NOMA: A Novel Reliability Improvement Methodology for 3-D IC-based Neuromorphic Systems

Ngo-Doanh Nguyen, *Member, IEEE,*[*] [†] Khanh N. Dang, *Member, IEEE,*[*] Akram Ben Ahmed, *Member, IEEE,*[‡] Abderazek Ben Abdallah, *Senior Member, IEEE,*[*] and Xuan-Tu Tran, *Senior Member, IEEE,*[†]

***Abstract*—Hardware is the foundation of trust for all real-time applications and is especially crucial in Artificial Intelligence (AI). The growing reliance on AI for various tasks has raised concerns about the reliability of the hardware. Extensive research has shown that hardware faults arising from the manufacturing process or device variation can significantly impact the precision and accuracy of AI applications. This also applies to 3-D IC (Three-Dimensional Integrated Circuit)-based Spiking Neural Networks (SNNs), despite their noise resilience, low-power, and low memory footprint advantages. This is because, in addition to the mentioned hardware faults, thermal dissipation on 3-D ICs can negatively affect the memory stacked on upper dies. Hence, this article proposes a methodology called NOMA (Network-of-Memory Architecture) to enhance the reliability of 3-D IC-based SNNs by replacing the defective critical synaptic weights in high-priority layers with lower-priority ones. The method is assessed on multiple memory-on-logic architectures with various memory technologies such as SRAM, eDRAM, STT-RAM, and RRAM. As a result, for the 45nm SRAM library, the accuracy degradation improves by roughly 56% at a BER of 0.05 with a timing overhead of 3.538μs and a power overhead of 390.796nJ.**

***Index Terms*—Spiking Neural Networks, 3-D IC-based Neuromorphic System, Fault Tolerance, Network-of-Memory**

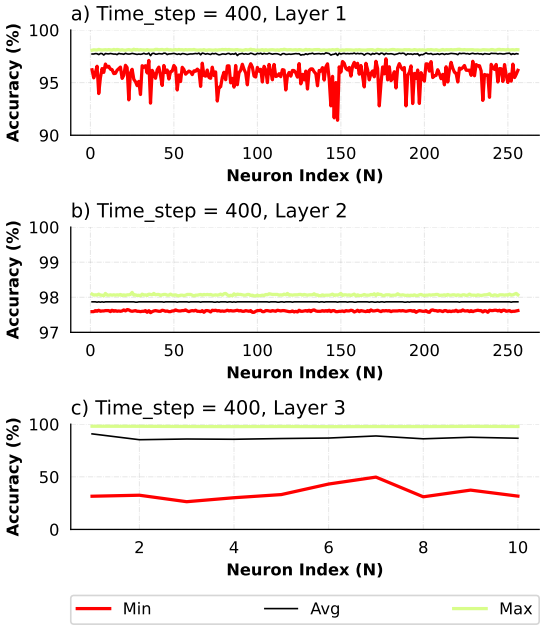

Fig. 1. The average accuracy with the faults occurring[1].

I. Introduction

THE combination of Spiking Neural Networks (SNNs) and Three-Dimensional Integrated Circuits (3-D ICs), 3-D IC-based SNNs, opens a new approach for Artificial Intelligence (AI) with multiple benefits. For example, SNNs introduce lightweight inferences with low-power characteristics [1]–[3], while 3-D IC-based technologies guarantee high bandwidth with high parallelism for effective computing [4], [5]. However, one of the biggest challenges of this approach is the low reliability caused by the 3-D ICs, such as aging, thermal and power issues, and manufacturing defects. Fortunately, SNNs can tolerate a certain number of those faults because of their noise resilience characteristics. In fact, with consideration of network topology, circuit implementation, hardware architecture, and fault-tolerance strategy, the failure of SNNs can be avoided when faults occur [6]–[9].

Although SNNs can tolerate faults, they are not always resilient against them, regardless of the type and location of the faults. To understand more about the impact of faults, especially memory faults, we experimented with fault injection into a multi-layer perception using the MNIST dataset classification, as depicted in Fig. 1[1]. In summary, this SNN model particularly has vulnerabilities when faults occur in the network's first and last layers, where it loses $2\% \sim 7\%$ and $65\% \sim 70\%$ of accuracy, respectively. In contrast, when faults occur in the middle layer(s) (Fig. 1 (b)), the SNN shows no considerable degradation in accuracy ($0.2\% \sim 0.5\%$ loss). In

[*] The Graduate School of Computer Science and Engineering, The University of Aizu, Aizu-Wakamatsu 965-8580, Fukushima, Japan. (e-mail: doanhnn@ieee.org; benab@u-aizu.ac.jp; khanh@u-aizu.ac.jp)

[†] The Information Technology Institute, Vietnam National University, Hanoi, 10000, Vietnam. (e-mail: tutx@vnu.edu.vn)

[‡] The Digital Architecture Research Center, National Institute of Advanced Industrial Sciences and Technology, 2-3-26 Aomi, Koto-ku, Tokyo, Japan. (e-mail: akram.benahmed@aist.go.jp)

This work is partly supported through the activities of VDEC, The University of Tokyo, in collaboration with NIHON SYNOPSYS G.K and Cadence Design Systems, and part by the Vietnam National University, Hanoi under Project No. QG.24.81. *(Corresponding author: Ngo-Doanh Nguyen.)*

[1]The faulty injection experiment covers every neuron in all layers (MLP $784 \times 256 \times 256 \times 10$, MNIST dataset, 400 time-steps). The SNN is trained using the ANN model and converted to SNN [10]. The weights are quantized into an 8-bit fixed point format, and faults are inserted as a flip-bit phenomenon. In this context, a soft-error rate of 50% for each neuron per run at multiple time steps. The accuracy of each faulty neuron is evaluated 1000 times in Monte Carlo simulations.

summary, fault impacts for SNN are still relevant, and we need to address them carefully.

To deal with memory faults in SNN, there are several existing works [11]–[15]. For instance, traditional fault-tolerance techniques, such as Triple Modular Redundancy (TMR), adding spares, and Error Correction Codes (ECCs), can be used [11], [12]. On the other hand, different approaches are proposed to deal with memory errors, especially non-volatile ones. In [14], Tosson *et al.* introduce a framework to manage soft errors in RRAM during the training phase of RRAM-based neuromorphic systems. Bhattacharjee *et al.* [13] explore the impact of RRAM crossbar non-idealities on the accuracy of SNNs. Anurup *et al.* [15] have proposed recursive linearised checks to detect synapse weight errors with high precision during inference. For the DRAM technology, Putra *et al.* [12] address soft error mitigation for digital SNN accelerators. It introduces the concept of fault-aware training and mapping (FATM) to address stuck-at-faults and low voltage-induced errors in memory. Although there are numerous works on addressing the reliability issue of SNN's memories, there are some existing challenges to be addressed:

- First, *conventional approaches are effective, but they have substantial hardware overhead*. This limitation also extends to memory-on-logic architectures, where thermal dissipation can adversely affect the memory stacked on upper dies farther from the heat sink [16], necessitating more hardware resources for fault correction. Moreover, 3-D IC-based neuromorphic systems have limited fault-tolerance solutions with low cost and power other than ECCs and alternations to maintain their low-power characteristics [17]. This underscores the pressing need for high reliability with low-power and low-cost solutions for 3-D IC-based neuromorphic systems.
- Second, *most of the aforementioned works only address the reliability problem in 2-D hardware architecture and are specific to particular memory technologies*. Meanwhile, 3-D hardware architectures and heterogeneous memory technologies are not addressed. Unlike 2-D ICs, 3-D ICs face much more significant reliability challenges due to manufacturing imperfections and high operating temperatures, which result in accelerated fault rates and mechanical failure.
- Third, *the existing approaches can be efficient in specific scenarios; however, they did not provide a comprehensive framework to adapt to different scenarios*. For instance, different SNN models have different fault impacts in different locations. On the other hand, different types of memory also have different area costs, read/write latency, and dynamic/static power consumption.

To tackle the challenges above, this article introduces NOMA (Network-of-Memory Architecture), a framework to enhance memory reliability in 3-D IC-based SNNs. The main contributions of this article are summarized as follows:

1) A novel methodology, called Network-of-Memory Architecture (NOMA), to tolerate the faults by reallocating synaptic weights between 3-D IC-based memories.
2) A comprehensive framework for priority-based reallocating synaptic weight of SNN that covers different fault models, fault rates, fault characteristics, and SNN architectures. Here, we classify synaptic weight memories into several categories based on the impact on overall performance during the occurrence of faults. Our framework focuses on reallocating the fault from critical locations into non-critical locations.
3) A trade-off evaluation between reliability and the hardware overhead across various memory technologies such as SRAM, eDRAM, STT-SRAM, and RRAM.

The rest of this article is organized as follows. Section II presents the background and related works for the fault tolerance of 3-D IC-based SNNs. Next, Section III introduces the challenges for the reliability of 3-D IC-based SNNs. Section IV explains the methodology and algorithm to improve the reliability. Section V evaluates the trade-off between the reliability improvement and the overhead in each memory technology and architecture. After that, Section VI discusses this work's challenges, remaining issues, and potential solutions. Finally, the article is concluded with summaries in Section VII.

II. Related Works

A. Fault Characterization for Neuromorphic Systems

The resilience of SNNs to faults varies depending on the specific training algorithms, and datasets used which have been reported in [8]. Contrary to popular belief, SNNs are not fully inherently resilient, and their ability to tolerate faults decreases significantly as fault rates rise. For example, Liu *et al.* [18] work on the fault-injection effect in memristor crossbars, showing that inference accuracy dropped by more than 50% with a stuck-at error rate of 20%.

On the other hand, Spyrou *et al.* [6] propose a method to accelerate fault injection in deep SNNs with a reused behavioral-level fault model [19]. The fault injection framework is built on SLAYER [20] and PyTorch [21] frameworks, where we can easily customize the faulty SNNs and map them onto a GPU. The overall conclusion of this experiment is that saturation neuron faults are the most detrimental and can significantly impact inference, regardless of the neuron's location in the network.

In [8], Schuman *et al.* investigated the impact of dead synapse faults in feed-forward SNNs trained using different algorithms. They found that the resilience of the SNNs depends primarily on the training algorithm. Another study by Zhu *et al.* [22] incorporated process variations and noise as random variables in neural network weights. At the same time, Gaol *et al.* [23] introduced a framework using a Bayesian neural network to train considering variations and defects, both for memristor crossbar-based architectures.

B. Fault Tolerance for Neuromorphic Systems

Putra *et al.* [12], [24] studied the fault tolerance of a Python-based SNN model using the flip-bit fault model, proposing fault-tolerance schemes (FAM and FATM) to mitigate memory failures. FAM identifies memory segments with non-faulty

cells, prioritizing the Most Significant Bits (MSBs) placement through circular shifting. FATM incorporates re-training to adapt accuracy to different bit-flip probabilities. On the other hand, our previous work [25] leverages 3-D neuromorphic systems to identify faults at multiple-bit levels and enhance operational resilience.

In 3-D Network-on-Chip (NoC) neuromorphic systems, potential faults may arise due to thermal imbalances resulting from neuron mapping. For example, the NASH system utilizes layer-to-layer mapping [26], leveraging the hardware's routing algorithm and the 3-D mesh topology. Dang *et al.* proposed migration methods named MigSpike [27], which utilize max-flow min-cut and genetic algorithms in response to these challenges. This strategy addresses the limitations of the previous proposal by introducing a fault-tolerant framework during neuron mapping. Despite the notable advantages of the MigSpike mapping framework, it does not support remapping if the number of faulty neurons exceeds the number of spare neurons.

III. Reliability Challenges for 3-D IC-based Neuromorphic Systems

This section outlines the challenges of implementing 3-D IC-based SNNs, including reliability issues related to thermal and electrical concerns, fault characterization, and locations. Although we do not address all of these challenges in the proposed methodology, it is important to highlight how 3D-IC adds another layer of complexity when implementing fault-tolerant methods to enhance the reliability in SNNs.

A. Thermal Issues in 3-D ICs

Thermal issues are the primary challenge encountered in 3-D ICs [28]. The increased integration leads to a more concentrated power supply for a given area, resulting in higher heat dissipation per unit footprint [29]. Consequently, the overall temperature of 3-D chips surpasses that of 2-D chips, potentially compromising system performance and reliability while introducing variability in circuit behavior. In this context, faults manifest as transient faults, where the elevated temperature alters a gate's Single Event Transient (SET) current, potentially causing pMOS transistors to activate and change the state of the gate output logic. This phenomenon is commonly referred to as a soft error in the circuits. The frequency of soft errors in the circuit, or soft error rate (SER), depends on the SET generation rate due to particle strikes and various masking probabilities along propagation paths. For example, Dang *et al.* [30] showed that the SER in 3D-NoCs at 80°C is much greater than the SER at 30°C (220×) and at 70°C (2.6×). Additionally, elevated temperature can induce instability in the threshold voltage, resulting in reduced transistor conductance and saturation current. This occurrence is called Negative Bias Temperature Instability (NBTI) [29]. For example, Lin *et al.* [31] showed that the difference in circuit delay (NBTI degradation) is larger than 3% when the temperature increases by 25°C (from 100°C to 125°C). In the case of memory-on-logic architecture, where memories are vertically stacked, memory reliability is compromised. For example, SRAMs exhibit diminished stability due to cell flipping caused by threshold voltage variation, erratic read operations, and increased cell access time.

B. Electrical Issues in 3-D ICs

The electrical challenges encountered within 3-D ICs predominantly stem from power delivery issues [29], particularly related to on-chip power supply noise. The increased power supply network (PSN) impedance, failing to keep pace with the growing device density and operating current, has led to heightened noise on-chip power supply. This is primarily due to limited wire resources and a consistent RC per wire length. The complexity is further exacerbated in 3-D ICs as the system necessitates a power supply network capable of delivering current through the power supply pins several times more than in 2-D chips. This is attributed to the reduced footprint of a 3-D die in comparison to a 2-D die, resulting in a significantly reduced number of power pins for the same circuit models, approximately one-third of the 2-D case. Moreover, Through-Silicon Vias (TSVs) compound the power supply problem by introducing additional resistance to the supply network. Consequently, the amplified supply noise yields a more pronounced operating speed variation, leading to increased timing violations in 3-D ICs. Furthermore, supply noise overshooting due to inductive parasitism may exacerbate reliability issues such as Time Dependent Dielectric Breakdown (TDDB), Hot Carrier Injection (HCI), and NBTI [29]. In the memory-on-logic architecture, the power supply to memory cells could significantly decrease to sub-threshold voltage levels based on the application's power budget. Consequently, the heightened DC supply noise induces variations in bit cells during read and write operations.

C. Challenges for Synapse Memory in 3-D IC-based SNNs

With the thermal and electrical issues, the 3-D IC-based neuromorphic systems become more vulnerable when faults occur [32], especially the memory-on-logic architectures. Since the synapse memories are stacked on top layers, the synapse weights for inference are then affected. This leads to a wrong classification or prediction of SNN, where the pre-trained weights are modified unexpectedly. For example, in the case of thermal hotspots, the values stored in SRAM memory cells around these regions can be modified because the NBTI effect shifts the threshold voltage of their transistors. In the case of non-volatile memory, thermal and electrical problems can cause a drifting effect, which also changes the stored values. On the other hand, the hotspots can also delay the path between memory blocks and processing elements because it increases the resistance of TSVs. This violates the timing constraints. Hence, the processing elements proceed to calculate with the wrong values. As a result, SNN has errors, which affects its final decision. However, depending on the location of faults, there are two types of errors, *dormant errors* and *critical errors* (explained in the following section). The *dormant errors* are the type of fault occurring in the system, but they do not change the system's outcomes. The *critical erros*, however, will cause the system's failure when they

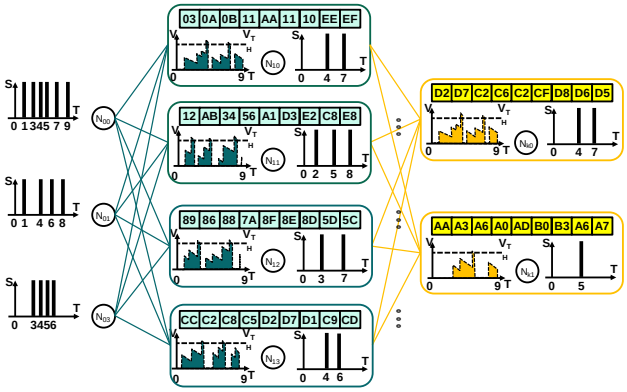

Fig. 2. An illustration of normal spiking neural network operations.

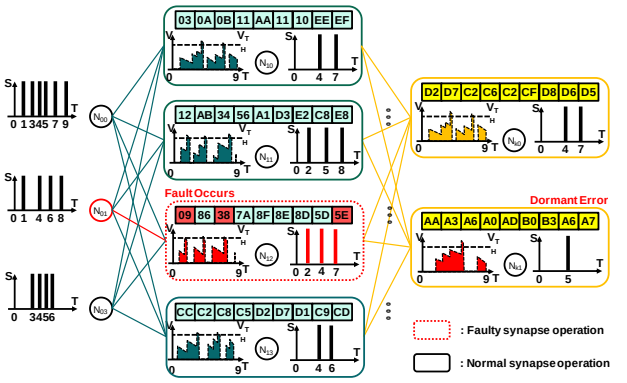

Fig. 3. An example of *dormant error* in spiking neural network operations.

appear in the system. The illustration of those affections is explained in the following section.

D. Different Effects of Faults in 3-D IC-based SNN

In this article, we focus on two main fault models: random flip bits and stuck-at faults caused by thermal and electrical issues.

- *Stuck-at*: This model demonstrates a data or control line stuck at either a high (*stuck-at-1*) or a low (*stuck-at-0*) state.
- *Random flip bits*: This model illustrates a data or memory element with an unpredictable and incorrect value.

Here, we select these two popular fault models for the evaluation; however, we would like to note that with different fault models, they can be easily integrated into the framework as in Section IV-B.

As shown previously in the Introduction section, the performance of a neural network is significantly impacted by the location of faults. For illustrative purposes, we here show two possible behaviors of the SNNs under faults. The SNN model is converted from an Artificial Neural Network (ANN-to-SNN conversion methodology) as a fully connected feed-forward neural network with RELU activation function. In the normal situation, the neuron operation using the Leaky Integrate-and-Fire (LIF) model [33] and rate coding is depicted in Fig. 2. Subsequently, Fig. 3 and 4 present the alternative operational outcomes of the SNN in the event of a fault.

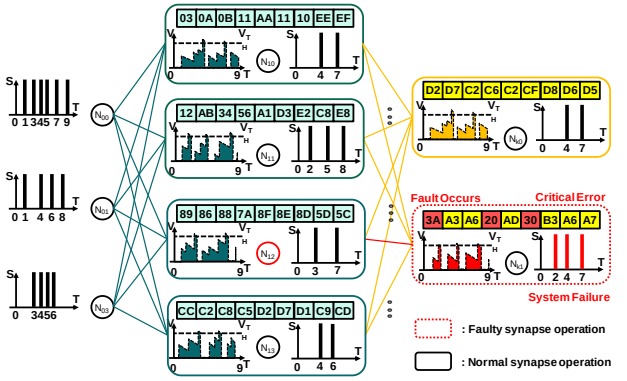

Fig. 4. An example of *critical error* in spiking neural network operations.

In the first case (Fig. 3), faulty bits occur in the neuron N_{12}; however, the SNN's performance remains unaffected as the number of output spikes of the SNN and their timing stay unchanged. This is because the fault occurs in the inactive neurons or the mostly zero-ed weight neurons. We call this type of fault *dormant errors* since the fault has an insignificant impact on the overall performance. Here, we could further relax the definition of *dormant errors* to some certain level of accuracy losses (i.e., 0.2-0.5% for the middle layer in Fig. 1) for better fault swapping performance.

In the second case (Fig. 4), the fault now occurs in the neuron N_{k1}. of the last layer, where the final decision is dependent on the number of output spikes. Here, we can observe that the output neuron N_{k1} fires three times, which outnumbers the expected firing neuron. In the rate coding method, this could lead to incorrect classification/prediction. Here, we call this type *critical errors*.

Accordingly, this article exploits this aspect *(dormant errors)* to enhance the neuromorphic system's fault tolerance by reordering synapse memories between the weights in the non-critical location. Our framework will analyze the characteristics of faults under Monte-Carlo simulation to decide what locations/neurons give *dormant errors* or *critical errors*.

IV. Reliability-Improvement Methodology for Network-of-Memory Architecture

This section introduces our proposed approach for achieving fault tolerance in NOMA. This work mainly focuses on the faults in synapse memories that occupy the major part of the neuromorphic chip.

A. The Network-of-Memory Architecture (NOMA)

The 3-D network-of-memory architecture (NOMA) extends our prior research efforts [34]. This architectural framework capitalizes on 3-D ICs, enabling the segmentation of synaptic weight memories into multiple bit levels across distinct dies. The hardware architecture in Fig. 5 delineates the logic and memory components into distinct layers. The logic components employ $L = 16$ LIF neurons [33] with a rate coding scheme chosen for their energy efficiency and bio-plausibility. Each memory layer comprises a subset of n-bit synaptic

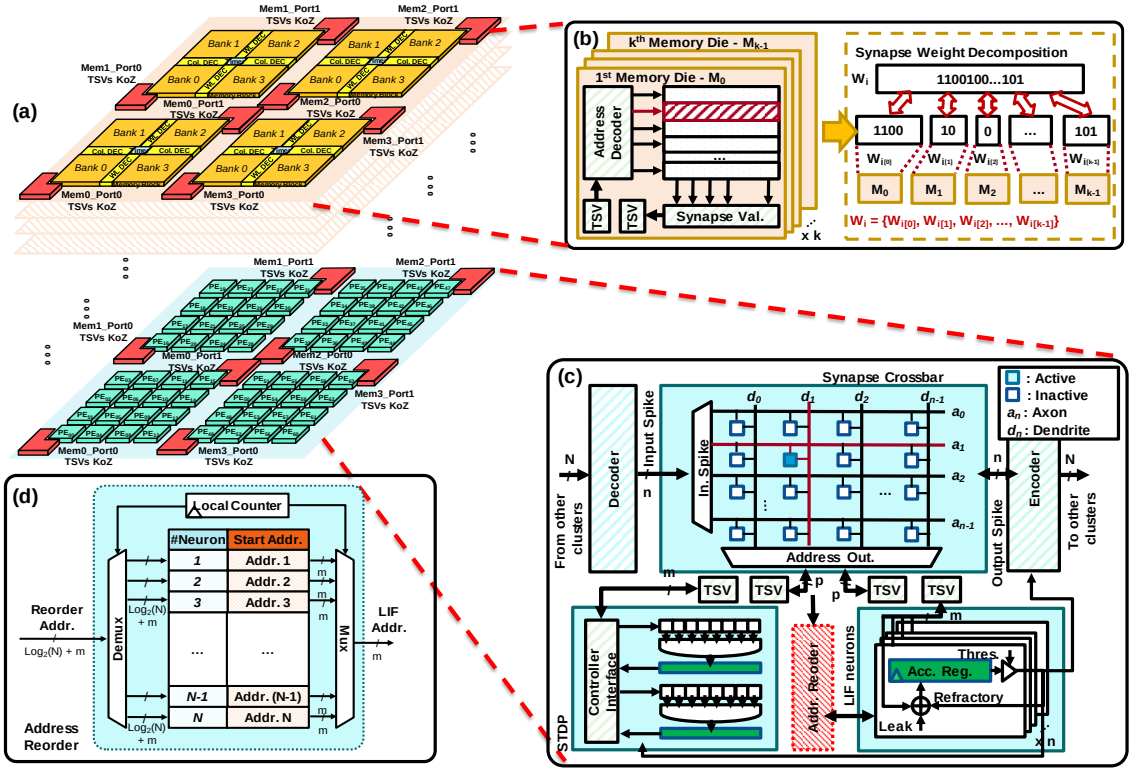

Fig. 5. An illustration of the 3-D memory-on-logic neuromorphic system based on our previous works [25] for fault tolerance. (a) Multiple memory layers are stacked on top of the logic layer. (b) Synopsis weights decomposition with 3-D stacking memory layers. (c) Hardware architecture of each processing element. (d) Address Reodering Module.

weights ($\{m_0, m_1,..., m_{M-1}\}$), denoted as m_i for the i^{th} subset and M for the total number of subsets or stacked layers. These numerical parameters remain configurable during the design phase. Moreover, the output of a neuron can be transmitted to other neurons within the same cluster or to those in different clusters. Its input triggers the crossbar to access the corresponding synaptic weight from the memory layers via TSV for LIF computations.

To support fault tolerance, our architecture relies on swapping the synaptic weight to shift *critical errors* into *dormant errors*. To do so, the address-reorder module is the alteration from our previous work designed to ensure accurate address extraction even in system faults, as shown in Fig. 5 (d).

This paper mainly considers the above stacking memory-based SNN architecture. However, there are other SNN architectures, such as pure 3D-IC or 2D-IC. Adapting to these topologies only requires modifications to memory architectures, resulting in different converted graphs and solutions. Nevertheless, the central principle of NOMA is still the same and could be widely applied.

B. Proposed framework

Fig. 6 shows the proposed NOMA framework. First, the SNN model, design constraints, and memory architecture are inputted into the framework. At this stage, the fault model and the fault rate are selected. Hence, the feasible fault positions in the network model based on design constraints and architecture are generated. Moreover, since *dormant faults* are essential in this work, fault position characterization is done by Monte-Carlo simulation to understand the impact of faults in the SNN models.

Then, the memory network is converted into a corresponding graph with those fault positions to find the relocating solutions for reliability improvement. Here, we solve the graph with the max-flow min-cut problem and detail the swapping path with shortest path solving.

The performance of the fixed and non-fix networks is then evaluated. Finally, this framework's output includes new memory reconstruction, resource overhead, and reliability improvement.

The following subsections explain how to formulate and solve the problem using graph theory.

C. Problem Formulation and Fault Injection

First, we define the problems based on the given design constraints, memory architectures, and the network's configuration, as shown in Fig. 6. We assume that the neuromorphic system consists of N nodes (or neuron clusters), each containing p_i processing elements (neurons) and m_i memory blocks. It is important to note that the number of processing elements and memory blocks can vary between nodes and clusters in heterogeneous systems. Consequently, the total number of processing elements is denoted as $P = \sum_{i=0}^{N-1} p_i$, and the total number of memory blocks is denoted as $M = \sum_{i=0}^{N-1} m_i$. Hence, a specific SNN application requires P' processing elements and M' memory blocks, where $P' \leqslant P$ and $M' \leqslant M$.

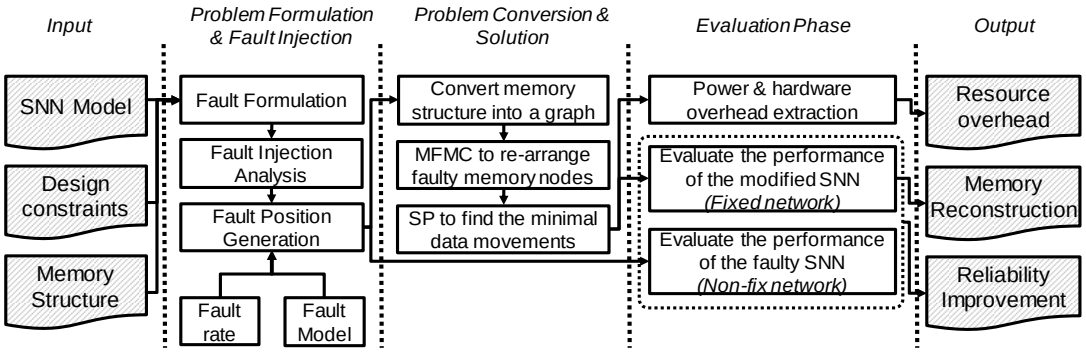

Fig. 6. The proposed NOMA framework's flow.

1) Definition of Faults: Next, we define the faults (F) that appeared in the neuromorphic system. Here, we denote F_{sa} as the number of stuck-at faults and F_{fb} as the number of flip-bit faults. The total number of faults is as follows:

$$F = F_{sa} + F_{fb} = \sum_{i=0}^{M'-1} f_{sa_i} + \sum_{i=0}^{M'-1} f_{fb_i} \tag{1}$$

where f_{sa_i} is the average number of stuck-at faults in the i^{th} memory block, and f_{fb_i} is the average number of flip-bit faults in the i^{th} memory block. Next, these faults could be divided into k different levels of importance:

$$F = \sum_{i=0}^{M'-1}\sum_{j=0}^{k-1} f_{sa_{i,j}} + \sum_{i=0}^{M'-1}\sum_{j=0}^{k-1} f_{fb_{i,j}} \tag{2}$$

As we analyzed earlier in Fig. 1, SNNs' noise resilience can be stronger in some layers, and weaker in others. Therefore, we consider that the first x level(s) in k levels will not strongly impact the network's results, where $x \leqslant k$. To decide the value of k, the NOMA framework goes through the phases of fault injection analysis, where we insert faults into the software model of SNN using Monte Carlo simulations.

With an appropriate value of k, we assume that the total number of faults that degrade SNNs' performance is formulated as follows:

$$F' = \sum_{i=0}^{M'-1}\sum_{j=x}^{k-1} f_{sa_{i,j}} + \sum_{i=0}^{M'-1}\sum_{j=x}^{k-1} f_{fb_{i,j}} \tag{3}$$

Hence, the methodology for fault tolerance in our approach is to maximize the number of faults in the middle layer(s) ($\sum_{j=0}^{x-1} f_{sa_j} + \sum_{j=0}^{x-1} f_{fb_j}$) and to minimize the number of faults in the critical layer(s) ($\sum_{j=x}^{k-1} f_{sa_j} + \sum_{j=x}^{k-1} f_{fb_j}$) by swapping their positions.

2) The Limitation of Dormant Faults: Since adding spare neurons or memory blocks is expensive, we mainly aim to avoid doing so in the scope of this work. However, one consequence is that we do not have spares to replace the faulty area, which leads to the point of accepting faults that occur inside the system.

Therefore, in this particular problem, the main priority is maximizing the faults in non-critical layer(s) (*dormant faults*) to minimize the impact on the overall performance. However, the research literature [6]–[9], [17] shows that SNNs can only tolerate faults to a certain level of Bit Error Rate (BER). We define the following equation to represent this BER:

$$BER = \frac{F}{\sum_{i=0}^{M'-1} w_i} = \frac{\sum_{i=0}^{M'-1} f_{sa_i} + \sum_{i=0}^{M'-1} f_{fb_i}}{\sum_{i=0}^{M'-1} w_i} \tag{4}$$

where w_i is the number of bits used in the corresponding m_i memory blocks. It is assumed that each synaptic-weight memory block has its maximal BER and can tolerate faults within this range. This maximal number depends on the critical levels of synaptic weights stored inside the memory blocks. Hence, the total maximal faults in every memory block or system's fault-tolerance capability, C_{BER}, can be represented as follows:

$$C_{BER} = \alpha \cdot F'_{max} = \sum_{i=0}^{M'-1} a_i \cdot f_{sa_i} + b_i \cdot f_{fb_i} \tag{5}$$

3) Final formulation: Since f_{sa_i} and f_{fb_i} represent the average number of stuck-at and flip-bit faults at the i^{th} memory block, the ratios of the maximal faults to those two faults are the coefficients of a_i and b_i, respectively. Hence, if the faults are within the range of the maximal numbers, the performance degradation of the system is acceptable, as shown in Fig. 1. Otherwise, the system performance degrades significantly, proportioning to the increase in BER.

On the other hand, because each synaptic neuron has its own critical level in the network, the coefficients for the ratio bring different weights to the maximal fault-tolerance capability. Hence, Equation 5 can be rewritten as follows.

$$C_{BER} = \sum_{i=0}^{M'-1}\sum_{j=0}^{k-1} a_{i,j} \cdot f_{sa_{i,j}} + b_{i,j} \cdot f_{fb_{i,j}} \tag{6}$$

where, $a_{i,j} << a_{i,(j+1)}$ and $b_{i,j} << b_{i,(j+1)}$. Therefore, the problem of system reliability can be solved by minimizing the faults leading to system failure or maximizing the dormant faults. Combining Equations 3 and 6, the equation for this problem is formulated as follows:

$$Max(C'_{BER}) = \sum_{i=0}^{M'-1} \sum_{j=0}^{x-1} a'_{i,j} \cdot f_{sa_{i,j}} + b'_{i,j} \cdot f_{fb_{i,j}} \quad (7)$$

D. Problem Conversion and Solution

In the second stage, as shown in Fig. 6, we convert the problem defined in the first stage into a max-flow problem in a corresponding graph. The problem conversion and how to solve it are illustrated by Algorithm 1.

Algorithm 1 - The Proposed Max-Flow Min-Cut with Minimal Neural Replacement Algorithm

Require: $N =$ *The total number of memory nodes*;
$F =$ *The number of faulty memory nodes*;
$D_{max} =$ *The critical path of the system*;
$D =$ *The distance between two nodes*;
$D_{min} =$ *The minimal distance between two nodes*;
$K =$ *The important level of each node*;
Ensure: $G_f =$ *The feasible flow of fault-tolerance graph*;
1: Initalise the graph G;
2: Add source S and sink T;
3: **for** (node: $i = 0 \rightarrow (N-1)$) **do**
4: **for** each node $n_j \in Adj[n_i]$ **do**
5: $d_{i,j} = distance[n_i, n_j]$; {*Based on the memory structure.*}
6: **if** $(d_{i,j} \leqslant D_{max} - D_{n_i})$ and $(K_{n_j} < K_{n_i})$ **then**
7: Add an *edge* from the node n_i to node n_j;
8: Add *weight* of the edge $E_{i,j} = d_{i,j}/D_{min}$; {*Based on the memory structure.*}
9: Add an *edge* from the node n_j to the sink T;
10: Add *weight* of the edge $E_{j,T} = K_{n_j}$;
11: **end if**
12: **end for**
13: **end for**
14: **for** (*faulty node:* $i = 0 \rightarrow (F-1)$) **do**
15: Add *edge* from the source S to the node n_i;
16: Add *weight* of the edge: $E_{S,i} = 1$;
17: **end for**
18: Find *max-flow* with the *push-relabel-max-flow* algorithm;
19: Find G_f with the shortest path from all flows with *Dijkstra's* algorithm;
20: **if** (*max-flow* $== F$) **then**
21: return G_f; {*Done.*}
22: **else**
23: return $Null$; {*Fail.*}
24: **end if**

1) Convert the memory structure into a graph: This subsection discusses how the neural network's converting graph flows while considering the design constraint and memory architecture.

To begin, we use the inputs, such as the memory structure, SNN's configuration, the number of faults, and the critical path of the hardware system, to initialize the graph. Each neuron's synapse memory weights will be represented as a node in the initialized graph G (line 1). Then, we add a virtual source S and a virtual sink T. When connecting two nodes, we need to ensure they are within the possible range (line 6). Additionally, these edges must originate from a higher-priority node and terminate at a lower-priority node. Finally, the nodes without faults will be connected to a virtual sink T (line 9).

In terms of weight, the edge's weight will be normalized based on the distance between the two memories in the given memory structure (line 8). Also, the weight of the edge connecting the virtual and faulty nodes will always be labeled as one unit. Notably, the weights of the edges connecting to the virtual sink follow k levels of importance, with higher-level nodes having larger weights (line 10).

If a fault appears in a node (neuron's synapse memory), it will be considered faulty and connected to a virtual source S (line 15).

At the end of this phase, we generate the directed graph to convert the problem into graph theory.

2) Using MFMC to re-arrange fault memory nodes: As previously mentioned, the objective is to minimize faults in critical synaptic weights or maximize faults in non-critical ones (or shifting *critical errors* into *dormant errors*), as illustrated in Equation 7. After converting the memory structure into a graph, the problem is translated into finding the swapping candidates for the faulty memories. In other words, we now consider solving the multi-source multi-sink max flow min cut (MFMC) where sources are the faulty memories and sinks are the possible swapping candidates. To solve the multi-source, multi-sink MFMC problem, an additional virtual source and sink are added to convert it into a single-source, single-sink MFMC problem.

For ease of understanding, the flow graph presented in Fig. 7 illustrates some cases for the fault tolerance of NOMA. Fig. 7 (a) shows the categorization of neurons in SNNs into three levels of importance ($k = 3$). The number of levels is selected from analyzing fault injection simulations as we did in Fig. 1.

Fig. 7 (b) visually represents the faults observed in NOMA, which typically conforms to a normal distribution pattern. For clarity, the illustration portrays the manifestation of faults across all levels of neurons. Henceforth, in the scenario of faulty recovery, provided reliable communication is ensured, NOMA can accommodate up to the number of low-priority neurons, as evidenced in Fig. 7 (c).

Fig. 8 illustrates the flow graph derived from the Mesh-based memory architecture. Initially, a virtual sink (T) and source (S) are established to facilitate the relocation of neurons within the flow graph. Please note that edges are added only when satisfying the design constraints (lines 7-8 of Algorithm 1). Subsequently, edges are constructed with direction from the source (S to the faulty neurons (N_{31}, N_{23}, and N_{12}). At the same time, a similar configuration is applied for edges from the lower-priority neurons (N_{33}, N_{30}, N_{20}, N_{10} and N_{13}) to the sink (T). The weight of each edge is calculated as in line 8 of Algorithm 1.

Fig. 9 portrays a viable solution for the graph presented in Fig. 8. As illustrated, the red arrows show the flows between two neurons. The final results show swapping flows between high-priority neurons to the low-priority ones. By doing so, we

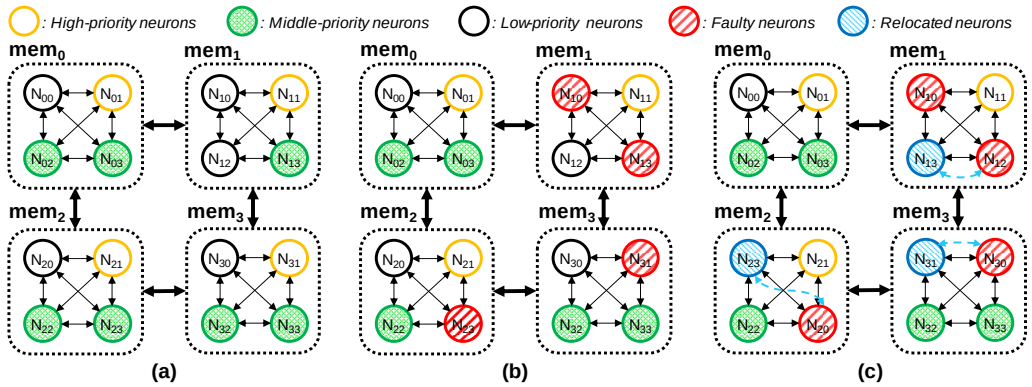

Fig. 7. System model for fault tolerance SNN. (a) Mapping an SNN system into NOMA using nodes of neurons with initial design constraints. (b) NOMA when faults occur. (c) Recovery from faults by swapping the faulty critical neurons to the neighbor non-critical ones.

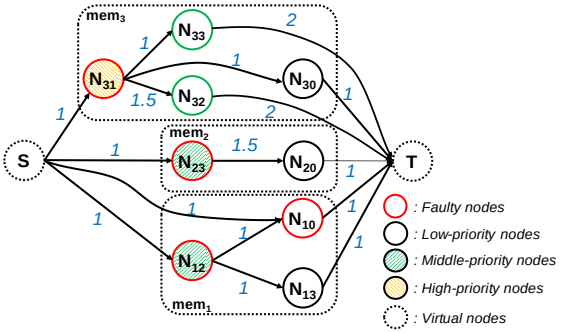

Fig. 8. Flow graph for the max-flow min-cut problem. The graph is converted from the Mesh-based memory architecture.

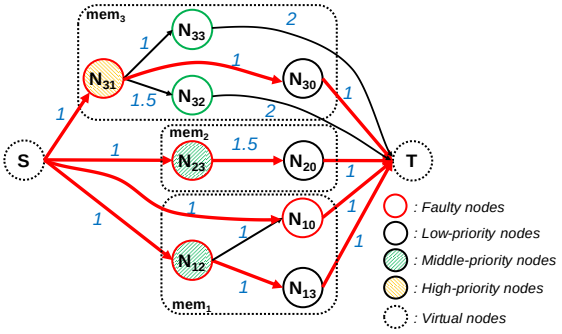

Fig. 9. A solution for the max-flow min-cut problem. The chosen solution should have the maximal flow consisting of minimal weighted edges.

shift the *critical errors* into *dormant errors*, improving overall accuracy.

3) Finding shortest path for minimal data movement: After solving the MFMC problem, as we showed in Fig. 9, the flows are decided between faulty memories and the swapping candidates. However, the flow itself does not contain the routing path since the locations of the neurons are omitted. To finalize the solution, we use *Dijkstra's* algorithm to find the shortest paths for all flows.

4) Complexity Analysis: In this analysis, the *push-relabel* algorithm was selected for its status as one of the field's most efficient maximum flow algorithms. This algorithm is characterized by a robust polynomial $O(V^2E)$ runtime, with E denoting the number of edges and V representing the number of vertices. Notably, the variant employing the highest label node selection rule exhibits a time complexity of $O(V^2E)$, rendering it applicable to the matter discussed in this article. Furthermore, *Dijkstra's* algorithm for determining the shortest path based on the given edge weights operates with a time complexity of $O(V^2)$. Consequently, the proposed algorithm typically operates at a time complexity of $O(V^2E)$. This translates to $O(N^3)$ for 2-D or 3-D IC-based neuromorphic systems, where N designates the count of memory nodes or blocks.

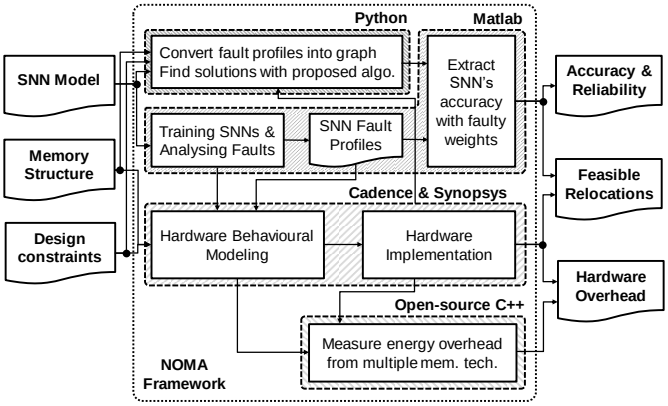

Fig. 10. The NOMA framework for evaluating the performance of the proposed methodology.

V. Evaluation

A. Evaluation Methodology and Framework

In the third phase represented in Fig. 6, to assess the efficiency of the proposed methodology, we implemented the NOMA framework to enhance its accessibility, as depicted in Fig. 10. The initial step involves defining the SNN configuration and the hardware design constraints. Consistent with the experiment outlined in the introduction, the same SNN configuration was employed for this evaluation. Subsequently, these parameters were utilized as input for the NOMA

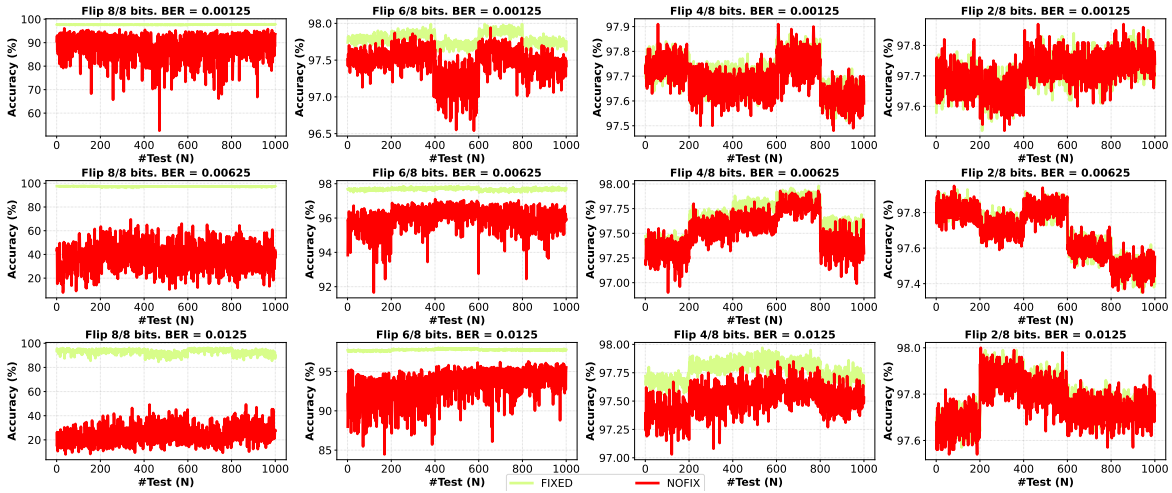

Fig. 11. The analysis compares the accuracy of two architectures with flip-bit faults. One does not have a reliability improvement methodology, while the other includes our proposed method. We used our previous 3-D IC-based SNN architecture with four stacked memory layers, splitting 8 bits into four pairs of two bits. The experiment explores the impact of faults on multiple stacked layers.

framework. The NOMA framework comprises four primary components: 1) The hardware implementation flow, which relies on commercial CAD tools from Cadence and Synopsys (Cadence Innovus, Synopsys Design Compiler, PrimeTime); 2) A Matlab program designed for training, generating fault profiles, and extracting accuracy; 3) A Python program, which implements our proposed software-based approach to identify relocation solutions; and 4) An open-source C++ program based on CACTI memory models [35], called Destiny [36], used for extracting hardware overhead. Additionally, the physical design of our hardware is undertaken with the NANGATE 45-nm PDK [37] and NCSU FreePDK3D45 TSV [38].

The evaluation process comprises several sequential steps. Initially, we executed the hardware implementation based on our previous work and SNN configurations to extract timing paths relevant to memory blocks. These timing paths encompass the critical path, timing between memory blocks, timing through TSV, and data movement timing between memory blocks and PEs. Following this, the Matlab program was used to generate fault profiles. As previously stated, we focus on two common fault models: stuck-at and flip-bit faults. Leveraging the fault profiles, the Matlab program derives the modified SNN weights. On the other hand, timing paths were inputted into a Python program as constraints to identify feasible relocation solutions. Based on these solutions, the accuracy of SNN was re-evaluated to compare the performance between fixed and non-fixed networks. Finally, the hardware overhead was computed using a selected solution based on commercial CAD tools and an open-source C++ program. In this context, the C++ program extracted information from CACTI memory models [35].

B. Accuracy Evaluation

To assess the accuracy enhancement or reliability improvement of our proposed software-based methodology, we generated two sets of faulty 8-bit SNN weights. Faults were randomly distributed across all SNN layers in one set based on a normal distribution. Conversely, in the other set, fault locations were shifted from the first and final layers to the middle layers following our proposed methodology. Each set of flawed weights underwent evaluation via Monte Carlo simulations (1000 iterations per BER per fault type) with three distinct BERs and two fault types (stuck-at and flip-bit faults).

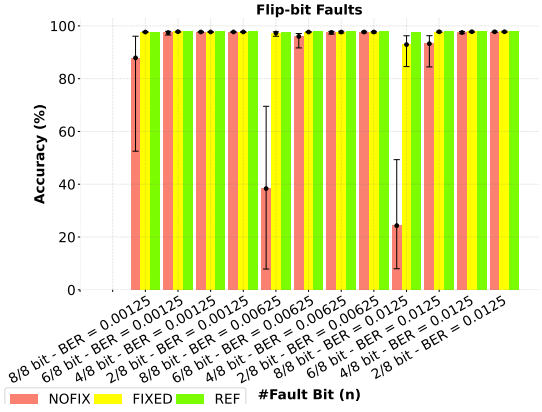

Fig. 12. Comparing the average accuracy of non-repaired and repaired architectures to the reference accuracy under the flip-bit-fault conditions.

1) Flip-bit fault: : For the flip-bit faults depicted in Fig. 11, we evaluated accuracy transformation considering fault rates of 0.00125, 0.00625, and 0.0125, respectively. Within our prior 3-D IC-based neuromorphic systems, we extensively analyzed the impact of faults on each level of synaptic weights' bits, spanning from MSBs to LSBs. Notably, each stacked memory layer comprises a pair of two bits in synaptic weights. Consequently, Fig. 11 illustrates the ensuing accuracy degradation across four distinct cases: 1) all four stacked memory layers are affected, 2) three upper memory layers are affected, 3) two upper memory layers are affected, and 4) solely the top memory layer. In summary, our method enhances accuracy up to 60%, showing strong fault-tolerance efficiency.

Fig. 12 illustrates the average performance of SNN in Monte Carlo simulations featuring varied BERs for flip-bit faults.

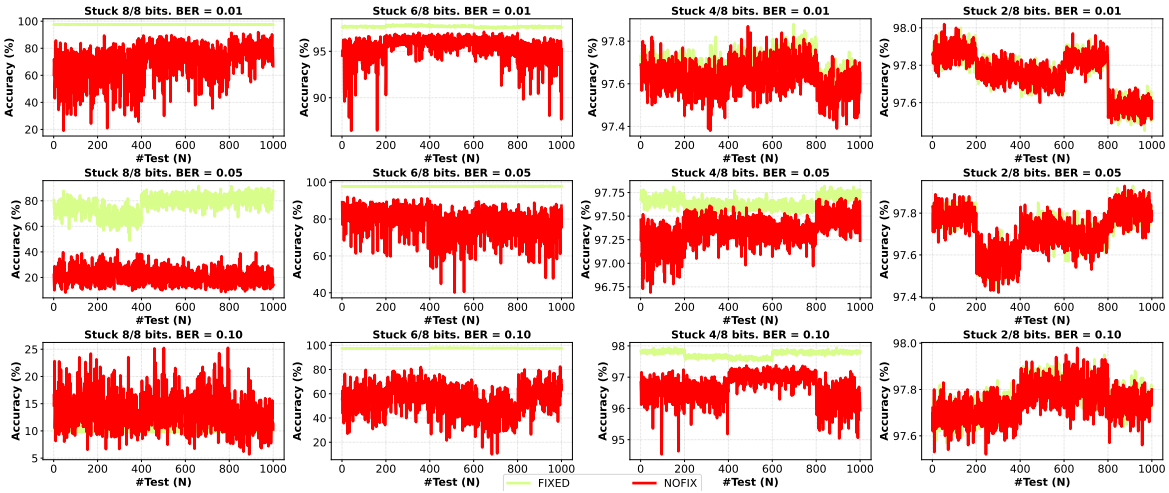

Fig. 13. The analysis compares the accuracy of two architectures with stuck-at faults. One does not have a reliability improvement methodology, while the other includes our proposed method. We used our previous 3-D IC-based SNN architecture with four stacked memory layers, splitting 8 bits into four pairs of two bits. The experiment explores the impact of faults on multiple stacked layers.

Our proposed algorithm demonstrates comparable outcomes to a standard neuromorphic system unaffected by faults across three BERs in most instances. However, when subjected to a BER of 0.0125 affecting all bits, the system's accuracy diminished by approximately 5% (from 97.75% to 92.92%) upon implementing our proposed method. In contrast, a similar hardware system lacking fault tolerance experienced a substantial decline in accuracy, averaging only 24.32% with the same BER. As a result, our proposal can yield a significant fault-resilient improvement for neuromorphic systems.

2) Stuck-at fault: : The outcomes presented in Fig. 13 demonstrate the effects of stuck-at faults in multiple stacked memory layers using consistent Monte Carlo iterations referred to in Fig. 11. Elevated BERs of 0.01, 0.05, and 0.10 are introduced into our 3-D IC-based neuromorphic system. As indicated by Fig. 13, our proposed algorithm produces analogous results for stuck-at faults occurring in the LSBs, which exhibit notable fault resilience. However, the performance does not show improvement when transitioning the weights of higher-priority layers to lower-priority ones in the event of faults affecting MSBs with a high BER of 0.10. Conversely, similar BERs do not result in accuracy degradation within the hardware system if our proposed method is implemented in scenarios where faults manifest in three or fewer stacked layers. Hence, this suggests incorporating additional important levels of MSBs to enhance fault tolerance.

Fig. 14 illustrates the average accuracy degradation of the hardware systems with and without fault tolerance methodology. At a BER of 0.05, our proposed algorithm leads to an accuracy reduction of approximately 20% (from 97.78% to 77.27%) compared to the reference performance. However, it improves by roughly 56% (from 21% to 77.27%) compared to the system's performance without fault tolerance. When considering a BER of 0.10, it is observed that only about 25% of the total neurons are low-priority in the evaluated SNN. For this BER, it is found that half of the neurons' weights in the middle layers are impacted by faults in the event of swapping.

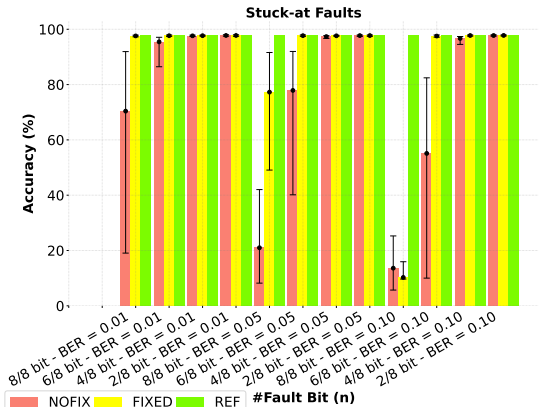

Fig. 14. Comparing the average accuracy of non-repaired and repaired architectures to the reference accuracy under the stuck-at-fault conditions.

Consequently, the accuracy does not show improvement if our proposed method is applied. This highlights a limitation of the relocating strategy and the fault resilience of SNNs.

In conclusion, our proposed methodology presents two key findings: 1) our approach can withstand a high BER through the simple replacement of faulty weights, and 2) the significance levels in our algorithm should encompass the MSBs of weights to yield substantial improvement, extending beyond the initial and final layers.

C. Performance Comparison

In this section, we compared the accuracy loss incurred by our proposed methodology to similar approaches, as shown in Table I. The hardware overhead information was unavailable in the compared works, so it cannot be factored into our evaluation. Our analysis focused on two proximate works, ReSpawn and SoftSNN, which are pertinent to fault tolerance in SNN. Notably, the hidden layers in our SNN are larger than those in the compared works. As a result, our baseline accuracy is higher. When applied to the MNIST benchmark,

TABLE I
COMPARISON RESULTS BETWEEN THE PROPOSED METHODOLOGY AND EXISTING WORKS.

	Our work	ReSpawn [9]	SoftSNN [12]
Network Size	784:256:256:10	784:400	784:400
Hardware Architecture	3-D SNN	2-D SNN	2-D SNN
Benchmark	MNIST	MNIST	MNIST
Tolerance Technique	Swapping Weights	Fault-Aware Mapping	Bound-and-Protect
Bit Error Rate	0.10	0.10	0.10
Baseline Accuracy	97.78%	$\sim 86\%$[1]	$\sim 86\%$[1]
Accuracy Loss	0.01-0.24%	$\sim 10\%$[1]	$\sim 12\%$[1]

[1] We calculated the accuracy loss based on the provided images.

these methods reduced around $10-12\%$ from the baseline accuracy. In our research, aside from the BER effect on MSBs, our strategies for bit distribution across stacked memory layers yielded a notable range of accuracy loss, from 0.01 to 0.24%. Notably, the significant accuracy diminution can be mitigated by assigning additional important levels to MSBs.

Additionally, in contrast to prior research employing 2-D hardware architecture, our proposed methodology capitalizes on the advantages of 3-D hardware architecture. This approach exploits the distribution of bits across multiple stacked layers to enhance fault tolerance. The Fault-Aware Mapping protects the MSBs by shifting the data word cyclically. However, it introduces a notable energy overhead $(6-20\times)$ attributable to the constraints imposed by 2-D hardware architecture prevalent in existing works.

In conclusion, our proposed approach delivers the same accuracy improvement as other methods simply by redistributing weights to address faults. This also results in minimal hardware area overhead and low energy consumption.

D. Hardware Overhead Evaluation

Table 1 assessed the hardware overhead associated with our proposed methodology using various memory technologies (SRAM@45-nm, eDRAM@32-nm, STT-RAM@32-nm, and ReRAM@180-nm). The initial configuration encompassed memory die areas of 446,951.901μm^2, 189,261.396μm^2, 42,361.132μm^2, and 875,285.979μm^2 corresponding to the respective memory technologies. Furthermore, Table II provides comprehensive insights into the energy consumption and latency of both read and write memory operations.

In this context, hardware overhead was assessed using three distinct BERs of 0.01, 0.05, and 0.10. The mean number of read and write operations necessary for data relocation upon fault occurrence was determined for each given BER. Consequently, considering each memory technology's respective attributes, the timing overhead and energy overhead for reading and writing were extracted. Our findings indicate that the ReRAM technology at the 180-nm node exhibits the highest energy consumption ($8.838\mu J@BER = 0.10$) for data relocation among the four evaluated memory technologies. Notwithstanding, the ReRAM technology evinces the lowest leakage power ($10.674mW$ per memory block), implying a potentially reduced long-term power consumption. Conversely, the eDRAM technology at the 32-nm node manifests the most efficient energy utilization ($44.698nJ@BER = 0.10$) for individual data relocations to accommodate faults. Nevertheless, its conspicuous high leakage power ($447.426mW$ per memory block) suggests an anticipated escalation in long-term power consumption relative to ReRAM. Regarding timing overhead, ReRAM also has the most extended delay ($425.974\mu s@BER = 0.10$) for relocating weights in the event of faults, while eDRAM has the fastest response time ($6.136\mu s@BER = 0.10$).

In summary, using the proposed methodology for fault tolerance, a suitable memory technology can be selected based on the budget and design constraints.

TABLE II
HARDWARE COMPLEXITY OF MULTIPLE MEMORY TECHNOLOGIES WITH THE PROPOSED METHODOLOGY.

Technology	**SRAM @ 45-nm**	**eDRAM @ 32-nm**	**STTSRAM @ 32-nm**	**ReRAM @ 180-nm**
SNN Configuration	3 Hidden Layers ($784 \times 256 \times 256 \times 10$)			
Memory Size	4 Stacked Layers (4×256 ***kB***)			
Hardware Area	446951.901 $\times 4\ \mu m^2$	189261.396 $\times 4\ \mu m^2$	42361.132 $\times 4\ \mu m^2$	875285.979 $\times 4\ \mu m^2$
Read Latency	96.541 *ps*	85.066 *ps*	1.385 *ns*	1.6483 *ns*
Write Latency	78.877 *ps*	66.676 *ps*	4.287 *ns*	8.885 *ns*
Read Dyn. Energy	9.711 *pJ*	5.607 *pJ*	5.861 *pJ*	111.957 *pJ*
Write Dyn. Energy	9.668 *pJ*	5.579 *pJ*	7.094 *pJ*	106.622 *pJ*
Leakage Power	540.339 $\times 4\ mW$	447.426 $\times 4\ mW$	93.645 $\times 4\ mW$	10.674 $\times 4\ mW$
Case 1: BER = 0.01				
Timing Overhead	700.97 *ns*	606.36 *ns*	22.664 μs	42.088 μs
Energy Overhead	77.438 *nJ*	44.698 *nJ*	51.768 *nJ*	873.44 *nJ*
Case 2: BER = 0.05				
Timing Overhead	3.538 μs	3.06 μs	114.38 μs	212.408 μs
Energy Overhead	390.796 *nJ*	225.576 *nJ*	261.35 *nJ*	4.406 μJ
Case 3: BER = 0.10				
Timing Overhead	7.094 μs	6.136 μs	229.386 μs	425.974 μs
Energy Overhead	783.724 *nJ*	452.384 *nJ*	523.926 *nJ*	8.838 μJ

TABLE III
EXECUTION TIME OF OUR PROPOSED ALGORITHM FOR 3-D MESH-BASED NOMA

BER	**3-D Mesh-based NOMA - SNN's Configuration**		
	784:256:256:10	784:512:256:10	784:512:512:10
0.01	2.457 ms	3.554 ms	7.034 ms
0.05	8.286 ms	11.416 ms	23.946 ms
0.10	14.422 ms	20.581 ms	42.470 ms

E. Execution Time

In Table III, the execution times of our proposed algorithm for various network sizes are presented. The evaluation utilized the 13^{th} Gen Intel Core i-series processor, specifically the i7-13000K with 16 cores, running Ubuntu 22.04.3 LTS. The algorithm was developed in Python without thread parallelism.

This study involved the introduction of faults at three distinct BERs (0.01, 0.05, and 0.1) on three SNNs. Our analysis indicates a direct correlation between the execution time of the algorithm and the occurrence of faults, as well as the magnitude of the SNN. As the network size grows or faults become more frequent, the execution time correspondingly increases. For example, when dealing with a network size of $784\times256\times256\times10$, the algorithm exhibited average execution times of $2.457ms$, $8.286ms$, and $14.422ms$ for operations under BER of 0.01, 0.05, and 0.10, respectively. With an expansion in the scale of the SNN to $784 \times 512 \times 512 \times 10$, the average execution times rose to $7.034ms$, $23.946ms$, and $42.470ms$, respectively, for the same BERs.

In summary, the execution time of the proposed method and framework is lightweight and friendly to the co-host CPU with a fast response.

VI. Discussion

The previous section shows that NOMA offers high-accuracy improvement with a fast execution time and low resource overhead. However, several points need to be addressed to improve its efficiency. Therefore, we dedicate this section to addressing the limitations of our work and proposing potential solutions. First, our current software-based method lacks suitable lightweight fault tolerance for hardware systems. Thus, future efforts must develop a novel hardware approach for migrating weights among memory blocks to ensure high fault tolerance in neuromorphic systems. Additionally, this method must account for the design budget and constraints.

Second, the evaluation only covers two types of faults (stuck-at faults and flip-bit faults), and their fault rates are provided at the early design stage, which may not fit the actual fault models in the fabricated hardware systems. Therefore, future works should evaluate more hardware fault behaviors such as stuck-short, stuck-open, and bridging faults to closely match the real hardware systems.

Third, Algorithm 1 uses distances to decide the edge in the generated graph; however, there is a gap between the distances and the actual latency of wires. In fact, detailed timing requires running the place and route, so it is impossible to have it during the problem-solving phase. Therefore, we believe having the distance to cut off long latency repairing paths is a feasible solution.

Fourth, given the distinct physical characteristics of 3-D ICs compared to 2-D ICs, fault models for transient and permanent faults must encompass these divergences. For example, the impact of thermal dissipation on logic operations in 3-D ICs elevates the likelihood of transient faults manifesting around areas of heightened temperature. Consequently, developing fault models tailored to 3-D ICs is essential to establish a robust and effective fault tolerance framework.

Fifth, this study builds upon prior research wherein 8-bit weights are evenly distributed across four stacked memory layers. However, optimizing fault tolerance may be achieved by adjusting the number of stacked layers or the bit allocation within each layer. This is particularly important due to the significance of the MSBs, which necessitate storage in a low-noise-impact layer. Consequently, the optimization process introduces a higher level of complexity that may only comprehensively encompass some potential scenarios outlined in this article.

Sixth, our evaluation is limited to a small-scale hardware system in the present study. However, it is essential to note that neuromorphic systems encompass a much broader scope, including NoC-based systems and Systems-on-Chip. Therefore, it is imperative to acknowledge that our proposed methodology needs to be revised because it does not comprehensively address high-complexity systems. To address this constraint, we will focus on expanding the scale of our work to reduce this limitation.

While this endeavor has several limitations, the proposed methodology has demonstrated the potential to enhance system reliability with minimal hardware and energy overhead.

VII. Conclusions

This research examines the varying impact of faults on Spiking Neural Networks (SNNs), emphasizing that faults in the neurons' weights in the first and last layers of the network result in a more significant reduction in accuracy than those occurring in the middle layers. To address this, we present a methodology, NOMA, for enhancing the dependability of 3-D IC-based SNNs by directly reallocating priority weights. The efficiency of this approach is demonstrated across diverse memory technologies, yielding accuracy improvements consistent with those observed in established studies. Subsequent research will seek to incorporate more realistic fault models for 3-D IC-based neuromorphic systems and develop a more suitable hardware strategy for fault tolerance.

References

[1] M. Davies *et al.*, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[2] B. V. Benjamin *et al.*, "Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[3] W. Guo *et al.*, "Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems," *Frontiers in Neuroscience*, vol. 15, 2021.

[4] B. Belhadj *et al.*, "The improbable but highly appropriate marriage of 3D stacking and neuromorphic accelerators," in *Proceedings of the 2014 CASES*, New York, USA, 2014.

[5] D. Kim *et al.*, "Neurocube: a programmable digital neuromorphic architecture with high-density 3D memory," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, p. 380–392, jun 2016.

[6] T. Spyrou *et al.*, "Neuron Fault Tolerance in Spiking Neural Networks," in *2021 DATE*, 2021, pp. 743–748.

[7] E.-I. Vatajelu *et al.*, "Special Session: Reliability of Hardware-Implemented Spiking Neural Networks (SNN)," in *2019 IEEE 37th VTS*, 2019, pp. 1–8.

[8] C. D. Schuman *et al.*, "Resilience and Robustness of Spiking Neural Networks for Neuromorphic Systems," in *2020 IJCNN*, 2020, pp. 1–10.

[9] R. V. W. Putra *et al.*, "ReSpawn: Energy-Efficient Fault-Tolerance for Spiking Neural Networks considering Unreliable Memories," in *2021 IEEE/ACM ICCAD*, 2021, pp. 1–9.

[10] N.-D. Ho and I.-J. Chang, "TCL: an ANN-to-SNN Conversion with Trainable Clipping Layers," in *2021 58th ACM/IEEE DAC*, 2021, pp. 793–798.

[11] S. Hassan, P. Dattilo, and A. Akoglu, "A Novel Implementation Methodology for Error Correction Codes on a Neuromorphic Architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 12, pp. 4706–4720, 2023.

[12] R. V. W. Putra *et al.*, "SoftSNN: low-cost fault tolerance for spiking neural network accelerators under soft errors," in *Proceedings of the 59th ACM/IEEE DAC*, New York, USA, 2022, p. 151–156.

[13] A. Bhattacharjee *et al.*, "Examining the Robustness of Spiking Neural Networks on Non-ideal Memristive Crossbars," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. New York, USA: Association for Computing Machinery, 2022.

[14] A. M. S. Tosson, S. Yu, M. H. Anis, and L. Wei, "A Study of the Effect of RRAM Reliability Soft Errors on the Performance of RRAM-Based Neuromorphic Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 11, pp. 3125–3137, 2017.

[15] A. Saha *et al.*, "A Resilience Framework for Synapse Weight Errors and Firing Threshold Perturbations in RRAM Spiking Neural Networks," in *2023 IEEE European Test Symposium (ETS)*, 2023, pp. 1–4.

[16] P. Shukla *et al.*, "An Overview of Thermal Challenges and Opportunities for Monolithic 3D ICs," in *Proceedings of the 2019 GLSVLSI*, New York, USA, 2019, p. 439–444.

[17] F. Su *et al.*, "Testability and Dependability of AI Hardware: Survey, Trends, Challenges, and Perspectives," *IEEE Design & Test*, vol. 40, no. 2, pp. 8–58, 2023.

[18] C. Liu *et al.*, "Rescuing Memristor-based Neuromorphic Design with High Defects," in *Proceedings of the 54th Annual DAC 2017*, New York, USA, 2017.

[19] S. A. El-Sayed *et al.*, "Spiking Neuron Hardware-Level Fault Modeling," in *2020 IEEE 26th IOLTS*, 2020, pp. 1–4.

[20] S. B. Shrestha and G. Orchard, "SLAYER: Spike Layer Error Reassignment in Time," in *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc., 2018.

[21] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[22] Y. Zhu *et al.*, "Statistical Training for Neuromorphic Computing using Memristor-based Crossbars Considering Process Variations and Noise," in *2020 DATE*, 2020, pp. 1590–1593.

[23] D. Gaol *et al.*, "Reliable Memristor-based Neuromorphic Design Using Variation- and Defect-Aware Training," in *2021 IEEE/ACM ICCAD*, 2021, pp. 1–9.

[24] R. V. W. Putra *et al.*, "EnforceSNN: Enabling resilient and energy-efficient spiking neural network inference considering approximate DRAMs for embedded systems," *Frontiers in Neuroscience*, vol. 16, 2022.

[25] N.-D. Nguyen *et al.*, "Power-Aware Neuromorphic Architecture With Partial Voltage Scaling 3-D Stacking Synaptic Memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 12, pp. 2016–2029, 2023.

[26] A. Ben Abdallah and K. N. Dang, "Toward Robust Cognitive 3D Brain-Inspired Cross-Paradigm System," *Frontiers in Neuroscience*, vol. 15, 2021.

[27] K. N. Dang *et al.*, "Migspike: A migration based algorithms and architecture for scalable robust neuromorphic systems," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 602–617, 2022.

[28] S. S. Salvi and A. Jain, "A Review of Recent Research on Heat Transfer in Three-Dimensional Integrated Circuits (3-D ICs)," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 11, no. 5, pp. 802–821, 2021.

[29] P. Jain *et al.*, "Thermal and power delivery challenges in 3D ICs," *Three Dimensional Integrated Circuit Design: EDA, Design and Microarchitectures*, pp. 33–61, 2010.

[30] K. N. Dang *et al.*, "Report on power, thermal and reliability prediction for 3d networks-on-chip," *CoRR*, vol. abs/2003.08648, 2020.

[31] C.-H. Lin *et al.*, "The effect of nbti on 3d integrated circuits," in *2012 IEEE EDAPS*, 2012, pp. 201–204.

[32] K. N. Dang *et al.*, "HotCluster: A Thermal-Aware Defect Recovery Method for Through-Silicon-Vias Toward Reliable 3-D ICs Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 799–812, 2022.

[33] A. Burkitt *et al.*, "A Review of the Integrate-and-Fire Neuron Model: I. Homogeneous Synaptic Input." *Biol. Cybern*, vol. 95, pp. 1–19, 2006.

[34] N.-D. Nguyen *et al.*, "An In-Situ Dynamic Quantization With 3D Stacking Synaptic Memory for Power-Aware Neuromorphic Architecture," *IEEE Access*, vol. 11, pp. 82 377–82 389, 2023.

[35] N. Muralimanohar and *et al.*, "CACTI 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[36] M. Poremba *et al.*, "DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches," in *2015 DATE*, 2015, pp. 1543–1546.

[37] N. Inc. Nangate Open Cell Library 45 nm. [Online]. Available: http://www.nangate.com/

[38] N. E. D. Automation. FreePDK3D45 3D-IC Process Design Kit. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK3D45

Ngo-Doanh Nguyen received a Master's degree in Computer Science and Engineering at the Graduate School of Computer Science and Engineering, The University of Aizu, Aizuwakamatsu, Japan, in 2024. He is currently a Research Engineer on system integration and VLSI design for artificial intelligence with the Information Technology Institute, Vietnam National University, Hanoi, where he worked from 2018 to 2022. His research interests include hardware/software co-design and verification, and low-power solutions for artificial intelligence.

Khanh N. Dang is currently an Associate Professor in the Department of Computer Science and Engineering at the University of Aizu. He received his Ph.D. from the University of Aizu and his M.Sc. from the University of Paris XI. His research interests include Network-on-Chips, 3D-ICs, neuromorphic computing, and fault-tolerant systems.

Akram Ben Ahmed received the M.S.E. and Ph.D. degrees in computer science and engineering from the University of Aizu, Aizuwakamatsu, Japan, in 2012 and 2015, respectively. He later joined Keio University, Tokyo, Japan, as a Postdoctoral Researcher. He is currently a Research Scientist with the National Institute of Advanced Industrial Science and Technology, Tokyo, Japan. His research interests include on-chip interconnection networks, reliable and fault-tolerant systems, and ultralow-power embedded systems.

Abderazek Ben Abdallah received the Ph.D. degree in computer engineering from The University of Electro-Communications, Tokyo, in 2002. From April 2014 to March 2022, he was the Head of the Computer Engineering Division, The University of Aizu, Japan. Since April 2022, he has been the Dean of the School of Computer Science and Engineering, The University of Aizu. He is currently a Full Professor at The University of Aizu. He is the author of four books, four registered and eight provisional Japanese patents, and more than 150 publications in peer-reviewed journal articles and conference papers. His research interests include adaptive/self-organizing systems, brain-inspired computing, interconnection networks, and AI-powered cyber-physical systems. He is a Senior Member of ACM.

Xuan-Tu Tran received a Ph.D. degree in 2008 from Grenoble INP (at the CEA-LETI), France, in Micro-Nano Electronics. He is currently a full professor at Vietnam National University, Hanoi (VNU), and the Director of VNU Information Technology Institute. He was an invited professor at the University Paris-Sud 11, France (2009, 2010, and 2015), the University of Electro-Communication, Tokyo (2019), Grenoble INP (2011), and adjunct professor at the University of Technology Sydney (2017–2023). He was the Director of the VNU Key Laboratory for Smart Integrated Systems (SISLAB) from 2016 to 2021. His research interests include designing and testing systems-on-chips, networks-on-chips, design-for-testability, asynchronous/synchronous VLSI design, low-power techniques, and hardware architectures for multimedia applications. He has published 3 books, 4 patents and more than 120 peer-reviewed publications in these areas. He is a Senior Member of the IEEE, IEEE Circuits and Systems (CAS), IEEE Solid-State Circuits and Systems (SSCS), a member of IEICE, and the Executive Board of the Radio Electronics Association of Vietnam (REV).