ApproxiMorph: Energy-efficient Neuromorphic System with Layer-wise Approximation of Spiking Neural Networks and 3D-Stacked SRAM

Ryoji Kobayashi[§], Ngo-Doanh Nguyen, *Graduate Student Member, IEEE,*[†] Abderazek Ben Abdallah, *Senior Member, IEEE,*[§] Nguyen Anh Vu Doan[‡], Khanh N. Dang, *Member, IEEE,*[§]

***Abstract*— This paper proposes *ApproxiMorph*, a comprehensive framework for both software and hardware co-design, targeting energy-efficient AI applications using 3D-IC-based neuromorphic systems. By leveraging parallel interconnections and high-bandwidth communications inherent to 3D-ICs, and the noise-resilience characteristics of spiking neural networks (SNNs), *ApproxiMorph* achieves significant power savings by exploiting (1) approximate implementation of neuron cells, (2) layer-wise approximation of SNNs through the heuristic exploration algorithm, (3) reduced-voltage operation in the 3D-stacked SRAM, and (4) incorporating a weight-tuning method. As a result, to search for the energy-optimal layer-wise approximation, *ApproxiMorph* explores only 0.44–0.67% of all possible combinations, achieving a 28.06% power saving for additions with a 0.60% accuracy loss in comparison to the baseline SNN for MNIST. In the VGG16 for CIFAR-10, *ApproxiMorph* searches around 10^3 combinations from over 10^{17} possible solutions, resulting in a 29.16% power saving with slight accuracy gain. Furthermore, integrating all methods enhances the accuracy of approximate implementations and demonstrates higher error resilience than accurate implementations.**

***Index Terms*—Spiking Neural Networks, 3D-IC-based Neuromorphic System, Approximate Computing, Optimization**

I. Introduction

RAPID growth of artificial intelligence (AI) applications has led to a substantial increase in power consumption [1], [2], showing a critical challenge for sustainable or carbon-efficient computing [3]. However, as highlighted in [4], the energy efficiency of CPU/GPU-based computation for deep neural network (DNN) applications has only improved by approximately an order of magnitude over the past decade. This limited improvement is primarily attributed to the increasing number of parameters and the growing complexity of models. Consequently, the power consumption from three key sources has significantly escalated: (1) *arithmetic computations* for activation functions (e.g., addition and multiplication), (2) *memory systems* required to store parameters, and (3) *communication* overhead among hardware components such as CPUs, GPUs, and memory.

On the other hand, Spiking Neural Networks (SNNs) or neuromorphic computing offer a biologically inspired alternative to traditional Artificial Neural Networks (ANNs) [5]. Unlike ANNs, where neurons produce continuous outputs via activation functions, SNNs operate using discrete action potentials or spikes. This mechanism enables simplified and low-power neuron cell operations, particularly with the Integrate-and-Fire model [6]. These advantages make SNNs ideal for resource-constrained applications, such as Internet of Things (IoT) devices and portable electronics. Additionally, SNNs exhibit inherent noise resilience, allowing tolerance to computational inaccuracies, which aligns well with approximate computing techniques [7]–[9].

To enhance the energy efficiency of arithmetic units, approximate computing has emerged as a promising paradigm that trades a limited degree of computational precision for significant energy savings [10]. Previous works, such as the approximate neural network (AxNN) introduced by *Venkataramani et al.* [11] and the approximate spiking neural network (AxSNN) proposed by *Sanchari et al.* [8], have demonstrated notable reductions in energy consumption for neural networks. However, in neuromorphic systems, memory power consumption remains a dominant factor, accounting for approximately 50–75% of total power usage, compared to 20–40% for processing elements [12].

Various strategies have been explored to address memory-related energy consumption. For instance, *Putra et al.* [12] utilized approximate DRAM with voltage scaling. In addition, *Nguyen et al.* [13] introduced a novel 3D-stacked SRAM memory-on-logic design leveraging partial voltage scaling and power gating for energy reduction. Another promising direction involves computation-in-memory (CiM) architectures, where processing elements are integrated with memory cells using memristive devices (ReRAM, FeFET, STT-RAM, etc.) [14]–[16]. However, CiM architectures face challenges with reliability due to their analog-based operations [17].

Despite these advancements, some underexplored areas are motivating us in this work:

1) *First*, a *combined approach integrating approximate neuron cells and memory systems* holds significant potential for achieving energy-efficient neuromorphic systems. However, this area remains underexplored, and to the best of our knowledge, no prior work has investigated this combination.
2) Second, optimizing the approximation levels for neuron cells before manufacturing is crucial, as these levels cannot be adjusted post-fabrication. However, *determining the optimal trade-offs between accuracy and energy savings presents a significant challenge*, primarily due to the computational demands of evaluating numerous con-

[§]The School of Computer Science and Engineering, The University of Aizu, Aizu-Wakamatsu 965-8580, Fukushima, Japan. (e-mail: k.ryoji0209@gmail.com; benab@u-aizu.ac.jp; khanh@u-aizu.ac.jp)

[†]Department of Electrical, Electronics and Telecommunication Engineering and Naval Architecture, University of Genoa, Italy. (e-mail: doanhnn@ieee.org)

[‡]Infineon Technologies AG, Germany (email: anhvu.doan@infineon.com)

(Corresponding author: Ryoji Kobayashi.)

figurations. For instance, with layer-wise approximation across five layers and ten different approximate adders, there are 10^5 possible combinations to consider. Incorporating additional parameters, such as voltage scaling, power-gating settings, or finer-granularity approximations, further increases the number of combinations exponentially, complicating the exploration process.

3) Third, existing approximation techniques typically accept accuracy losses as a trade-off for reduced power consumption. However, *regaining the lost accuracy after combining multiple approximation techniques* is feasible, as the computational and storage distortions can complement each other. This synergy enables the design of systems with significantly lower power consumption while maintaining comparable accuracy.

Building on these motivations, this paper presents *ApproxiMorph*, a comprehensive framework that reduces the energy consumption of Spiking Neural Networks through approximate computing and approximate 3D stacking SRAM. The main contributions of this work are as follows:

- A novel framework integrating approximate neuron cells and 3D-IC-based stacking memory to achieve energy-efficient neuromorphic systems.
- Implementation of memory approximation using partial voltage scaling and power-gating techniques, combined with approximate adders for neuron cells due to their multiplier-less design.
- Development of a time-efficient heuristic exploration methodology to determine optimal approximation levels for each layer in the target SNNs, targeting low-power solutions while compensating for quality.
- Comprehensive evaluations demonstrating the benefits of combining approximate neuron cells and memory systems, with detailed quality-energy trade-off analyses.

The rest of this paper is organized as follows. Section II reviews related works. Section III describes the implementation of the 3D-IC-based architecture with the proposed approximate design approach. The exploration methodology is detailed in Section IV. Section V presents performance analyses and assessments of quality-energy trade-offs. Discussions are provided in Section VI, and conclusions are drawn in Section VII.

II. Related Works

In this section, we will cover existing work on power-efficient AI using neuromorphic computing. Then, we discuss the optimization approach for the approximation flow of hardware design

A. Neuromorphic Systems for low-power AI Applications

1) 2D-IC-based design: In recent years, numerous studies have explored low-power solutions for AI applications by leveraging the resource-efficient nature of spiking neural networks (SNNs) [18]. For example, Loihi by Intel [19] and TrueNorth by IBM [20] are prominent 2D-IC-based neuromorphic systems that have demonstrated significant improvements in energy efficiency. However, in the 2D architecture, memory segments for synaptic weights and processing elements are located on the same layer, leading to an inherent bottleneck due to the high communication cost required for data transfer between them.

2) 3D-IC-based design: To address this challenge, 3D-IC-based neuromorphic systems have been introduced. *An et al.* [21] proposed a 3D neuromorphic system employing memristive devices as synapses. *Kim et al.* [22] developed an architecture named Neurocube, which stacks DRAM dies on a logic die using Through-Silicon-Vias (TSVs) technology. Similarly, *Ueyoshi et al.* [23] designed a DNN accelerator with 3D SRAM, using Inductive Coupling (TCI) technology, which offers advantages in manufacturing cost, yield rate, and reliability [24]. These studies demonstrated that 3D implementations can significantly reduce communication costs, hardware footprint, and power consumption due to the vertical alignment of memory architecture.

3) Approximate design: Another approach to enhancing the energy efficiency of neuromorphic systems is the application of approximation-based techniques, leveraging the noise-resilient nature of SNNs. Weight quantization is a widely adopted method in many studies to reduce energy consumption and memory usage. For instance, *Putra and Muhammad* [25] proposed the Q-SpiNN framework, which quantizes SNN parameters. This approach achieved approximately $4\times$ and $2\times$ memory savings for unsupervised and supervised SNN models, respectively, while maintaining similar accuracy. Similarly, *Hasssan et al.* [26] introduced SpQuant-SNN, combining quantization with dynamic pruning. Their method demonstrated up to $13\times$ memory reduction and over $4.7\times$ FLOPs reduction compared to state-of-the-art baseline models.

Since memory is the most energy-intensive component of neuromorphic systems, approximating memory using voltage scaling techniques offers significant potential for reducing the total power. Voltron, developed by *Chang et al.* [27], explored the effects of reduced-voltage DRAM operation across various workloads and proposed the DRAM energy reduction mechanism. This approach achieved an average energy savings of 10.5% while minimizing performance degradation. Similarly, Minerva, introduced by *Reagen et al.* [28], adopted a co-design approach involving algorithms, architecture, and circuits to optimize DNN accelerators, achieving a $2.7\times$ energy reduction by lowering SRAM voltages. *Putra et al.* considered the reduced-voltage operation of two-layer SNN for both 2D-based SRAM and DRAM with fault-tolerant approaches, which identifies faulty memory cells for weight mapping [29]. For 3D architectures, *Zhao et al.* [30] applied voltage scaling to processing cores and cache hierarchies, resulting in a 34% total energy reduction. Additionally, an *in-situ* dynamic quantization technique using a 3D-stacked SRAM architecture was introduced in [31], demonstrating successful energy reductions while preserving the performance quality of SNN applications.

After memory, computation units are the next largest consumer of power in neuromorphic systems [12]. Given the extensive number of calculations performed by neuron cells, applying approximate arithmetic operators, such as approximate adders and multipliers, emerges as a promising solution for

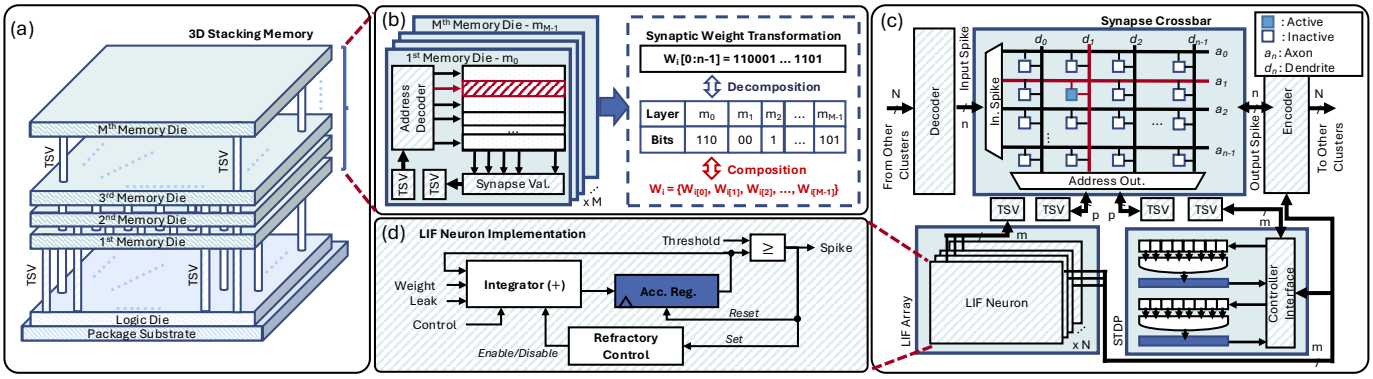

Fig. 1. The overview of our 3D-IC-based neuromorphic system: (a) Overview of the 3D stacking memory with M layers; (b) Approximate Stack Memory architecture with an example of synaptic weight transformation (decomposition/composition). (c) Block diagram of the logic component (computing core); (d) Detail of the LIF neuron implementation.

low-power operation. Numerous studies on approximate arithmetic operators and their applications have demonstrated their effectiveness in hardware implementations [32]. However, to the best of the authors' knowledge, research on employing approximate units in SNN applications remains limited. A recent study applied approximate adders and multipliers to the Izhikevich neuron model, achieving energy reductions with minimal accuracy loss in a two-layer neural network [33].

Although computations for neurons and memory for synaptic weights are the largest consumers of power, no prior works advocated the benefits of applying the approximation paradigm to both at once to facilitate further energy efficiency of neuromorphic systems. Therefore, this work considers the use of approximate arithmetic operators and 3D-stacked memory in multi-layer perceptrons and convolutional neural network models to evaluate their potential in neuromorphic applications.

B. Optimization for Approximate Hardware Systems

1) Circuit Approximation: When multiple approximation schemes are available, designers can select the specific combination that provides the best energy-quality trade-offs for the target application. However, as the number of configurations increases, the possible combinations grow exponentially, making exhaustive exploration of the design space infeasible. Numerous approaches have been proposed to address this challenge, spanning from the arithmetic operator level to the application level. *Mrazek et al.* [34] utilized multi-objective Cartesian Genetic Programming (CGP) to optimize circuits for approximate adders and multipliers. *Zervakis et al.* [35] introduced a systematic methodology combining multi-level approximation with the Voltage Over-Scaling (VOS) technique. Furthermore, frameworks such as autoAx by *Mrazek et al.* [36] and ApproxFPGAs by *Prabakaran et al.* [37] were developed to optimize configurations of approximate circuits tailored to target applications. While these approaches achieve optimal energy-quality trade-offs for less complex applications, their scalability to larger design spaces remains limited as they rely on the bit-wise design space exploration. The neural network circuit typically requires embedding a large number of neurons to gain performance improvement, and thus, these fine-grained design searches are inefficient in most cases.

2) System Approximation: To overcome this limitation, prior studies have demonstrated the effectiveness of approximation techniques for more complex applications through coarse-grained optimization approaches. For example, *Manuel et al.* [38] employed a GA-based process, where the search space is confined to a user-defined region, for image color processing applications. *ALWANN* by *Mrazek et al.* [39] utilized non-uniform layer-wise approximation for the convolutional layers in CNNs, showing its advantage over uniform approximation. Furthermore, *Nguyen et al.* [40] proposed a framework that searches for the optimal quantization level for each layer in SNNs. As demonstrated in several studies, coarse-grained optimization enables designers to effectively navigate the design space, achieving the best energy-quality trade-offs. Building on this, we propose a time-efficient heuristic exploration algorithm and a framework called ApproxiMorph to derive non-uniform layer-wise approximation specifically for target SNNs.

III. 3D-IC-based Neuromorphic Systems

This section introduces the implementation of our hardware architecture. First, we show the details of 3D-IC-based stacking memory. Second, the strategy for low-power operation through *in-situ* memory approximations. Then, we present the methodology for approximating neuron cells.

A. Overall Neuromorphic System

An overview of our hardware architecture is shown in Fig. 1. Fig. 1(a) presents the block diagram of the architecture. The memory dies are stacked vertically on top of the logic die, and the stored bits in memory are accessed and transferred to the computing segments via TSV connections. This strategy effectively reduces the data transfer distance between the memory and computing segments, resulting in a decrease in the power consumption and latency required for communication.

Fig. 1(b) shows the implementation of each memory die and the methodology for synaptic weight transformation. Thanks to our 3D implementation, memory words (e.g., 8-bit, 16-bit) can be split into smaller subsets (e.g., 1-bit, 2-bit, 4-bit) and stored in the corresponding memory dies. In this approach, synaptic weights are distributed across different memory dies. When the stored synaptic weight is used in the computing

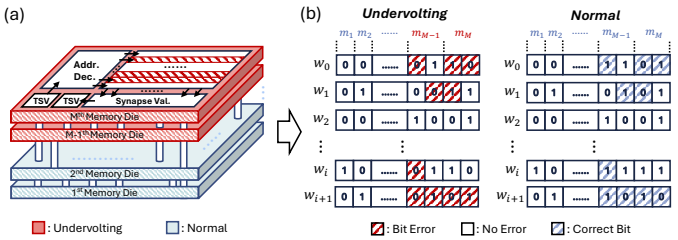

Fig. 2. The example of operations with undervolting: (a) Memory layers with undervolting. (b) Faults' occurrence in comparison to normal operations.

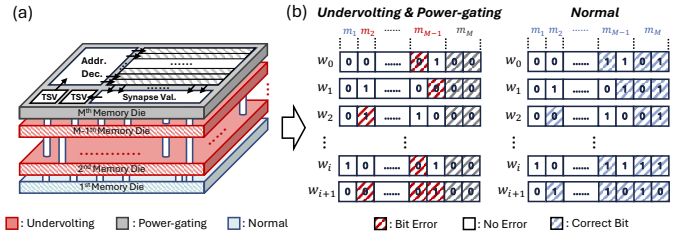

Fig. 3. The example of operations with undervolting and power-gating: (a) Memory layers with undervolting and power-gating; (b) The occurrence of the faults with comparison to the normal operations.

segments, the subset bits are transferred and concatenated to form a single meaningful value.

B. Mapping with 3D Stacking Memory

With the stacked memory structure shown in Fig. 1, the SNN workload should be carefully mapped onto the hardware for inference. Given an n-bit weight format and M memory layers, several approaches exist for mapping the weights, assuming on-chip learning is excluded. Due to their inherent susceptibility to faults, these memories are preferred for storing weights rather than configuration parameters or communication data, as faulty bits in the latter can be catastrophic.

A common strategy is to map the most significant bits (MSBs) of the weights to the layer closest to the logic die, and the least significant bits (LSBs) to the furthest layer. This strategy has two main advantages: First, faults may occur post-bonding or during runtime, and defective layers are more likely to appear in the furthest memory layer or in the interconnects (e.g., Through-Silicon Vias), leading to it. Second, this allows the bits-evaluating mechanism for fault-tolerance, as discussed in [41]. For example, assuming the bit-width of the synaptic weight is 8 bits ($n = 8$), the number of stacked memory dies is four ($M = 4$), and each memory die contains 2-bit subsets, the synaptic weight $W[7:0] = 01101100$ can be stored in bottom-up order as $\{m_1, m_2, m_3, m_4\} = \{01, 10, 11, 00\}$, where m_i represents the i^{th} memory die. In this case, m_1 contains the most significant bits (MSBs), while m_4 includes the least significant bits (LSBs). In the case of m_4 become faulty, the system can read $\{m_1, m_2, m_3, m_4'\} = \{01, 10, 11, XX\}$ where XX is the faulty bits.

However, it is worth noting that in an ideal system, the mapping location of each bit can be flexible, since after reading from the memory layers, the bits can be reordered to reconstruct the correct bit positions in the final weight representation.

C. Approximation with 3D Stacking Memory

Unlike the traditional 2D memory, where the voltage control affects the overall memory cells at once, 3D stacked memory

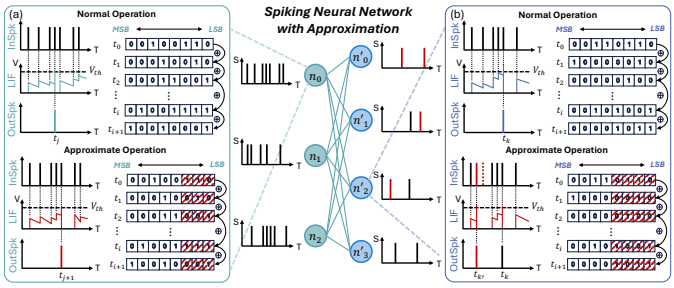

Fig. 4. The spiking neural network (SNN) with approximate neuron cells: (a) 3-bit approximation on LSBs; (b) 4-bit approximation on LSBs.

can enable partial voltage scaling. By exploiting this unique advantage, memory can be approximated by reducing or turning off the supply voltage of the different memory dies independently. More specifically, since active and inactive bits are stored separately, it is possible to lower the supply voltage only for the memory dies containing inactive bits. Consequently, bit errors arising from voltage scaling impact the least significant bits (LSBs), while the most significant bits (MSBs) remain unaffected. This approach minimizes quality degradation while maximizing energy efficiency.

In our previous work, we explored the potential impact on SNN applications and the bit errors induced by the reduced voltage [13]. In this context, the bit error rate (BER) of SRAM cells is examined, as our implementation is based on the SRAM model. According to previous works [42], [43], the BER of an SRAM cell can be defined as the probability that the static noise margin (SNM) is approximately zero. Fig. 1 in Appendix A illustrates the relationship between the BER of a 45-nm 6T SRAM cell and the supply voltage in the near-threshold region ($0.7V$ to $0.85V$), which is obtained using PrimeSim HSPICE and mathematical equations [42]–[44].

Examples of approximate operations are shown in Fig. 2 and Fig. 3, where the bit errors resulting from voltage scaling and power-gating techniques are illustrated. In these figures, the red-square areas indicate flip-bit errors due to reduced-voltage operations (undervolting), the grey-square areas show the quantized bits resulting from power gating, and the blue-square areas represent the bits affected by either undervolting or power gating.

To handle approximation adaptively, we apply different power supply modes based on the activity of memory regions and the error tolerance of the stored data. The normal mode is used for critical data that requires full accuracy, undervolting is applied to regions where minor bit flips are tolerable, and power-gating is employed for inactive regions to achieve maximum energy savings. For example, in an SNN inference task, synaptic weights can be split into different subsets (from MSB to LSB), as discussed in Section III-C. Our strategy prioritizes lowering the voltage of the LSB region first and then progressively moving toward the MSB. When undervolting reaches its limit and the bit error rate (BER) becomes excessively high, maintaining power for the layer containing these bits becomes inefficient, leading to the application of power gating. This approach allows energy reduction while tolerating a small accuracy degradation. Moreover, quantization using power gating can result in significant energy savings.

In this work, we analyzed the trade-off between power saving and accuracy of the SNN before mapping. In other words, our system currently does not support on-chip (dynamic) decision-making regarding the supply voltage and power gating. Therefore, we need off-chip support for the voltage controller to employ undervolting and power gating during operation based on the pre-mapping analysis about the expected trade-off.

By exploiting this voltage-mode flexibility and understanding the impact of approximation on SNN behavior, we can maximize the energy efficiency of the memory system while maintaining acceptable inference accuracy.

D. Approximation in Neuron Cells

To achieve further energy efficiency of neuromorphic systems, the computation unit, especially the neuron cell, is also the subject of approximation.

Fig. 1(c) illustrates the overview architecture of the logic component, and Fig. 1(d) shows the detailed implementation of the neuron cell. In this work, we used the Leaky integrate-and-fire (LIF) neuron as a component of SNN due to its potential for hardware-friendly implementation. Since the neuron cell operates in an accumulation and firing manner, it comprises an accumulation register and an adder, as shown in Fig. 1(d). At each time step, the synaptic weights are summed based on the input spikes from the pre-synaptic neuron cells, and the output spike from one neuron can be transferred to either the same cluster or other clusters. In summary, the behavior of the LIF neuron can be represented mathematically as follows:

$$V_i(t) = V_i(t-1) + \sum_j w_{ij} x_j(t-1) - \lambda \tag{1}$$

$$x_i(t) = \begin{cases} 1 & \text{if } V_i(t) \geq V_{th}. \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

where $V_i(t)$ is the membrane potential of the neuron i at time step t, w_{ij} is the synaptic weight between neuron i and j, $x_i(t)$ is the spike at time step t, and λ is the leakage value.

The approximation methodology for neuron cells is straightforward; an adder in the neuron cell is replaced with an approximate adder. While the computation errors are induced by approximate implementation, they are usually limited to the LSBs. Fig. 4 shows an example of SNN operation with approximate neuron cells. Here, the red-square areas indicate bits under the influence of the approximation. For instance, Fig. 4(a) and (b) present the approximate operation of neuron cells in comparison to the normal one. In the case of (a), even if the input spike train is the same, the time the output spike is emitted is delayed by a single time step. In (b), there is one undesired input spike (red bar) and one missing input spike (red dashed bar), which is because of the computation errors in the previous layer. As a result, one undesired output spike is emitted, but another output spike is the exact timing as normal operation. Since output spike errors are propagated from one layer to another, careful selection of an approximate adder is needed to maximize energy savings.

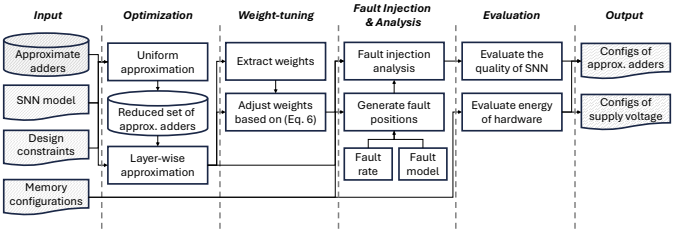

Fig. 5. The proposed *ApproxiMorph* framework's flow.

E. Raising Question About Approximation Selection

In general, the approximate stacking memory architecture offers the ability to update its strategy on the fly by tuning the supply voltages. However, the primary challenge is selecting an appropriate approximate circuit for the neuron, as this choice significantly impacts performance and cannot be altered post-fabrication. Selecting these circuits is essential, as the complexity increases with the number of available adders and the granularity of the selection process.

Here, let's consider a system with N neurons divided into B subsets (e.g. layers), where subset i contains n_i neurons:

$$N = \sum_{i=0}^{B-1} n_i \tag{3}$$

Additionally, let A be the number of adder types available for selection. For simplicity, we assume that all neurons in a given subset (e.g., a layer) use the same type of adder, regardless of granularity. Under this assumption, the total number of combinations C to be validated for the approximate neurons is:

$$C_{apx\_neuron} = A^B \tag{4}$$

If we extend this to include memory approximations, consider a memory system with L silicon layers and V different voltage levels, where $V_{DD} = 0V$ is considered power-gating. The total number of combinations, including both neuron and memory approximations, is then given by:

$$C_{final} = V^L \cdot A^B \tag{5}$$

For instance, if we have $B = 5$ subsets, $A = 10$ adder types, $V = 7$ voltage levels, and $L = 2$ memory layers, the total number of combinations to evaluate would be approximately 4.9×10^6. Even with fixed supply voltages, we end up with 10^5 combinations. It is important to note that for each combination, not just one inference is performed; instead, a Monte Carlo simulation (typically with 1,000 $\sim$ 10,000 tests) is required to analyze the impact of flip-bit errors at different memory locations. This emphasizes the significant challenge of efficiently selecting optimal approximation techniques, a topic that will be addressed in the following section.

IV. Proposed Approximation Selection Methodology

This section introduces our proposed *ApproxiMorph*, a methodology for optimizing the configuration of the approximate SNN implementation. We present an overview of the proposed framework, followed by a detailed discussion of the optimization process and weight-tuning method.

A. Proposed Framework

Fig. 5 shows the proposed *ApproxiMorph* framework. The inputs consist of the set of approximate adders (used for the approximate implementation of neurons), the target SNN model, design constraints, and memory configuration. In the optimization phase, we explore the optimal approximate configurations based on the quality constraint (Section IV-B).

Then, the weight-tuning method is applied to fit the synaptic weights to the approximated SNN model (Section IV-C). To evaluate the effectiveness of this phase, we also examine the quality without the weight-tuning method.

Next, the fault scenarios on reduced voltage operation are emulated. The fault position is determined based on the fault characteristic (e.g., fault rate, fault model) and memory configuration. Here, the Monte Carlo simulation is used to analyze the impacts of faults on the model's quality.

Finally, the quality of approximate SNN and the energy consumption of the hardware system are evaluated, and the framework outputs the resulting hardware configurations.

B. Optimization Process for Approximation Selection

We consider a heuristic exploration approach to search for combinations of approximate adders used in different layers of the SNN model that provide optimal design choices. However, the naive GA-based approach consumes a lot of time because it starts from random states. Thus, in this work, we initiate the optimization process from a specific initial state instead of random states. However, this approach sacrifices some variety in the final solutions to some extent. Furthermore, the proposed algorithm is designed to focus on lowering the power consumption as much as possible while maintaining accuracy. To this end, we also proposed the uni-directional generation algorithm instead of GA-based crossover/mutation operations. The proposed optimization algorithm is described in Algorithm 1, and a detailed description is provided in Appendix B of the supplemental document.

In summary, the proposed method has the following four key steps:

- **Step 1**: We investigate the quality-power trade-offs in the case of uniform approximation using the candidate approximate adders (lines 1-4). The quality is evaluated by executing the approximated SNN using the validation dataset D, and power is obtained by borrowing estimated power from EvoApprox library [34]. Since *ApproxiMorph* and the process of generating the approximate circuits are independent, arbitrary approximate adders can be used.
- **Step 2**: Based on the quality and power evaluation of uniform selection obtained in **Step 1**, the list of approximate adders A is reorganized such that the approximations resulting in lower quality always correspond to lower power (lines 5-6). This is because the approximate adder leading to lower quality does not always result in lower power consumption [7]. This process eliminates certain candidate approximate adders from the initial list and generates a new list with reduced candidates. As a consequence, this helps reduce the search space of the optimization process. Although some useful adders may be removed from candidates, our proposal tolerates this concern for the sake of efficient search.
- **Step 3**: Before entering the primary optimization process, the algorithm checks if the approximate implementation is possible (line 7) and if further exploration is needed (line 9). Next, the algorithm determines the boundary index k, which is the maximum index satisfying the initial quality requirement Q_{init} (line 12). In other words, the SNN with the uniform approximation with A_k is an optimal solution when only Q_{init} is considered.
- **Step 4**: The heuristic exploration loop is executed. The difference from the naive GA approach is that *ApproxiMorph* specifies the initial state, which is initialized by the uniform approximate configuration with A_k derived in **Step 3** (line 14). The new approximate configurations are generated (line 16) by randomly selecting a layer and increasing its approximation level, as presented in Algorithm 2. Their quality and power are then evaluated for each solution. Then, the population is sorted and se-

Algorithm 1 - The Proposed Optimization Algorithm for Multi-level Layer-wise Approximation of SNN

Require: $model_{snn}$ = *The pre-trained SNN model*;
L = *The number of layers in SNN model*;
N = *The number of approximate adders*;
D = *Validation dataset*;
$A = [A_0, A_1, ..., A_{N-1}]$ *The list of approximate adders*;
Q_{init} = *Quality constraint for initial state*;
Q_{sol} = *Quality constraint for solutions*;
M_{pop} = *Maximum number of solutions*;
Ensure: $P \subseteq \{A_0 \times A_1 \times ... \times A_{N-1}\}$ *The set of solutions*;

1: **for** (A_i: $i = 0 \rightarrow (N-1)$) **do**
2: $model_{approx} \leftarrow uniform\_approx(model_{snn}, A_i)$;
3: $A_i^{quality}, A_i^{power} \leftarrow evaluate(model_{approx}, D)$;
4: **end for**
5: Apply non-dominated sorting to A and extract top-rank;
6: Sort A in descending order of quality;
7: **if** $A_0^{quality} < Q_{sol}$ **then**
8: **return** ϕ; // $fails\ to\ find$
9: **else if** $A_{size(A)-1}^{quality} \geq Q_{sol}$ **then**
10: **return** $[A_{size(A)-1}^{(0)}, A_{size(A)-1}^{(1)}, ..., A_{size(A)-1}^{(L-1)}]$;
11: **end if**
12: Find index k s.t. $A_k^{quality} \geq Q_{init} > A_{k+1}^{quality}$;
13: // $initialize\ population$
14: $P \leftarrow$ Duplicate $[A_k^{(0)}, A_k^{(1)}, ..., A_k^{(L-1)}]$ by L;
15: **while** ($\neg TerminationCondition$) **do**
16: $P \leftarrow generate\_population(P, L)$; // $Algorithm2$
17: **for** (P_i: $i = 0 \rightarrow (size(P)-1)$) **do**
18: **if** P_i is already evaluated **then**
19: continue; // $skip\ evaluation$
20: **end if**
21: $model_{approx} \leftarrow layerwise\_approx(model_{snn}, P_i)$;
22: $P_i^{quality}, P_i^{power} \leftarrow evaluate(model_{approx}, D)$;
23: **end for**
24: $P \leftarrow nondominated\_sort\_select(P, Q_{sol}, M_{pop})$;
25: **end while**

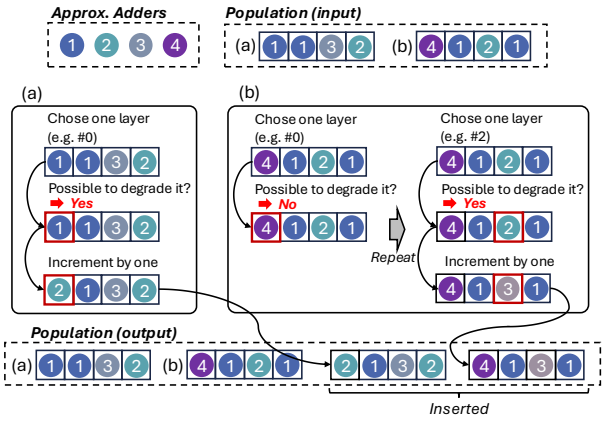

Fig. 6. The process flow of population generation (Algorithm 2). (a) and (b) are the examples of input solutions.

lected based on the evaluation result and solution quality Q_{sol} at each step (line 24). The termination condition can be defined by setting the maximum number of iterations and/or the user-defined constraint.

Algorithm 2 - Population Generation Algorithm

Require: $P =$ *The current population*;
$L =$ *The number of layers in SNN model*;
Ensure: $P_{new} =$ *The new population*;
1: $P_{new} = \phi$;
2: **for** $(P_i$: $i = 0 \rightarrow (size(P) - 1))$ **do**
3: Insert P_i to P_{new}; // *before increment*
4: **repeat**
5: $l \leftarrow random\_layer(0, L-1, P_i)$;
6: **until** valid l is found
7: Increment approximate adder's index of $P_i^{(l)}$ by one;
8: Insert P_i to P_{new}; // *after increment*
9: **end for**

As stated in **Step 4**, new approximate configurations are generated based on Algorithm 2. The example of Algorithm 2 is illustrated in Fig. 6. Here, the approximate adders are represented by (encoded into) the integer (1 to 4), and the input population has two solutions (a) and (b), assuming the target neural network is a five-layer network including the input layer. Please note that the input layer is not approximated. For the solution (a), the first layer (#0) is selected for approximation. Since further degradation is available in this case, the corresponding value is incremented by one (1 $\rightarrow$ 2), and the new solution is generated. On the other hand, for solution (b), although the first layer (#0) is selected, it is already saturated and is not possible to approximate further. Thus, the algorithm chooses other layers. The third layer (#2) is selected in the example, and since it can be degraded more, the value is incremented by one (2 $\rightarrow$ 3). Finally, the generated solution is added to the output population. Here, the original solutions (a) and (b) are added to the output to preserve the possibility of different variations that can be obtained from them.

Although the step-by-step approximation method (increment by one) is adopted in the Algorithm 2, the designer can use other strategies, such as incrementing by two, probabilistically choosing the number of steps, or integrating a crossover- and mutation-based approach. However, this work uses the one-by-one increment method at each iteration for simplicity.

C. Weight-Tuning for Post-Approximation Models

ApproxiMorph evaluates the accuracy of approximate models without changing the synaptic weights from the original values. Therefore, the weight tuning method can be used to fit the synaptic weights to the post-approximation model for better accuracy. Although this problem is not trivial in general, the new weights can be decided based on the characteristics of each approximate adder type.

To this end, we assume that the weight value that minimizes the arithmetic error against all possible combinations of operands (two input values to the adder) can contribute to better accuracy of the approximate model. The weight-tuning method is based on the previous work [39] and is represented as follows:

$$w_{ij} \leftarrow w' \quad \text{s.t.} \quad \underset{w' \in W, v \in V}{\operatorname{argmin}} \ \mathrm{MRE}(w_{ij}, w', v) \tag{6}$$

where w_{ij} is the synaptic weight between neuron i and j, V is the set of possible values that the accumulation register holds, W is the set of weights from the possible range, and w' is an element in W. This means that w_{ij} is adjusted to w', which minimizes the mean-relative-error (MRE) between the calculation of the approximate and accurate adder against all values $v \in V$. In this work, all synaptic weights in the post-approximation spiking neural network (SNN) model are updated using this method.

V. Evaluation

A. Evaluation Methodology

To evaluate our approximation methodology, we utilize a multi-layer perceptron for the MNIST dataset and a convolutional neural network with the VGG16 model for the CIFAR-10 dataset. Both models are trained using the ANN-to-SNN conversion method. The neural network is trained on the software, and its synaptic weights are quantized to 9-bit signed fixed-point values (the sign bit + 8-bit fractional) for MNIST and 13-bit signed fixed-point values (the sign bit + 12-bit fractional) for CIFAR-10. In addition, we select all 12-bit signed approximate adders for MNIST and 16-bit signed approximate adders for CIFAR-10 from the open-source EvoApprox library [34] based on the error metrics and hardware parameters. In this case, we consider power and area efficiency to be the key evaluation metrics. Since these setups were experimentally decided, designers can consider different implementation schemes depending on the requirements.

The hardware architecture is developed in Verilog-HDL, synthesized using Synopsys Design Compiler, and evaluated with PrimeTime. The physical design of our hardware is implemented using the NANGATE 45-nm library [45] and NCSU FreePDK3D45 TSV [46]. The memory is constructed with 6T SRAM generated from OpenRAM [47]. Its BER characteristic is obtained by examining the static noise margin (SNM) of SRAM cells at near-threshold supply voltages through the Monte Carlo simulation with PrimeSim HSPICE, and

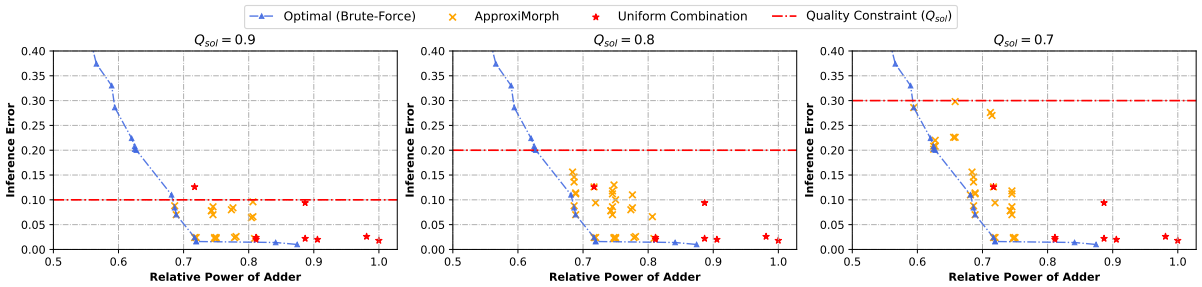

Fig. 7. The final result of the proposed algorithm for the four-layer neural network [784:256:128:10] using the MNIST dataset.

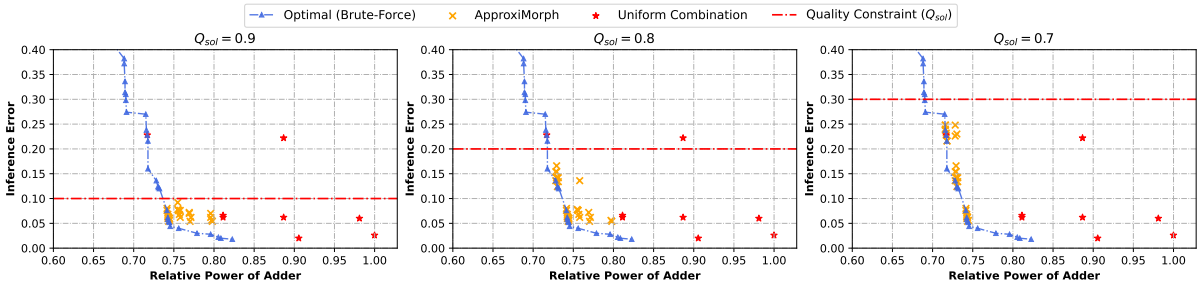

Fig. 8. The final result of the proposed algorithm for the five-layer neural network [784:512:256:128:10] using the MNIST dataset.

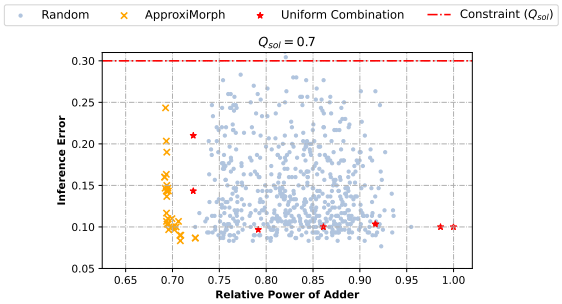

Fig. 9. The final result of the proposed algorithm for the VGG16 model using the CIFAR-10 dataset.

calculating the mathematical equation presented in previous works [42]–[44].

First, we examine the performance of the proposed optimization algorithm for MNIST and CIFAR-10, focusing on its validity and time efficiency. To prove the validity, we execute the brute-force search for MNIST, which provides ground truth results, and compare it with the solutions obtained from *ApproxiMorph*. For CIFAR-10, we select random combinations to compare the results with the obtained solutions, as the brute-force search is unavailable due to the size of the feasible decision space. (For instance, since we use 12 approximate adders for VGG16 model, $12^{16} \sim 1.8 \times 10^{17}$ combination is possible in total. In our machine, since a single evaluation for the VGG16 model requires about 8.38 min (8.38 min/combination), even with only 100 validation data, it will be over 1.07×10^{15} days, which is over 2.9×10^{12} years.) In addition, the number of execution times of the target neural network model is measured and compared with all possible combinations to evaluate the time efficiency of *ApproxiMorph*. This is because executing the neural network is the most time-consuming part of the algorithm.

Second, the transformation of power consumption and accuracy for different voltage scaling scenarios is evaluated using the derived configurations for layer-wise approximation from

TABLE I
THE COMPARISON OF THE NUMBER OF NEURAL NETWORK EXECUTIONS

Dataset	Configuration	Quality Constraint (Q_{sol})		
		$= 90\%$	$= 80\%$	$= 70\%$
MNIST	Four-layer, This work	≤ 66	≤ 78	≤ 101
	Four-layer, Brute-force	2197		
	Five-layer, This work	≤ 127	≤ 172	≤ 192
	Five-layer, Brute-force	28561		
CIFAR-10	VGG16, This work	N/A	N/A	1392[1]
	VGG16, Brute-force	1.85×10^{17}		

[1] The value is the maximum number obtained by 10 times independent executions.

Algorithm 1. Since there is no available hardware design for the VGG16 model in this work, we evaluate only the case of MNIST. To this end, we implemented the memory system with five stacked layers ($M = 5$), where one layer is dedicated solely for the sign bit, and the other four layers are committed to the 8-bit fractional part. In addition, we investigate two network models, the four-layer model [784:256:128:10] and the five-layer model [784:512:256:128:10]. To simulate the influence of memory approximation under the approximate neuron implementation, we insert faults into synaptic weights of the trained network model based on the SRAM's bit error rate (BER) characteristic and evaluate the impact on the quality. The fault positions are distributed randomly using a Monte Carlo simulation, following a uniform distribution. After that, the power consumption of our hardware system is evaluated using Synopsys PrimeTime.

B. Performance of Optimization Algorithm

This subsection presents the proposed algorithm's performance in terms of 1) validity and 2) execution time. Here, the approximation is only applied to neuron cells, so memory is assumed to be in normal operation.

1) Validity of solutions: Fig. 7 and Fig. 8 show the comparison results of the proposed algorithm for the MNIST. In the experiment, we evaluate the accuracy of the approximate model using reduced datasets. The power is assessed by calculating the relative power consumption of the approximate

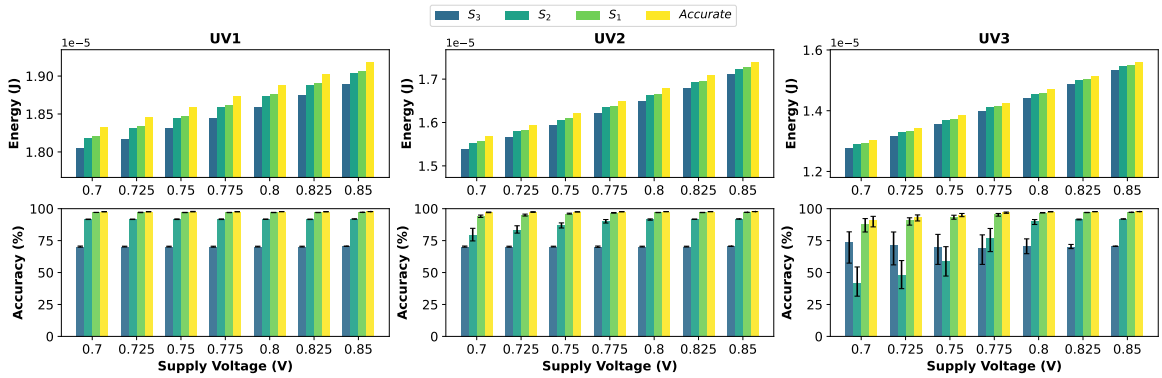

Fig. 10. Evaluation of energy and accuracy **before** weight-tuning with undervolting for the four-layer neural network.

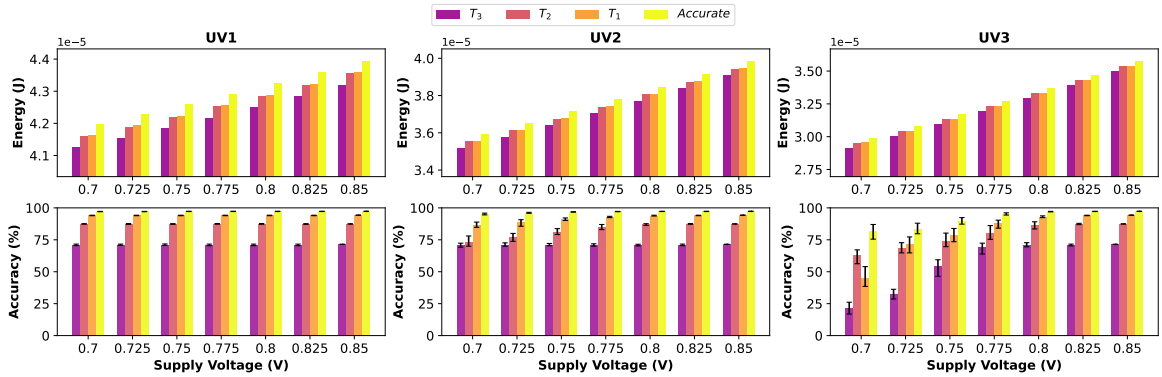

Fig. 11. Evaluation of energy and accuracy **before** weight-tuning with undervolting for the five-layer neural network.

TABLE II
ACCURACY (%) AND POWER SAVING (%) OF APPROXIMATE ADDERS

Model	Metric	Accurate	Approximated Version
MNIST: four-layer	Version	fixed	S_1 / S_2 / S_3
	Accuracy (%)	97.74	97.14 / 91.92 / 70.60
	Power Save (%)	0.00	28.06 / 31.85 / 40.56
MNIST: five-layer	Version	fixed	T_1 / T_2 / T_3
	Accuracy (%)	97.30	94.28 / 87.43 / 71.45
	Power Save (%)	0.00	25.64 / 27.18 / 30.87
CIFAR10-VGG16	Version	fixed	U_1 / U_2 / U_3
	Accuracy (%)	87.46	89.63 / 89.15 / 84.90
	Power Save (%)	0.00	29.16 / 30.11 / 30.83

adder to the non-approximate circuit. We experimentally set the quality constraint for the initial state to $Q_{init} = 90\%$ accuracy, observing the quality of the uniform approximation, and the maximum number of solutions to $M_{pop} = 30$. The exploration loop is iterated 30 times regardless of constraints. Here, we also consider three cases for solution quality constraints: $Q_{sol} = 70\%$, 80%, and 90% accuracy. As shown in Fig. 7 and Fig. 8, the proposed algorithm can derive solutions on the Pareto-front or its neighbor below the quality threshold (red-dashed line). Thus, we conclude that the algorithm can find the optimal solutions for the multi-layer neural networks.

Fig. 9 presents the result of the proposed algorithm for CIFAR-10 with the VGG16 model. We set the quality constraint for the initial state to $Q_{init} = 90\%$ accuracy, and the maximum number of solutions to $M_{pop} = 30$. The exploration loop is iterated 50 times regardless of constraints. Here, we examine only one case of solution quality constraints, $Q_{sol} = 70\%$ accuracy. As a result of the random selection, we cannot find a combination of approximate adders that results in better quality-energy trade-offs than the obtained solutions. Therefore, although our approach assumes that optimal or near-optimal solutions can be obtained from a reduced decision space (i.e., reduced set of approximate adders), these results indicate that such an approach is available, at least in the SNN applications discussed in this work.

From the Pareto front of the obtained final solutions in the case of $Q_{sol} = 70\%$ accuracy, we select six (S_1 to S_3, T_1 to T_3) and three (U_1 to U_3) configurations for MNIST and CIFAR-10, respectively, as examples. Here, S_1 to S_3, T_1 to T_3, and U_1 to U_3 are optimal approximate configurations of the four-layer, five-layer, and VGG16 models, respectively. Table II shows the accuracy against all test data and power savings on adder circuits for selected samples. For instance, S_1 achieves 28.06% power savings with 97.14% accuracy, and U_1 saves 29.16% of power consumption while having 89.63% accuracy, which is a slightly better result than an accurate implementation.

By choosing different Q_{sol} values, the impact of this hyperparameter becomes evident. However, selecting an appropriate Q_{sol} remains an open problem, as designers must adjust it according to the neural network models, datasets, and other constraints such as power and accuracy.

2) Time efficiency of algorithm: Table I summarizes the number of execution times of neural networks with *ApproxiMorph* and the brute-force search. In our experiment, since the type of approximate adders for MNIST is 13 ($A = 13$) and is 12 ($A = 12$) for CIFAR-10, the possible combinations are $13^3 = 2197$ for the four-layer model ($B = 3$), $13^4 = 28561$ for the five-layer model ($B = 4$), and about 1.85×10^{17} for the VGG16 model. However, *ApproxiMorph* requires at most

101 executions for the four-layer model and 192 times for the five-layer model in the case of $Q_{sol} = 70\%$. The value decreases to 66 and 127 for the four-layer and five-layer models, respectively, when the quality constraint is $Q_{sol} = 90\%$. Thus, in cases of MNIST, the total number of executions was significantly reduced, and the required number is only 3.00-4.59% and 0.44-0.67% of total combinations for the four- and five-layer models, respectively. We can reasonably expect the number to decrease further as the possible combinations increase, as indicated by the VGG16 result, where the required number is only 1392 out of over 10^{17} at $Q_{sol} = 70\%$.

In summary, ApproxiMorph provide a reasonable number of excution time (i.e., 192 executions on five layers MNIST for $Q_{sol} = 70\%$ took 17.71 min to run or 1391 executions on VGG-16 for CIFAR-10 requires 194.47 hrs of execution time in our program and machine), therefore, designers can opt to run our approach in parallel to other designing phases as approximate adders can be swapped out without significant modification to the source code. In case execution time becomes problematic, our uniform approximation (all adders in all layers are the same type) design on [7] can be a viable and alternative solution.

C. Operation with Approximate Memory

So far, we have applied the approximation technique only to neuron cells. In this subsection, the impact of the combined use of approximate neurons and memory on quality and energy consumption is evaluated using the voltage scaling strategy. The experimental setup of the memory for the evaluation is shown in Appendix C. Our hardware operates with a supply voltage of $1.1V$ in normal operation. When driving the memory layer in the undervolting condition, we reduce the supply voltage to the near-threshold voltage in the $0.7V$ to $0.85V$ range. To apply the power-gating technique, the supply voltage is turned off, and all bits inside of the memory layer become zero. In this work, we fix the supply voltage of m_1 to $V_{DD} = 1.1V$ to keep the sign bit unchanged.

Fig. 10 summarizes the energy and accuracy transformation for selected samples with undervolting (UV1 to UV3) in the four-layer neural network. In UV1 mode, at the supply voltage $V_{DD} = 0.7V$, S_1, S_2, and S_3 save 13.18%, 13.33%, and 13.97% of energy consumption compared to the non-approximate implementation, while reducing 0.12%, 0.17%, and 0.35% accuracy from the values in Table II (baseline), respectively. Similarly, in UV2 mode, power savings of up to 25.37% for S_1, 25.40% for S_2, and 25.57% for S_3 can be achieved at the cost of slightly greater accuracy loss. Although S_2 originally shows better accuracy than S_3 in UV3 mode, S_2 results in significant accuracy degradation from $V_{DD} = 0.775V$ and finally shows worse results than S_3. Considering the configuration having above 70% accuracy, for example, S_1, S_2, and S_3 achieve $38.05\%|_{V_{DD}=0.7V}$, $32.25\%|_{V_{DD}=0.775V}$, and $38.36\%|_{V_{DD}=0.7V}$ energy savings, respectively, in UV3 mode. This example indicates that noise resilience varies depending on the approximate implementation, which implies that detailed investigations must be needed for practical uses.

Fig. 11 shows the transformation of energy and accuracy with undervolting (UV1 to UV3) in the five-layer neural network. At the supply voltage $V_{DD} = 0.7V$ in the UV1 mode, approximate implementations can reduce energy by 13.33-14.12% while restricting accuracy loss within 1% from their baseline. In UV2 mode, the accuracy of T_1 and T_2 is maintained up to around $V_{DD} = 0.8V$ and decreases thereafter, while the accuracy of T_3 is maintained. Moreover, T_1, T_2, and T_3 can save 30.11%, 30.13%, and 30.35% of energy consumption at $V_{DD} = 0.8V$ in the UV3 mode, while having about 1% accuracy loss from their baseline.

On the other hand, as shown in Fig. 12 and Fig. 13, applying the power-gating to the top memory layer (m_5) greatly influences the accuracy of selected samples, while the non-approximate model still maintains the accuracy. This result indicates that 6-bit quantization of the fractional parts is available in the non-approximate model but not in the approximate implementation without weight-tuning. In this case, the benefit of energy savings from the approximate implementation of neuron cells will disappear since the energy consumption of the logic segment is only about 5.7% of the total. To address this drawback, we examine the impact of the weight-tuning method on the approximate implementation in the following subsection.

D. Impact of Weight-Tuning

We have used the synaptic weights without changing their values to evaluate the accuracy under the reduced voltage conditions. In this subsection, we discuss the impact of the weight adjustment method and evaluate it using approximate neurons and memory. Fig. 14 and Fig. 15 show the effect on the classification accuracy after the weight-tuning with different supply voltages. In both cases, we observed an improvement in accuracy and the capability of the operation combining undervolting and power-gating strategies, which is not available without weight-tuning as described in the previous subsection. For instance, with the use of the undervolting and power-gating methods, S_1, S_2, and S_3 achieve 96.7%, 88.54%, and 69.12% accuracy, which are similar to the original accuracy, at the supply voltage $V_{DD} = 0.8V$ in the UV-PG2 mode. Furthermore, T_1, T_2, and T_3 in UV3 mode contribute to 95.09%, 91.25%, and 90.44% accuracy, which are 1.95%, 4.72%, and 19.36% higher accuracy, respectively, compared to that without weight-tuning.

In the operation using undervolting and power-gating (UV-PG1 to UV-PG3), the selected samples show the error tolerance trends compared to the accurate implementation. For example, S_1 achieves 82.19% accuracy at the supply voltage $V_{DD} = 0.7V$ in UV-PG2, which is over 50% higher accuracy than an accurate implementation. In addition, T_3 in UV-PG1 to UV-PG3 mode shows outstanding fault tolerance after the weight-tuning. For instance, it provides 90.30% accuracy at $V_{DD} = 0.7V$ in UV-PG1 mode and 86.38% accuracy at $V_{DD} = 0.8V$ in UV-PG3 mode. These results indicate that the approximate implementation of the SNN application is a promising solution for low-power hardware systems.

E. Comparison

Table III shows the comparison results between our work and other previous works [19], [20], [48], [49]. In all cases,

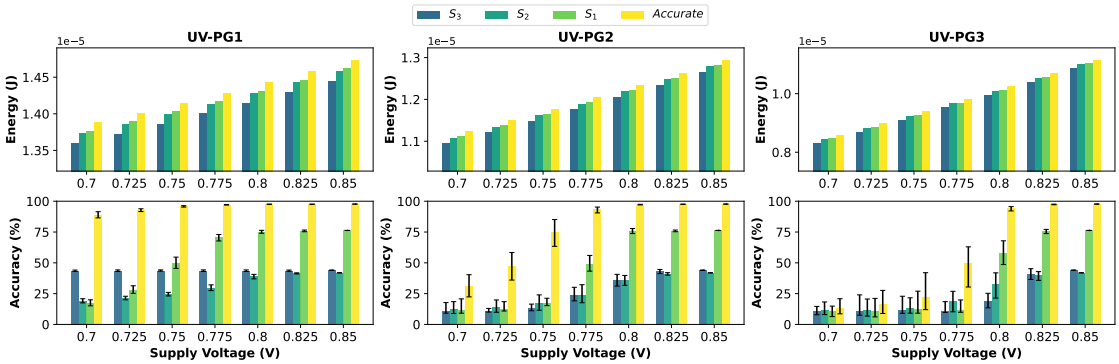

Fig. 12. Evaluation of energy and accuracy **before** weight-tuning with undervolting and power-gating for the four-layer neural network.

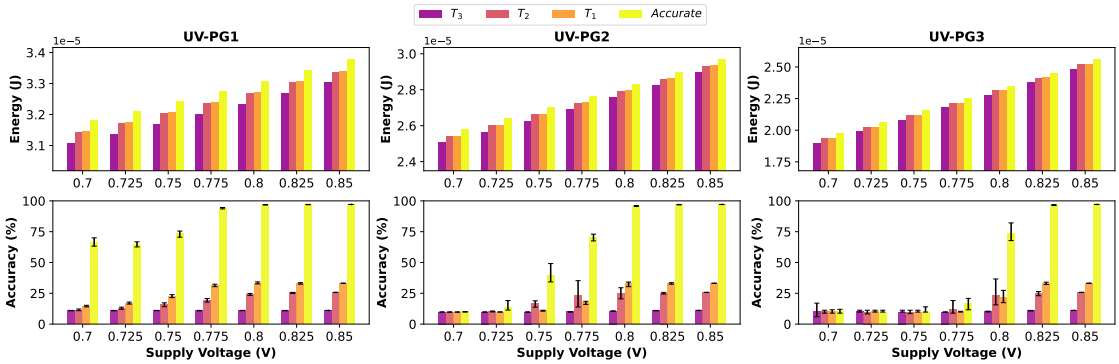

Fig. 13. Evaluation of energy and accuracy **before** weight-tuning with undervolting and power-gating for the five-layer neural network.

accuracy is based on the MNIST benchmark after weight tuning. Besides the accurate implementation (case 1), we selected four approximate configurations (cases 2 to 5) for comparison. As shown in Table III, our system has competitive accuracy compared to the prior works, even with the approximate implementation.

In terms of energy consumption, we compare our work with others using the energy per Synaptic Operation (SOP). For a fair comparison, we use the well-known scaling equation presented in [50] to scale down to the 14-nm technology node. As a result, our hardware consumes 8.797pJ with the accurate implementation (case 1), and the value decreases to 5.163pJ and 3.057pJ with 0.63% (case 2) and 3.17% (case 3) accuracy loss. After scaling down to the 14-nm technology node, their energy per SOP achieves 0.504pJ, 0.296pJ, and 0.175pJ accordingly. In the five-layer model (case 4 and case 5), the energy per SOP is 0.338pJ and 0.223pJ, which is achieved by 7.00% and 10.92% accuracy loss, respectively, compared to the baseline result. Here, please note that the value of energy per SOP is based on the amount of synaptic activities and thus differs depending on the implementation and algorithm of SNNs. In conclusion, these results prove that our system can achieve competitive implementation of the existing works.

VI. Discussion

In this section, the limitations of this work and their potential solutions are discussed.

First, 3D-IC-based implementation offers network-on-chip (NoC) scalability and high-bandwidth communication capabilities with a three-dimensional architecture, but reliability and thermal issues are major concerns [51], [52]. To address reliability issues, many previous works have added redundancies to mitigate the defects of TSV connections in conjunction with the routing algorithm. The thermal-aware design and algorithm also contribute to dealing with dissipation issues. Therefore, although we did not consider these problems in this paper, the methods above can be implemented in our hardware system to cover the architectural concerns.

Second, although we examine only the case of uniform approximation for memory systems to keep evaluation simple, there are many choices regarding the supply voltage and memory splitting patterns. For instance, the hardware designer can consider non-uniform voltage scaling schemes (e.g. $\{m_1, m_2, m_3, m_4\} = \{1.1V, 0.8V, 0.75V, 0.7V\}$) and the different number of memory layers (e.g. $M = 2$, 4, 8, or more). Since examining all possible configurations is mostly infeasible, we only covered the cases summarized in Table I of the supplemental document.

Third, although we utilized the SRAM to store the synaptic weights in this work, other advanced memory technologies, such as ReRAM, FeFET, and STT-RAM, can also be integrated into the memory. They have near-zero leakage power consumption and can retain their values even after applying the power-gating method. Thanks to the die-stacking implementation, it is possible to use them only for LSBs, which mitigates the negative impact of signal noise resulting from their analog operation, considering the noise-resilient nature of SNNs. In addition, since the SRAM's internal power dominates the total power in our implementation, using non-volatile memory can significantly reduce the total energy consumption.

Fourth, our work employed the ANN-to-SNN conversion method, utilizing floating-point values and subsequently quantizing the trained weights to fixed-point values. However, quantization-aware or approximation-aware training can also

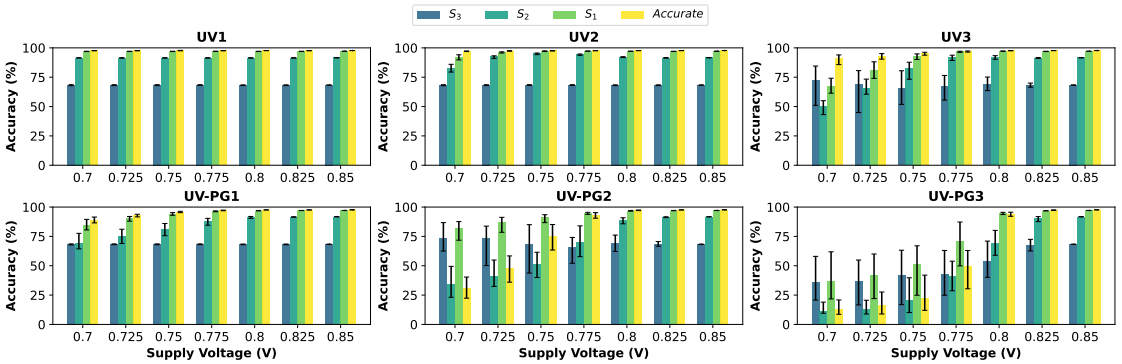

Fig. 14. Evaluation of accuracy **after** weight-tuning in the four-layer neural network.

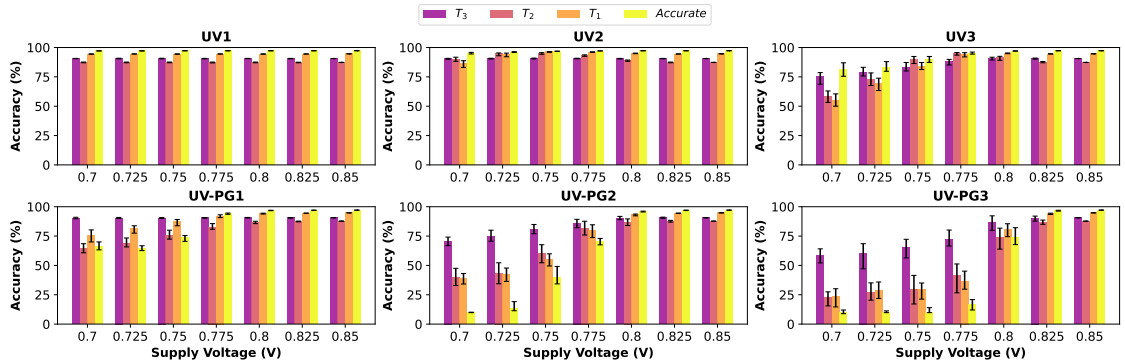

Fig. 15. Evaluation of accuracy **after** weight-tuning in the five-layer neural network.

be exploited. We can expect to perform with better accuracy and improve noise resilience during approximate operations, but the extent of this improvement is not clearly understood currently.

Fifth, since we focus on lowering the power consumption, the shape of the results is an I-shape frontier rather than an L-shape one, as observed in the GA-based approach. This is especially attributed to the definition of the initial state and the uni-directional generation algorithm. Therefore, to explore the horizontal line of an L-shape frontier further, the hardware designer is required to use the standard GA-based approach in place of the proposed methodology.

Sixth, as shown in Algorithm 1, the ApproxiMorph relies on a traditional flow of Genetic Algorithm; therefore, it is expected that the run-time complexity of this work is similar to traditional Genetic Algorithm or Reinforcement Learning under the assumption that the number of generations and evaluations per generation are equivalent.

Seventh, although this work evaluates the benefits of our approach only on multi-layer perceptrons and the VGG16 model, it can be readily extended to more advanced architectures, provided that the approximate operator (e.g., approximate adder or multiplier) is defined. However, we note that emulating approximate circuits on a CPU is not sufficient for evaluating each configuration in more complex networks. Therefore, a strategy for leveraging GPUs to accelerate approximate implementations is necessary. For instance, the work in [53] introduced a fast DNN emulation system that supports approximate operators using GPU computation, enabling agile testing of approximate networks.

Eighth, the benefit of weight-tuning after the approximation is only evaluated in less-complex network architectures in this work. For more complex networks, a previous work [39] has shown the accuracy improvement in the ResNet architecture with approximate operators. Although it focuses on the ANN, we can see the potential to be applied to SNNs as well, and room for further exploration remains. Furthermore, the metric used in weight-tuning (ours is MRE) needs more investigation and analysis on its impact.

Ninth, when approximating the earlier layer, it is empirically shown that the succeeding computation compensates for the error propagated from previous layers. However, the approximation of the deeper layer significantly affects and changes the final accuracy depending on the type of approximate adder. This problem necessitates further investigation, and the impact of error propagation could suggest more optimal approaches for design exploration for approximate design.

Tenth, in this work, the neural architecture deployment is fixed. However, in general, large-scale SNN implementations often rely on Network-on-Chip (NoC) communication among clusters of neurons [19], [20]. Although this work does not address issues related to large-scale on-chip networks and mapping strategies, it is reasonable to expect that ApproxiMorph can be extended to such designs. One concern with NoC-based systems is that the approximation of neuron circuits may need to be applied on a cluster-by-cluster basis, rather than the layer-by-layer approach used in this study. This shift may introduce new challenges related to cluster partitioning and mapping.

Eleventh, in this work, we did not perform weight tuning for VGG-16 due to the excessive execution time (estimated at 7.079×10^5 minutes, or approximately 491 days). However, this can be executed on the more optimized program, and further exploration should be needed. Solving this limitation

TABLE III
COMPARISON RESULTS BETWEEN THE PROPOSED APPROACH AND THE EXISTING WORKS

Model	Acc.(%)	Arch.	Tech.	Energy per SOP (pJ)	Energy per SOP (pJ) (in 14nm)
TrueNorth [20]	91.94	2D	28nm	26 (0.775V)	4.902
Loihi [19]	96	2D	14nm FinFET	23.6 (0.75V)	23.6
ODIN [48]	84.5	2D	28nm FD-SOI	8.4	1.078
NASH [49]	79.4	3D	45nm	11.3 (1.1V)	0.648
[13]	95.35	3D	45nm	244.28	14.02
	94.84			191.46	10.98
	88.77			81.16	4.65
[7]	94.8	3D	45nm	20.33	1.167
	93.9			13.28	0.762
	77.6			8.374	0.48
This work	97.74[1]	3D	45nm	8.797[1]	0.504[1]
	97.11[2]			5.163[2]	0.296[2]
	94.57[3]			3.057[3]	0.175[3]
	90.30[4]			5.900[4]	0.338[4]
	86.38[5]			3.898[5]	0.223[5]

[1] Case 1: Accurate implementation (four-layer model).
[2] Case 2: $V_{DD} = 0.8V$ in UV3 mode using the configuration S_1.
[3] Case 3: $V_{DD} = 0.8V$ in UV-PG3 mode using the configuration S_1.
[4] Case 4: $V_{DD} = 0.7V$ in UV-PG1 mode using the configuration T_3.
[5] Case 5: $V_{DD} = 0.8V$ in UV-PG3 mode using the configuration T_3.

could be one of the potential future works.

Although this work has several drawbacks, as being discussed above, there are many studies that also aim to address them. Therefore, the proposed approach has the potential to realize low-power neuromorphic systems with optimal quality-energy trade-offs, which enables use in power-constrained edge devices.

VII. CONCLUSIONS

This work presents *ApproxiMorph*, a low-power neuromorphic framework that combines a time-efficient heuristic design exploration for layer-wise SNN approximation with 3D-stacked SRAM. The proposed algorithm significantly reduces search time, requiring only 0.67% of evaluations among all possible configurations in a five-layer SNN using 13 approximate adders. Additionally, weight tuning enables integration with approximate memory, demonstrating error tolerance compared to accurate implementations. Future work could involve extending the proposed method to support the execution of multiple SNN models, including larger models and more complex datasets. The selection of post-approximation weight-tuning metrics and hyperparameters for ApproxiMorph also needs to be explored and addressed.

ACKNOWLEDGEMENT

We gratefully acknowledge the reviewers for their insightful comments, which have helped improve the quality and clarity of this paper.

This work is supported by the Competitive Research Funding Ref. 2025-21.

REFERENCES

[1] D. Patterson *et al.*, "The carbon footprint of machine learning training will plateau, then shrink," *Computer*, vol. 55, no. 7, pp. 18–28, 2022.

[2] C.-J. Wu *et al.*, "Sustainable ai: Environmental implications, challenges and opportunities," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 795–813, 2022.

[3] B. C. Lee *et al.*, "Carbon connect: An ecosystem for sustainable computing," *arXiv preprint arXiv:2405.13858*, 2024.

[4] R. Desislavov *et al.*, "Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100857, 2023.

[5] A. Tavanaei *et al.*, "Deep Learning in Spiking Neural Networks," *Neural networks*, vol. 111, pp. 47–63, 2019.

[6] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. Homogeneous synaptic input," *Biological cybernetics*, vol. 95, pp. 1–19, 2006.

[7] R. Kobayashi *et al.*, "Energy-Efficient Spiking Neural Networks Using Approximate Neuron Circuits and 3D Stacking Memory," in *2024 IEEE 17th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 2024, pp. 421–425.

[8] S. Sen *et al.*, "Approximate Computing for Spiking Neural Networks," in *Design, Automation & Test in Europe Conference & Exhibition, 2017*. IEEE, 2017, pp. 193–198.

[9] R. Putra *et al.*, "Sparkxd: A framework for resilient and energy-efficient spiking neural network inference using approximate dram," in *2021 58th ACM/IEEE Design Automation Conference*. IEEE, 2021, pp. 379–384.

[10] J. Han and M. Orshansky, "Approximate Computing: An Emerging Paradigm For Energy-Efficient Design," in *2013 18th IEEE European Test Symposium*. IEEE, 2013, pp. 1–6.

[11] S. Venkataramani *et al.*, "AxNN: Energy-efficient neuromorphic systems using approximate computing," in *Proceedings of the 2014 international symposium on Low power electronics and design*, 2014, pp. 27–32.

[12] 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, p. 937782, 2022.

[13] N.-D. Nguyen *et al.*, "Power-Aware Neuromorphic Architecture With Partial Voltage Scaling 3-D Stacking Synaptic Memory," *IEEE Trans. VLSI Syst.*, vol. 31, no. 12, pp. 2016–2029, 2023.

[14] M. Prezioso *et al.*, "Training and Operation of an Integrated Neuromorphic Network Based on Metal-Oxide Memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[15] B. Gao *et al.*, "Metal oxide resistive random access memory based synaptic devices for brain-inspired computing," *Japanese Journal of Applied Physics*, vol. 55, no. 4S, p. 04EA06, 2016.

[16] G. Zhong *et al.*, "Flexible electronic synapse enabled by ferroelectric field effect transistor for robust neuromorphic computing," *Applied Physics Letters*, vol. 117, no. 9, p. 092903, 09 2020.

[17] M. Hu *et al.*, "Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Matrix-Vector Multiplication," in *53rd annual Design Automation Conference*, 2016, pp. 1–6.

[18] K. Roy *et al.*, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

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

[20] F. Akopyan *et al.*, "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, 2015.

[21] H. An *et al.*, "Three-Dimensional Neuromorphic Computing System With Two-Layer and Low-Variation Memristive Synapses," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 3, pp. 400–409, 2021.

[22] D. Kim *et al.*, "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 380–392, 2016.

[23] K. Ueyoshi *et al.*, "QUEST: Multi-Purpose Log-Quantized DNN Inference Engine Stacked on 96-MB 3-D SRAM Using Inductive Coupling Technology in 40-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 186–196, 2018.

[24] K. Shiba *et al.*, "A 96-MB 3D-Stacked SRAM Using Inductive Coupling With 0.4-V Transmitter, Termination Scheme and 12:1 SerDes in 40-nm CMOS," *IEEE Trans. Circuits Syst. I*, vol. 68, no. 2, pp. 692–703, 2020.

[25] R. V. W. Putra and M. Shafique, "Q-SpiNN: A Framework for Quantizing Spiking Neural Networks," in *2021 International Joint Conference on Neural Networks*. IEEE, 2021, pp. 1–8.

[26] A. Hasssan *et al.*, "SpQuant-SNN: ultra-low precision membrane potential with sparse activations unlock the potential of on-device spiking neural networks applications," *Frontiers in Neuroscience*, vol. 18, p. 1440000, 2024.

[27] K. K. Chang *et al.*, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, pp. 1–42, 2017.

[28] B. Reagen *et al.*, "Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 267–278, 2016.

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

[30] J. Zhao *et al.*, "An Energy-Efficient 3D CMP Design with Fine-Grained Voltage Scaling," in *2011 Design, Automation & Test in Europe*. IEEE, 2011, pp. 1–4.

[31] N.-D. Nguyen *et al.*, "An In-Situ Dynamic Quantization With 3D Stacking Synaptic Memory for Power-Aware Neuromorphic Architecture," *IEEE Access*, 2023.

[32] H. Jiang *et al.*, "Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.

[33] R. Kobayashi and K. N. Dang, "An Efficient Hardware Implementation of Spiking Neural Network Using Approximate Izhikevich Neuron," in *9th International Conference on Integrated Circuits, Design, and Verification*. IEEE, 2024, pp. 13–18.

[34] V. Mrazek *et al.*, "EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods," in *Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2017, pp. 258–261.

[35] G. Zervakis *et al.*, "Multi-Level Approximate Accelerator Synthesis Under Voltage Island Constraints," *IEEE Trans. Circuits Syst. II*, vol. 66, no. 4, pp. 607–611, 2018.

[36] V. Mrazek *et al.*, "autoAx: An Automatic Design Space Exploration and Circuit Building Methodology utilizing Libraries of Approximate Components," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[37] B. S. Prabakaran *et al.*, "ApproxFPGAs: Embracing ASIC-Based Approximate Arithmetic Components for FPGA-Based Systems," in *2020 57th ACM/IEEE Design Automation Conference*. IEEE, 2020, pp. 1–6.

[38] M. Manuel *et al.*, "Region of Interest-Based Parameter Optimization for Approximate Image Processing on FPGAs," *International Journal of Networking and Computing*, vol. 11, no. 2, pp. 438–462, 2021.

[39] V. Mrazek *et al.*, "ALWANN: Automatic Layer-Wise Approximation of Deep Neural Network Accelerators without Retraining," in *2019 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2019, pp. 1–8.

[40] D.-A. Nguyen *et al.*, "GAQ-SNN: A Genetic Algorithm based Quantization Framework for Deep Spiking Neural Networks," in *2022 International Conference on IC Design and Technology*, 2022, pp. 93–96.

[41] N.-D. Nguyen *et al.*, "Noma: a novel reliability improvement methodology for 3-d ic-based neuromorphic systems," *IEEE Trans. Compon. Packag. Manuf. Technol.*, 2024.

[42] P. Royer *et al.*, "Using pMOS Pass-Gates to Boost SRAM Performance by Exploiting Strain Effects in Sub-20-nm FinFET Technologies," *IEEE Trans. Nanotechnol.*, vol. 13, no. 6, pp. 1226–1233, 2014.

[43] P. Reviriego *et al.*, "Error-Tolerant Data Sketches Using Approximate Nanoscale Memories and Voltage Scaling," *IEEE Trans. Nanotechnol.*, vol. 21, pp. 16–22, 2021.

[44] E. Seevinck *et al.*, "Static-noise margin analysis of MOS SRAM cells," *IEEE J. Solid-State Circuits*, vol. 22, no. 5, pp. 748–754, 1987.

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

[46] NCSU EDA. FreePDK3D45 3D-IC Process Design Kit. [Online]. Available: https://eda.ncsu.edu/freepdk/freepdk3d45

[47] M. R. Guthaus *et al.*, "OpenRAM: An open-source memory compiler," in *2016 IEEE/ACM International Conference on Computer-Aided Design*, 2016, pp. 1–6.

[48] C. Frenkel *et al.*, "A 0.086-mm^2 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28-nm CMOS," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 1, pp. 145–158, 2018.

[49] O. M. Ikechukwu *et al.*, "On the Design of a Fault-Tolerant Scalable Three Dimensional NoC-Based Digital Neuromorphic System With On-Chip Learning," *IEEE Access*, vol. 9, pp. 64 331–64 345, 2021.

[50] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," *Integration*, vol. 58, pp. 74–81, 2017.

[51] F. Ye and K. Chakrabarty, "TSV Open Defects in 3D Integrated Circuits: Characterization, Test, and Optimal Spare Allocation," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1024–1030.

[52] J. Cong *et al.*, "Thermal-Aware 3D IC Placement Via Transformation," in *2007 Asia and South Pacific Design Automation Conference*. IEEE, 2007, pp. 780–785.

[53] F. Vaverka *et al.*, "TFApprox: Towards a Fast Emulation of DNN Approximate Hardware Accelerators on GPU," in *2020 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2020, pp. 294–297.

Ryoji Kobayashi is currently a Master student at Graduate School of Engineering, University of Tokyo, Japan. He received his B.Sc. from the University of Aizu, Aizu-Wakamatsu, Japan in 2025. His work was conducted at the University of Aizu under the supervision of Assoc. Prof. Khanh N. Dang. His research interests include energy-efficient computing, fault-tolerant/reliable systems, and 3D-ICs.

Ngo-Doanh Nguyen is currently a Ph.D. candidate in Electromagnetism, Electronics, and Telecommunications at the University of Genoa, Italy. He received an M.Sc. degree in Computer Science and Engineering at the Graduate School of Computer Science and Engineering, The University of Aizu, Aizuwakamatsu, Japan, in 2024. He was a Research Engineer with the Information Technology Institute, Vietnam National University, Hanoi (2018-2022), where he received a B.Sc. in Electronics and Communications Engineering (ECE), in 2019. His research interests include hardware/software co-design and verification, as well as low-power solutions for artificial intelligence.

Abderazek Ben Abdallah (Senior Member, IEEE) is a Full Professor at the University of Aizu, Japan, serving as Regent and Dean of the School of Computer Science and Engineering since April 2022. Previously, he was Head of the Computer Engineering Division (2014–2022). Dr. Ben Abdallah holds a Ph.D. in Computer Engineering from the University of Electro-communications (UEC), Tokyo, in 2002. With over 20 years of experience in academia and research, his expertise encompasses computer architecture, neuromorphic circuits and systems, advanced on-chip interconnects, fault tolerance and reliability, and embedded systems. He has authored four books, published 160+ papers, secured 15 grants, and holds 14 patents. He is a Senior Member of IEEE and ACM, as well as a member of Sigma Xi, the Scientific Research Honor Society.

Nguyen Anh Vu Doan is currently with Infineon Technologies AG, Germany. He was previously with Fraunhofer IKS as a Senior Scientist, having worked as a Postdoctoral Researcher first with the Amano lab at Keio University (Japan) and then with the Chair of Integrated Systems at the Technical University of Munich (Germany). He received a Ph.D. degree in electrical engineering from Université libre de Bruxelles (Belgium) in 2015. His research interests include design space exploration, design automation, multiobjective optimization, and multi-criteria decision aiding.

Khanh N. Dang (Member, IEEE) is currently an Associate Professor at the University of Aizu. He received his B.Sc., M.Sc., and Ph.D. degree from Vietnam National University Hanoi, University of Paris-XI, and University of Aizu, in 2011, 2014, and 2017, respectively. He has co-authored two books, 60+ journal and conference papers, and three Japanese patents. His research interests include 3D-ICs, on-chip communication, neuromorphic computing, low-power, and fault-tolerant systems.