips induce two failure modes: (1) unbounded numerical deviations or (2) exceptional states (*NaN/Infinity*). In *INT8*, a single bit-flip in the most significant bit (MSB) inverts sign magnitude, introducing 100% relative error. For *FP8(E3M4)* where the exponent consists of 3 bits while the mantissa consists of 4 bits, an MSB flip results in a sign change of the number, amplifying errors to 200%. Similarly, for *FP16*, a bit flip in the MSB of the number (0111011110000000) causes a sign change (highlighted red in Table I), leading to a large absolute error of 61440, but no exception is triggered. In contrast, as depicted in Table I, a soft error in the exponent results in *NaN*, triggering an exception. *BF16* shows similar vulnerabilities—exponent field perturbations produce either unbounded value distortions or exceptional states. *FP32* highlights silent criticality: for a number $(01111111011010000000000000000000)$ represented in *FP32* format, a bit-flip in the sign bit alters a large number (3.08e+38) to a small value (0.90625) without triggering an exception, while a flip in the exponent field causes *NaN* and triggers an exception. These examples demonstrate how soft errors can corrupt floating-point values and trigger unexpected exceptions, such as *NaN*, which can propagate through computations and lead to significant inaccuracies.

Table II presents the results of a motivating experiment where a set of 10^6 randomly generated floating-point numbers are processed through a floating-point multiplier while the numbers are exposed to soft errors. In this analysis, we have injected 1-bit flips at 4%, 3%, 2%, and 1% error-injection rates to simulate soft errors in the sign, exponent, and mantissa fields of the floating-point representation. The target of this analysis is to observe the impact of these soft errors on exception triggering. Table II reports the total number of exceptions triggered in the presence and absence of soft errors under varying bit-flip rates. The recorded exceptions include *Overflow, Underflow, and NaN*. In the absence of soft error protection, 6382 exceptions are recorded. To address these issues, we propose *SERA-Float*, a floating-point format designed to handle errors and ensure computational correctness in error-prone environments (details in Section III). With *SERA-Float*, the number of triggered exceptions is reduced, e.g., to 2512 for the 4% 1-bit flips. On average, the absence of soft-error protection causes $\frac{6382}{2512} = 2.54\times$ more exceptions compared to the scenario without any soft errors. This highlights the increased sensitivity of floating-point operations to soft errors and the consequent rise in exception triggering.

Table III presents a qualitative comparison of *SERA-Float* against widely used integer and floating-point formats, including *INT8, FP8, FP16, BF16, and FP32*. Notably, *SERA-Float* is the only format among those compared that explicitly considers soft-error resilience, offering built-in mechanisms for error correction. Additionally, *SERA-Float* supports efficient mantissa compression, a feature not present in other custom floating-point formats such as *FP16 and BF16*, which rely on naive truncation [13]. While *FP32 and BF16* offer a broad dynamic range, *SERA-Float* achieves comparable range coverage while sustaining precision levels akin to *FP32*.

III. Soft Error Resilient Approximate Floating-Point Format (SERA-Float)

This section initially describes the considered fault model and then presents the proposed approximate floating-point storage format and the conversion process.

A. Fault Model

The computing system is susceptible to soft errors in floating-point representations, which can impact the sign, exponent, and mantissa bits. In this section, we introduce two fault models that account for both single-bit and multiple-bit faults, respectively. The occurrence of soft errors in storage elements is influenced by various factors. As established in [7], the soft-error generation in a hardware component x can be quantified using the Soft Error Rate (SER), as defined in Equation (1):

$$\text{SER}_x = e_x \cdot p_x \tag{1}$$

where e_x represents the error-generation rate (which is dependent on technology and operational conditions), while p_x accounts for the conditional probability of a system-visible failure, which is typically measured in failures-in-time (FIT). According to [10], single-bit flips are typically observed with error-generation rates ranging from 1% to 4% (dependent on environment, operating conditions, and technology node). In contrast, multi-bit faults generally occur with error-generation rates between 0.1% and 0.4%, with 85% of these being 2-bit faults [10]. Additionally, [19] provides details on the nature of 2-bit faults, which can occur in either spatial or temporal contexts. In a spatial setting, contiguous bit locations get corrupted, while in a temporal setting, non-contiguous bits are affected. Existing

TABLE I: Errors and exception triggering due to soft errors in INT8, FP8(E3M4), FP16, BF16 and FP32

Format	Original Number	Number after Soft Error	Absolute Error	Exception Triggering
INT8	(10000000) = -128	(00000000) = 0	128	✗
FP8(E3M4)	(0 100 0101) = 2.625	(1 100 0101) = -2.625	5.25	✗
BF16	(0 11111110 1101000) = 3.08309e+38	(0 01111110 1101000) = 0.90625	3.08e+38	✗
		(0 11111111 1101000) = NaN	NaN	✓
FP16	(0 11101 1110000000) = 30720	(1 11101 1110000000) = -30720	61440	✗
	(0 11101 1110000000) = 30720	(0 11111 1110000000) = NaN	NaN	✓
FP32	(0 11111110 11010000000000000000000) = 3.083809e+38	(0 01111110 11010000000000000000000) 0.90625	3.08e+38	✗
	(0 11111110 11010000000000000000000) = 3.083809e+38	(0 11111111 11010000000000000000000) = NaN	NaN	✓

TABLE II: Exceptions in FP arithmetic with(out) soft-error protection

Event/ Exceptions	Total exceptions with *SERA-Float*			Total exceptions (in FP32) without soft-error protection
	Overflow	Underflow	NaN	Overflow/Underflow + NaN
4% 1-bit flip	4	5	2503	6382
3% 1-bit flip	3	4	1794	6382
2% 1-bit flip	2	3	1018	6382
1% 1-bit flip	0	0	581	6382

TABLE III: Comparison of SERA-Float with FP Formats

Feature/Format	INT8	FP8	FP16	BF16	FP32	SERA-Float
Soft Error Consideration	✗	✗	✗	✗	✗	✓
Error Correction Ability	✗	✗	✗	✗	✗	✓
Efficient Mantissa Compression	N/A	✗	✗	✗	N/A	✓
Wide Range	✗	✗	✗	✓	✓	✓
High Precision	N/A	✗	✓	✗	✓	✓

studies have further demonstrated that temporal occurrences are rare and can often be mitigated [20]. According to the extensive research on error-generation rates and fault occurrence patterns, we focus on single-bit flips and spatial double-bit flips for our fault models.

Assuming an equal likelihood of failures across FIT under a uniform probability distribution (i.e., $p_x = 0.5$), we estimate the range of SER for 2-bit flips as: $SER = 0.85 \times 0.5 \times (0.1\% \text{ to } 0.4\%) = 0.02\% \text{ to } 0.08\%$. Furthermore, as noted in [19], single-bit flips have the highest likelihood of occurring in terms of FIT i.e., $p_x \approx 1$. Consequently, the SER for single-bit flips is expected to range from 1% to 4%.

Based on these derived SER ranges for 1-bit and 2-bit flips, we propose two distinct fault models. In the first model (denoted as F1), we consider only single-bit flips. In the second model (denoted as F2), we include both single-bit flips and spatial 2-bit flips, occurring simultaneously with respective SERs. Correspondingly, each fault model consists of four distinct events determined by the single-bit flip SER. A summary is provided in Table IV, and the mentioned notations are followed in the rest of the paper.

TABLE IV: Proposed Fault Models

Event (Based on SER for 1-bit flip)	Fault Model #1 (F1) Single-bit Flip		Fault Model #2 (F2) Multiple-bit Flips		
	SER for 1-bit flip	Notation	SER for 1-bit flip	SER for 2-bit Spatial flips	Notation
E1	1%	F1E1	1%	0.02%	F2E1
E2	2%	F1E2	2%	0.04%	F2E2
E3	3%	F1E3	3%	0.06%	F2E3
E4	4%	F1E4	4%	0.08%	F2E4

B. Proposed Format

SERA-Float is an approximate floating-point storage format designed to enhance resilience against soft errors and mitigate exception-triggering bit-flips, particularly in hardware and memory systems susceptible to faults. *SERA-Float* converts an IEEE 754 FP32 number into a 32-bit format, where the sign and exponent bits remain unchanged while the 23-bit mantissa is compressed to 8 bits. The free 15 bits resulting from the compression are utilized for error detection and correction, mantissa decompression, and exception-avoidance mechanisms. Additionally, since the mantissa is compressed to 8 bits - matching the length of the exponent - *the computation of both the exponent and mantissa can be performed using 8-bit arithmetic units*. This design leads to significant energy and performance benefits. The proposed soft error resilient approximate floating-point format (*SERA-Float*) is illustrated in Fig. 1. *SERA-Float* consists of five components: a single parity bit, 14-bit protected exponent, 7-bit control signal, 8-bit compressed mantissa, and 2-bit identifier (I-bits). The 8-bit exponent is protected using 6-bit error-correcting code following a hamming code error correction scheme [21]. Mantissa is compressed from 23-bits to 8-bits via a compression scheme that sets a control signal for mantissa decompression. Identifier bits are used to recognize the nature of the number as normal, overflow, underflow, and NaN. The remaining paragraphs discuss *SERA-Float* components in detail and provide an illustrative example.

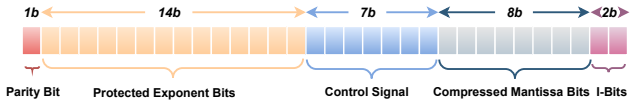


Fig. 1: The proposed SERA-Float format

Parity Bit (1 bit): While *SERA-Float* safeguards sign and exponent bits, the mantissa bits undergo compression. Since the compressed mantissa bits are not protected, *SERA-Float* adds a 1-bit even parity to the compressed mantissa bits for error detection. This mechanism helps to detect 1-bit errors in the compressed mantissa bits, which is critical for floating-point accuracy [8].

Protected Exponent (14 bits): The exponent in floating-point representations determines the range of values a number can represent. A bit-flip in the exponent can either lead to a drastic change in the magnitude of the number or cause an *overflow, underflow, or NaN* exception. Thus, protecting the exponent bits is crucial for maintaining the integrity of the number. Without this protection, even single bit-flips in the exponent bit could result in catastrophic computational errors. In the *SERA-Float* format, the 8-bit exponent is extended to 14 bits by adding 6-bit error-correcting code. These bits protect the exponent from soft errors by providing 2-bit error detection and correction capability. The exponent is encoded with a lightweight (7,4) error-correcting hamming code [21] to detect and, when possible, correct errors. The following example illustrates the exponent encoding, error detection, and correction.

Example of Exponent Encoding: Consider an 8-bit exponent, represented as $(11001010)_2$ in Fig. 2(a). The exponent is divided into two

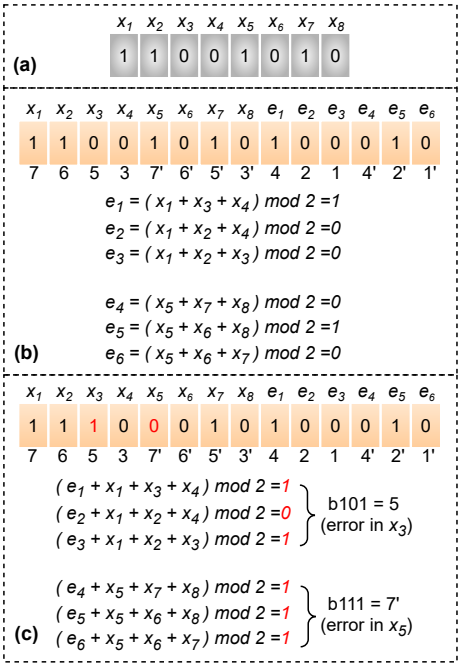


Fig. 2: Error detection and correction in exponent-bits of SERA-Float. e_1 to e_3 protect the left half and e_4 to e_6 protect the right half of the original 8-bit exponent. (a) Original 8-bit exponent; (b) 14-bit encoded exponent of SERA-Float; (c) example of error detection and correction at x_3 and x_5.

4-bit chunks: left-half and right-half. Each chunk is protected using a (7,4) hamming code, which adds 3 parity bits to 4 data bits, resulting in a 7-bit encoding for each chunk. Combining both encoded halves gives a 14-bit encoded exponent. The (7,4) Hamming code can correct 1-bit flips within each half, and the 14-bit encoding can correct up to 2-bit flips, provided the flips occur in separate halves. Since 2-bit flips are rare (0.02% to 0.08% occurrence), the 1-bit error correction for each half is sufficient for error recovery in typical scenarios.

Example of Error Correction: For the exponent bits $(x_1, x_2.....x_8)$, the error-correcting bits $(e_1, e_2...., e_6)$ are computed on different parts of the data bits, as shown in Fig. 2(b). We have used even parity, which means the error-correcting bit is logical 1, if the number of 1's in the exponent bit chunk is odd; otherwise, it is 0. In Fig. 2(c), we showcase the error correction example when two 1-bit flips (soft errors) occur at position x_3 and x_5, respectively. To detect the error, we perform a parity checkover and get the exact location of the error as '101' and '111', respectively. Once the positions of the faulty bits are known, they can be corrected.

Compressed Mantissa (8 bits): To efficiently compress a 23-bit floating-point mantissa into an 8-bit representation, we employ a structured segmentation and selection-based approach. Initially, the 23-bit mantissa is divided into 7 blocks of 3 bits each, starting from the MSB, with the remaining bits forming the 8^{th} block, which is shorter. The next step involves identifying the first non-zero block from the left-hand side. This block is denoted as the *first valid block*. Hence, we use the first valid block and the two subsequent blocks to form the compressed 8-bit mantissa. Specifically, 3 bits are taken from each of the first valid block and the next block; two bits are taken from the third block (see the example below). The least significant bit (LSB) of the third block undergoes bit-OR with the second LSB of the same block to compensate for the error induced due to compression.

If a valid block is not found within the first six blocks (positions 8 to 2), the mantissa is considered to be fully compressed to zero. The final 8-bit compressed mantissa is then used in subsequent floating-point operations, with a significant reduction in storage and computation costs. For clarity, an illustrative example of mantissa compression is presented below.

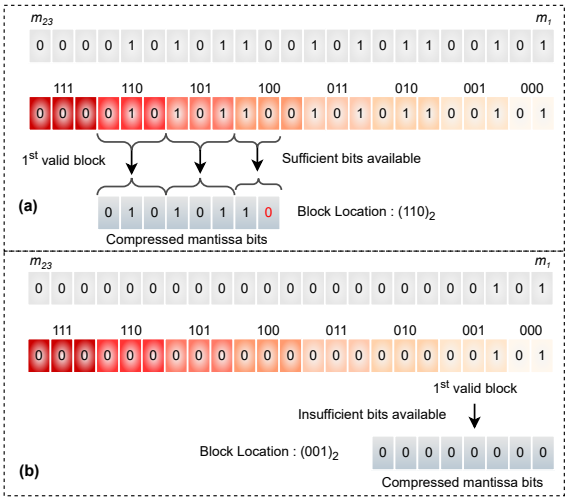


Fig. 3: Illustrative example of mantissa compression in SERA-Float: (a) for position of first valid block being greater than 1, and (b) other cases.

Example of Mantissa Compression: Fig. 3 presents an illustrative example of mantissa compression. In Fig. 3(a), the first valid block of the 23-bit mantissa is identified at position 6. Correspondingly, the bits of the identified *first valid block*, the block adjacent to it, and the following two bits are selected as the compressed 8-bit mantissa. In Fig. 3(b), the first valid block is identified at position 1, resulting in the compressed mantissa set to all zeros.

Control Signal (7 bits): The control signal is designed to protect the sign bit and the position of the first valid block of mantissa bits. Since the position of the first valid block requires 3 bits, and the additional sign bit brings the total to 4 bits, we employ a (7,4) hamming code scheme to generate the control signal. Note that while the position of the first valid block can range from 0 to 7, the meaningful positions are constrained to the range between 2 and 6 (counted from the right-hand side). Consequently, we assign a unique control signal corresponding to the position of the first valid block and the sign bit, as detailed in Table V. Additionally, for the edge cases where the position of the first valid block is $(000)_2$ or $(001)_2$, a separate control signal is provided. This approach ensures that the control signal provides an additional layer of reliability, safeguarding critical components of the floating-point number, such as the sign bit and the position of the first valid block, which determines the significant mantissa bits.

Identifier Bits (2 bits): The identifier bits (I-bits) provide information regarding the category of the floating-point number: i) normal numbers, ii) underflow, iii) overflow, and iv) NaN (Not a Number) conditions. I-bits are initialized before execution and updated at runtime using Equation (2).

$$I_0 = BitAND(x_i)(s + bitOR(m_i)) \qquad (2)$$
$$I_1 = BitAND(x_i)(\overline{s} + \overline{bitOR(m_i)})$$

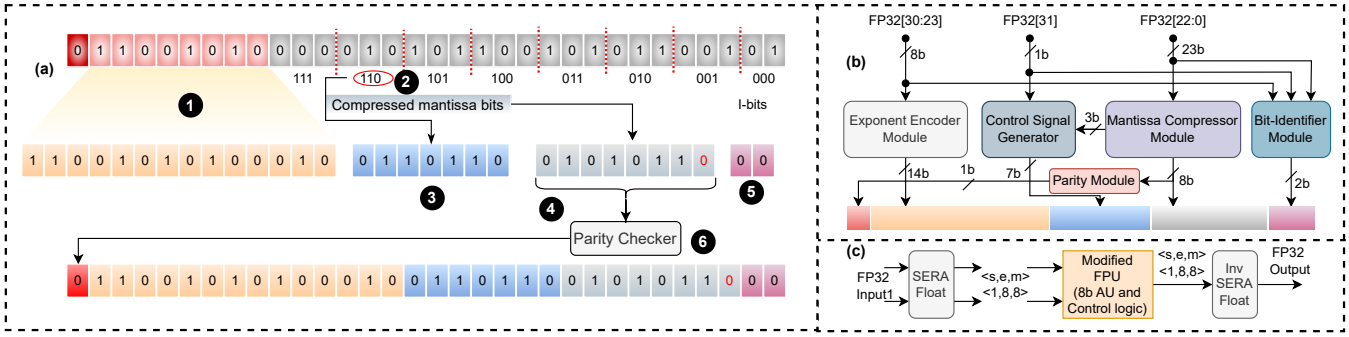


Fig. 4: (a) Illustration of SERA-Float conversion process, (b) Block diagram of conversion modules and, (c) SERA-Float based floating-point computing framework.

TABLE V: Sign bit, position of first valid block, and control signal

Mantissa with sufficient bits available					
Sign bit	Position of first valid block	Control signal	Sign bit	Position of first valid block	Control signal
1	010	1010 010	0	010	0010 101
1	011	1011 100	0	011	0011 011
1	100	1100 100	0	100	0100 011
1	101	1101 010	0	101	0101 101
1	110	1110 001	0	110	0110 110
1	111	1111 111	0	111	0111 000
Mantissa with insufficient bits available					
1	001/000	1000 000	0	001/000	0100 000

I_0 and I_1 represent the least significant bit and most significant bit, respectively. $BitAND(x_i)$ corresponds to the bit-wise AND operation applied to all exponent bits, 's' denotes the sign bit, and $BitOR(m_i)$ represents the bit-wise OR operation applied to all mantissa bits. In this way, the 2-bit identifier provides information about the type of number stored, as depicted in Table VI.

TABLE VI: 2-bit Identifier for Floating-point Numbers

2-Bit Identifier	Description
00	Normal Number (neither NaN nor overflow/underflow)
01	Underflow (the number is too small to be represented)
10	Overflow (the number exceeds the maximum value)
11	NaN (Not a Number)

C. Conversion process

Fig. 4(a) illustrates the conversion process of a 32-bit IEEE 754 FP32 number to the *SERA-Float* format. Initially, the 8-bit exponent is protected using 6-bit error-correcting code ①. The chosen error-correction scheme is capable of correcting up to two 1-bit flips. Next, the mantissa bits are divided into blocks of 3 bits, and the position of the first valid block is determined ②. This first valid block position, in combination with the sign bit, is used to generate the 7-bit control signal ③. The data bits within the valid block, along with the data from two subsequent blocks, are then used to form the compressed mantissa bits ④. Finally, identifier bits are appended to define the number type ⑤, and a parity bit is included to protect the compressed mantissa bits ⑥. This results in the final 32-bit output.

IV. Experimental Evaluation

This section initially presents a comparative analysis of *SERA-Float* with other formats such as *FP8(E3M4), FP16*, and *BF16* with respect to precision and exception triggering. Then, the *SERA-Float* format performance is evaluated for five applications. For a comprehensive performance evaluation comparison, the *INT8* format is also included due to its widespread adoption in DNN inference. Finally, the hardware implementation results are reported.

A. Format Comparison

We perform a comparative analysis of the *SERA-Float* against two well-known 16-bit compressed formats, *BF16* and *FP16*, as well as the conventional *FP32* format. Note that, although *SERA-Float* is a 32-bit format, we have selected two 16-bit formats for comparison because in *SERA-Float* format 17 data bits are used, i.e., the sign bit, exponent bits (8 bits), and compressed mantissa bits (8 bits). Also, we have included two low-precision formats, *FP8(E3M4) and INT8* due to their widespread adoption in machine learning tasks. We have generated $num = 10^6$ *FP32* floating-point numbers following a uniform distribution, with values lying within the maximum and minimum range of the format. These numbers are then converted from *FP32* to *INT8, FP8, FP16, BF16*, and *SERA-Float* format, respectively. Note that INT8 conversions are enabled using the quantization technique described in [22], wherein the scaling factor is updated every 100 iterations. After conversion, the numbers are reverted back to the *FP32* format, and the error in precision is recorded using two metrics: Mean Relative Error Difference (MRED) and Maximum Relative Error Difference (MaxRED) as defined in Equations (3) and (4), respectively.

$$MRED = \frac{1}{num}\sum_{i=1}^{num}\left|\frac{(N_{FP32_i} - N_{x_i})}{N_{FP32_i}}\right| \times 100 \quad (3)$$

$$MaxRED = max\left\{\left|\frac{(N_{FP32_i} - N_{x_i})}{N_{FP32_i}} \times 100\right|\right\},\ i\epsilon\,[1, num] \quad (4)$$

In Equations (3) and (4), N_{FP32_i} represents the number expressed in *FP32* format, whereas N_{x_i} denotes the number stored in one of the selected formats. During analysis, we considered both fault-free and faulty scenarios. In the first scenario, the floating-point numbers are processed without introducing any soft errors. The conversion from *FP32* to the selected format and vice versa is performed under normal conditions to evaluate the precision loss due to the compression and decompression operations. The second scenario considers soft errors introduced during the storage. Specifically, bit-flips are injected into the considered formats and *SERA-Float* representation, following four error rates corresponding to two fault models as depicted in Table IV. The bit flips can occur at any bit location in the entire bit width. For the second scenario, we measure the precision error and the number of exceptions under multiplication, which is one of the dominant operations in DNN.

Precision: Table VII presents the baseline MRED and MaxRED in the absence of soft errors. *SERA-Float* shows an MRED of

TABLE VII: Precision comparison of SERA-float, FP16, BF16, FP8(E3M4) and INT8 (lower is better)

Event	MRED (in %)					MaxRED (in %)				
	SERA-Float	FP16	BF16	FP8	INT8	SERA-Float	FP16	BF16	FP8	INT8
No Soft Error	0.11	0.08	0.13	0.18	0.25	2.5	1.9	3.1	3.8	4.3
1% 1-bit flip (F1E1)	0.14	6.1×10^5	9.4×10^6	1.0×10^7	1.5×10^7	39.5	7.3×10^7	1.3×10^8	9.0×10^8	1.5×10^9
2% 1-bit flip (F1E2)	0.15	6.6×10^5	9.7×10^6	1.4×10^7	1.7×10^7	39.8	9.4×10^7	2.9×10^8	1.1×10^9	1.7×10^{10}
3% 1-bit flip (F1E3)	0.18	7.4×10^6	8.9×10^7	1.2×10^8	9.5×10^8	41.2	3.6×10^8	6.9×10^9	7.3×10^9	8.0×10^{10}
4% 1-bit flip (F1E4)	0.20	8.3×10^8	7.2×10^9	1.9×10^{10}	8.0×10^{10}	44.6	9.3×10^{10}	3.8×10^{11}	5.2×10^{11}	9.6×10^{11}
1% 1-bit flip and 0.02% 2-bit flip (F2E1)	0.32	3.8×10^9	4.1×10^{10}	9.2×10^{10}	4.5×10^{11}	48.4	2.3×10^{11}	4.3×10^{12}	5.2×10^{12}	8.0×10^{12}
2% 1-bit flip and 0.04% 2-bit flip (F2E3)	0.33	4.2×10^9	4.5×10^{10}	9.8×10^{10}	6.0×10^{11}	50.0	2.5×10^{11}	4.7×10^{12}	7.2×10^{12}	9.5×10^{12}
3% 1-bit flip and 0.06% 2-bit flip (F2E3)	0.35	4.8×10^9	5.2×10^{10}	1.1×10^{11}	9.7×10^{11}	51.5	2.8×10^{11}	5.0×10^{12}	8.3×10^{12}	1.6×10^{13}
4% 1-bit flip and 0.08% 2-bit flip (F2E4)	0.39	5.0×10^{10}	9.5×10^{11}	7.3×10^{11}	5.9×10^{12}	53.0	3.0×10^{11}	6.2×10^{12}	3.3×10^{13}	6.5×10^{13}

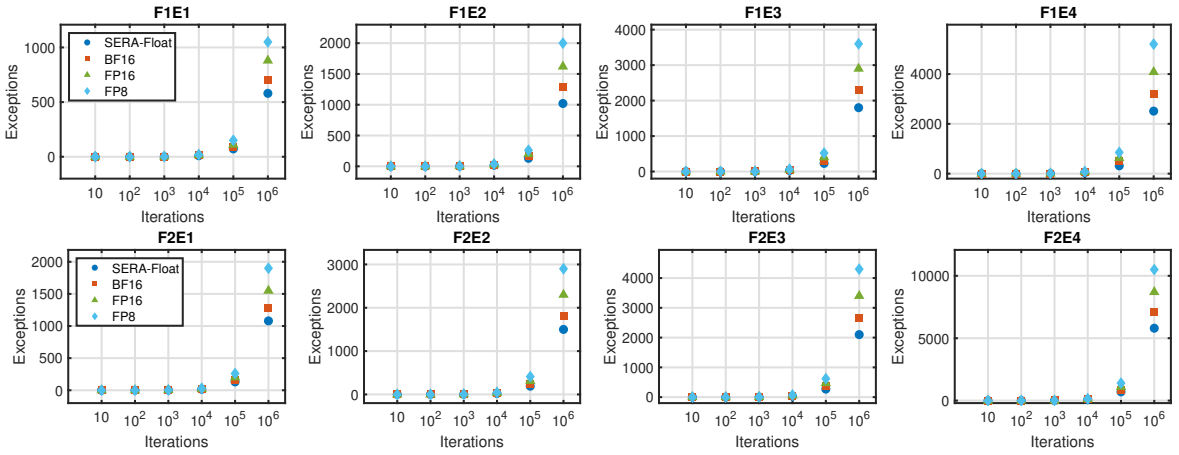


Fig. 5: Total exceptions in the presence of soft errors: Fault Model #1 [Top], Fault Model #2 [Bottom] (lower is better)

0.11% and MaxRED of 2.5%, indicating high precision with minimal conversion error. As *FP16* accounts for 10 mantissa bits, it has slightly lower MRED (0.08%) and MaxRED (1.9%), offering better precision, but at the cost of a limited value range. *BF16* shows a higher MRED (0.13%) and MaxRED (3.1%) compared to the other formats, allowing a trade-off between precision and other benefits. *FP8* shows the MRED (0.18%) and MaxRED (3.8%) due to its narrow range and only 4-bits in the mantissa representation. Further, *INT8* has highest MRED and MaxRED due to quantization as well as conversion error. Overall, despite mantissa compression, *SERA-Float* maintains precision when compared with other 16-bit formats.

The bottom part of Table VII shows the obtained results under the faulty scenario and the four different error rates corresponding to two fault models, i.e., (F1E1-F1E4) and (F2E1-F2E4). When soft errors are introduced, each floating-point format exhibits varying degrees of resilience depending on the severity of the soft-error rate. For instance, under a 1% 1-bit flip, *SERA-Float* demonstrates minimal impact, with MRED at 0.14% and MaxRED at 39.5%. In contrast, *FP16* and *BF16* experience substantial degradation, with MRED values of 6.1×10^5 and 9.4×10^6, and MaxRED values of 7.3×10^7 and 1.3×10^8, respectively. Similarly, in the case of a 2% 1-bit flip, *SERA-Float* shows only a slight increase in MRED (0.15%) and MaxRED (39.8%). However, *FP16* and *BF16* exhibit significant increases in the measured errors, with MREDs of 6.6×10^5 and 9.7×10^6, and MaxREDs of 9.4×10^7 and 2.9×10^8, respectively. Further, *FP8 and INT8* formats are worse than 16-bit formats. Similar trends are observed for the 4% 1-bit flip and 1-bit combined with 2-bit fault injection scenarios (F2E1 - F2E4). Overall, the obtained results show that *SERA-Float* maintains data integrity under increasing error rates, whereas *INT8, FP8, FP16* and *BF16* exhibit a significant loss of accuracy as the error rate rises.

Exception Triggering: Fig. 5 shows the number of exceptions that occurred due to soft errors during multiplication of *num* pairs of floating-point numbers. The experiments were conducted with the SER for F1E1 to F1E4 (Fig. 5 [Top]) and the SER for F2E1 to F2E4 (Fig. 5 [Bottom]), tuning *num* from 10 up to 10^6. The results indicate that for *num* = 10, 100, 1000, and 10,000, the number of exceptions is low. However, for *num* = 100,000 and 1,000,000, the number of exceptions becomes significant. Furthermore, we observe that the *FP16* format exhibits 1.6× more exceptions than *SERA-Float* and 1.2× more exceptions than the standard *FP32* floating-point representation. This suggests that soft errors contribute noticeably to the total number of exceptions triggered. As shown in Fig. 5, the total number of exceptions in the case of *SERA-Float* is the lowest, demonstrating that it is effective in mitigating soft-error-induced exceptions.

B. Application Evaluation

We evaluate the accuracy obtained with the floating-point formats *SERA-Float*, *FP16, BF16, FP8*, and *INT8* for five applications in image processing and machine learning: Image Sharpening (IS), Gaussian Smoothing (GS), K-Nearest Neighbors (KNN), Deep Neural Networks (DNN), and Support Vector Machines (SVM), under fault-free and faulty scenarios. The baseline is computed using FP32.

For Image Sharpening, we apply the unsharp mask technique to enhance image edges, using a female portrait as input. The quality

TABLE VIII: Application-level results of FP16, BF16, FP8(E4M3), INT8 and SERA-Float with different soft errors (higher value is better). Left half: F1 (1-bit Flips); Right half: F2 (1-bit + additional 2-bit Flips).

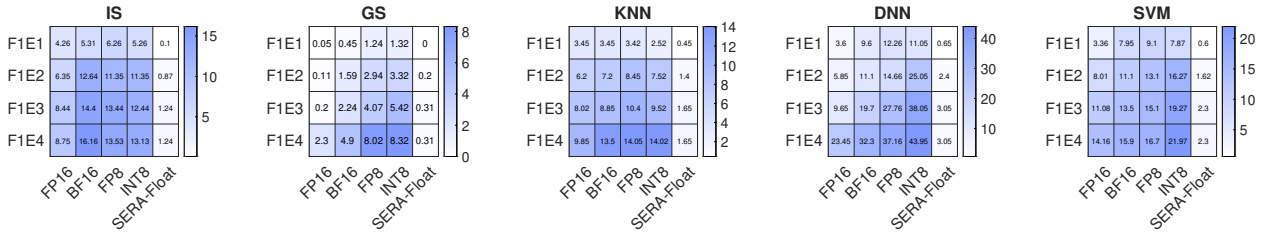
Fault Model #1 (Single-bit Flips)							Fault Model #2 Multiple-bit Flips						
Application		IS	GS	KNN	DNN	SVM	Application		IS	GS	KNN	DNN	SVM
Error Metric		PSNR (higher better)		Accuracy (higher better)			Error Metric		PSNR (higher better)		Accuracy (higher better)		
Baseline (FP32)		38.70	33.96	96.60%	98.85%	95.75%	Baseline (FP32)		38.70	33.96	96.60%	98.85%	95.75%
No soft Error	FP16	27.84	24.05	95.75%	98.10%	95.16%	No soft Error	FP16	27.84	24.05	95.75%	98.10%	95.16%
	BF16	27.84	24.05	94.50%	96.80%	93.70%		BF16	27.84	24.05	94.50%	96.80%	93.70%
	FP8	24.61	21.02	93.45%	95.26%	91.10%		FP8	24.61	21.02	93.45%	95.26%	91.10%
	INT8	23.16	20.82	91.02%	90.05%	88.27%		INT8	23.16	20.82	91.02%	90.05%	88.27%
	SERA-Float	33.05	24.91	95.25%	97.90%	94.55%		SERA-Float	33.05	24.91	95.25%	97.90%	94.55%
F1E1: 1% 1-bit Flips	FP16	23.58	24.00	92.30%	94.50%	91.80%	F2E1: 1% 1-bit + 0.02% 2-bit Flips	FP16	21.01	22.90	90.08%	92.10%	89.95%
	BF16	22.53	23.60	91.05%	87.20%	85.75%		BF16	20.20	22.50	89.10%	84.90%	84.02%
	FP8	18.35	19.78	90.03%	83.00%	82.00%		FP8	17.38	18.50	88.02%	80.27%	80.35%
	INT8	17.90	19.50	88.50%	79.00%	80.40%		INT8	16.50	18.00	86.50%	74.75%	78.00%
	SERA-Float	32.95	24.91	94.80%	97.25%	93.95%		SERA-Float	31.45	24.70	94.10%	95.85%	93.50%
F1E2: 2% 1-bit Flips	FP16	21.49	23.94	89.55%	92.25%	87.15%	F2E2: 2% 1-bit + 0.04% 2-bit Flips	FP16	19.17	22.01	88.02%	90.40%	83.95%
	BF16	15.20	22.46	87.30%	85.70%	82.60%		BF16	13.28	21.73	86.50%	82.16%	80.50%
	FP8	13.26	18.08	85.00%	80.60%	78.00%		FP8	11.50	15.59	82.18%	77.05%	75.14%
	INT8	11.81	17.50	83.50%	65.00%	72.00%		INT8	10.30	15.46	80.07%	59.46%	63.50%
	SERA-Float	32.18	24.71	93.85%	95.50%	92.93%		SERA-Float	30.04	24.30	93.05%	92.16%	90.10%
F1E3: 3% 1-bit Flips	FP16	19.40	23.85	87.73%	88.45%	84.08%	F2E3: 3% 1-bit + 0.06% 2-bit Flips	FP16	16.20	21.00	85.80%	82.02%	83.05%
	BF16	13.44	21.81	85.65%	77.10%	80.20%		BF16	11.50	18.43	83.92%	71.80%	78.90%
	FP8	11.17	16.95	83.05%	67.50%	76.00%		FP8	9.80	14.17	81.50%	63.10%	75.02%
	INT8	10.72	15.40	81.50%	52.00%	69.00%		INT8	8.05	13.48	78.43%	48.25%	61.10%
	SERA-Float	31.81	24.60	93.60%	94.85%	92.25%		SERA-Float	29.85	23.30	92.04%	92.70%	89.75%
F1E4: 4% 1-bit Flips	FP16	19.09	21.75	85.90%	74.65%	81.00%	F2E4: 4% 1-bit + 0.08% 2-bit Flips	FP16	13.72	19.72	82.15%	72.90%	79.50%
	BF16	11.68	19.15	81.00%	64.50%	77.80%		BF16	09.44	18.01	78.95%	57.68%	73.42%
	FP8	11.08	13.00	79.40%	58.10%	74.40%		FP8	09.40	12.18	75.05%	54.05%	71.07%
	INT8	10.03	12.50	77.00%	46.10%	66.30%		INT8	07.50	10.36	73.50%	39.50%	59.30%
	SERA-Float	31.81	24.60	93.60%	94.85%	92.25%		SERA-Float	29.50	23.23	91.50%	88.05%	87.40%

of the sharpened images is measured using Peak Signal-to-Noise Ratio (PSNR) [23]. In Gaussian Smoothing, a Gaussian blur is applied to reduce noise and detail, and the quality is measured using PSNR with the same portrait image. For KNN, we evaluate classification accuracy using the CIFAR-100 dataset, consisting of 100 classes and 600 images per class. The model is tested with $k = 5$. In DNN, we use the pre-trained ResNet-50 model, trained on CIFAR-100, and perform inference using the selected floating-point formats. Finally, for SVM, we train a linear kernel classifier on CIFAR-100 and evaluate accuracy under both error-free and soft-error conditions. Note that we have employed Fisher vectors as our feature representation for SVM.

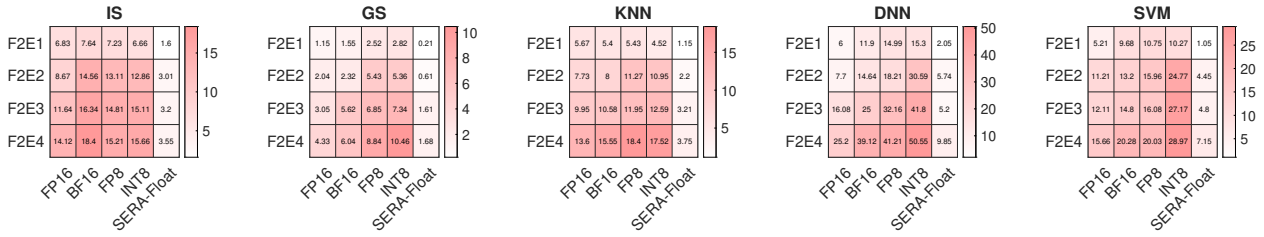
Table VIII illustrates how different floating-point formats experience a reduction in the quality of results with and without faults. As the *SERA-Float* format is designed with error resilience in mind, it maintains higher accuracy under faults across all applications, particularly in tasks like DNN and image processing. In the absence of soft errors, the *FP16* format demonstrates higher metric values, which can be attributed to its 10-bit mantissa, offering higher precision compared to the other formats. Specifically for DNN, while *FP16* shows an accuracy degradation of 0.7%, *SERA-Float* shows an accuracy degradation of 0.9%. However, as the soft error events increase (F1E1 to F1E4 and F2E1 to F2E4), performance degradation in all formats is inevitable. These formats, which lack built-in error-correction mechanisms, show significant performance degradation under higher error rates. For instance, in the case of DNN, the accuracy of *BF16* drops from 96.80% in the no-soft-error condition to 57.68% under F2E4. Similarly, for *FP16*, accuracy drops from 98.10% to 72.90% under F2E4. In contrast, *SERA-Float* shows a more modest drop in accuracy, from 97.90% to 88.05% (for F2E4), highlighting its robustness in error-prone environments. Fig. 6 depicts the absolute reduction in accuracy (%) in the presence of soft errors. As shown in Fig. 6, while *SERA-Float* maintains application performance, other formats experience a drastic drop in accuracy and image quality (qualitative comparison shown in Fig. 7). This emphasizes the importance of using robust floating-point formats, such as *SERA-Float*, in environments where soft errors are prevalent or where precision is critical.

C. Hardware evaluation

For hardware-level evaluation, we have implemented the *SERA-Float* conversion module, as depicted in Fig. 4(b), in Verilog and synthesized it using Synopsys Design Compiler with the NanGate 45nm Open Cell Library. The design was synthesized under the operating conditions of a clock period of 1.0 ns, a supply voltage of 1.10 V, and a temperature of 25.0°C. Subsequently, we performed the multiplication of two floating-point numbers: (a) first using the conventional FP32 multiplier and (b) then using the computation framework shown in Fig. 4(c). Fig. 4(b) illustrates the 32-bit FP32 to 32-bit *SERA-Float* conversion module, which is comprised of four submodules: the exponent encoder, control signal generator, mantissa compressor, and bit identifier. The exponent encoder implements a (7,4) Hamming code logic, while the other submodules are designed as described in Section III. Table IX presents the hardware metrics for the FP32 multiplier (Baseline) and the multiplication operation



(a) Fault Model #1: Single-bit Faults



(b) Fault Model #2: Multiple-bit Faults

Fig. 6: Quality of Results (QoR) degradation across various applications for Fault Model #1 (top) and Fault Model #2 (bottom). Lighter shades indicate lower QoR degradation, while darker shades represent higher degradation. Among all formats, *SERA-Float* consistently exhibits the lowest QoR degradation, highlighting its resilience under both fault models.

Fig. 7: Image sharpening evaluation results for a female portrait under the F2E2 fault model, involving 2% 1-bit flips and a 0.04% 2-bit flips, across various data formats.

TABLE IX: Hardware evaluation results

Technique	Area ($mu\ m^2$)	Power (mW)	Delay (ns)	Energy (pJ)
Baseline	7690	0.15	3.60	0.54
SERA-Float	7386	0.09	1.18	0.10

enabled by *SERA-Float* conversion module. As shown in Table IX, we observe up to 80.3% energy savings through the use of *SERA-Float*. Notably, these savings are achieved because both the mantissa and exponent computations are handled by the 8-bit arithmetic unit, eliminating the need for higher-bit arithmetic in the mantissa computation.

D. Single-bit flip worst-case error analysis

In this subsection, we evaluate the worst-case error caused by single-bit flips. Although our study considers two fault models, we focus here exclusively on single-bit flip faults, as the second model, characterized by lower soft error rates for double-bit upsets, has a comparatively negligible impact in the worst-case analysis. Table X highlights the worst-case absolute errors resulting from single-bit flips across selected FP formats, including *FP32, BF16, FP16, FP8, INT8*, and the proposed *SERA-Float*. Table X captures two critical error scenarios for floating-point formats: (i) maximum error (*MaxE*), representing the transition from a benign value to a special value$\pm\infty$, and (ii) maximum finite error (*MaxFE*), denoting the maximum finite numeric deviation caused by a single-bit error. Table X lists the hexadecimal representations of the original value (Before Soft Error) and the corrupted value (After Soft Error) for various formats,

TABLE X: Worst-Case Absolute Errors for Single-bit Flips

Format	Error Metric	Before Soft Error (in hexa-decimal)	After Soft Error (in hexa-decimal)	Value
IEEE FP32	MaxE	0X3F800000	0X7F800000	$\pm\infty$
	MaxFE	0X7F7FFFFF	0XFF7FFFFF	6.8×10^{38}
BF16	MaxE	0X3F80	0X7F80	$\pm\infty$
	MaxFE	0X7F7F	0XFF7F	3.4×10^{38}
FP16	MaxE	0X3C00	0X7C00	$\pm\infty$
	MaxFE	0X7BFF	0XFBFF	1.3×10^{5}
FP8	MaxE	0X30	0X70	$\pm\infty$
	MaxFE	0X3F	0XEF	31
INT8	MaxFE	0X80	0X00	128
SERA-Float	MaxFE	0X7F7FFFFF	0X7F3FFFFF	0.9×10^{38}

along with the corresponding absolute error magnitude (Value). For example, in IEEE FP32, a bit flip in the exponent field can change 0x3F800000 (representing 1.0) to 0x7F800000 ($+\infty$), while a flip in the most significant mantissa bits leads to a maximum finite deviation of approximately 6.8×10^{38}. Similar exception-triggering behavior is seen in *BF16, FP16, and FP8. INT8*, lacking special values and having a limited range, exhibits a maximum absolute error of 128 due to a bit flip in the sign bit. In contrast, *SERA-Float* avoids transitions to non-numeric exceptions and confines the worst-case finite error to 0.9×10^{38} ($7.5\times$ less than FP32 and BF16), demonstrating its improved resilience against soft errors without sacrificing representational range.

V. Conclusion

This paper introduces *SERA-Float*, a novel storage format for approximate floating-point numbers designed to be resilient against soft errors. *SERA-Float* protects critical bits by enabling up to 2-bit error detection and correction for the exponent and 1-bit error detection for the mantissa. It also compresses the 23-bit mantissa into 8 bits without losing data integrity. Experimental results demonstrate that *SERA-Float* enhances resilience against soft errors and provides up to 80.3% energy savings in floating-point operations, making it a practical solution for real-world applications.

REFERENCES

[1] W. Liu *et al.*, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[2] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, pp. 1–33, 2016.

[3] S. Jain *et al.*, "Compensated-dnn: Energy efficient low-precision deep neural networks by compensating quantization errors," in *Proceedings of the 55th annual design automation conference (DAC)*, 2018, pp. 1–6.

[4] V. Mrazek *et al.*, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *DATE*, 2017, pp. 258–261.

[5] S. Ullah *et al.*, "Smapproxlib: Library of fpga-based approximate multipliers," in *Proceedings of the 55th Annual Design Automation Conference (DAC)*, 2018, pp. 1–6.

[6] M. Traiola, S. Pappalardo, A. Piri, A. Ruospo, B. Deveautour, E. Sanchez, A. Bosio, S. Saeedi, A. Carpegna, A. B. Göğebakan, E. Magliano, and A. Savino, "Approximate fault-tolerant neural network systems," in *2024 IEEE European Test Symposium (ETS)*, 2024, pp. 1–10.

[7] A. Vijayan *et al.*, "Online soft-error vulnerability estimation for memory arrays and logic cores," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 37, no. 2, pp. 499–511, 2017.

[8] Y. Wang *et al.*, "Ftapprox: A fault-tolerant approximate arithmetic computing data format," in *DATE*, 2021, pp. 1548–1551.

[9] V. Mishra *et al.*, "Vadf: Versatile approximate data formats for energy-efficient computing," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 22, no. 5s, pp. 1–21, 2023.

[10] M. Hashimoto, K. Kobayashi, J. Furuta, S.-I. Abe, and Y. Watanabe, "Characterizing sram and ff soft error rates with measurement and simulation," *Integration*, vol. 69, pp. 161–179, 2019.

[11] G. P. Saggese, N. J. Wang, Z. T. Kalbarczyk, S. J. Patel, and R. K. Iyer, "An experimental study of soft errors in microprocessors," *IEEE micro*, vol. 25, no. 6, pp. 30–39, 2005.

[12] IEEE, "Ieee standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, 2008.

[13] N. Burgess *et al.*, "Bfloat16 processing for neural networks," in 26^{th} *Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 88–91.

[14] P. Micikevicius *et al.*, "Fp8 formats for deep learning," *arXiv preprint arXiv:2209.05433*, 2022.

[15] M. Yousefloo and O. Akbari, "Design exploration of fault-tolerant deep neural networks using posit number representation system," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 32, no. 7, pp. 1350–1363, 2024.

[16] W. Kahan, "Ieee standard 754 for binary floating-point arithmetic," *Lecture Notes on the Status of IEEE*, vol. 754, no. 94720-1776, p. 11, 1996.

[17] A. Agrawal *et al.*, "Dlfloat: A 16-b floating point format designed for deep learning training and inference," in 26^{th} *Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 92–95.

[18] M. Van Baalen, A. Kuzmin, S. S. Nair, Y. Ren, E. Mahurin, C. Patel, S. Subramanian, S. Lee, M. Nagel, J. Soriaga *et al.*, "Fp8 versus int8 for efficient deep learning inference," *arXiv preprint arXiv:2303.17951*, 2023.

[19] S. S. Mukherjee, J. Emer, and S. K. Reinhardt, "The soft error problem: An architectural perspective," in *11th International Symposium on High-Performance Computer Architecture*. IEEE, 2005, pp. 243–247.

[20] S. S. Mukherjee, J. Emer, T. Fossum, and S. K. Reinhardt, "Cache scrubbing in microprocessors: Myth or necessity?" in *10th IEEE Pacific Rim International Symposium on Dependable Computing, 2004. Proceedings.* IEEE, 2004, pp. 37–42.

[21] Z. Cao *et al.*, "High capacity data hiding scheme based on (7, 4) hamming code," *SpringerPlus*, vol. 5, pp. 1–13, 2016.

[22] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards unified int8 training for convolutional neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1969–1979.

[23] A. Hore and D. Ziou, "Image quality metrics: Psnr vs. ssim," in 20^{th} *international conference on pattern recognition*. IEEE, 2010, pp. 2366–2369.