Architecture and Algorithms for Robust Reconfigurable Neuromorphic Systems

WILLIAMS YOHANNA YERIMA

A DISSERTATION

SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

IN COMPUTER SCIENCE AND ENGINEERING

Graduate Department of Computer and Information Systems

The University of Aizu

2024

Declaration

I confirm that the research contained in this dissertation unless otherwise specified is my original work. I further attest that this dissertation has not been submitted for any other degree at any other university. I have not incorporated in part or in whole any data, figures, graphs, or information from other individuals, researchers, or online sources, unless properly acknowledged and cited.

Signed: ____________

Date: ____________

THE THESIS TITLED

Architecture and Algorithms for Robust Reconfigurable Neuromorphic Systems

BY

WILLIAMS YOHANNA YERIMA

IS REVIEWED AND APPROVED BY:

Chief referee

Professor Date Feb. 22, 2024

BEN Abderazak Abdallah [illegible]

Professor Date 2024.2.22

NAKASATO Naohito [illegible] 中里

Senior associate professor Date 2024.2.22

OKUYAMA Yuichi [illegible] 奥山

Associate professor Date Feb. 22, 2024

DAISUKE Suzuki Daisuke Suzuki 鈴木

Associate professor Date Feb 22. 2024

KHANH Dang Nam [illegible]

THE UNIVERSITY OF AIZU

2024

Contents

Chapter 1 INTRODUCTION **1**

1.1 Evolution of Neuromorphic Computing systems 1
1.2 Robustness in Neuromorphic Systems: Problems and Motivation 5
1.3 Dissertation Contributions 9
1.4 Dissertation Structure 10

Chapter 2 FUNDAMENTALS OF NEUROMORPHIC SYSTEMS **12**

2.1 Spiking Neural Networks 13
2.1.1 Learning rules 13
Supervised Learning 14
Unsupervised Learning 14
Reinforcement Learning 15
2.2 Spiking Neural Network Models 16
2.2.1 Hodgkin–Huxley Model 18
2.2.2 Leaky Integrate and Fire Model 19
2.2.3 Izhikevich Model 19
2.3 Coding Schemes 20
2.3.1 Rate coding 20
2.3.2 Temporal coding 20
Time-to-First-Spike 21
Rank Order Coding 21
Phase Coding 22
Latency Coding 23
2.4 Hardware Implementation 23
2.4.1 Platforms 23
2.4.2 Memory Technology 24
2.4.3 Learning 25
2.5 Chapter Summary 26

Chapter 3 RELATED WORKS **27**

3.1 Neuromorphic systems: State of art and Advancement 28

3.2 Neuromorphic systems: Robustness concerns 29
3.3 Existing Solutions To Areas of Neuromorphic Systems For Robustness 30
3.3.1 Fault Recovery In Neural Computation 30
3.3.2 Reliable Communication 31
3.3.3 Reliable Memory Operation 31
3.3.4 Reliable Application Mapping 32
3.4 Chapter Summary 34

Chapter 4 FAULT RECOVERY METHODS FOR NEURAL COMPUTATIONS (SAP and TSM) 35

4.1 Introduction to Fault Recovery in Neural Networks 36
4.2 Impacts of Faults in Spiking Neural Networks 37
4.2.1 Fault Modelling 38
4.3 The Proposed Fault Recovery Algorithms 40
4.3.1 Stuck-at Augmented Pruning Algorithm (SAP) 40
Weak Point 42
4.3.2 Target and Selection Method (TSM) 42
4.4 Evaluation 44
4.4.1 Evaluation Methodology 44
4.4.2 Evaluation Results 44
4.4.3 Impacts of Faults in Spiking Neural Networks 44
4.4.4 Stuck-at Pruning (SAP) Algorithm 46
4.4.5 Target and Selection Method 48
MLP-MNIST 48
CSNN-NMNIST 52
4.5 Chapter Summary 56

Chapter 5 FAULT-TOLERANT MAPPING ALGORITHM OF SNNs TO NEUROMORPHIC SYSTEMS 57

5.1 Fault-tolerant Mapping Overview 58
5.2 Migration-Based Mapping 59
5.3 Migration-Based Mapping with the Proposed Ranking and Selection Mechanism 60
5.3.1 Selection Throughout The System Algorithm 62
Weak Point 65
5.3.2 Selection Cluster by Cluster Algorithm 65
5.6 Evaluation 67
5.7 Evaluation Methodology 67
5.8 Evaluation Results 68
5.8.1 Mapping without RSM 68
5.8.2 Mapping with RSM 72
Mapping efficiency 72

Time complexity 78
Reliability analysis 79
5.9 Chapter Summary 86

Chapter 6 ROBUST MAPPING TO NEUROMORPHIC SYSTEMS 87

6.1 The Proposed Mapping Method (R-MaS3N) 88
6.1.1 Initial Mapping to Neuron Clusters 89
6.1.2 Robust Mapping to Neuron Clusters 89
SNN Layer Partitioning (SLP) Algorithm 93
Neuron Partitioning (NP) Algorithm 96
6.3 Evaluation 99
6.3.1 Evaluation Methodology 99
6.3.2 Evaluation Results 100
Mapping efficiency 100
Mapping cost 104
Neuron utilization behavior 106
Time complexity 108
Reliability Assessment: Mean Time To Failure (MTTF) 110
6.4 Chapter Summary 113

Chapter 7 TOWARDS A ROBUST RECONFIGURABLE NEUROMORPHIC ARCHITECTURE 114

7.1 System Architecture 115
7.2 Neural Tile 116
7.2.1 3D Multi-Cast Router 116
7.2.2 Spiking Neuro Processing Core Architecture 117
Network Interface (NI) 118
7.3 Evaluation methodology 120
7.4 Evaluation results 120
7.5 Conclusion 121

Chapter 8 DISSERTATION SUMMARY AND FUTURE OUTLOOK 122

8.1 Contributions Summary 122
8.2 Future outlook 124
Major Journals 125
Major Conference(s) 125

List of Figures

1.1 Computing power demand of machine learning algorithms on conventional computing architectures over the past 4 decades expressed in PETAFLOPS [1]. *2*

1.2 Costs associated with training AI models have shown a significant increase since 2011. An exponential rise of this magnitude is unsustainable [1]. *3*

1.3 Comparison of the von Neumann architecture with the neuromorphic architecture: (a) Von Neumann-based computing systems architecture, (b) Neuromorphic computing systems architecture. *4*

1.4 Example of neural computation in a simple connected neural network with no faulty neuron. *6*

1.5 Example of neural computation in a simple connected neural network with a faulty neuron. *6*

1.6 Illustrative examples of the mapping of a neural network application to a neural circuit: (i) With sufficient redundancy elements, (ii) With depleted redundancy elements following multiple failures. . . . *8*

1.7 Correlation between scaling factor and number of redundant neurons in large-scale NoC-based neuromorphic systems. *8*

2.1 STDP illustrating how spike timing influences synaptic weight change. When the presynaptic spike precedes (follows) the postsynaptic spike within a window of milliseconds, it leads to weight increase (decrease), resulting in LTP or LTD, respectively. [2] *15*

2.2 Neuron diagram essentially consists of three parts: dendrites, cell bodies, and axons. [2] *16*

2.3 A neuron analog implementation as described in [3] *17*

2.4 A neuron digital implementation as described in [4] *17*

2.5 Schematic illustration of neural coding: (a) Rate coding: Input intensity is converted to firing rates, (b) Time to first spike coding: Information is encoded based on the precise timing of the first spike generated by a neuron with high-intensity input to a neuron firing first, (c) Phase coding: High- and medium-intensity input neurons spike almost at the same time as neurons with low-intensity input that are not synchronized, (d) Latency coding: Spike timing t_1 and t_2 encode information in neurons with medium and least intensity input relative to neurons with the highest intensity. *22*

4.1 MLP-MNIST Architecture. *39*

4.2 CSNN-NMNIST Architecture. *39*

4.3 The impact of SA faults on accuracy in MLP for MNIST dataset. . . 45

4.4 The impact of SA faults on accuracy in CSNN for NMNIST dataset. . 45

4.5 Effect of the SAP method on the accuracy of MLP-MNIST with stuck-at-1 fault. 47

4.6 Effect of the SAP method on the accuracy of MLP-MNIST with stuck-at-0 fault. 47

4.7 Performance comparison of SAP (with 10% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for MLP-MNIST. . 48

4.8 Performance comparison of SAP (with 50% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for MLP-MNIST. . 49

4.9 Performance comparison of SAP (with 80% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for MLP-MNIST. . 49

4.10 Performance comparison of SAP (with 10% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for MLP-MNIST. . 50

4.11 Performance comparison of SAP (with 50% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for MLP-MNIST. . 50

4.12 Performance comparison of SAP (with 80% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for MLP-MNIST. . 51

4.13 Performance comparison of SAP (with 10% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for CSNN-NMNIST. . 52

4.14 Performance comparison of SAP (with 50% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for CSNN-NMNIST. . 53
4.15 Performance comparison of SAP (with 80% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for CSNN-NMNIST. . 53
4.16 Performance comparison of SAP (with 10% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for CSNN-NMNIST. . 54
4.17 Performance comparison of SAP (with 50% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for CSNN-NMNIST. . 54
4.18 Performance comparison of SAP (with 80% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for CSNN-NMNIST. . 55
5.1 An illustration of migration-based mapping on a $3 \times 3 \times 3$ NoC-based neuromorphic system. 60
5.2 An illustration of the migration-based mapping with the proposed RSM on a $3 \times 3 \times 3$ NoC-based neuromorphic system. 61
5.3 An illustration of post-remapping in a $3 \times 3 \times 1$ NoC-based neuromorphic system where each cluster comprises 256 neurons: (a) Post-remapping results following system-wide selection, (b) Post-remapping results with a cluster-by-cluster selection approach within the system. . 63
5.4 Output mapping behavior without the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 256 neurons per cluster. 69

5.5 Output mapping behavior without the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 256 neurons per cluster. *69*

5.6 Output mapping behavior without the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 256 neurons per cluster. *70*

5.7 Output mapping behavior without the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 64 neurons per cluster. *70*

5.8 Output mapping behavior without the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 64 neurons per cluster. *71*

5.9 Output mapping behavior without the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 64 neurons per cluster. *71*

5.10 Output mapping behavior with the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks different). *72*

5.11 Output mapping behavior with the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks different). *73*

5.12 Output mapping behavior with the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks different). 73

5.13 Output mapping behavior with the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks the same). 74

5.14 Output mapping behavior with the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks the same). 74

5.15 Output mapping behavior with the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks the same). 75

5.16 Output mapping behavior with the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks different). 75

5.17 Output mapping behavior with the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks different). 76

5.18 Output mapping behavior with the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks different). 76

5.19 Output mapping behavior with the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks the same). *77*

5.20 Output mapping behavior with the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks the same). *77*

5.21 Output mapping behavior with the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks the same). *78*

5.22 Reliability Plot of RSM for a $4 \times 4 \times 4$ NoC-based neuromorphic system throughout the simulation duration. *82*

5.23 Reliability Plot of RSM for a $5 \times 5 \times 5$ NoC-based neuromorphic system throughout the simulation duration. *82*

5.24 Reliability Plot of RSM for a $6 \times 6 \times 6$ NoC-based neuromorphic system throughout the simulation duration. *83*

5.25 Operational availability plot of RSM for a $4 \times 4 \times 4$ NoC-based neuromorphic system throughout the simulation duration. *83*

5.26 Operational availability plot of RSM for a $5 \times 5 \times 5$ NoC-based neuromorphic system throughout the simulation duration. *84*

5.27 Operational availability plot of RSM for a $6 \times 6 \times 6$ NoC-based neuromorphic system throughout the simulation duration. *84*

5.28 Distribution plot of TTF across three different 3D NoC-based neuromorphic system. 85

5.29 Distribution plot of MTTF across three different 3D NoC-based neuromorphic system sizes. 85

6.1 An illustration of the initial SNN mapping process on a $3 \times 3 \times 3$ NoC-based neuromorphic system: (a) Represents the neural network application, while (b) showcases the neural network application mapped onto the neuro cores of the 3D NoC-based neuromorphic system. Our mapping method, as introduced in [5], is employed for this mapping. It's noteworthy that certain clusters, specifically C_{11}, C_{12}, C_{14}, C_{15}, and C_{16} within NL_1, contain faulty neurons. Consequently, the mapped neurons in these clusters are fewer than the expected maximum capacity. . . 90

6.2 Descriptive overview of the mapping sequence executed on the 3D NoC-based neuromorphic hardware, as detailed in Figure 6.1. The process involves mapping neurons from each layer of the network application to their corresponding layers within the neuromorphic system. . . 91

6.3 An illustrative representation of the proposed solution within a $3 \times 3 \times 1$ NoC-based neuromorphic system, addressing the mapping challenge detailed in Figure 6.1. The sequence unfolds as follows: (a) Depicts the unmapped neurons originating from L_1 of the neural network application after the initial mapping. (b) Illustrates the partitioning of these unmapped neurons into two distinct groups: the most active and least active partitions. (c) Demonstrates the subsequent remapping process where partitioned, unmapped neurons from both the most active and least active partitions are reassigned to neurons located within the less-active regions of the respective layer in the 3D NoC-based neuromorphic system. (d) Offers a visual representation of the sequential steps involved in the remapping process within NL_1. 92

6.4 An illustration of an SNN layer divided into two distinct neuron partitions using the SLP algorithm: (a) The SNN application (b) Sample layout of the characterized neurons (c) Layout of the clustered neurons. . 95

6.5 An illustrative representation of neurons in clusters within a $3 \times 3 \times 1$ NoC-based neuromorphic system, divided into C_{high} and C_{less} partitions utilizing the NP algorithm. 97

6.6 Behavior of output neuron remapping across different fault rates within a $3 \times 3 \times 3$ NoC-based neuromorphic system for *MLP*_1794. . . . 101

6.7 Behavior of output neuron remapping across different fault rates within a $3 \times 3 \times 3$ NoC-based neuromorphic system for *MLP*_2794. . . . *102*

6.8 Behavior of output neuron remapping across different fault rates within a $4 \times 4 \times 4$ NoC-based neuromorphic system for *MLP*_4794. . . . *102*

6.9 Behavior of output neuron remapping across different fault rates within a $4 \times 4 \times 4$ NoC-based neuromorphic system for *MLP*_8794. . . . *103*

6.10 Behavior of output neuron remapping across different fault rates within a $5 \times 5 \times 5$ NoC-based neuromorphic system for *MLP*_9794. . . . *103*

6.11 Behavior of output neuron remapping across different fault rates within a $5 \times 5 \times 5$ NoC-based neuromorphic system for *MLP*_18794. . . *104*

6.12 Plot illustrating applications mapping cost for a $3 \times 3 \times 3$ NoC-based neuromorphic system under various fault rates. *105*

6.13 Plot illustrating applications mapping cost for a $4 \times 4 \times 4$ NoC-based neuromorphic system under various fault rates. *105*

6.14 Plot illustrating applications mapping cost for a $5 \times 5 \times 5$ NoC-based neuromorphic system under various fault rates. *106*

6.15 Plot illustrating neuron utilization across various stages of the mapping method for a $3 \times 3 \times 3$ 3D NoC-based neuromorphic system. . . . *107*

6.16 Plot illustrating neuron utilization across various stages of the mapping method for a $4 \times 4 \times 4$ 3D NoC-based neuromorphic system. . . . *107*

6.17 Plot illustrating neuron utilization across various stages of the mapping method for a $5 \times 5 \times 5$ 3D NoC-based neuromorphic system. . . . 108

6.18 The mean time to failure (MTTF) of R-MaS3N when MLP_{1794} and MLP_{2794} applications are mapped onto a $3 \times 3 \times 3$ NoC-based neuromorphic system. 112

6.19 The mean time to failure (MTTF) of R-MaS3N when MLP_{4794} and MLP_{8794} applications are mapped onto a $3 \times 3 \times 3$ NoC-based neuromorphic system. 112

6.20 The mean time to failure (MTTF) of R-MaS3N when MLP_{9794} and MLP_{18794} applications are mapped onto a $3 \times 3 \times 3$ NoC-based neuromorphic system. 113

7.1 High-level view of the neuromorphic architecture. 115

7.2 High level view of the 3D multicast router [6]. 117

7.3 High-level view of the spiking neuro processing core [6]. . . . 118

7.4 A block diagram of the network interface supporting the MigSpike + RSM mapping method. 119

7.5 A block diagram of the network interface supporting the R-MaS3N mapping method. 119

7.6 Hardware physical design layout of the proposed robust reconfigurable neuromorphic system for a 2×2 NoC size: (a) Layout (b) Schematic layout comprising 256 neuron logic cells and 65K synapses for the crossbar. 121

List of Tables

5.1 Configuration for the evaluation. *68*

5.2 Execution time of RSM for faulty neuron selection in 3D NoCs. . . *79*

6.1 Configuration used for evaluating R-MaS3N. *100*

6.2 Applications used for evaluating R-MaS3N *100*

6.3 R-MaS3N remapping time in the 3D-NoC-based neuromorphic system for different SNN applications. *109*

7.1 Hardware complexity of the proposed robust reconfigurable system. . *121*

List of Abbreviations

SNN	Spiking Neural Network
NoC	Network-on-Chip
MTTF	Mean Time To Failure
IFR	Initial Mapping Failure Rate
RFR	Remap Failure Rate
GA	Genetic Algorithm
MLP	Multi-Layer Perceptron
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FPGA	Field-Programmable Gate Array
AI	Artificial Intelligence
ML	Machine Learning
NP	Neuron Partitioning
SLP	SNN Layer Partitioning
HPC	High-Performance Computing
S_C	Spiking Counts

F_{CS} Frequency of Consecutive Spike Counts

NC Number of Negative Connection

PC Number of Positive Connection

UM Unmapped Neurons

F_N Faulty Neurons

UA_{BR} Unmapped Application Neurons Before Repairs

UA_{AR} Unmapped Application Neurons After Repairs

UN_{NS} Utilized Neurons of Neuromorphic System

3D Three-Dimensional

2D Two-Dimensional

NN Neural Network

UI User Interface

MLP Multilayer Perceptron

LIF Leaky Integrate and Fire

NC Neural Circuit

RSM Ranking and Selection Mechanism

FPGA Field-Programmable Gate Array

RAM Random Access Memory

ASIC Application-Specific Integrated Circuit

SNPC Spiking neuro-processing core

NI Network interface

SAP	Stuck-at Augmented Pruning
TSM	Target and Selection Method
FT	Fault-Tolerant
AER	Address-Event-Representation

TO MY LATE PARENTS, LATE GUARDIAN, AND MY FAMILY

Acknowledgment

First and foremost, I would like to thank my supervisor Prof. Abderazak Ben Abdallah. Throughout this research journey, his guidance, expertise, patience, and unwavering support have been invaluable. This work has been shaped by his encouraging and insightful feedback. Also, I wish to extend my sincere thanks to my co-referees, Prof. Nakasato Naohito, Yuichi, Prof. Okuyama Yuichi, Prof. Daisuke Suzuki, and Prof. Dang Nam Khanh, who have been instrumental in improving the quality of my thesis through their constructive criticism and valuable suggestions.

For their generous financial support, I am particularly grateful to the Fukushima Global Rotary Club, through the Rotary Yoneyama Foundation. As a result of their sponsorship, this research has been made possible. It is truly an honor that they believe in what I'm doing. It is impossible to ignore the significant role played by my club counselors, Mr. Chintaka, Mr. Odera, and Ms. Mieko Suzuki. As well as serving as mentors for club and rotary activities, they treated me as if they were my parents here in Japan. Their support, encouragement, and understanding have greatly contributed to my academic success in Japan.

As well, I would like to express my gratitude to Dr. Wang for taking the time to review my articles before their submission to Dr. Jiang Wang for his immeasurable contributions and assistance throughout my program. I would also like to thank my colleagues Maatar and Dr. Liang for providing me with stimulating discussions and advice that have enhanced my understanding and helped develop a conducive research environment.

I extend heartfelt thanks to the late Mr. D.D. Sani and his family, who warmly embraced me into their lives. My siblings and fiancée, Anna Jonah, have been pillars of unwavering support and understanding, constantly motivating me with their love and belief throughout this journey.

Thesis advisor: Professor Abderazek Ben Abdallah Williams Yohanna Yerima

Architecture and Algorithms for Robust Reconfigurable Neuromorphic Systems

ABSTRACT

NEuromorphic computing is gaining momentum as a revolutionary technology for advancing artificial intelligence (AI). By drawing inspiration from the operations of the human brain, neuromorphic computing aims to create computing systems mirroring the brain's computational structure and architecture. These computing systems hold the promise of processing information with greater efficiency and accuracy than traditional computing systems. In contrast to conventional computing systems that employ artificial neural networks (ANNs), where information flows sequentially through network layers, neuromorphic systems implement spiking neural networks (SNNs). This enables asynchronous and event-driven computation aligning with the temporal dynamics of biological neurons and synapses. Additionally, it is noteworthy that SNNs demonstrate inherent fault-tolerant properties providing neuromorphic systems with a degree of resilience against errors. Nevertheless, as the number of faults increases, their performance begins to degrade. In light of these concerns and to guarantee precise output results, the implementation of fault recovery algorithms becomes imperative.

Furthermore, given the growing adoption of neuromorphic systems in critical applications, resiliency and robustness have emerged as paramount concerns. While prevailing research primarily focuses on enhancing performance, the optimal utilization of neuromorphic systems in the future necessitates incorporating robust computing capabilities for seamless application execution. Effectively executing SNN applications on neuromorphic hardware requires the mapping of neurons and synapses onto the hardware architecture. Mapping to neuromorphic hardware remains a non-trivial task. Furthermore, neural circuits are prone to faults caused by variability in the manufacturing flow, process variations, and manufacturing defects which adds complexity. Strategies employing redundancies and large search spaces for mapping come with substantial resource and computational overheads when dealing with scalable neuromorphic systems. It's also essential to know that redundancy is finite. Consequently, it is imperative to find the most efficient approach for mapping SNN applications onto neuromorphic hardware. Such an approach would have significant implications for application performance and reliability. In this dissertation, we propose architectures and algorithms to facilitate robust neural computations and application mapping onto reconfigurable neuromorphic systems.

First, we conduct experiments and analyses to determine the effects of faults on neural computations. These faults may arise from internal and external interference, noise in synaptic transmission, and fluctuations in post-synaptic potentials within a neural model. After understanding how these faults affect the outcomes of neural computations, we develop algorithms to recover these computations from the impact of these faults. Secondly, to ensure reliable computation outcomes in application execution, we propose a mapping algorithm that explicitly leverages redundancy through a novel selection mechanism for fault tolerance. This mechanism ensures successful mapping even in situations where redundancies are outnumbered. Thirdly, we present "R-MaS3N," an approach for robustly mapping SNN applications onto scalable neuromorphic systems with a novel fault-tolerant mechanism. The novel fault-tolerant mechanism, inspired by the neural reuse theory observed in the human brain significantly reduces mapping computational and resource requirements. Notably, it also addresses the challenge of finite resource availability.

Lastly, we provide a comprehensive evaluation of the proposed fault-tolerant mapping algorithms within the context of scalable 3D NoC-based neuromorphic systems and a hardware complexity analysis of the proposed robust reconfigurable neuromorphic chip.

Thesis advisor: Professor Abderazek Ben Abdallah　　　　　Williams Yohanna Yerima

「ロバスト再構成可能なニューロモーフィックシステムのためのアーキテクチャとアルゴリズム」

要旨

ニューロモーフィック・コンピューティングは、人工知能（AI）を進化させる画期的な技術として勢いを増している。ニューロモーフィック・コンピューティングは、人間の脳の動作からインスピレーションを得ることで、脳の計算構造とアーキテクチャを反映したコンピューティング・システムを構築することを目指す。これらのコンピューティング・システムは、従来のコンピューティング・システムよりも高い効率と精度で情報を処理することが期待されている。人工ニューラルネットワーク（ANN）を採用した従来のコンピューティング・システムでは、情報がネットワーク層を通して順次流れるのに対し、ニューロモーフィック・システムでは、スパイキング・ニューラル・ネットワーク（SNN）が実装されている。これにより、生物学的ニューロンとシナプスの時間的ダイナミクスに沿った、非同期かつイベント駆動型の計算が可能になる。さらに、SNNが固有の故障耐性を発揮することで、ニューロモーフィック・システムにエラーに対するある程度の回復力を与えていることも注目に値する。とはいえ、故障の数が増えるにつれて、その性能は低下し始める。これらの懸念に鑑み、また正確な出力結果を保証するためには、故障回復アルゴリズムの実装が必須となる。

さらに、重要なアプリケーションにおけるニューロモーフィック・システムの採用が増加していることから、回復力と堅牢性が最も重要な懸念事項として浮上している。一般的な研究は主に性能の向上に焦点を当てているが、将来的にニューロモーフィック・システムを最適に活用するためには、シームレスなアプリケーション実行のための堅牢なコンピューティング機能を組み込む必要がある。ニューロモーフィック・ハードウェア上でSNNアプリケーションを効果的に実行するには、

ニューロンとシナプスをハードウェア・アーキテクチャにマッピングする必要がある。ニューロモルフィック・ハードウェアへのマッピングは、依然として非自明な課題である。さらに、神経回路は、製造フローのばらつき、プロセスのばらつき、製造上の欠陥によって欠陥が発生しやすく、これらにより複雑さを増している。スケーラブルなニューロモルフィック・システムを扱う場合、マッピングに冗長性と大きな探索空間を採用する戦略は、かなりのリソースと計算オーバーヘッドを伴う。また、冗長性は有限であることも考慮すべき点だ。したがって、SNNアプリケーションをニューロモーフィック・ハードウェアにマッピングするための最も効率的なアプローチを見つけることが不可欠である。このようなアプローチは、アプリケーションの性能と信頼性に大きな影響を与えるだろう。本論文では、再構成可能なニューロモルフィック・システム上へのロバストなニューラル計算とアプリケーション・マッピングを容易にするアーキテクチャとアルゴリズムを提案する。

まず、神経計算における欠陥の影響を明らかにするための実験と分析を行う。これらの欠陥は、内部干渉や外部干渉、シナプス伝達のノイズ、神経モデル内のシナプス後電位のゆらぎなどから生じる可能性がある。これらの障害が神経計算の結果にどのような影響を与えるかを理解した後、これらの障害の影響からこれらの計算を回復するアルゴリズムを開発する。第二に、アプリケーションの実行において信頼性の高い計算結果を保証するために、故障耐性のための新しい選択メカニズムを通じて冗長性を明示的に活用するマッピングアルゴリズムを提案する。このメカニズムにより、冗長性が劣る状況でもマッピングを成功させることができる。第三に、SNNアプリケーションをスケーラブルなニューロモルフィック・システムに頑健にマッピングするための新しいアプローチである「R-MaS3N」を紹介する。この方法は、人間の脳で観察された神経再利用理論に着想を得ており、マッピングに必要な計算量とリソースを大幅に削減する。特筆すべきは、リソースの有限性という課題にも対処していることである。

最後に、本稿はスケーラブルな3D NoCベースのニューロモルフィック・システムの文脈における、提案する耐障害性マッピング・アルゴリズムの包括的な評価と、提案するロバストなリコンフィギュラブル・ニューロモルフィック・チップのハードウェア複雑性解析を提供する。

1

INTRODUCTION

1.1 Evolution of Neuromorphic Computing systems

In recent years, the rapid advancement of artificial intelligence (AI) driven by artificial neural networks (ANNs) has given rise to unprecedented demands for computing hardware. However, traditional computing architectures based on the von Neumann model struggle to meet these extraordinary demands as they face the limitations of Moore's Law. While not all data-intensive computing applications require deep learning algorithms, we must consider AI applications like the Internet of Things (IoT), bio-medical systems, and autonomous vehicles where computational efficiency is paramount. Recent surveys reveal that the ever-growing need for computing power is outpacing the progress achieved through Moore's Law scaling as depicted in Figure 1.1 and 1.2. This computing power challenge predominantly stems from the separation of data storage and processing in conventional computing systems as described in Figure 1.1. Such architectures force processors to expend a significant portion of their time and energy on data transfers. Given these trends, it's increasingly unlikely that conventional hardware can adequately address these demands over the long term. This is especially evident when observing the growth in the cost of training required for AI models illustrated in Figure 1.2.

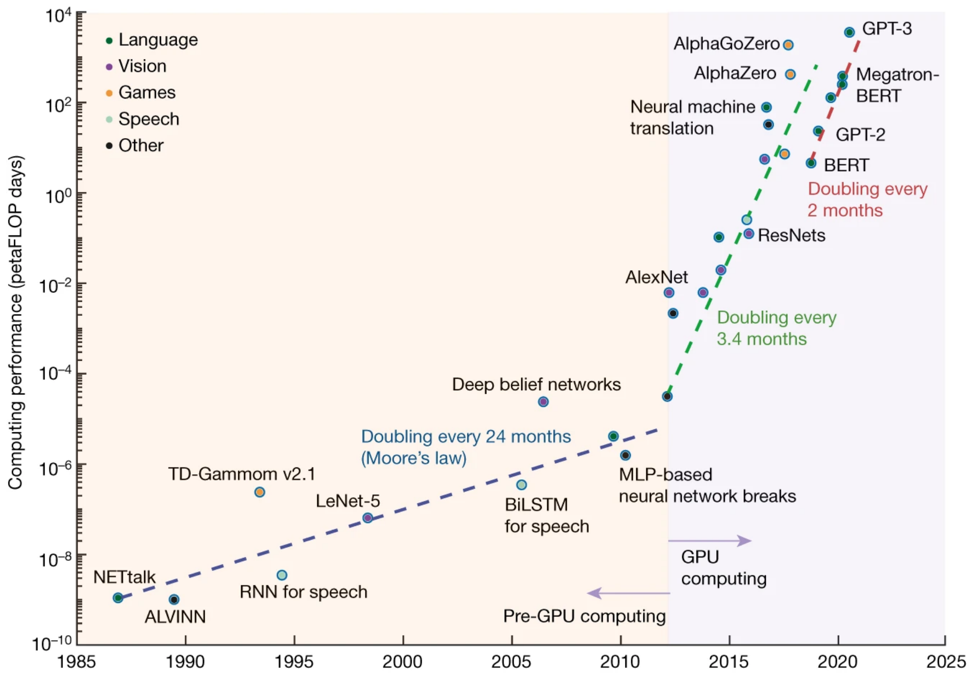


Figure 1.1: Computing power demand of machine learning algorithms on conventional computing architectures over the past 4 decades expressed in PETAFLOPS [1].

However, a hopeful solution lies in adopting principles from biology. Recent explorations within the research community have highlighted a system that not only achieves remarkable energy efficiency but also offers advanced functionality: the human brain [1]. Drawing inspiration from the brain allows approaching information processing differently from the approach in conventional systems. Unlike in conventional systems, the brain co-locates memory and processing within a single compartment, encoding information through signals and harnessing massive parallelism [7]. This arrangement alleviates the constant need to access main memory as seen in conventional systems thereby reducing significant energy consumption [8]. Neuromorphic computing is a rapidly evolving engineering field that emulates the brain's computing

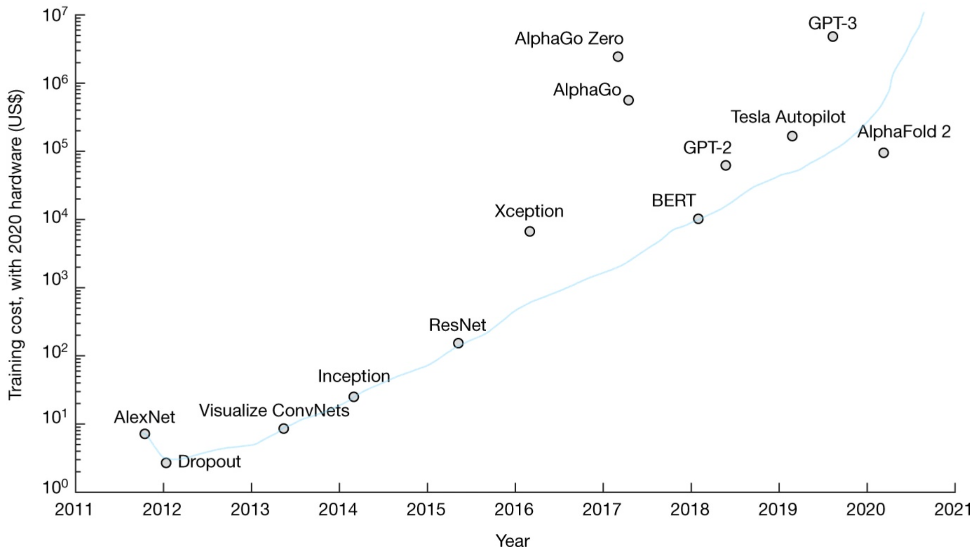


Figure 1.2: Costs associated with training AI models have shown a significant increase since 2011. An exponential rise of this magnitude is unsustainable [1].

functionality to develop neuromorphic systems. Our definition of neuromorphic systems is computers inspired by the brain's neuronal functions and composed of synapses and neurons. While conventional computing systems consist of separate units for processing and memory, in a neuromorphic system, both processing and memory are contained in a single compartment controlled by a neural network as described in Figure 1.3b. Instead of explicit instructions as in conventional computing systems, neuromorphic systems define applications based on the neural network's structure and parameters [8]. The term "neuromorphic" was coined by an American scientist in the late 1980s at the California Institute of Technology and is sometimes referred to as mixed analog and digital implementation [9]. This innovative technology revolutionizes the design and implementation of machine learning algorithms.

Recent strides in neuromorphic computing have led to the development of machine learning algorithms that are not only more efficient but also more powerful than ever before. These algorithms when implemented on neuromorphic systems exhibit the ability to rapidly and accurately process vast amounts of data enabling more complex analysis and fast decision-making. Typically, spiking neural networks (SNNs) are employed on neuromorphic systems despite their poor performance in terms of accuracy compared to ANNs [10]. This is due to their unique biological plausibility [11]. Unlike other types of artificial neural networks such as convolu-

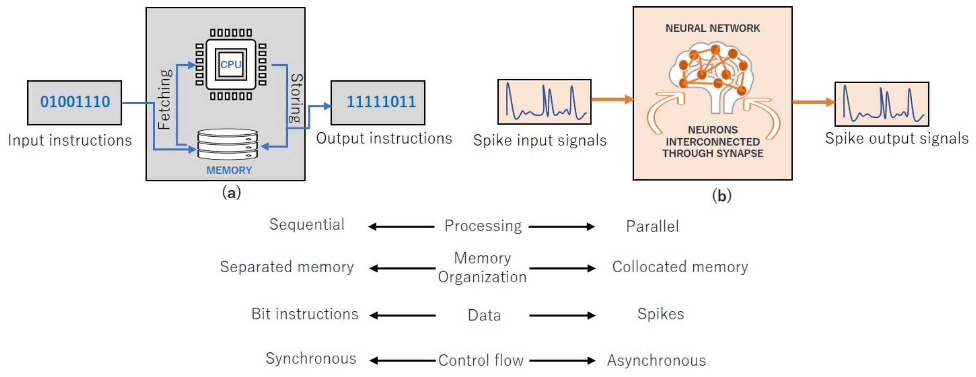


Figure 1.3: Comparison of the von Neumann architecture with the neuromorphic architecture: (a) Von Neumann-based computing systems architecture, (b) Neuromorphic computing systems architecture.

tional neural networks (CNNs) and recurrent neural networks (RNNs), spiking neural networks attempt to replicate key aspects of neural behavior observed in real biological systems such as synaptic plasticity [12], sparse activity [13], temporal dynamics [14], and asynchronous processing [15]. SNNs, unlike traditional artificial neural networks, consider timing as part of their operation [15]. Thus, they can be implemented on neuromorphic hardware to fit the temporal dynamics of spiking synapses and neurons to operate event-driven.

Spiking neuron models such as the integrate-and-fire or Hodgkin-Huxley model describe membrane potential and spike generation differently. In the integrate and fire model (LIF), output spikes are fired when accumulated membrane potentials from other interconnected neurons reach a threshold. A complex and biologically plausible model like the Hodgkin-Huxley neuron model approximates specific aspects of biological neurons like ion channels [16]. The Izhikevich model combines the biological plausibility of Hodgkin-Huxley dynamics with the computational efficiency of the integrate-and-fire neurons. Using the Izhikevich model, thousands of spiking neurons can be simulated in real-time. It's worth noting that the selection of the spiking neural model's implementation plays a significant role in exploring the neuronal capabilities of a neuromorphic system. The Hodgkin-Huxley and Izhikevich models provide the neuromorphic system with a remarkable resemblance to the biological brain in software simulation [16]. However, their hardware implementation is complex due to the numerous parameters inherent in these models. This complexity often poses challenges. Consequently, the Integrate-and-Fire (IF) model emerges as a preferred choice due to its simplified structure.

1.2 Robustness in Neuromorphic Systems: Problems and Motivation

The neuromorphic system excels at performing machine learning and some non-machine learning functions [7]. Beyond Moore's Law, it holds tremendous potential for computing [17]. While neuromorphic systems offer a promising future in computing, robustness and resiliency against faults leading to computational and operational errors are crucial, particularly for critical applications requiring safety, reliability, and dependability [18]. The accumulation of faults resulting from the independent failure or malfunction of a neuron at any point within a large-scale spiking neural network gives rise to reliability concerns [19] regarding computation results. These malfunctions or failures occur due to external interference, random actions by other components (i.e., neurons), and the potential impact of noise on neurons [20].

Figure 1.4 (i) illustrates a post-synaptic neuron N4 receiving spikes from three presynaptic neurons. In the absence of any faulty presynaptic neurons, the output of neuron N4 results in a 1 spike as shown in Figure 1.4 (ii). However, in Figure 1.5 (iii), where neuron N3 is identified as faulty, it is unable to generate an output spike. Consequently, neuron N4 is also incapable of firing an output spike resulting in an output spike value of 0 as shown in Figure 1.5 (iv).

Recent neurobiological and computational studies have demonstrated that neurons can tolerate minimal faults and are resilient to noisy inputs [21]. However, this resilience is not universally applicable to all neural network applications. While SNNs exhibit resilience to faults, recent experiments in [19] have shown that their performance deteriorates with an increasing number of faults. To address these concerns, fault recovery algorithms must be developed to recover the performance of these applications from the effects of an even higher fault rate. These algorithms aim to identify, isolate, and rectify faults within the network thereby ensuring performance and reliability recovery.

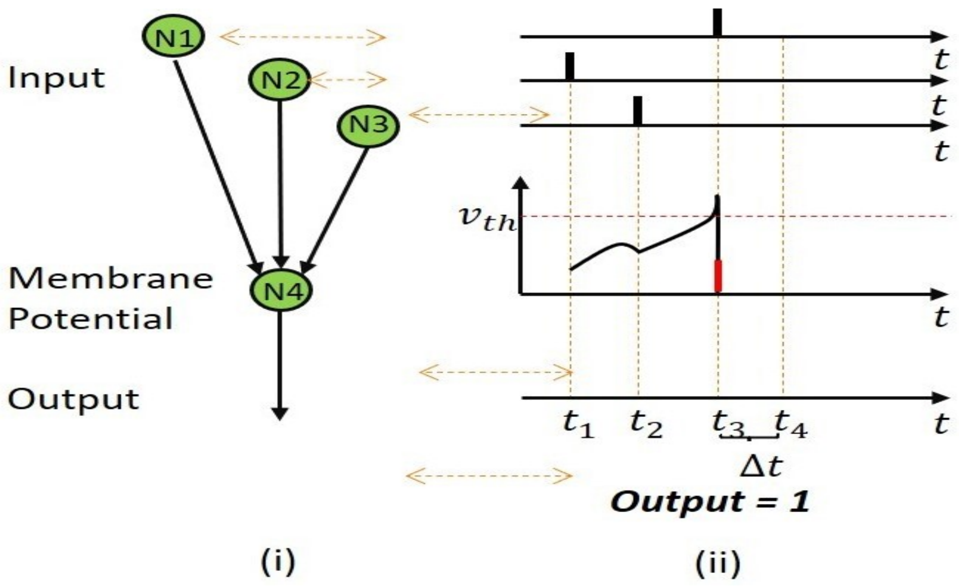


Figure 1.4: Example of neural computation in a simple connected neural network with no faulty neuron.

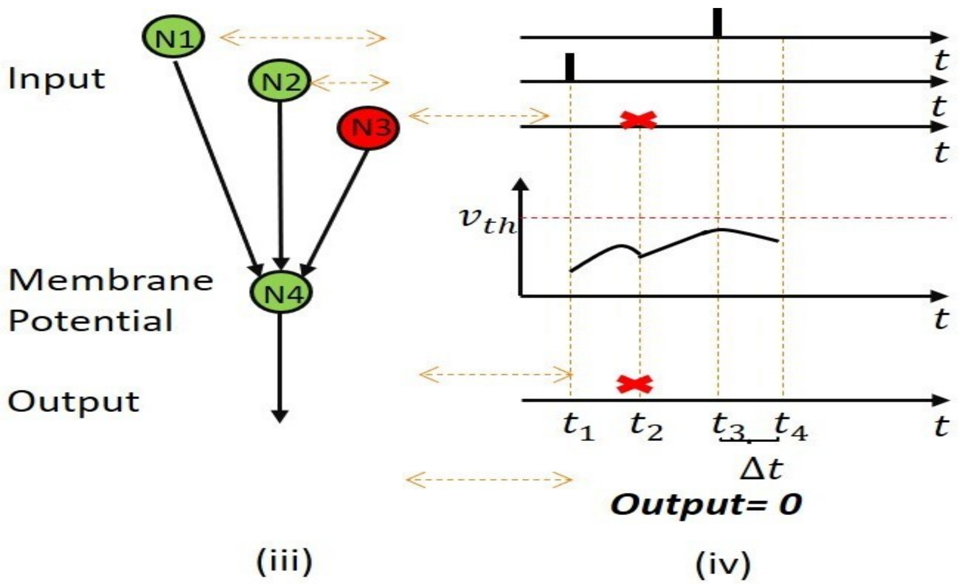


Figure 1.5: Example of neural computation in a simple connected neural network with a faulty neuron.

Furthermore, To execute applications on neuromorphic systems, it is essential to map them to the system. However, reliability issues may arise due to various factors including the inherent complexity of the hardware, the potential for faults within the system, and the challenges of scaling both hardware and application sizes.
To address these reliability issues, fault-tolerant mapping techniques are commonly employed. These techniques include fault detection, redundancy, and dynamic reconfiguration, often using methods such as genetic algorithms, integer linear programming, and min-max optimization [21] [22]. It's important to note that scalability in neuromorphic systems specifically NoC-based (Network-on-Chip) systems comes with its challenges. The extended time required by genetic fault-tolerant approaches to find the optimal solution leads to significantly lengthy repair times. The approach of periodically adding redundancies to integrated chips to achieve fault tolerance is not a sustainable solution in the long term. Moreover, the number of redundant neurons could be exceeded by the number of faulty neurons resulting in the need to drop some neurons. As shown in Figure 1.6, the redundancy element could deplete over time when multiple faults occur during run-time. Another notable challenge illustrated in Figure 1.7 is that as NoC sizes increase, the required redundancies for repairs also increase exponentially.
While techniques such as fault detection, redundancy, and reconfiguration contribute to reliability recovery, their effectiveness may be limited within a short time frame, often accompanied by high computational and resource costs. It is more efficient to re-purpose existing neurons in the neuromorphic system, thus ensuring enhanced reliability at minimal cost and with reduced repair time. To the best of our knowledge, existing fault-tolerant approaches never considered re-purposing some existing neurons for fault tolerance during mapping. It's worth mentioning that this dissertation does not encompass thermal and energy analyses; our primary focus is on ensuring the robustness and reliability of the neuromorphic system.

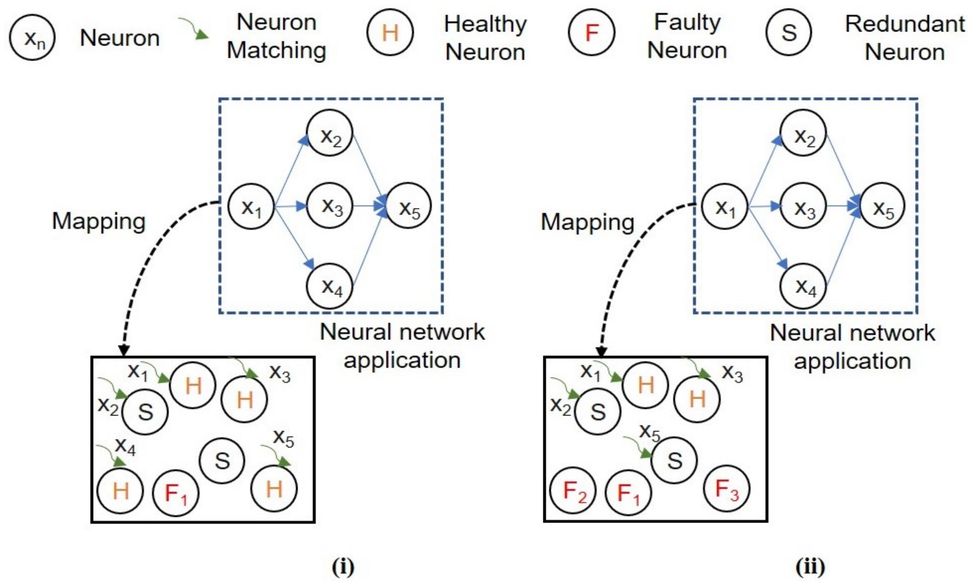


Figure 1.6: Illustrative examples of the mapping of a neural network application to a neural circuit: (i) With sufficient redundancy elements, (ii) With depleted redundancy elements following multiple failures.

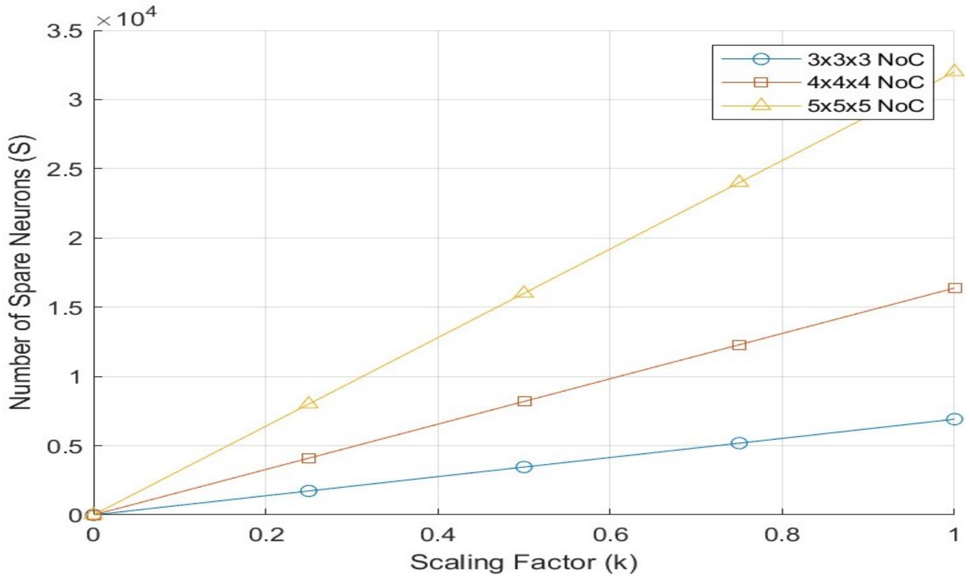


Figure 1.7: Correlation between scaling factor and number of redundant neurons in large-scale NoC-based neuromorphic systems.

1.3 Dissertation Contributions

In this dissertation, we contribute to the field of reliable neuromorphic computing by presenting architectures and algorithms designed to enhance the robustness of neuromorphic systems. Our contribution begins with an in-depth analysis of the impact of faults on neural computation performance. To recover neuromorphic computing applications from the impact of faults, we propose novel fault recovery algorithms. Furthermore, we address the complex task of mapping applications onto neuromorphic systems, a challenge exacerbated by potential faults including high-rate capable of compromising neural computation reliability. Building upon our prior work in migration-based mapping, we introduce an advanced fault-tolerant mapping algorithm and architecture with a novel ranking and selection mechanism. This addition aims to ensure the continued effectiveness of the mapping process even in the presence of high-rate faults. Lastly, we recognize the inherent drawbacks associated with existing fault-tolerant mechanisms particularly concerning resource costs and finite bottleneck issues when striving for a balance between system efficiency and reliability during the mapping of applications onto neuromorphic systems. In response, we present a novel mapping approach featuring a unique fault-tolerance mechanism inspired by the theory of neural reuse. This innovative strategy seeks to overcome the limitations of traditional methods, offering a more sustainable and efficient solution for system mapping within the realm of neuromorphic computing.

The main contributions of this dissertation are as follows:

- A stuck-at augmented pruning (SAP) method for recovery from the impact of fault in SNNs during neural computation.

- A target and selection method (TSM) for improved fault recovery in SNNs during neural computation that identifies and removes faulty neurons during neural computation.

- A ranking and selection mechanism (RSM) for fault-tolerant neuron mapping. The use of this mechanism solves the problem of repairing many defective neurons while having fewer spares.

- A cluster-by-cluster-based ranking and selection method to solve the issue of defective neurons not being selected for fault-tolerant neuron mapping in neuromorphic hardware. In this approach, all clusters with at least one defective neuron are selected for repair.

- A robust mapping scheme (R-MaS3N) featuring an innovative fault-tolerant mechanism designed for mapping SNNs onto 3D-NoC-based neuromorphic systems. The novel fault-tolerant mechanism effectively overcomes limitations in resources and computational costs associated with traditional redundancy-based fault-tolerant methods.
- A heuristic-based method for partitioning neurons in the layer of an SNN application. This approach offers an optimal balance between performance and reliability.
- A partitioning technique that clusters neurons within the layers of the neuromorphic system. This approach ensures that underutilized neurons are first leveraged and prioritized for fault tolerance and neuron utilization recovery before most utilized.

1.4 Dissertation Structure

The rest of this dissertation is organized as follows:

- In Chapter 2, we discuss the fundamental components of a neuromorphic system.
- In Chapter 3, we provide an in-depth exploration of the concept of neuromorphic computing and the progress achieved thus far. We also examine related works dedicated to enhancing robustness in a neuromorphic system focusing on three crucial domains: fault recovery in neural computation, reliable communication, reliable memory operation, and application mapping.
- In Chapter 4 we introduce algorithms to recover SNNs applications performance from faults in their neural computations.
- In Chapter 5, we present a fault-tolerant SNN algorithm tailored for mapping onto 3D NoC-based neuromorphic systems using the proposed RSM.

- In Chapter 6, we also present R-MaS3N, a robust mapping method for mapping SNNs onto 3D NoC-based neuromorphic systems.
- In Chapter 7, we present an implementation of the proposed algorithms and schemes to achieve a robust reconfigurable neuromorphic chip including hardware evaluation and sample layout design.
- Finally, in Chapter 8, we conclude this dissertation with a comprehensive summary of our contributions and a road map for future research.

2 FUNDAMENTALS OF NEUROMORPHIC SYSTEMS

Neuromorphic systems emulate the neural computations observed in biological brains, showcasing impressive learning capabilities and adaptability. Drawing inspiration from the architecture and logical structure of the human brain has paved the way for the development of biological neural networks extensively applied in realms such as machine learning and artificial intelligence. The shift from biological neural network models to neuromorphic computing systems necessitates the design of architectures that mimic their logical structure and computational processes. This chapter discusses biological neural network models, exploring the governing learning rules and information processing coding schemes. It concludes by examining how these computing models with their specific learning rules and coding schemes are used to design and develop neuromorphic systems.

2.1 Spiking Neural Networks

Biological and artificial neural networks (ANNs) are composed of neurons connected by synapses where neurons serve as the fundamental processing units. Regarding their computational paradigm, ANNs can be categorized into three generations [23]. The first generation encompasses McCulloch-Pitts neurons, often referred to as perceptrons. A multilayer perceptron has a single hidden layer and can efficiently compute various boolean functions. In the second generation, neurons incorporate activation functions like sigmoid functions. Consequently, second-generation neural networks have analog inputs and outputs making them more realistic than their first-generation counterparts. Both generational models significantly differ from biological neural networks in terms of functionality.

Brain neurons communicate through discrete pulses known as spikes, typically firing at frequencies lower than 100MHz. This implies that only a window of 20 or 30 ms is required to calculate the current firing rate [24]. Experimental evidence suggests visual processing tasks can be accomplished in as little as 20 ms [25] [26]. As such, it appears unlikely that firing rate serves as the principal coding mechanism for biological neurons instead, timing appears to play a crucial role [27]. These insights led to the development of third-generation neural networks which employ spiking neurons as their basis. Unlike the previous generations, spiking neurons are inspired by the neuron dynamics in artificial neural networks observed in biology [28] however, spiking neurons communicate using spikes and encode information both spatially and temporally which explains the brain's energy efficiency [29].

2.1.1 Learning rules

Learning rules in SNNs are crucial for enabling these networks to perform complex tasks and adapt to changing environments. These rules govern how synaptic weights between neurons are adjusted based on input data and the network's responses. Similar to conventional ANNs, SNNs have three major categories of learning: unsupervised learning, supervised learning, and reinforcement learning.

SUPERVISED LEARNING

SNNs use supervised learning to optimize the network parameters based on the network's current output spike pattern and the desired output spike pattern [16] [30]. Based on the network task, the desired output spike pattern represents a specific value. As an example, in classification tasks, desired output spike patterns encode class labels while in regression tasks, they encode real values [16]. Many supervised learning methods in SNNs have their foundations rooted in the concept of back-propagation which is widely used in traditional ANNs for training. However, there are some differences when applying back-propagation to SNNs due to their spiking nature and temporal dynamics. An early application of spike-based back-propagation in multilayer SNNs is pikeProp [31]. Furthermore, the Tempotron [32] has demonstrated its effectiveness in binary classification tasks drawing parallels with the perceptron concept.

UNSUPERVISED LEARNING

An unsupervised learning approach to SNNs relies on correlations between neural activities to adjust network parameters without relying on class labels [30]. Input spike patterns that are represented using unsupervised learning can be used for clustering and classification tasks. Biology's most fundamental unsupervised learning rule is spike-timing-dependent plasticity (STDP) as Hebb described it, neurons that fire together wire together [2].

The Hebbian learning rule can be expressed mathematically using equations:

$$\Delta w_{ij} = \sum_{k=1}^{N} \begin{cases} A_{+}e^{-\frac{\Delta t}{\tau_{+}}}, & \text{if } \Delta t > 0 \\ A_{-}e^{-\frac{\Delta t}{\tau_{-}}}, & \text{if } \Delta t \leq 0 \end{cases} \tag{2.1}$$

where Δw_{ij} represents the change in synaptic weight between lets say neurons i and j, A_{+} and A_{-} are constants for potentiation and depression, Δt is used to represent the change in time $(t_j - t_k)$, , and τ_{+} and τ_{-} are time constants. As illustrated in Figure 2.1, the STDP process leads to synaptic potentiation (LTP) when the pre-synaptic neuron fires just before the post-synaptic neuron $(t_i < t_j)$, and depression (LTD) when the pre-synaptic neuron fires just after the post-synaptic neuron $(t_i > t_j)$. In unsupervised learning, STDP enables neuromorphic

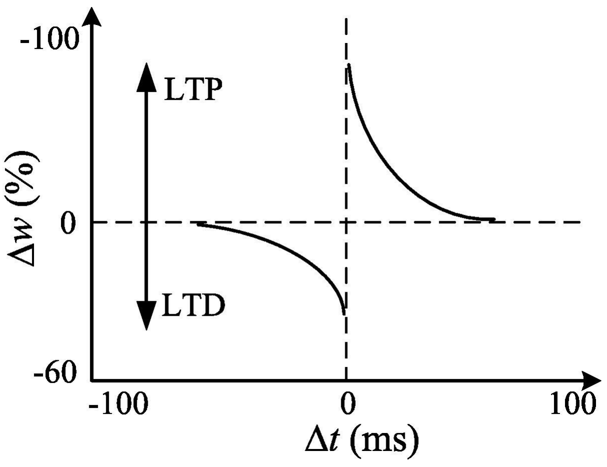


Figure 2.1: STDP illustrating how spike timing influences synaptic weight change. When the presynaptic spike precedes (follows) the postsynaptic spike within a window of milliseconds, it leads to weight increase (decrease), resulting in LTP or LTD, respectively. [2]

systems to adapt synaptic connections based on input spike patterns capturing temporal relationships and efficiently representing data within neural networks. This biologically inspired mechanism plays a crucial role in various neuromorphic computing applications including pattern recognition and sensory data processing.

Reinforcement Learning

Reinforcement learning is another learning paradigm applicable to SNNs. In reinforcement learning (RL), the parameters of an SNN are altered based on external feedback based on the predictions generated by the network [30]. Reward-modulated STDP is one of the well-known approaches used in RL with SNNs to provide an end-to-end learning approach to train two different SNN-based sub-controllers [33]. A reinforcement learning approach for SNNs has been developed recently using policy gradients. To this end, the SNN receives feedback concerning the chosen outputs which then facilitates the adjustment of its parameters [2].

2.2 Spiking Neural Network Models

Figure 2.2 shows the basic structure of a neuron in the human brain. Each neuron has a dendrite as a terminal for incoming signals. Figure 2.3 and 2.4 illustrates two distinct design implementations of a neuron. In the analog design as shown in Figure 2.3, spikes on the 'axon' wire undergo integration via the capacitance of the 'dendrite' wire. The resultant voltage is subsequently compared to a predefined threshold utilizing a comparator, which triggers a spike generation when the voltage surpasses this threshold. Conversely, the digital design described in Figure 2.4 is characterized by a counter. This counter registers increments each time an incoming spike triggers a '1' state within the bit cell.

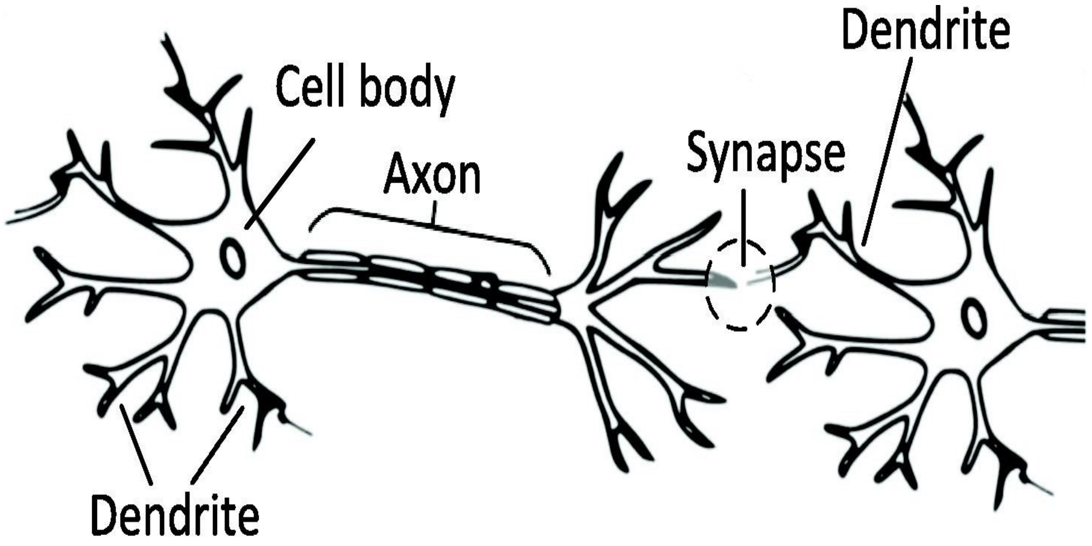


Figure 2.2: Neuron diagram essentially consists of three parts: dendrites, cell bodies, and axons. [2]

To emulate the generation of spikes with different levels of bio-plausibility and computational efficiency, a variety of spiking neuron models have been proposed. The Hodgkin-Huxley model [34] initially proposed by Hodgkin and Huxley mimics the computational and information propagation in axons [35]. However, for applications involving intensive computational processes, the Hodgkin-Huxley model is less suitable for implementation due to the significant computational overhead it incurs. For computational efficiency with low computational overhead, the Leaky Integrate-and-Fire (LIF) model [36] was proposed. However, the neurons in the LIF model do not accurately emulate the biological plausibility of neurons in biology.

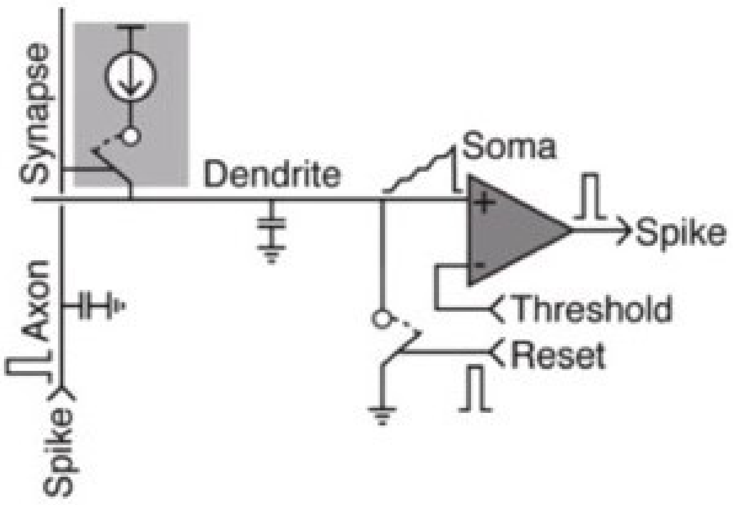


Figure 2.3: A neuron analog implementation as described in [3]

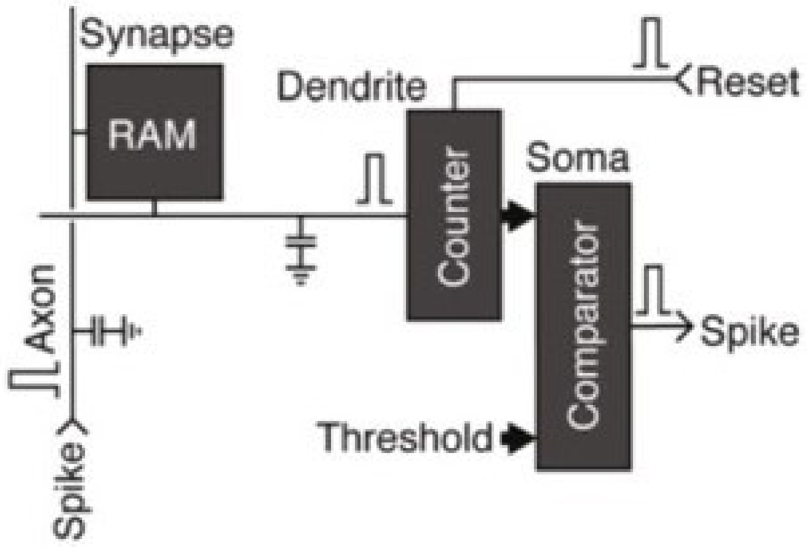


Figure 2.4: A neuron digital implementation as described in [4]

To have a computationally efficient and biologically plausible neuron model, the Izhikevich model [37] was proposed. To better formalize these models, we provide a brief discussion about them.

2.2.1 Hodgkin–Huxley Model

As one of the earliest computational neuron models, the Hodgkin-Huxley model computes the neuronal membrane potential using four differential equations [38]. A neuron's ionic flow dynamics is captured by these four differential equations.

The membrane potential (V) adheres to the equation:

$$C_m \frac{dV}{dt} = I_{\text{injected}} - I_{\text{Na}} - I_{\text{K}} - I_{\text{leak}} \tag{2.2}$$

Here, C_m represents membrane capacitance, $\frac{dV}{dt}$ denotes the rate of potential change, and I_{injected}, I_{Na}, I_{K}, and I_{leak} signify injected current, sodium ion current, potassium ion current, and leak current, respectively.

The sodium ion current (I_{Na}) is formulated as:

$$I_{\text{Na}} = g_{\text{Na}} m^3 h (E_{\text{Na}} - V) \tag{2.3}$$

Similarly, the potassium ion current (I_{K}) is determined by:

$$I_{\text{K}} = g_{\text{K}} n^4 (E_{\text{K}} - V) \tag{2.4}$$

The leak current (I_{leak}), representing passive ion flow, can be expressed as:

$$I_{\text{leak}} = g_{\text{leak}} (E_{\text{leak}} - V) \tag{2.5}$$

These equations collectively form the foundation of the Hodgkin-Huxley model, providing a comprehensive framework to understand and analyze the neuronal membrane's electrical dynamics. Known for its ability to simulate neural activity, the model captures neuronal spikes with the highest accuracy and the parameters correspond to physiological parameters [38]. In spite of this, Hodgkin-Huxley models are however, computationally expensive.

2.2.2 Leaky Integrate and Fire Model

In this model, a neuron's membrane potential is characterized by resistance and capacitance, similar to an RC circuit [39]. The differential equation below describes the membrane potential of an LIF neuron at time t (v(t)):

$$\tau \frac{dv(t)}{dt} = -(v(t) - v_r) + RI(t) \tag{2.6}$$

Here, v(t) denotes the membrane potential of the neuron at time t, and vr represents its resting potential. τ represents the neuron's time constant, and R represents its resistance. I(t) is the input current the neuron receives through the synapses.

2.2.3 Izhikevich Model

The Izhikevich model, a popular spiking neuron model, is described by a pair of differential equations [37]. The membrane potential (v) dynamics are given by:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{2.7}$$

The recovery variable (u) dynamics are described by the equation:

$$\frac{du}{dt} = a(bv - u) \tag{2.8}$$

Here, a and b are parameters that govern the recovery variable's behavior, and I represents the input current. The Izhikevich model is known for its versatility in capturing a variety of spiking patterns in neurons.

Using u as feedback, a neuron is unable to spike again after generating a spike, as u provides negative feedback to v. When a neuron's potential crosses a threshold value (c), it generates a spike. Each spike resets v to resting potential and increments u by d. Parameters a and b

represent the time scale in which u observes variations in v. These equations enable spiking behaviors by cortical neurons to be simulated by varying the values for the parameters a, b, c, and d.

2.3 Coding Schemes

Real-world data and signals are inherently analog and continuous. To enable information processing in SNNs, it is essential to encode these real-world data into spike trains. Yet, an unresolved question pertains to determining the optimal method for encoding this data, whether through rate encoding or temporal encoding. In the following subsections, we will discuss some established neural coding methods in neuroscience.

2.3.1 Rate coding

There are numerous coding schemes used in neural network models with rate coding being the most popular. In this scheme, each input intensity is treated as a firing rate and converted into a Poisson spike train corresponding to its firing rate [40]. The firing rate refers to the spike count over a brief period and such spike counts typically carry limited information. An input neuron fires more frequently if the input intensity is high as shown in Figure 2.5a.

2.3.2 Temporal coding

Unlike rate coding, temporal coding encodes information by analyzing the precise timing of spikes. This approach finds biological support in various types of biological neurons [39]. The first application of this coding scheme in SNN-based supervised learning was demonstrated in [41]. Below, we will discuss some of the most commonly used coding schemes based on temporal spikes.

Time-to-First-Spike

By using first spikes, time-to-first spike encodes information for fast responses within a few milliseconds such as tactile stimuli. As shown in Figure 2.5b, in time to first spike coding, the larger the input pixel intensity, the more information it contains and the sooner it emits a spike [40]. To convert input pixels into first-spike patterns, the authors in [42] proposed an exponential-decaying dynamic threshold. In the computational framework, an exponential function is employed to calculate the threshold P_{th}, denoted as

$$P_{\text{th}}(t) = \theta_0 \exp\left(-\frac{t}{\tau_{\text{th}}}\right), \tag{2.9}$$

where θ_0 is a threshold constant, typically set to 1, and τ_{th} represents the time constant. In this encoding scheme, input pixels are transformed into precise timing information particularly the timing of the first spikes. The exact timing of these input spikes plays a pivotal role during the decoding phase determining the amount of information conveyed to the post-synaptic neurons [40]. The input spikes serve to excite the synapse, generating synaptic input as the sum of post-synaptic potentials (PSPs) according to the equation:

$$z_j(t) = \sum_i \text{PSP}_{ij}(t) = w_s(t) \sum_i w_{ij} s_i(t) \tag{2.10}$$

Here, $z_j(t)$ signifies the synaptic input received by the postsynaptic neuron j, $\text{PSP}_{ij}(t)$ represents the post-synaptic potentials originating from input neuron i, $s_i(t)$ denotes the input spike train emitted by the presynaptic neuron i, and w_{ij} signifies the synaptic weight. Furthermore, $w_s(t)$ corresponds to the spike weight at time t, characterized by an exponential decay function (i.e., $w_s(t) = \exp\left(-\frac{t}{\tau_s}\right)$).

Rank Order Coding

Ranking-order coding encodes information in the order of spikes emitted by neurons in a population. In this scheme, each neuron fires once after a stimulus is presented [39]. This coding scheme has gained attention in neuroscience for its potential to capture fine temporal

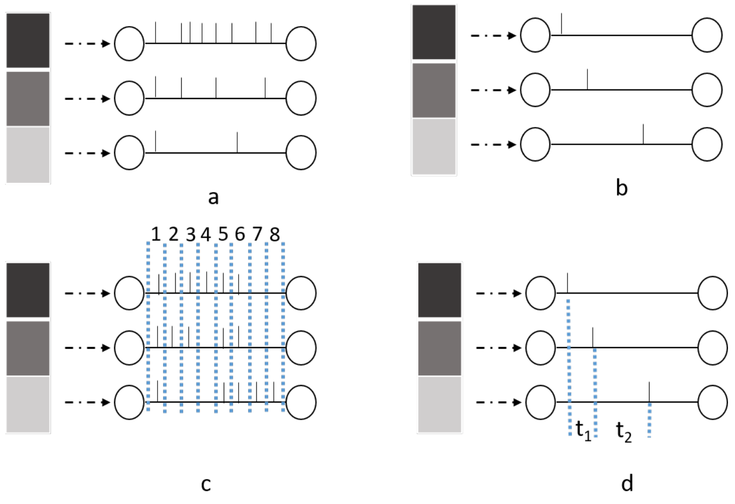


Figure 2.5: Schematic illustration of neural coding: (a) Rate coding: Input intensity is converted to firing rates, (b) Time to first spike coding: Information is encoded based on the precise timing of the first spike generated by a neuron with high-intensity input to a neuron firing first, (c) Phase coding: High- and medium-intensity input neurons spike almost at the same time as neurons with low-intensity input that are not synchronized, (d) Latency coding: Spike timing t_1 and t_2 encode information in neurons with medium and least intensity input relative to neurons with the highest intensity.

details in sensory information. It's particularly relevant in scenarios where the exact timing of events is critical for accurate perception and response. For example, in tasks such as speech recognition or object identification, the precise order of neural responses can convey essential information that might be missed in traditional rate coding schemes.

Phase Coding

In this encoding scheme, neurons spike at different phases about the reference oscillation [39]. Using a binary representation, the authors in [43] came up with a phase coding scheme using the bit "1" which signifies a spike. Each bit in the representation is weighted differently to add phase information to the spikes. Also, the largest pixel intensity determines the number of phases. As shown in Figure 2.5c, the number of phases is 8 and varies periodically with time following the formula:

$$w_s(t) = 2^{-(1+\mathrm{mod}(t-1,8))} \tag{2.11}$$

This expression reflects the relative importance of each bit within the binary representation. Based on Equation 2.11, the synaptic input is produced by decoding the weighted spikes. Typically, larger pixels produce more significant spikes and transmit more information.

Latency Coding

Latency coding represents a fundamental aspect of neural information processing that heavily depends on the precise timing of spikes within neuronal networks as shown in Figure 2.5d. In this coding scheme, the temporal aspect of neural activity takes center stage and plays a pivotal role in determining whether a synapse undergoes Long-Term Potentiation (LTP) or Long-Term Depression (LTD) [39] thus mirroring the concept of STDP.

2.4 Hardware Implementation

In hardware, neuromorphic systems are implemented by developing specialized architectures that closely emulate the learning, memory, and computational processes of biological brains. Our discussion in this section will focus on how neuromorphic hardware systems implement sub-components such as learning, memory, and platform.

2.4.1 Platforms

Neuromorphic computing offers a promising avenue for mimicking and harnessing the capabilities of the human brain in electronic systems. For hardware implementation, two primary platforms are widely adopted: ASICs (Application-Specific Integrated Circuits) and FPGAs (Field-Programmable Gate Arrays). ASIC integrated circuits are known for their efficiency and performance in specialized neuromorphic applications. They offer fixed functionality, making them ideal for power-efficient, real-time, and large-scale neuromorphic simulations. An application that effectively harnesses ASICs is the deployment of SNN for real-time object recognition in autonomous vehicles. The inherent aforementioned characteristics of ASICs are crucial for rapid decision-making in dynamic environments. Neuromorphic devices such as

Intel's Loihi chip [44] and IBM's TrueNorth chip [45] are both renowned for their prowess in efficient and real-time processing utilizing ASIC platforms.
On the other hand, FPGAs are versatile, programmable hardware platforms, widely known for their flexibility and suitability for rapid prototyping for research and experimentation. While FPGAs may have slightly lower power efficiency, they allow adaptation to various neural network architectures and algorithms. For instance, the BrainScaleS [46] system developed by Heidelberg University represents an embodiment of the utility of FPGAs in cutting-edge research. Similarly, the SpiNNaker platform [47] originating from the University of Manchester showcases the power of FPGAs in facilitating large-scale neuromorphic simulations.

2.4.2 Memory Technology

Memory technologies play a pivotal role in the development of neuromorphic systems especially when dealing with SNNs. Many prominent neuromorphic systems have traditionally relied on circuits based on commercial CMOS technology [48]. However, in recent years several efforts have been made to advance neuromorphic systems by harnessing emerging memory and device technologies. These cutting-edge technologies find primary applications in synaptic and learning models within the domain of neuromorphic engineering [48].
Notably, research in nano-scale materials has unveiled the potential for new devices to emulate real synapses particularly their ability to retain state information within artificial neural networks [49]. This breakthrough opens avenues for overcoming traditional CMOS memory limitations. Several emerging memory devices have emerged as promising alternatives. Among these technologies, Resistive Random-Access Memories (R-RAMs) [50] have gained prominence. R-RAMs leverage resistance-switching phenomena providing distinct advantages over memory based on CMOS technology [49] [51]. Typically, R-RAM devices consist of two terminals: a top electrode and a bottom electrode, separated by a thin film. Voltage application between the electrodes enables gradual switching between high-conductive and high-resistive states, allowing the storage of corresponding conductance values.
In addition to R-RAM, other emerging memory technologies such as Spin Transfer Torque Magnetic Random Access Memory (STT-MRAM), ferroelectric devices, and phase-change

materials offer unique methods for storing memory states as resistance, each exhibiting its own set of distinctive behaviors [52]. These innovations mark a significant stride in enhancing the capabilities of neuromorphic systems, enabling them to more closely replicate the intricate processes of biological synapses and fostering advancements in artificial intelligence and cognitive computing.

2.4.3 Learning

Learning in neuromorphic systems can be implemented through two distinct approaches: off-chip learning and on-chip learning. Off-chip learning involves utilizing external computing resources often traditional digital computers to train neural networks for a target application for the neuromorphic hardware. This approach allows for training spiking neural networks with supervised training algorithms like BP-based learning algorithms and their variants on datasets. The resulting neural network configuration and parameters post-training are mapped to the neuromorphic chip for the desired intended task. Another approach to off-chip learning is through ANN-SNN conversion demonstrated in [53], [54], and [55]. This conversion process enables the utilization of pre-trained deep learning models while preserving much of the knowledge learned during the off-chip training.

Conversely, on-chip learning integrates the learning process directly within the neuromorphic hardware utilizing analog and digital circuits to update synaptic weights. Demonstrated in [5], this approach mimics the real-time learning capabilities of biological brains and offers low-latency and energy-efficient learning. One of the prominent on-chip learning algorithms is STDP which is a biologically inspired learning rule that adjusts synaptic weights based on the precise timing of spikes between connected neurons. When a pre-synaptic neuron consistently fires before a post-synaptic neuron, the synapse between them strengthens, and vice versa. This real-time adaptation ability of STDP makes it an ideal choice for on-chip learning because it enables immediate responses to changing data patterns minimizing latency.

Using supervised learning algorithms for on-chip learning can have significant disadvantages. This is because supervised learning algorithms typically involve iterative computations, backpropagating, and error adjustment which can be computationally intensive and energy-consuming. These characteristics make supervised learning algorithms less suitable for on-chip learning as they can introduce high latency, increase power consumption, and limit the real-time adaptability in a neuromorphic system.

2.5 Chapter Summary

In this chapter, we discussed the fundamentals of neuromorphic systems particularly SNNs, their computational models, and coding schemes used to process information. Additionally, we explored three pivotal modules for developing a neuromorphic system in hardware. The next chapter will discuss advancements in neuromorphic systems and the inherent challenges of robustness. Additionally, the chapter will also discuss how previous researchers have addressed the robustness issue within neuromorphic systems.

3

RELATED WORKS

IN addition to their primary functions, neuromorphic systems must consistently perform across real-world scenarios adapting to unpredictable sensor data, sustaining functionality in adverse conditions, and accommodating hardware variations. Achieving robustness is paramount to unlocking the full potential of neuromorphic systems in critical applications like autonomous driving, space exploration, and biomedical. This chapter explores the development and advancement of neuromorphic systems, the challenges to their reliability, and the necessity for robustness with a focus on neural computation, communication, memory operations, and application mapping.

3.1 Neuromorphic systems: State of art and Advancement

Early theoretical work by Frank Rosenblatt between the 1950s and 1960s on the perceptron led to the concept of neural networks. A lack of computing power, however, hindered practical implementation. As computational capabilities advanced between the 1980s and 1990s, neural networks attracted renewed attention. Researchers developed more complex neural network models including backpropagation algorithms for training multi-layer perceptrons. It was these developments that laid the foundation for modern deep learning. Neuromorphic engineering took shape after Carver Mead began developing silicon neurons based on analog VLSI (Large-scale integration) that closely resembled their biological counterparts [4] [56].

Advancements in hardware development such as specialized neuromorphic chips, vision sensors, and learning algorithms have advanced neuromorphic computing into the realm of real-time data processing. To achieve this, the development of spiking neural network models with adaptable features is imperative offering the ability to configure network connectivity, parameters, and even constituent element models such as neurons and synapses.

One pioneering approach in this direction is the SpiNNaker2 project [47] which employs a multicore computer system designed to simulate the behavior of up to a billion neurons in real time. This system integrates 57,600 custom VLSI chips connected through a dedicated global asynchronous communication infrastructure based on the AER communication protocol [57] [58]. This infrastructure is optimized for handling large volumes of small packets, such as those representing neuron spikes, in real-time. Another development is IBM's 'TrueNorth' ASIC [59] which represents a departure from traditional von Neumann architectures. TrueNorth's electronic circuits utilize transistors as digital gates but operate asynchronously and communicate through event-driven methods. This innovative design comprises 4,096 cores of spiking neural networks integrated into a single CMOS chip.

In contrast to SpiNNaker and TrueNorth, the NeuroGrid system [60] follows the original vision of neuromorphic engineering [59] [60]. It leverages analog/digital mixed-signal sub-threshold circuits to model continuous-time neural processing elements. NeuroGrid shares the objective of implementing large-scale neural models and emulating their real-time function distinguishing itself by embracing this unique approach.

The concept of reconfigurability within these systems represents an exciting intersection of neuroscience-inspired computing and adaptability. Designed to replicate the adaptability observed in biological brains, the capability to reconfigure these systems in response to changing environmental conditions endows them with unique attributes notably adaptability and scalability making them versatile for a broad spectrum of applications. In our prior research, we introduce the NASH system [5], a cutting-edge 3D NoC-based neuromorphic system comprising multiple tiles. Each tile features a spiking neuro-processing core (SNPC) housing 256 leaky integrate and fire (LIF) neurons, an SRAM-based synapse memory and 64k crossbar, a network interface (NI), and a 3D fault-tolerant (FT) router with TSV. The NI is equipped with an encoder and decoder responsible for interpreting spike data exchanges between sending and receiving cores. Within the neuromorphic system, the NI earlier supports a layer-to-layer mapping method proposed in previous work [22] and most recently in [6] and [19], migration-based methods. Furthermore, for inter-core communication, the 3D router employs K-means-based multicast routing algorithms as proposed in [22].

3.2 Neuromorphic systems: Robustness concerns

In later years, neuromorphic chips have been used for applications such as real-time object recognition, speech processing, and autonomous robotics. They consume significantly less power than traditional computing systems making them ideal for edge computing and IoT devices [3]. As VLSI [4] [61] technology advances, reliability concerns arise. As a result, finding solutions to enhance the reliability and robustness of these systems is essential. Moreover, neural networks in neuromorphic systems are susceptible to environmental factors such as noise and variability [62]. Also, neural circuits may develop hardware defects due to variations in manufacturing processes and flow [63]. As neuromorphic systems find applications in fields like autonomous vehicles and medical devices, ensuring their reliable and consistent performance, particularly in challenging environments, becomes crucial. Techniques for ensuring and improving the robustness of neuromorphic systems such as fault-tolerant and recovery algorithms, fault-tolerant and recovery mechanisms, and robust training methodologies are actively being researched.

3.3 Existing Solutions To Areas of Neuromorphic Systems For Robustness

Numerous efforts are underway to improve neuromorphic computing systems' robustness, particularly in critical areas such as neural computation, communication, memory operation, and application mapping as the demand for resilient systems grows. In addition to addressing these challenges, existing solutions provide a solid foundation for the evolution of fault-tolerant, recovery, and detection algorithms. The ensuing discussion offers an overview of the advancements in these critical areas aimed at ensuring the robustness of neuromorphic systems.

3.3.1 Fault Recovery In Neural Computation

A neural network often referred to as an ANN is a computational system designed for information processing and computation. However, like any other computational system, neural computations are susceptible to errors. Factors such as process variations, thermal issues, and leakages can lead to computational inaccuracies [64]. Consequently, these inaccuracies may result in output data lacking reliability, accuracy, or data integrity [64]. To tackle this challenge, researchers have put forward several strategies. In the works of [65] and [66], the authors proposed a method to mitigate and recover from these errors by introducing explicit redundancy in both neurons and synapses. This dissertation partially adopts this approach as an initial solution for achieving fault-tolerant neural network execution on neuromorphic systems. Another approach, as presented by [64] involves segregating neural network components based on their functionalities. These components are trained separately each with distinct objectives not only to handle errors but also to enhance overall performance and generalization. Another method for managing errors in neural computation is through network application training and retraining with diversified parameters, as discussed by [67] and [68]. However, it's important to note that this approach can be time-consuming and expensive especially when dealing with hardware implementations

3.3.2 Reliable Communication

Effective communication is a fundamental aspect of computing systems playing a pivotal role in the seamless exchange of data and information between various components and devices. In neuromorphic systems, accurate and reliable data transmission is critical for the proper functioning of the hardware and software layers. However, the reliability of these data transmissions is often challenged by various factors including hardware limitation [69], defective hardware components [69] [70], power surges [70], and in many cases electrical interference leading to memory errors. In this context, data transmission errors can be detected and corrected by Error Correction Codes (ECCs) and Detection Codes (DCs) as reported in [6] and [65]. As a result, data communications within a system can be trusted to be accurate. Despite its benefits, the idea introduces redundancy during data computation. A key characteristic of effective communication is the timely delivery of data. As timing violations introduce latency during communication, they are detrimental to SNNs' performance [5] [22]. As a result of this challenge, the work in [22] suggested implementing alternate links in 3D NoC-based systems to provide continuous spike transfer within cores in the case of a failure of a communication link. Based on ILP and PSO algorithms, the authors in [71] proposed the use of another spare router in the case of router failure. By using this method, extra routers can be replaced with lower communication overheads. While this approach is generally applicable to 2D NoC systems, it does not extend to 3D systems. Therefore, 3D NoC-based systems are not subject to the lower communication costs achieved by 2D NoC-based systems.

3.3.3 Reliable Memory Operation

Memory operations are the backbone of modern computing systems playing a pivotal role in the storage and retrieval of data. The efficiency and reliability of these operations are essential for the overall performance of a computing system. In-memory neuromorphic computations remain susceptible to data retention faults as highlighted by the authors in [67]. To address these faults, efforts have been made to enhance the thermal stability of memory [67]. However, this approach comes at the cost of increased energy consumption for writing data across the entire memory. An alternative strategy involves the use of error correction and detection mechanisms such as Hamming distances, parity bits, and cyclic redundancy which are widely employed [68]. Nevertheless, it's important to note that these techniques introduce a significant overhead in memory cells. Memory operations are also susceptible to soft errors [72] [73] [74] which are often transient and they occur due to fluctuations in expected power optimizations. To address these faults like in high-performance computing (HPC) systems, they typically employ various error correction mechanisms such as parity and checksum bit addition [72]. However, these mechanisms also increase the memory area overhead and energy consumption. In essence, addressing memory faults often involves a trade-off between energy consumption and area overhead.

3.3.4 Reliable Application Mapping

The complex routing architecture inherent in neuromorphic systems adds complexity to the task of mapping applications onto such hardware. Despite these inherent challenges, numerous strategies have been proposed to achieve optimal system performance while also considering the trade-offs between hardware performance and reliability post-mapping. However, the process of mapping applications such as SNNs onto neuromorphic hardware remains a non-trivial endeavor. Generally, the process of mapping an application such as SNNs to neuromorphic hardware involves two key steps: partitioning neurons into clusters based on hardware constraints and subsequently assigning these clusters to specific hardware processors. Several mapping strategies including Espine [75], Neumap [76], SpineMap [77], PACMAN [77], and PSOPART [78] employ variations of these two-step procedures. Within these approaches, heuristic algorithms and particle swarm optimization (PSO) techniques are commonly employed for optimization.

For instance, in PSOPART, neurons find direct placements on hardware cores through PSO, while PACMAN assigns neurons to SpiNNaker cores following a first-come, first-served principle. In Neumap, Espine, and SpineMap, neuron partitioning precedes the mapping process. Nevertheless, despite the effectiveness of these existing methods, they are not without limitations, particularly when facing the challenges of scaling both hardware and application sizes.

These limitations primarily arise from two factors: firstly, the potential for individual neurons within an application to independently fail due to various internal and external factors; and secondly, as application sizes scale up, the number of neurons susceptible to failure correspondingly increases [62]. Consequently, these mapping techniques entail a trade-off between reliability and resilience on one hand and performance metrics such as power consumption, spike latency, on-chip network congestion, and throughput on the other.

While existing mapping strategies for neuromorphic hardware have been explored to some extent, there remains a notable gap in the development of fault-tolerant mapping techniques tailored specifically for such hardware. In the quest to enhance fault tolerance within these mapping strategies, researchers have commonly integrated mechanisms like redundancy, dynamic reconfiguration, and fault detection [79]. These fault-tolerant mechanisms have often been complemented by conventional optimization approaches such as genetic algorithms [64], min-max optimization [64], quadratic programming [77] [80], and integer linear programming [77] to craft resilient mapping techniques. For example, in [81], researchers introduced a fault-tolerant mapping strategy for a memristor-based crossbar leveraging integer linear programming and hierarchical clustering to elevate mapping rates and reduce hardware costs.

Nevertheless, this method comes at the expense of increased computational and execution times and is confined to 2D neuromorphic hardware. Another noteworthy approach, presented in [82], revolves around fault-tolerant mapping achieved through pruning. This technique, while proficient in mapping applications to the hardware, introduces intricacies into the mapping process and the potential for compatibility challenges. Additionally, the authors in [79] proposed a runtime mapping scheme with a lifetime constraint. This scheme dynamically allocates incoming applications to multi-core systems and employs a borrowing strategy to manage many core resources across various scales. However, its primary focus lies in considering the aging components of the hardware, overlooking the intricacies of the neural circuit.

3.4 Chapter Summary

In this chapter, we outline the historical milestones that have shaped the evolution of neuromorphic systems. We discuss the reliability concerns inherent in neuromorphic systems, exploring previous efforts aimed at ensuring their robustness. Building upon this foundational understanding, the subsequent chapters will introduce our contributions to the ongoing efforts dedicated to achieving and ensuring the robustness of neuromorphic systems.

4

FAULT RECOVERY METHODS FOR NEURAL COMPUTATIONS (SAP AND TSM)

RECOVERY from fault in neural computation is crucial, especially in computational neuroscience and the context of neuromorphic systems. While exploring SNNs and simulating their neural processes, we discovered vulnerabilities to faults that can result in errors that must be addressed. To address these challenges, the adoption of fault recovery algorithms becomes imperative. In this chapter, our approach begins with an exploration of fault tolerance in neural networks, delving into the impact of faults on neural network accuracy. Through conducting fault injection experiments on trained SNN models, we measure the impact on classification or prediction accuracy following the introduction of faults. Subsequently, we introduce our proposed algorithms designed to ensure the reliability of output results in neural computations, especially after recovering from faults. These algorithms strategically employ fault detection and recovery strategies to mitigate the potential influence of faults that could compromise the accuracy of neural networks. By exploring fault-tolerant and recovery algorithms in neural computation within neuromorphic systems, our goal is to underscore their pivotal role in achieving dependable results.

4.1 Introduction to Fault Recovery in Neural Networks

Recovery from faults in neural networks is essential, especially in applications requiring reliable computation output such as autonomous vehicles, space exploration systems, and biomedical devices. In the context of neural networks, faults typically refer to the failure of individual neurons or synaptic connections. To address these issues, fault recovery mechanisms and techniques are specifically crafted and implemented. These mechanisms aim not only to recover a network from faults but also to guarantee that the network can persistently function correctly, even in the presence of such failures.

Mathematically, we can illustrate a basic feed-forward neural network with multiple layers having neurons interconnected together as follows:

$$a_i = f\left(\sum_{j=1}^{n} w_{ij} \cdot x_j + b_i\right) \tag{4.1}$$

Where:

a_i represents the output of neuron i.

f denotes the activation function of the neuron.

w_{ij} signifies the weight of the connection between neuron i and neuron j.

x_j is the output of neuron j in the previous layer.

b_i stands for the bias of neuron i.

Moreover, in a feed-forward neural network represented by Equation 4.1, a fault can manifest in several ways. Let's consider a fault in the weight of a connection w_{ij} between two neurons i and j. This fault can be quantified as the difference between the output of neuron i with the original weight w_{ij} and the output with the faulty weight w_{ij}^{fault}. We can represent this fault as follows:

$$\Delta a_i = f\left(\sum_{j=1}^{n} w_{ij} \cdot x_j + b_i\right) - f\left(\sum_{j=1}^{n} w_{ij}^{fault} \cdot x_j + b_i^{fault}\right) \tag{4.2}$$

Where:

Δa_i represents the change in the output of neuron i due to the weight fault.

f is the activation function of the neuron.

w_{ij} is the original weight between neurons i and j.

w_{ij}^{fault} is the faulty weight between neurons i and j.

x_j is the output of neuron j in the previous layer.

b_i is the bias of neuron i.

Fault recovery as defined by Equation 4.3 is a critical aspect aimed at maintaining the network's functionality and ensuring consistent outputs even when faults are present. In the context of a feed-forward network described by Equation 4.1, with a fault characterized by Equation 4.2, the tolerance to such faults is expressed as follows:

$$\|f\left(\sum_{j=1}^{n} w_{ij} \cdot x_j + b_i\right) - f\left(\sum_{j=1}^{n} w_{ij}^{fault} \cdot x_j + b_i^{fault}\right)\| \leq \varepsilon, \forall X \in T \tag{4.3}$$

Here, ε represents a small tolerance value for any given input data X from the training set T.

4.2 Impacts of Faults in Spiking Neural Networks

Neurobiological studies suggest that the human brain can tolerate only a limited number of errors during computation [21]. Computational studies indicate that neural networks exhibit resilience to noisy inputs further highlighting their ability to gracefully degrade when implemented at the physical level. However, it's essential to note that without proper design considerations, neural networks have minimal tolerance for intrinsic faults [21] [83] [84]. Recent experiments have demonstrated their vulnerability to hardware fault injections particularly following training [85]. To propose a fault recovery method, a comprehensive understanding of the impact of these faults on SNNs is imperative. We proposed to randomly introduce hardware-level faults into post-trained SNN models which include Multi-Layer Perceptron (MLP) and Convolutional Spiking Neural Network (CSNN) models and observe the performance of the faulty models in classification tasks like digit recognition. The architectures of these models are described in Figures 4.1 and 4.2. Our design for these SNN architectures draws inspiration from the work in [86].

4.2.1 Fault Modelling

In the context of this research, a foundational assumption has been made regarding the output layer of a system or application, asserting that faults cannot occur at this layer. This assumption is grounded in the understanding that if a fault were to manifest at the output layer, it would render the entire output non-existent, thereby diminishing the predictability and utility of the system. However, it is acknowledged that faults are not restricted to specific layers; they can potentially occur at any location within the system. In our fault modelling, in particular, we focus on 'stuck-at' (SA) faults, which are a common occurrence following the fabrication of circuits [87]. Our assumption concerning the 'stuck-at' fault is predicated on two scenarios: when it arises, either a neuron becomes incapable of generating the requisite output essential for computation (SA-0), or it persistently produces an output even in the absence of an input responsible for triggering the output (SA-1), effectively becoming 'stuck' in a particular state. This assumption harmonizes seamlessly with the nature of our system, which consists of neurons organized into clusters responsible for accumulation computation. In this architecture, each neuron plays a role in contributing to the final output. Therefore, the SA fault manifests as a disruption in the coordinated functioning of these neurons, potentially impacting the overall computation process. In our SA-0 fault models, we model to simulate a situation where a neuron's output becomes zero when the weight associated with that neuron is closest to a specified minimum weight threshold. For the SA-1 fault model, we emulated a condition where a neuron output becomes one continuously when the weight associated with the neuron approaches or reaches the maximum allowable value.

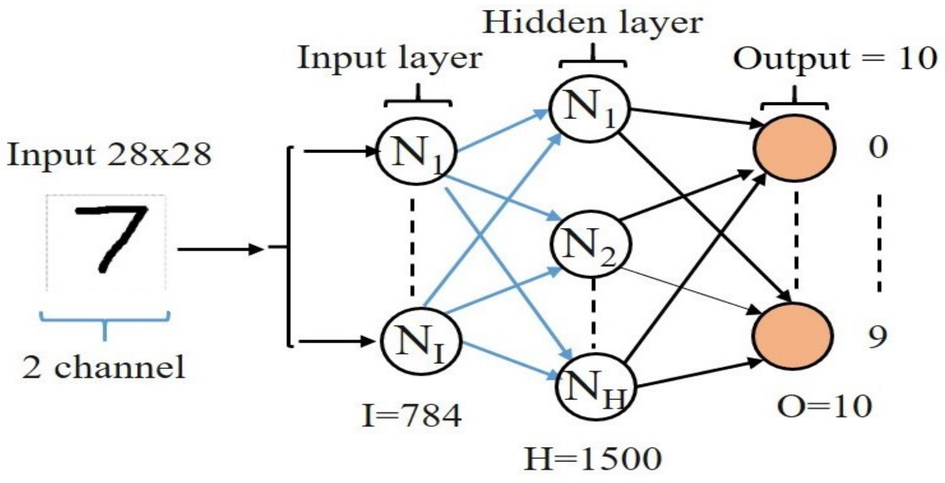


Figure 4.1: MLP-MNIST Architecture.

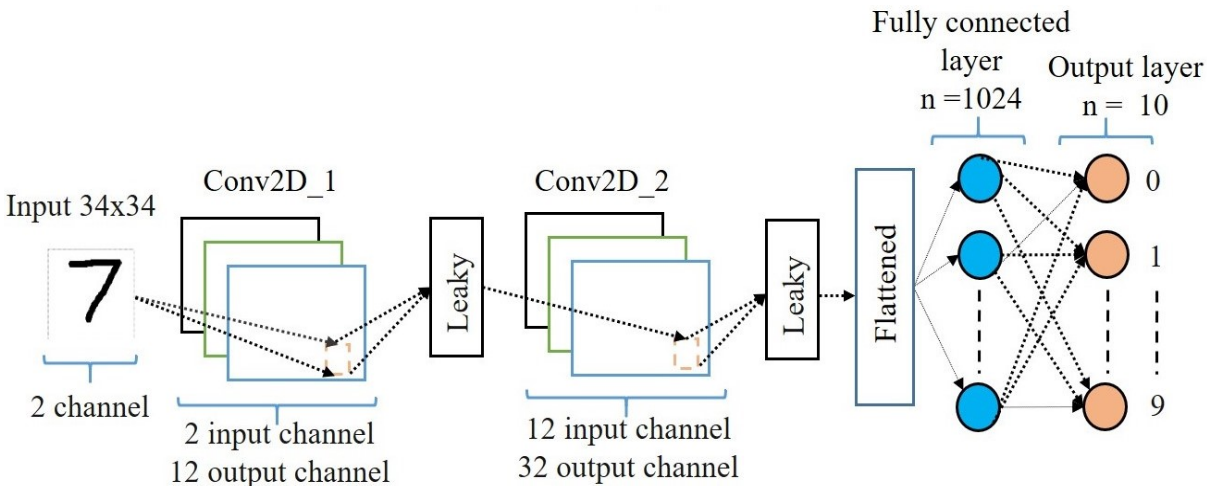


Figure 4.2: CSNN-NMNIST Architecture.

4.3 The Proposed Fault Recovery Algorithms

4.3.1 Stuck-at Augmented Pruning Algorithm (SAP)

To recover an SNN application from the impact of faults, in this research, we considered removing faulty neurons in the faulty application. However, authors in [88] suggest retraining the neural network to consider the positions of the defects when dealing with stuck-at-fault in neural networks. According to [89], the authors suggest rearranging the crossbar rows and columns but system complexity may arise as a result of these methods. Pruning is a technique widely used to compress, simplify, and speed up the inference of neural networks [88] [90] through the removal of unwanted connections or neurons. Given the degradation in neural network performance observed in the aftermath of fault injection experiments (refer to Figures 4.3 and 4.4 in section 4.4.4), we introduced SAP, an augmented pruning fault recovery strategy. It is important to note that this strategy differs from conventional pruning methodologies by incorporating additional criteria to guide the identification of faulty neurons targeted for removal. In the SAP approach, we suggest a paradigm shift from the conventional practice of solely removing synaptic connections, proposing instead the elimination of entire neurons. We introduced the term 'lopping' to describe this process which specifically entails removing neurons based on their normalized value as defined by Equation 4.4. The decision criteria for 'lopping' involve whether the normalized value surpasses a minimum weight threshold (K_{mn}) for stuck-at-0 or equals a maximum weight threshold (K_{mx}) for stuck-at-1, as dictated by Equation 4.5. Using neuron synapse weight values, SAP calculates each neuron's maximum and minimum weight threshold. Algorithm 4.3.1 formalizes the SAP method.

$$||X_w|| = \sum_{i=0}^{i=n} |X_i| \tag{4.4}$$

$$K_{mn} = floor(X_w * p), K_{mx} = ceil(X_w * p) \tag{4.5}$$

where X is a neuron, $|X_i|$ is a single synaptic connection value of the neuron, and $||X_w||$ is the normalized value for all its synaptic connections of the neuron.

Algorithm 4.3.1 Stuck-at Augmented Pruning (SAP) Algorithm

```
1:  Input
2:      $S_N$  Pre-trained weights
3:      $P$  Lopping percentage
4:  Output
5:      $\hat{X}$  Matrix graph of neurons
6:  for $n \leftarrow 1$ to $n-1$ layers in $S_N$ model do
7:      // Add stuck-at-fault;
8:      for neuron $i = 1, 2, \ldots$ do
9:          // Normalize ($N_{wr}$) $i$ using Eqn. 4.1;
10:         // Determine $K_{mn}$ and $K_{mx}$ using Eqn. 4.2;
11:         if $N_{wr} = K_{mn}$ or $N_{wr} \geq K_{mx}$ then
12:             // Lop $i$ with $P$ percentage;
13:         end if
14:     end for
15: end for
16: $\hat{X}$ = Lopped model($i$, $n$)
17: Return: Lopped model
```

Weak Point

To determine whether a neuron is faulty, SAP calculates a weight threshold for each neuron. Furthermore, the method removes faulty neurons by percentage. Furthermore, application models experience a significant decline in performance when a larger number of neurons presumed to be faulty are removed as evidenced by the outcomes presented in Figures 4.3 and 4.4 of Section 4.4.4. Additionally, the computational step involved in determining the weight threshold introduces added complexity to the overall strategy.

4.3.2 Target and Selection Method (TSM)

In light of the substantial drawbacks associated with SAP, we propose an alternative approach known as the target selection method (TSM). TSM eliminates all faulty neurons simultaneously guided by predetermined minimum and maximum threshold values. This stands in contrast to SAP which removes neurons based on a dynamic weight threshold and percentage criterion that can change over time. An additional advantage of TSM is its capability to accurately detect faults in SA fault models involving non-positive weighted neurons. In contrast, SAP is unable to identify faults in such neurons. We describe the TSM fault recovery algorithm in detail in Algorithm 4.3.2. Notably, TSM diverges from SAP by using the output state or weight rather than individual synaptic weights to identify faulty neurons.

Algorithm 4.3.2 Target and Selection Method (TSM) Algorithm

```
    $S_k$ : Pre-trained weights
 2: $W_{mn}$ : Minimum set weight
    $W_{mx}$ : Maximum set weight
 4: $\hat{X}$ : Matrix graph of neurons
    for $k \leftarrow 1$ to $k-1$ layers in $S_k$ model do
 6: Identification phase:
        for neuron $a = 1, 2, ....$ do
 8:         // Retrieve the weight of each neuron ($W_a$);
            if $W_a = 0$ or $W_a > W_{mx}$ then
10:             // Mark the neuron as faulty ($a_f$);
            end if
12: Removal phase:
            for neuron $a_f = 1, 2, ....$ do
14:             // $W_a \leftarrow 0$ of $a_f$
            end for
16:     end for
        $\hat{X}$ = Fault-free model ($a$, $k$)
18:     Return: Fault-free model
        Return: Faulty neurons and their indexes
```

4.4 Evaluation

This section presents the results of our experiment on fault recovery in SNN applications using the proposed SAP and TSM methods. In our evaluation, we follow a three-stage approach namely; model training, fault introduction and recovery, and, model evaluation.

4.4.1 Evaluation Methodology

First, we trained the proposed SNN architectures described in Figures 4.1 and 4.2 using the snntorch [86] platform with MNIST and NMNIST datasets. The MLP model was trained with the MNIST dataset and is identified as MLP-MNIST, while the CSNN model was trained with the NMNIST dataset and is identified as CSNN-NMNIST. We evaluate the trained model to determine its baseline accuracy. We then randomly inserted SA-1 and SA-0 faults on the post-trained models at 20%, 30%, and 40% fault rates.
First, we assessed the impact of faults in MLP-MNIST and CSNN-NMNIST SA fault models. Following this, we applied the SAP method and evaluated its performance on first, the MLP-MNIST. Subsequently, we implemented the TSM method for fault recovery in the MLP-MNIST and CSNN-NMNIST SA fault models and examined its effectiveness. When evaluating the TSM method, we used the same fault models for evaluating SAP as we did in [62]. In this way, the results of the SAP evaluation can be compared with the results of the TSM evaluation.

4.4.2 Evaluation Results

4.4.3 Impacts of Faults in Spiking Neural Networks

Figures 4.3 and 4.4 illustrate the impact of SA-0 and SA-1 faults on the classification accuracy of our proposed MLP and CSNN models respectively. In the case of MLP-MNIST, as shown in Figure 4.3, the baseline accuracy without any faults is 95.10%. When faults are introduced at rates of 20%, 30%, and 40%, the model's accuracy experiences only a slight drop. This can be attributed in part to the inherent fault resilience found in SNNs. However, when SA-1 faults are introduced, the accuracy decreases significantly with an increasing fault rate. Conversely, for SA-0 faults, the accuracy drop is not as pronounced compared to the baseline accuracy.

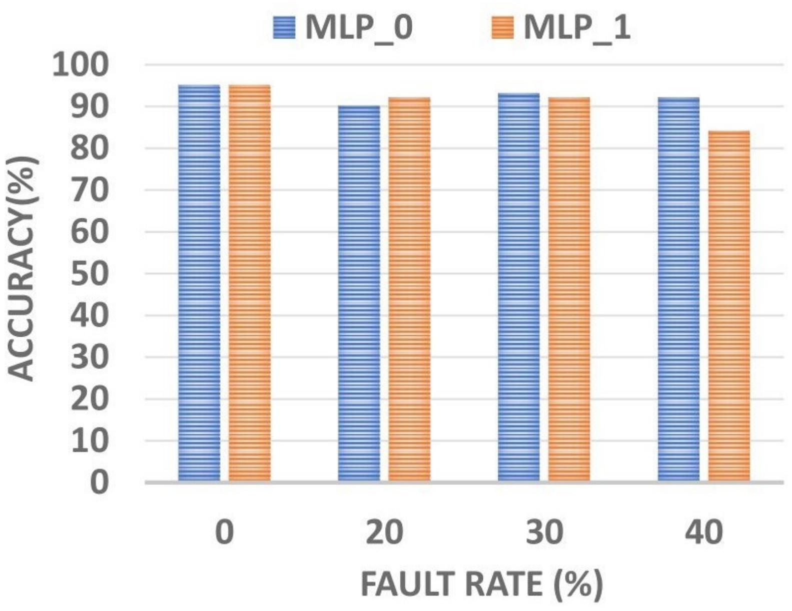


Figure 4.3: The impact of SA faults on accuracy in MLP for MNIST dataset.

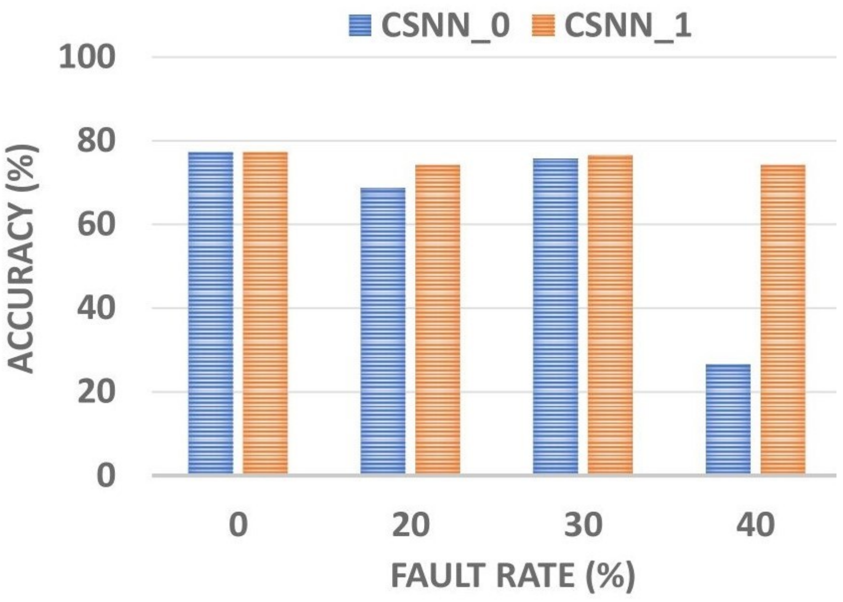


Figure 4.4: The impact of SA faults on accuracy in CSNN for NMNIST dataset.

On the other hand, for the CSNN-NMNIST as shown in Figure 4.4, the baseline accuracy with no faults is 82.03%. However, the introduction of an SA-0 fault results in a different behavior compared to MLP-MNIST. In this case, SA-0 faults have a critical impact while SA-1 faults exhibit behavior similar to SA-0 faults in MLP-MNIST. When the fault rate reaches 40%, the accuracy of CSNN-NMNIST drops to less than 30%. Therefore, it becomes imperative to target and select the faulty neurons to recover the classification accuracy of CSNN-NMNIST.

4.4.4 Stuck-at Pruning (SAP) Algorithm

Our experimental results aimed at recovering SNN using the SAP method in the presence of stuck-at faults are presented in Figures 4.5 and 4.6. The original model's accuracy when no faults are introduced is 95.31%. However, when SA-1 faults are introduced at rates of 20%, 30%, and 40%, as shown in Figure 4.5, the accuracy experiences reductions to 92.19% (with 20% and 30% faults) and 84% respectively. Similarly, for SA-0 faults, the SNN accuracy drops to 90.7%, 93.75%, and 92.19% as described in Figure 4.6. By applying the SAP technique and removing 80% of the faulty neurons, As shown in Figure 4.5, the accuracy of the SNN recovers to 93.75% for all three fault rates. This demonstrates that by selectively targeting and eliminating the faulty neurons, it is possible to recover the accuracy loss caused by SA faults up to 1.56% (with 20% and 30% faults) and 9.75% (with 40% fault) when using an 80% lopping rate. In comparison to the pre-fault state, all three fault rates show a 1.56% loss in accuracy.

A similar trend is observed for the SA-0 fault, as shown in Figure 4.6. The SNN accuracy decreases to 90.8%, 93.75%, and 92.19% at 20%, 30%, and 40% fault rates respectively. Utilizing the SAP method and removing 80% of the faulty neurons, the accuracy recovers to 93.75% (for 20% fault) and remains at 93.75% (for 30% fault) and 92.19% (for 40% fault). This reveals that by specifically targeting and removing the faulty neurons, the accuracy loss due to SA faults can be recuperated by up to 2.95% (with 20% fault) and 1.56% (with 40% fault) with an 80% lopping rate. In comparison to the pre-fault state, the accuracy loss for the SNN with 20% and 30% faults is 1.56% while it increases to 3.12% when the fault rate is 40%. Therefore, the SAP method adeptly restores the performance of an SNN application affected

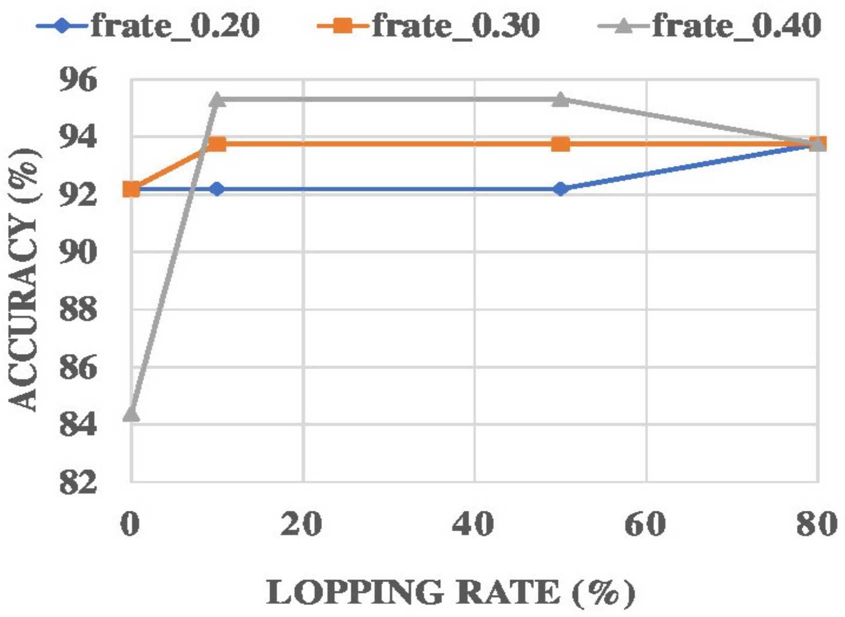


Figure 4.5: Effect of the SAP method on the accuracy of MLP-MNIST with stuck-at-1 fault.

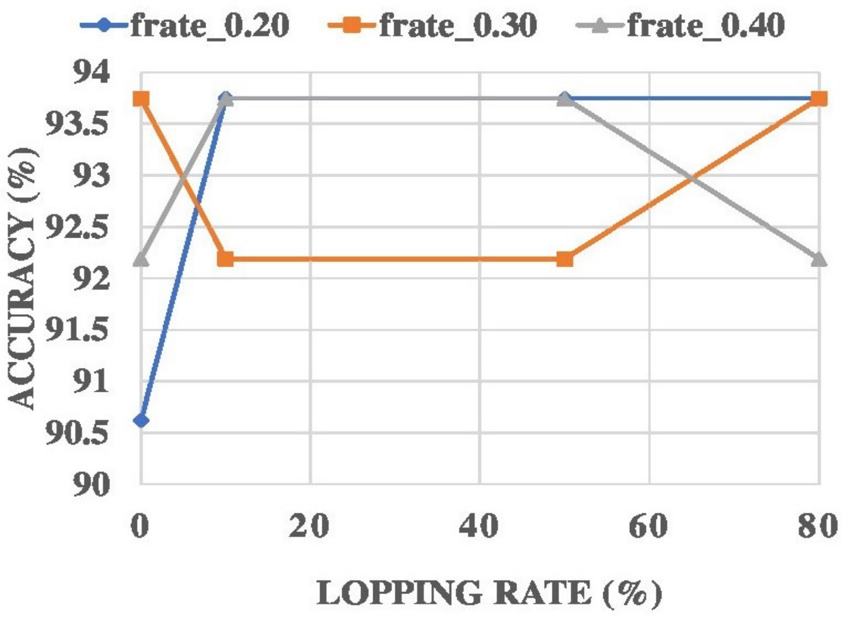


Figure 4.6: Effect of the SAP method on the accuracy of MLP-MNIST with stuck-at-0 fault.

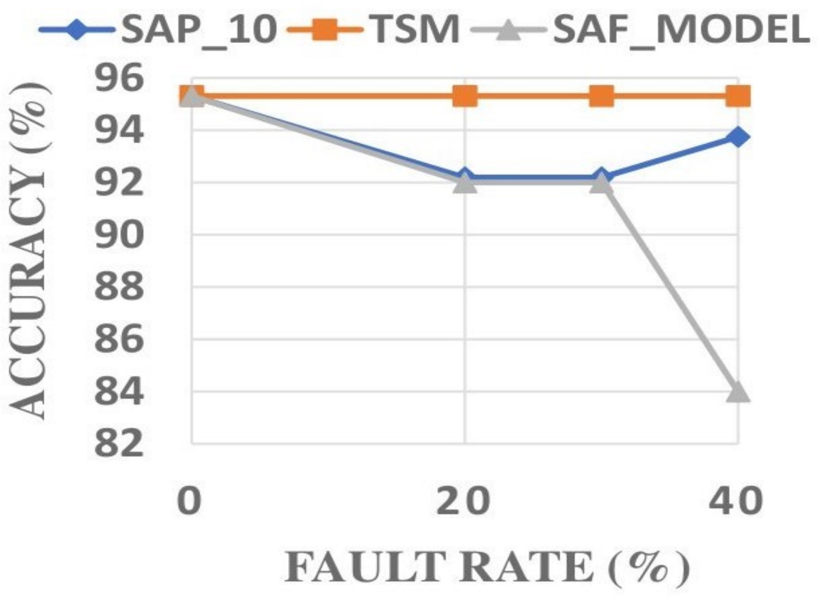


Figure 4.7: Performance comparison of SAP (with 10% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for MLP-MNIST.

by SA faults even in instances involving higher fault rates.

4.4.5 Target and Selection Method

MLP-MNIST

Figures 4.7 - 4.12 provide a comparative analysis of fault recovery methods (SAP and TSM) for an SA fault in the MLP-MNIST model. The baseline accuracy for the fault-free MLP-MNIST (0% fault rate) stands at 95.10%. When an SA-1 fault is introduced at a 40% fault rate, as shown in Figures 4.7 - 4.9, the model's classification accuracy drops to 84%. Employing the SAP method for fault recovery, with the removal of 10% of faulty neurons (SAP_{10}) as illustrated in Figure 4.7, the classification accuracy is restored to 93% at the 40% fault rate. Furthermore, even with 50% and 80% of faulty neurons removed (SAP_{50} and SAP_{80}) as shown in Figures 4.8 and 4.9, the classification accuracy remains stable at 93% resulting in a loss of only 2.10%. In contrast, utilizing TSM enhances classification accuracy recovery to 95% regardless of the fault rate. This represents a marginal 0.10% accuracy loss when compared to the baseline accuracy.

Similarly, in the case of the SA-0 fault in the MLP-MNIST model, the classification ac-

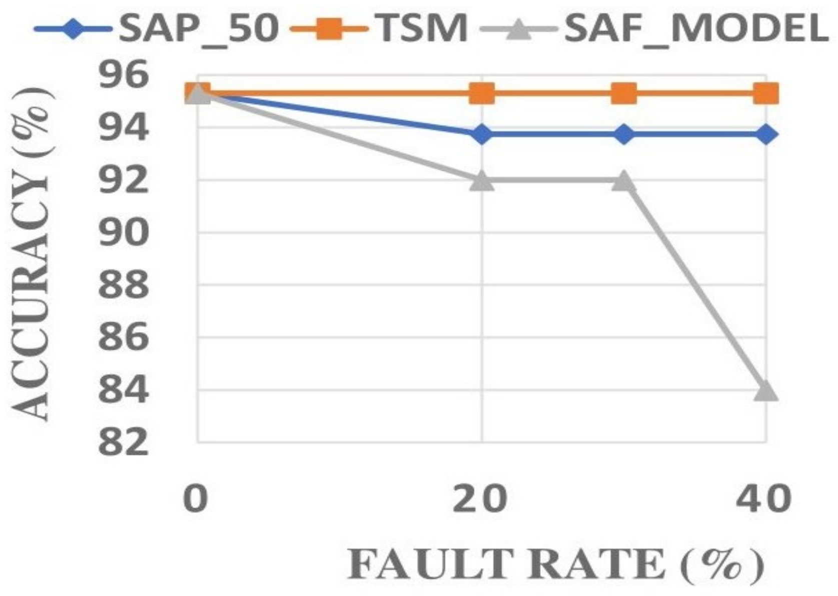


Figure 4.8: Performance comparison of SAP (with 50% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for MLP-MNIST.

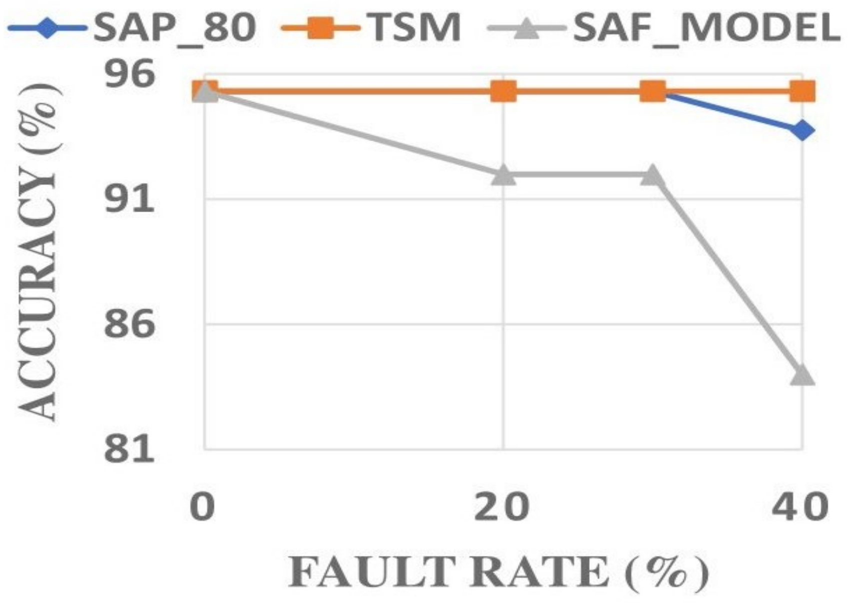


Figure 4.9: Performance comparison of SAP (with 80% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for MLP-MNIST.

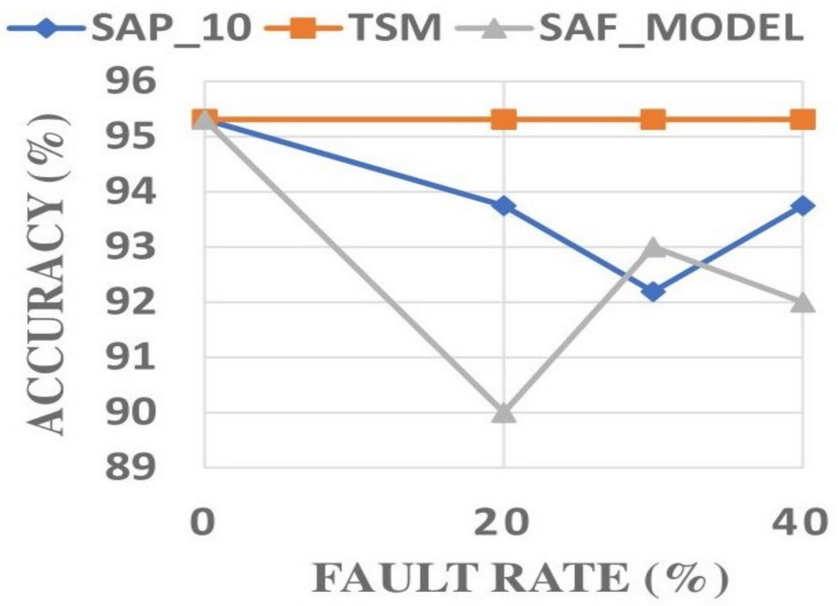


Figure 4.10: Performance comparison of SAP (with 10% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for MLP-MNIST.

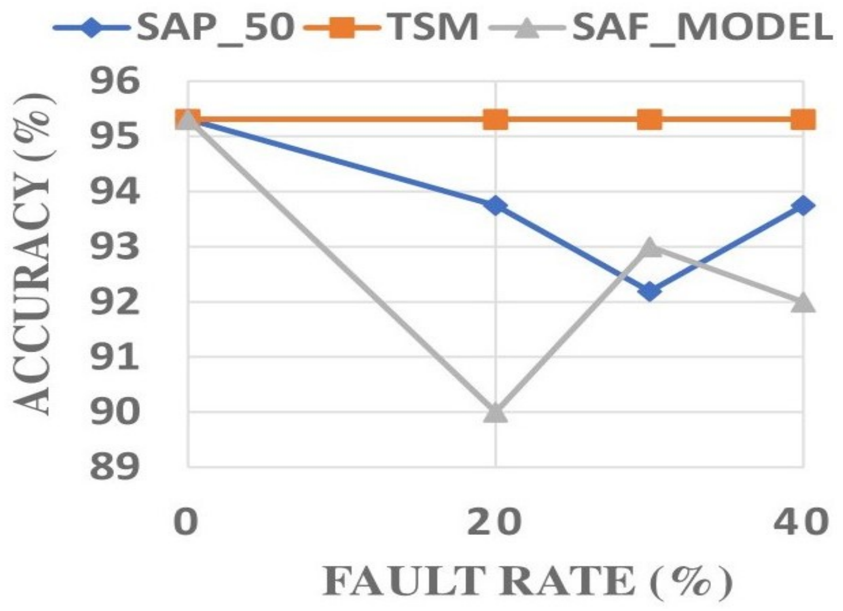


Figure 4.11: Performance comparison of SAP (with 50% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for MLP-MNIST.

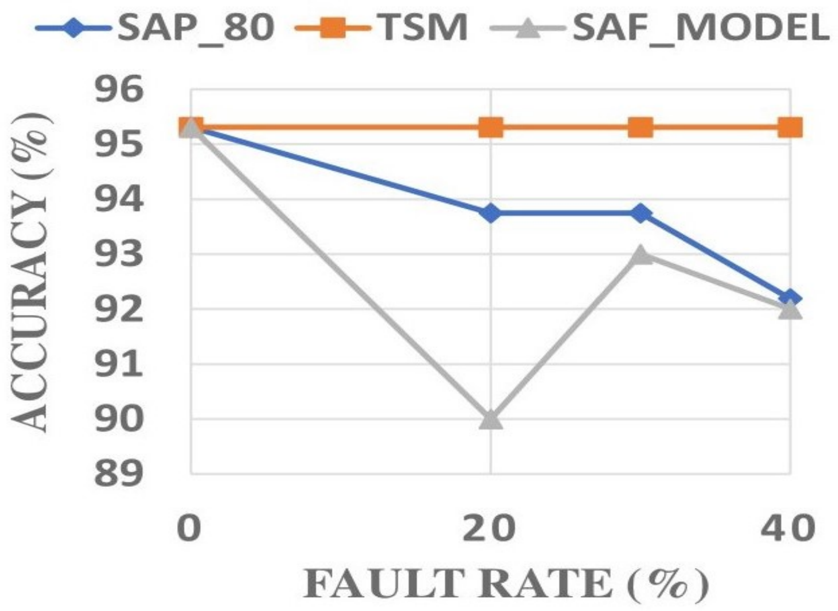


Figure 4.12: Performance comparison of SAP (with 80% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for MLP-MNIST.

curacy drops from 95.10% to 92% at a 40% fault rate. Leveraging the SAP method for fault recovery, with the removal of 10% (SAP_{10}) and 50% (SAP_{50}) of faulty neurons as shown in Figures 4.10 and 4.11, the accuracy is restored to 93% at the 40% fault rate. However, when 80% of faulty neurons are removed (SAP_{80}) as shown in Figure 4.12, the accuracy regresses and stabilizes at 92% under the same 40% fault rate. This results in no net increase in classification accuracy recovery. In contrast, employing TSM leads to an improvement in accuracy to 95%, albeit with a marginal 0.10% accuracy loss compared to the baseline accuracy.

CSNN-NMNIST

Figures 4.13 - 4.18 provide a comparison of the fault recovery performance between the SAP and TSM methods on an SA fault CSNN-NMNIST model. The CSNN-NMNIST model initially exhibits a baseline accuracy of 82.03%. In the case of SA-1 fault as shown in Figures 4.13-4.15, the classification accuracy slightly drops to 79% when subjected to a 40% fault rate. Employing the SAP method for fault recovery specifically by removing 10% of faulty neurons (SAP_{10}), as shown in Figure 4.13 results in an identical accuracy to the one observed under SA-1 fault condition. Furthermore, even with the removal of 50% and 80% of faulty neurons (SAP_{50} and SAP_{80}) as shown in Figures 4.14 and 4.15, the classification accuracy still remains 79%. More so, when using TSM for recovery, the accuracy is 79.10% on average irrespective of the fault rate. In this scenario, both the SAP and TSM methods perform comparably with each demonstrating a 3.03% accuracy loss compared to the baseline accuracy.

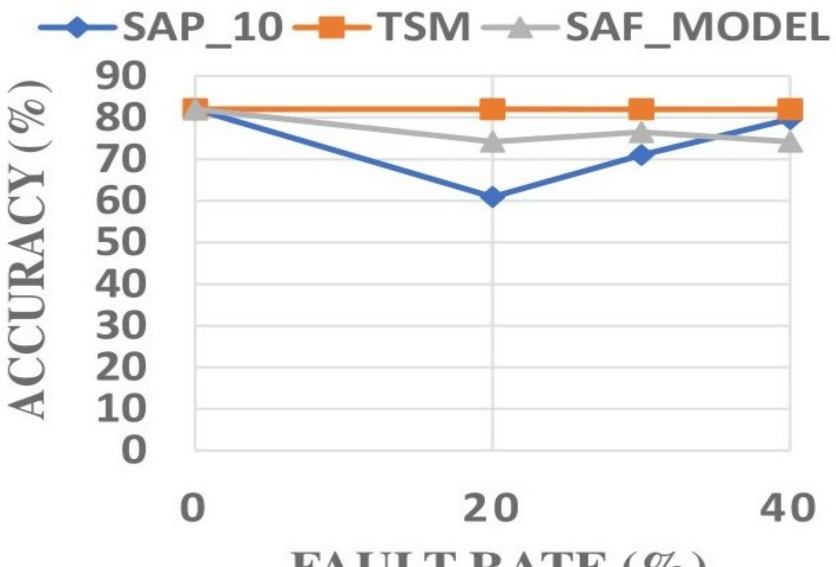


Figure 4.13: Performance comparison of SAP (with 10% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for CSNN-NMNIST.

Similarly, for SA-0 fault within the CSNN-NMNIST model, the SA fault model exhibits a classification accuracy of 26% at a 40% fault rate. Applying the SAP method for fault recovery

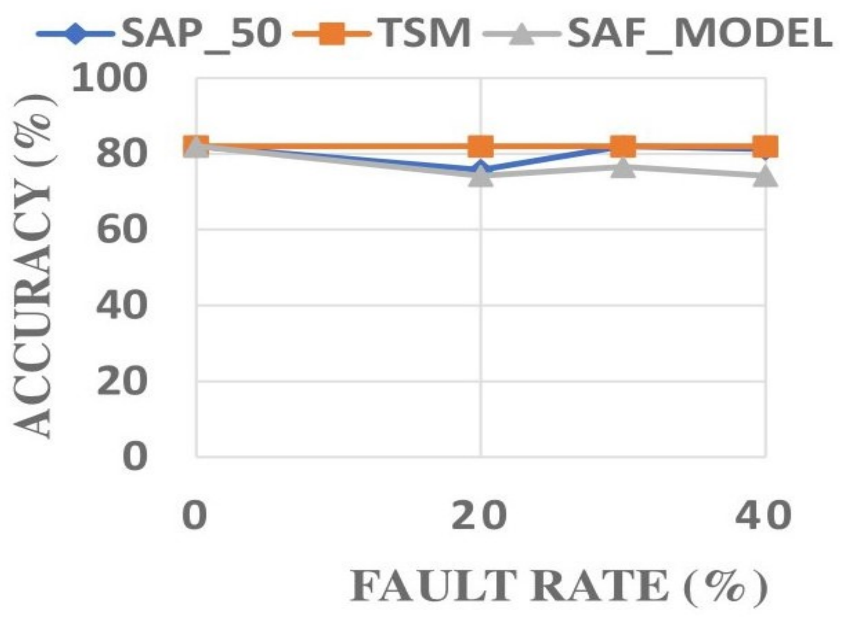


Figure 4.14: Performance comparison of SAP (with 50% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for CSNN-NMNIST.

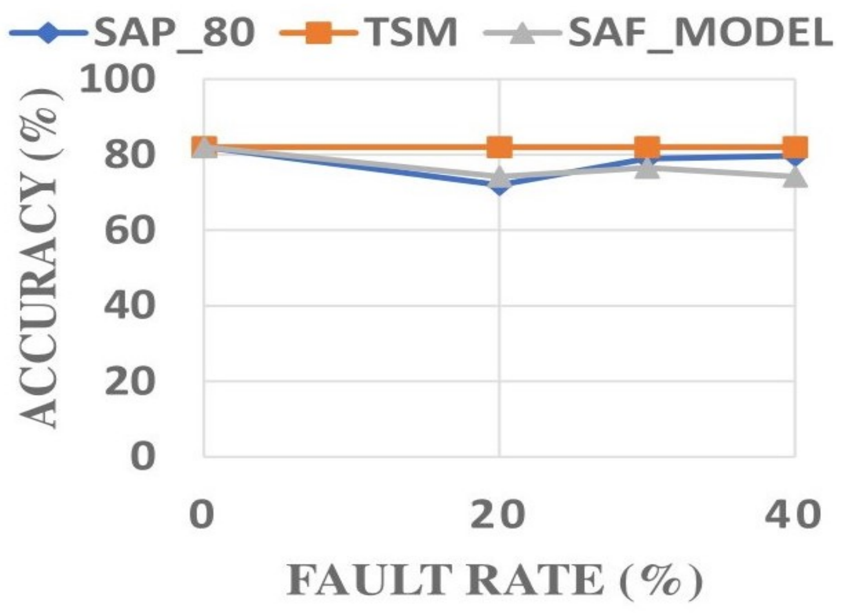


Figure 4.15: Performance comparison of SAP (with 80% faulty neurons removed) and TSM fault recovery methods on a SA-1 fault for CSNN-NMNIST.

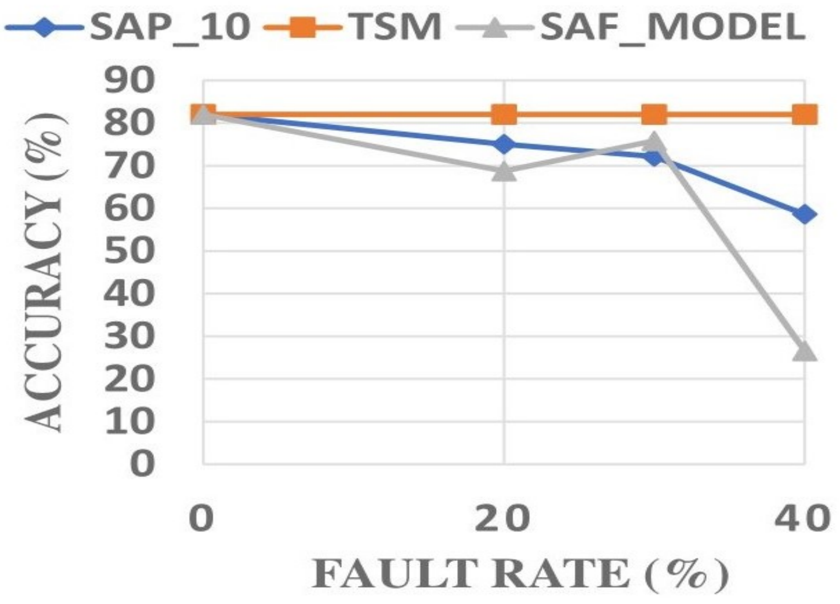


Figure 4.16: Performance comparison of SAP (with 10% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for CSNN-NMNIST.

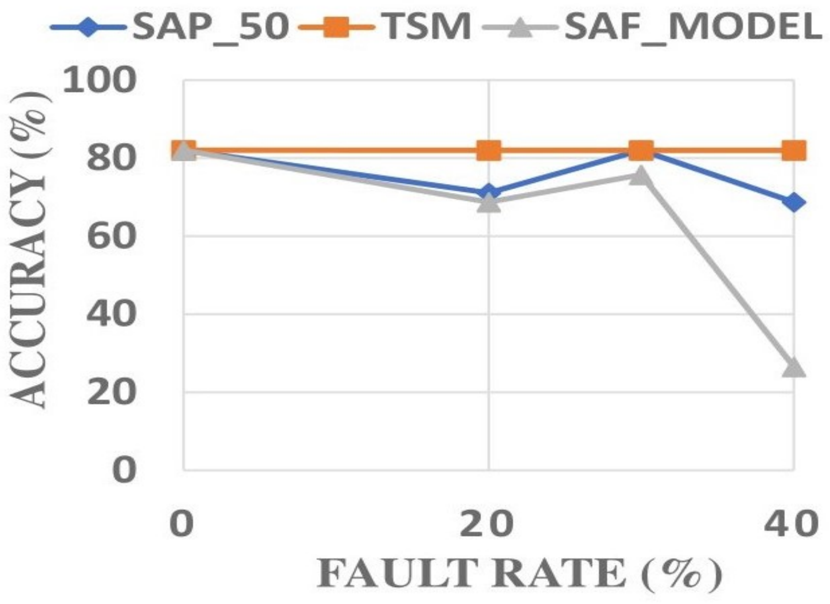


Figure 4.17: Performance comparison of SAP (with 50% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for CSNN-NMNIST.

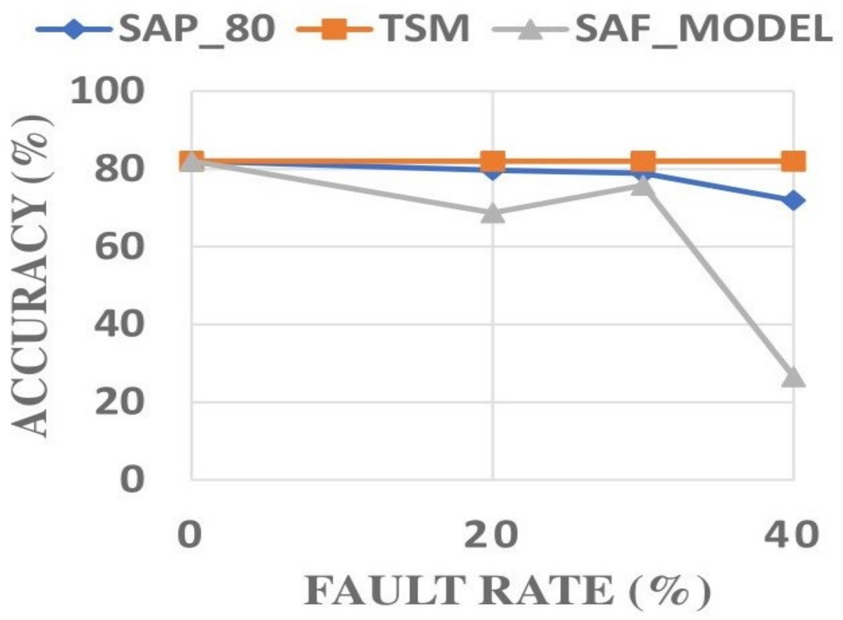


Figure 4.18: Performance comparison of SAP (with 80% faulty neurons removed) and TSM fault recovery methods on a SA-0 fault for CSNN-NMNIST.

and removing 10% (SAP_{10}), 50% (SAP_{50}), and 80% (SAP_{80}) of faulty neurons as shown in Figures 4.16,4.17, and4.18 leads to accuracy recoveries of 58%, 68%, and 71% respectively at a 40% fault rate. This trend highlights that the SAP method's effectiveness in restoring accuracy improves as more faulty neurons are removed. Nevertheless, it's important to note that even with the improved accuracy using SAP, these results still represent accuracy losses of 23%, 14%, and 11% when compared to the baseline accuracy. In contrast, the TSM method achieves an accuracy recovery of 80% on average across different fault rates. This demonstrates a notably lower accuracy loss of 2.31% in comparison to the baseline accuracy.

4.5 Chapter Summary

In this chapter, we discussed fault recovery in neural networks. Furthermore, we explore the impact of faults on neural network applications specifically SNNs. Following an understanding of these impacts, we present algorithms for fault recovery in neural computations needed in neuromorphic systems. Ultimately, these algorithms contribute to the advancement of dependable results by recovering accuracy after hardware faults occur in neural computations. We assumed 40% as the highest fault fault rate in our experiments because the maximum permissible fault rate in even in a system-on-chip (SoC) or any device is contingent upon the specific application and the criticality of the system. In safety-critical domains like aerospace, medical devices, or automotive applications, stringent standards demand exceptionally low fault rates to safeguard users and ensure optimal system performance. Conversely, in applications where safety concerns are less critical, the acceptable fault rate may be higher. However, even in these scenarios, the tolerable fault rate remains subject to the specific requirements and expectations of users. In essence, the fault rate considerations are tailored to align with the criticality of the system and the potential consequences of failures. The next chapter will present a fault-tolerant mapping architecture and algorithm of neuromorphic applications to 3D-NoC-based neuromorphic systems.

5

FAULT-TOLERANT MAPPING ALGORITHM OF SNNS TO NEUROMORPHIC SYSTEMS

It remains a challenge with non-trivial solutions to map SNN applications onto neuromorphic hardware. The mapping technique used can have a significant impact on the system's overall performance. However, this task is increasingly challenging once the neural circuits of neuromorphic systems suffer from potential faults, including high-rate faults that can disrupt the process and compromise neural computation reliability. In this chapter, we introduce a fault-tolerant mapping algorithm and architecture, extending our earlier work on migration-based mapping for recovery to tackle this challenge. We aim to not only improve mapping reliability and robustness but also ensure its efficacy in the face of high-rate faults. Through these advancements, our objective is to make meaningful contributions to the development of dependable and resilient neuromorphic computing systems.

5.1 Fault-tolerant Mapping Overview

Mapping synapses and neurons onto the hardware's cores and circuits is essential for enabling applications to run on neuromorphic hardware. Over the past two decades, numerous studies have introduced mainly performance-driven mapping techniques aiming to translate the brain's operational principles into digital hardware [91]. These techniques primarily focus on minimizing energy consumption and reducing latency [91]. The authors in [5] and [22] propose layer-to-layer mapping to capitalize on the routing algorithm and the 3D mesh topology of the hardware. However, as neurons can fail independently and at any time, the increase in system size raises the probability of more faults in the system, which may impact the performance of the application. To tackle this challenge, our work in [19] introduced migration methods that integrate spare neurons into each cluster of a neuromorphic system using techniques like max-flow min-cut flow, and genetic algorithms. This approach is proposed with considerations from two distinct perspectives. From the first perspective, the approach aims to replicate the redundant elements present in the neural dynamics of the brain. From the second perspective, the strategic inclusion of spare neurons within each cluster is geared towards preserving the performance of the mapping method with a specific focus on communication costs. By incorporating spare neurons into the system, our objective is to guarantee the mapping of all required neurons for executing an application thereby mitigating any potential impact on the application's performance. This strategy establishes a fault-tolerant framework during neuron mapping. While this approach offers several advantages, it does not remap when the number of faulty neurons surpasses the number of spare neurons. Possible solutions include retraining the system with fewer neurons or migrating the entire system to another chip or hardware [92]. These solutions would be effective although they have limitations. Re-training in a hardware-based system can be a time-consuming process, especially for larger applications. Additionally, the cost of migrating to a new chip is often high and may require multiple reconfigurations.
In light of the drawbacks associated with the approaches proposed by the authors in [92] and

the limitations of alternative methods, we investigated the possibility of selectively dropping specific neurons during the mapping process. Our methodology involves integrating a ranking and selection (RSM) mechanism, strategically designed to identify and exclude neurons based on their contribution levels. Improving our migration-based mapping strategy, we introduced a novel Ranking and Selection Mechanism (RSM). This mechanism selectively identifies faulty neurons based on their contribution levels whether across the entire system or within a specific neural circuit. In the upcoming subsections, we will provide detailed insights into how the RSM mechanism contributes to the refinement of the overall mapping approach as part of our migration-based mapping.

5.2 Migration-Based Mapping

A 3D NoC-based neuromorphic system is employed to implement the migration-based mapping approach we proposed in [6] and [19]. In contrast to conventional mapping methods which rely on clustering and partitioning, the migration-based approach also known as the MigSpike approach enables the transfer of tasks from faulty neurons to spare ones within the system. When dealing with faulty neurons, the initial focus is on repairing neurons within a cluster at the node level. If the available spare neurons within a cluster are insufficient for these repairs, the tasks of the faulty neurons are then migrated to other spare neurons across the entire system at the system level. Figure 5.1 illustrates the operational principle of the MigSpike method in a system denoted as S having three layers each composed of nine nodes. Within each of these nodes, there are $E = 256$ neurons along with spare neurons. In this mapping example, there are 5 spare neurons allocated per node providing each node with the capacity to correct up to 5 faulty neurons. Additionally, certain nodes do not possess any faulty neurons making their spare neurons accessible for system repairs. As shown in Figure 5.1a, both layers L_{12} and L_{13} have 10 faulty neurons each but only 5 of them are corrected within their respective nodes due to the limited availability of spare neurons. Consequently, the tasks associated with the remaining 5 faulty neurons are migrated to the neighboring node, L_{15}, which has available spare neurons. This mapping is performed using the genetic algorithm (GA) remapping method

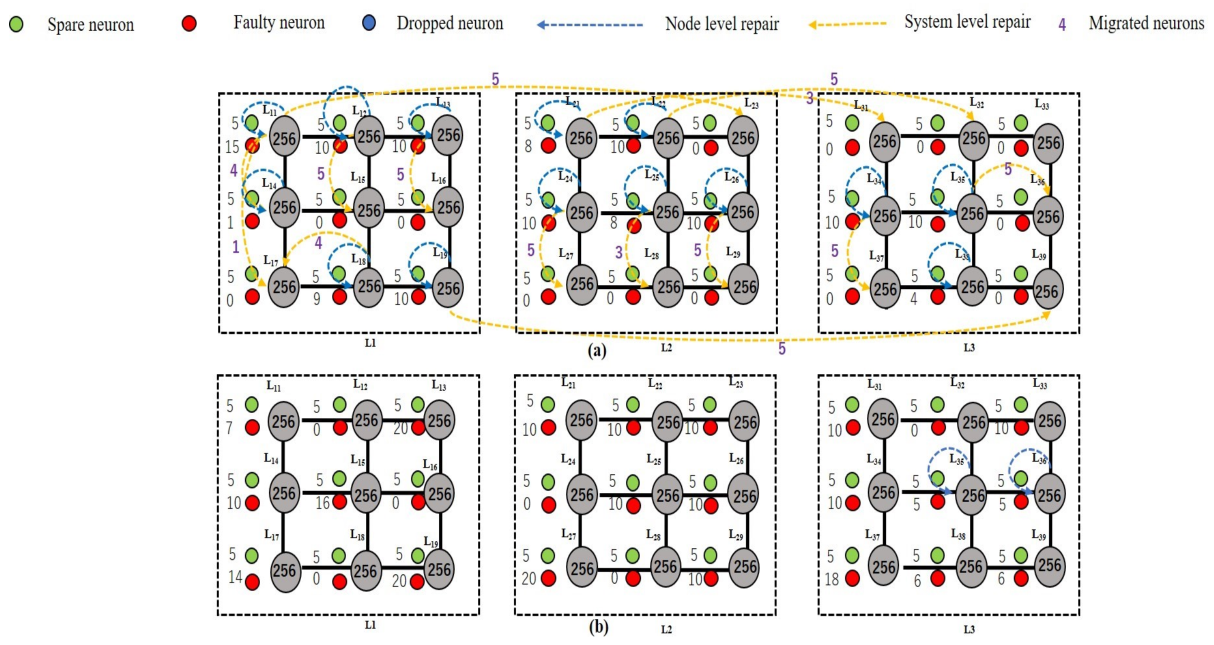


Figure 5.1: An illustration of migration-based mapping on a $3 \times 3 \times 3$ NoC-based neuromorphic system.

introduced in [6]. As for layer L_{11}, 5 faulty neurons are successfully corrected within the node while 10 remains faulty. In this case, L_{14} has 4 spare neurons following node-level repairs and would receive 4 faulty neurons from L_{11}, while 1 faulty neuron is migrated to L_{17}. With spare neurons in neighboring nodes and throughout the layer exhausted, the remaining 5 uncorrected faulty neurons in L_{11} are subsequently migrated to L_{23}. A similar process is applied to nodes L_{19}, L_{21}, and L_{22}. Figure 5.1b illustrates a situation in which a shortage of spare neurons causes system-level repairs to encounter difficulties after node-level repairs. L_2, for instance, has 135 spare neurons available for repairs, whereas 237 faulty neurons exist. When such scenarios arise, it becomes evident that migration-based mapping cannot perform the necessary remapping.

5.3 Migration-Based Mapping with the Proposed Ranking and Selection Mechanism

Our primary concern lies in ensuring neuron mapping is successful even when the number of faulty neurons (k) exceeds the available spare neurons (R). Building upon the principles of the SAP method introduced earlier, we apply a logical adaptation of the concept involving the selective exclusion of neurons within the neuromorphic system during the mapping process using the migration-based mapping technique. Figure 5.2 demonstrates how our proposed mechanism operates during fault-tolerant mapping. The nodes L_{13}, L_{15}, L_{17}, L_{19}, L_{34}, and L_{37} can migrate the tasks of 5, 3, 2, 5, and 4 neurons respectively to other nodes within or outside their clusters by selectively dropping faulty neurons before mapping.

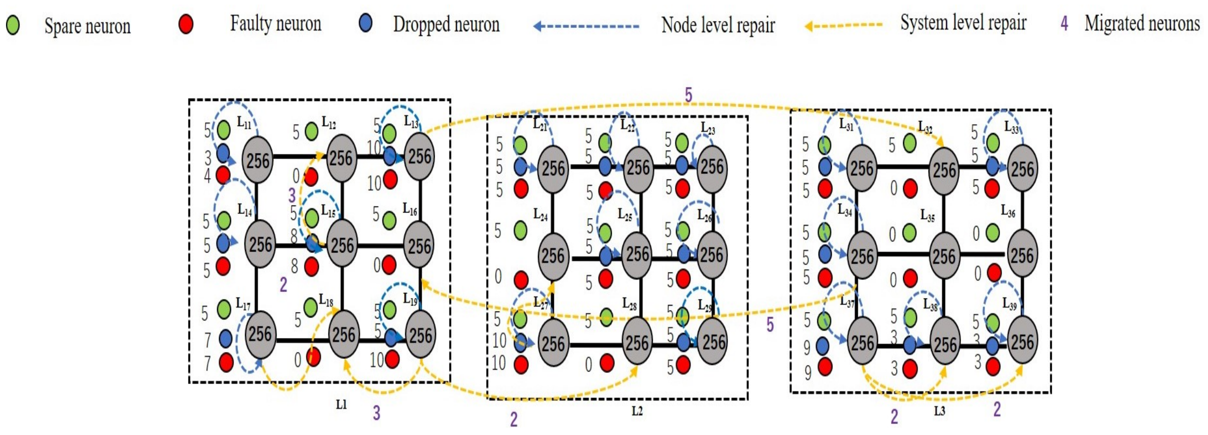


Figure 5.2: An illustration of the migration-based mapping with the proposed RSM on a $3 \times 3 \times 3$ NoC-based neuromorphic system.

In the initial implementation of our migration-based mapping approach presented in [6] and [19], the repair process is applied uniformly to all neurons irrespective of their contributions to a process in the system. In scenarios where the number of spare neurons falls short resulting in an outnumbered situation, the mapping process fails subsequently impacting application performance and system reliability. To address this issue and enhance system stability and reliability, we suggest the removal of the least contributing faulty neurons while retaining the most contributing ones for mapping. Selection procedures are crucial for ensuring the optimal choice from a given set of candidates is selected [93]. To address our challenge, we introduced the proposed RSM which systematically assesses and selects faulty neurons based on their individual contribution levels. With the RSM mechanism, neurons with the highest contributions receive top rank while those with lower contributions are ranked lower. Subsequently, only the highest-ranked neurons are chosen for the remapping process.
A neuron's contribution level (C_l) is determined by its weight value (W_v), as outlined in Equa-

tion 5.1. With Equation 5.2, we compute the average contribution level for all neurons. According to Equation 5.3, each neuron is categorized as either low-ranked (*LR*) or high-ranked (*HR*) according to its average contribution level (A_{C_l}). For neurons with distinct ranks, we select a sum of 'k' and the maximum fault rate ($max_{frate} + k$) number of *HR* neurons. $1 - max_{frate}$ neurons are selected when all neurons share the same rank.

$$C_{l_i} = W_v \tag{5.1}$$

$$A_{C_l} = \sum_{i=0}^{i=p} (C_l)/p \tag{5.2}$$

$$R_k = \begin{cases} HR, & \text{If } C_l > A_{C_l} \\ LR & \text{If } C_l < A_{C_l} \end{cases} \tag{5.3}$$

Here, W_v represents a neuron's total weight value, C_l signifies its contribution level, R_k denotes the neuron's rank, A_{C_l} represents the average contribution level, and p the total number of neurons.

5.3.1 SELECTION THROUGHOUT THE SYSTEM ALGORITHM

Using Algorithm 5.3.3, the first approach in the RSM mechanism is to select faulty neurons that contribute significantly across the entire system during a computation process. The primary objective is to prioritize the removal of the least contributing neurons while selecting those with the highest presumed contribution for mapping. This approach aims to efficiently conserve spare neurons for subsequent remapping, ultimately enhancing the effectiveness and robustness of the mapping process. To illustrate the operation of the RSM mechanism using Algorithm 5.3.3, we will use Figure 5.3. The figure demonstrates scenarios where in a 3D NoC-based neuromorphic system, cluster faulty neurons have different ranks or the same rank. In Figure 5.3a, the system is composed of 9 clusters each containing 256 neurons with 3 reserved as spares. This results in a total of 2304 neurons (256×9) out of which 2213 are actively mapped leaving 27 neurons (3×9) as spares. Unfortunately, we observe that 64 neurons have become faulty. During the ranking phase, a classification based on contribution levels is applied to these faulty neurons resulting in 43 categorized as *HR* (High Rank) and 21 as *LR* (Low Rank). Despite the presence of spare neurons, their numbers fall short of what is required for the remapping process. In the subsequent selection phase, let's assume a fixed max_{frate} and k of 40%. Initially, 32 *HR* neurons are selected. However, this number exceeds the available spare capacity. Consequently, the algorithm initiates a re-ranking and re-selection process among the previously chosen 32 neurons. Following this process, the number of selected *HR* neurons reduces to 26 satisfying the mapping condition ($k < R$). These 26 *HR* neurons are distributed across clusters C_{11}, C_{14}, C_{15}, and C_{17} which initially node-level repairs was performed. Nevertheless, some selected neurons in C_{14}, C_{15}, and C_{17} remain unmapped and are subsequently remapped to clusters C_{12}, C_{13}, C_{16}, C_{18}, and C_{19}.

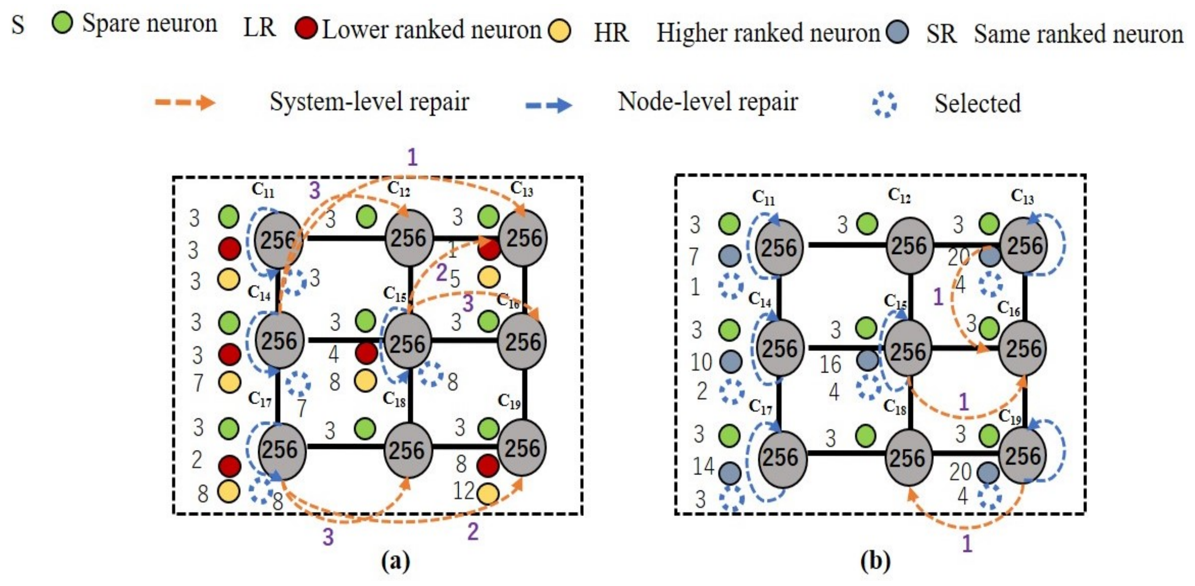


Figure 5.3: An illustration of post-remapping in a $3 \times 3 \times 1$ NoC-based neuromorphic system where each cluster comprises 256 neurons: (a) Post-remapping results following system-wide selection, (b) Post-remapping results with a cluster-by-cluster selection approach within the system.

Algorithm 5.3.3 Ranking and Selection Method (RSM): Throughout the system

```
 1: $F_n$ :Faulty neurons
 2: $k$ :Fault rate
 3: $Max_{frate}$
 4: $HR$ : Higher ranked neurons
 5: **for** faulty neurons $F_n$ in system S **do**
 6:     **Ranking phase:**
 7:     // Calculate the contribution level $C_l$ (Eqn. 5.2)
 8:     // Rank ($R_k$) $F_n$ into $HR$ and $LR$ based on $C_l$ (Eqn. 5.3)
 9:     **Selection phase:**
10:     **if** $F_n$ have different $R_k$ values **then**
11:         // Select $HR \times (Max_{frate} + k)$ neurons
12:     **else**
13:         // $R_k$ of all $F_n$ is the same
14:         // Select $F_n \times (1 - Max_{frate})$
15:     **end if**
16: **end for**
17: **Return:** $HR$ =0
```

Weak Point

With Algorithm 5.3.3, faulty neurons are randomly selected throughout the system by the RSM mechanism. Consequently, some higher-ranked neurons in specific clusters may remain unselected for repairs because only a percentage of higher-ranked neurons are chosen. The random selection throughout an entire system can lead to variations in the choice of faulty neurons potentially excluding those with more significant contributions. While this approach efficiently selects spare neurons for further remapping, optimizing it is necessary to ensure the selection of highly contributing faulty neurons across all clusters regardless of their location within the neuromorphic system.

5.3.2 Selection Cluster by Cluster Algorithm

With the drawback of using the selection approach in Algorithm 5.3.3 apparent, we recognized the need for a more efficient and robust selection strategy. This realization led us to introduce the cluster-by-cluster selection approach formalized in Algorithm 5.3.4. One of its significant advantages is ensuring that at a minimum, clusters containing faulty neurons have at least one neuron selected for repairs. This selection not only enhances the system's fault tolerance but also ensures a more efficient and balanced selection process especially when dealing with randomly scattered faulty neurons throughout the system. Additionally, this strategic shift allows us to prioritize the selection of highly contributing faulty neurons within clusters for repairs ultimately improving the overall fault tolerance and efficiency of the neuromorphic system.
Figure 5.3b illustrates the operation of the RSM mechanism using Algorithm 5.3.4. In this scenario, let's consider a situation where all neurons are ranked equally during the ranking phase. During the subsequent selection phase, the algorithm ensures that at least one faulty HR neuron is selected from each cluster in the system that contains faulty neurons. Following the selection process, clusters containing selected faulty HR neurons perform node-level repairs. However, clusters C_{13}, C_{15}, and C_{19} encounter a shortage of resources to repair all the selected neurons. Consequently, the remaining selected and unmapped neurons within clusters C_{13}, C_{15}, and C_{19} undergo remapping to neighboring clusters C_{16} and C_{18} for system-level repair.

Algorithm 5.3.4 Ranking and Selection (RSM): Cluster by cluster

```
    $F_n$ : Faulty neurons
    $k$ : Fault rate
 3: $Max_{frate}$
    $HR$ : Higher ranked neurons
    for cluster $C = 1, 2, \ldots, N$ in system S do
 6: Ranking phase:
        if $F_n > 1$ then
            // Group all faulty neurons $F_n$
 9:         // Get their contribution levels $C_l$ (Eqn.5.2)
            // Rank ($R_k$) all $F_n$ (Eqn.5.3) into $HR$ and $LR$
    Selection phase:
12:         if $F_n$ have different $R_k$(s) then
                // Select $HR \times (Max_{frate} + k)$
            else
15:             if $R_k$ of all $F_n$ is the same then
                    // Select $F_n \times (1 - Max_{frate})$
                end if
18:         end if
        end if
        if $F_n$ = 1 then
21:         // Select $F_n$
            Return:$F_n$
        end if
24: end for
    Return:$HR$
```

5.6 Evaluation

In this section, we present the results of two distinct experiments each designed to assess the efficacy of the fault-tolerant mapping strategy within 3D NoC-based neuromorphic systems of different sizes. Initially, we conducted fault-tolerant mapping without the application of the RSM. Subsequently, we repeated the mapping procedure, this time incorporating the RSM. Additionally, we conducted a comprehensive Monte Carlo reliability analysis to evaluate the efficiency and effectiveness of the proposed RSM.

5.7 Evaluation Methodology

For the initial experiment, we utilized an SNN application with the configuration specified in Table 6.1 to assess fault-tolerant mapping. This experiment comprised two distinct phases: first, we performed mapping using the GA-based migration method from [6] and [19] without integrating the RSM. Subsequently, we repeated the mapping process, this time incorporating the RSM mechanism. We evaluated these mappings across 3D NoC-based neuromorphic system configurations ranging from $4 \times 4 \times 4$ to $6 \times 6 \times 6$, featuring 64 and 256 neurons per cluster while maintaining a fault rate of $k > 20\%$ as outlined in Table 6.1. Our main objective was to assess the impact of integrating the RSM on mapping efficiency, particularly when there are insufficient spare neurons. The configuration details for the evaluation are provided in Table 6.1. The total number of neurons T is computed using Equation 5.4 for W system network sizes with N neurons per cluster, and I represents the utilized neurons calculated with Equation 5.5. Additionally, for the experiments, we set the maximum fault rate at 20% (for $k < R$) and 40% (for $k > R$).

$$T = N \times W \tag{5.4}$$

$$I = T \times U \tag{5.5}$$

By comparing the results of mapping with and without the RSM mechanism, we aimed to illustrate any behavioral differences and the advantages of utilizing the RSM in the mapping process. To evaluate the RSM's effectiveness in terms of robustness, we conducted a comprehensive reliability analysis assessment.

Table 5.1: Configuration for the evaluation.

Parameter	Value
Neurons per node (N)	64 and 256
Nodes (W)	$4 \times 4 \times 4$, $5 \times 5 \times 5$ and $6 \times 6 \times 6$
Spare neurons (R) (%)	20
Fault rates (k)(%)	5, 15, 20, 25, 30, 35, and 40
Max. fault rate (%)	40
Utilization rate (U) (%)	60
SNN layers	4
SNN	784:0.4*$(N \times W)$:0.4*$(N \times W)$:10
Realizations	100
Duration (Monte Carlo) (yrs)	5

**For mapping, the configuration 784:0.4*(N × W):0.4*(N × W):10 is used for mapping on different 3D NoC-based neuromorphic system sizes. For example an SNN configuration for N=64 and W=*$5 \times 5 \times 5$ *is 784:3200:3200:10.*

5.8 Evaluation Results

5.8.1 Mapping without RSM

We assessed the mapping efficiency of the fault-tolerant mapping method for the SNN application using the configuration specified in Table 6.1 across various sizes of 3D NoC-based neuromorphic systems: $4 \times 4 \times 4$, $5 \times 5 \times 5$, and $6 \times 6 \times 6$, each with 64 and 256 neurons per node. This assessment was specifically carried out for k exceeding R. As illustrated from Figure 5.4 - 5.9, when we set R at 20% and kept k below or equal to 20%, we observed that the number of unmapped neurons remained consistently at zero. However, when k exceeded 20%, the mapping method encountered challenges across all three network sizes. In these cases, the method failed to successfully remap neurons, and the available pool of spare neurons was depleted settling at zero.

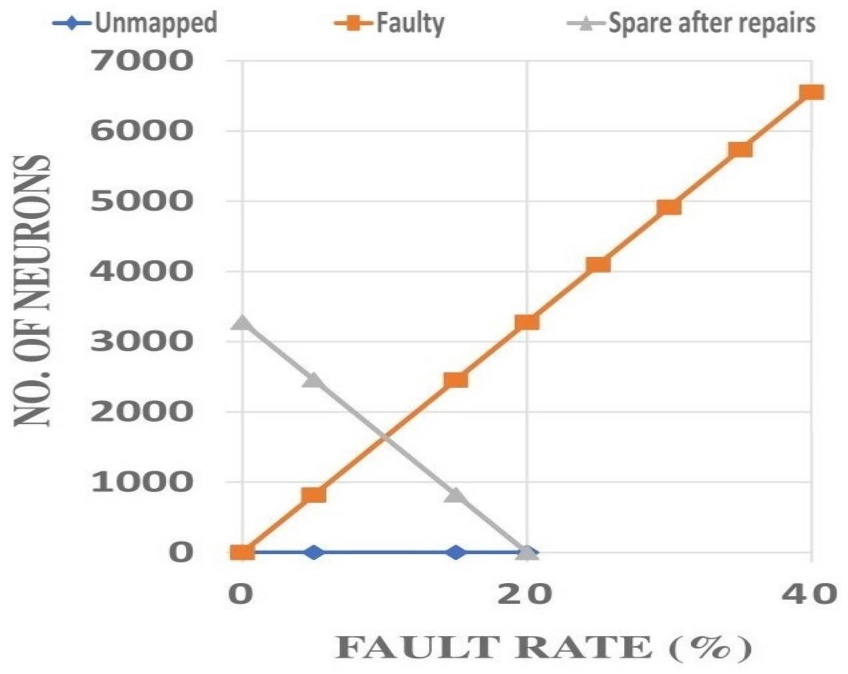


Figure 5.4: Output mapping behavior without the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 256 neurons per cluster.

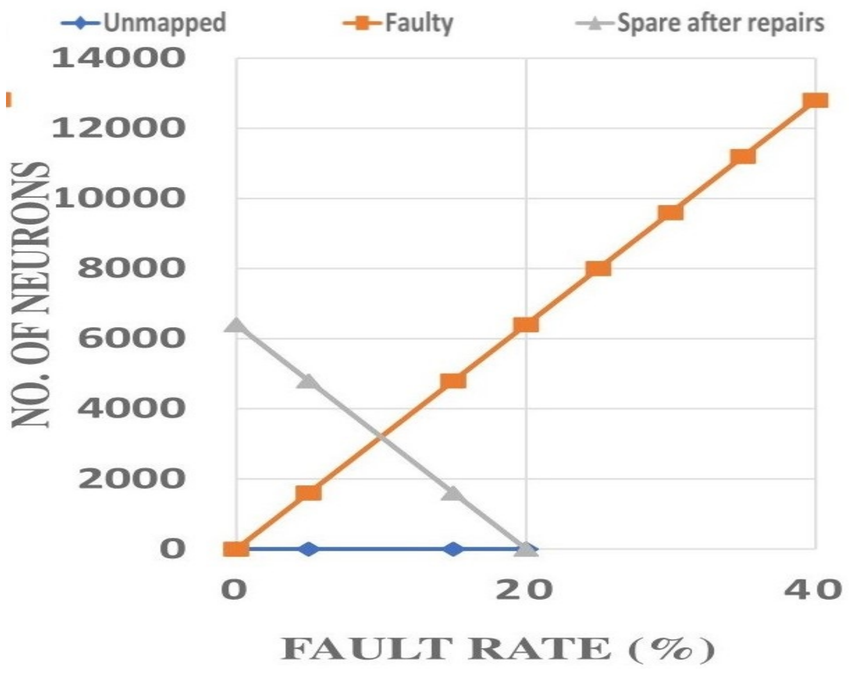


Figure 5.5: Output mapping behavior without the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 256 neurons per cluster.

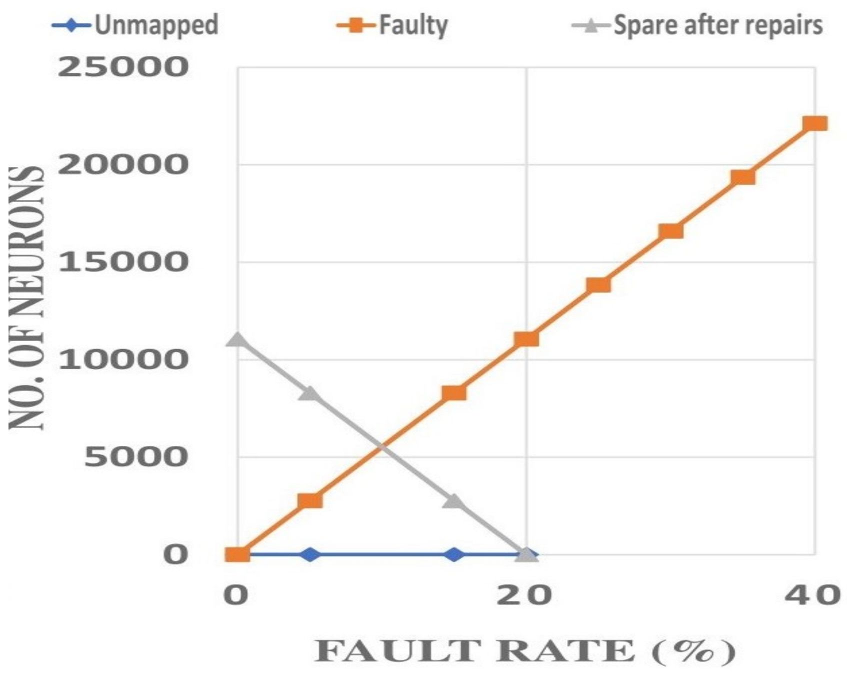


Figure 5.6: Output mapping behavior without the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 256 neurons per cluster.

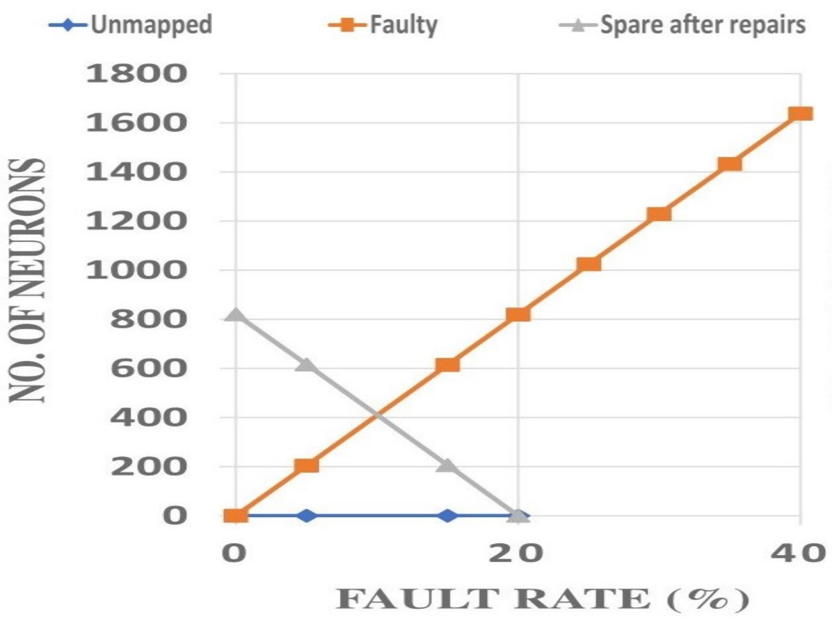


Figure 5.7: Output mapping behavior without the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 64 neurons per cluster.

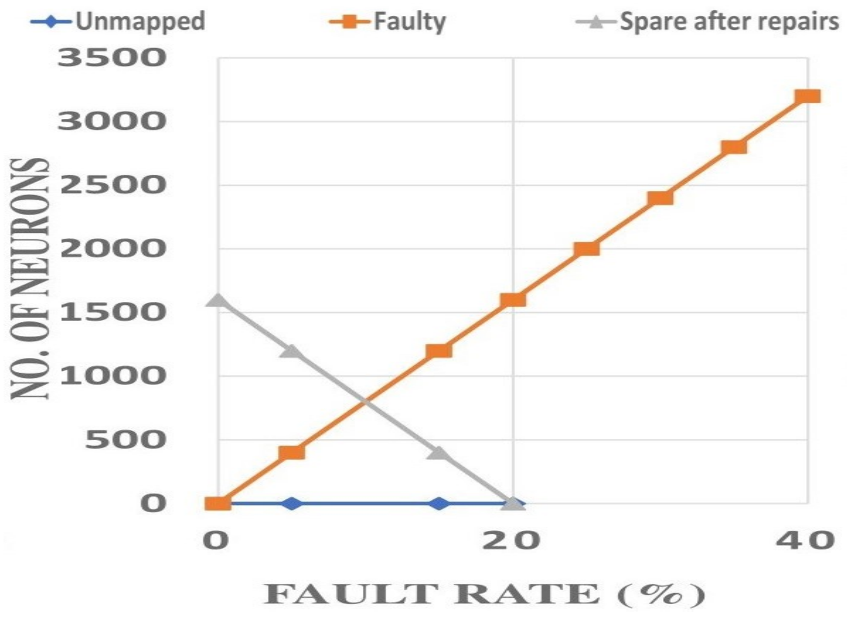


Figure 5.8: Output mapping behavior without the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 64 neurons per cluster.

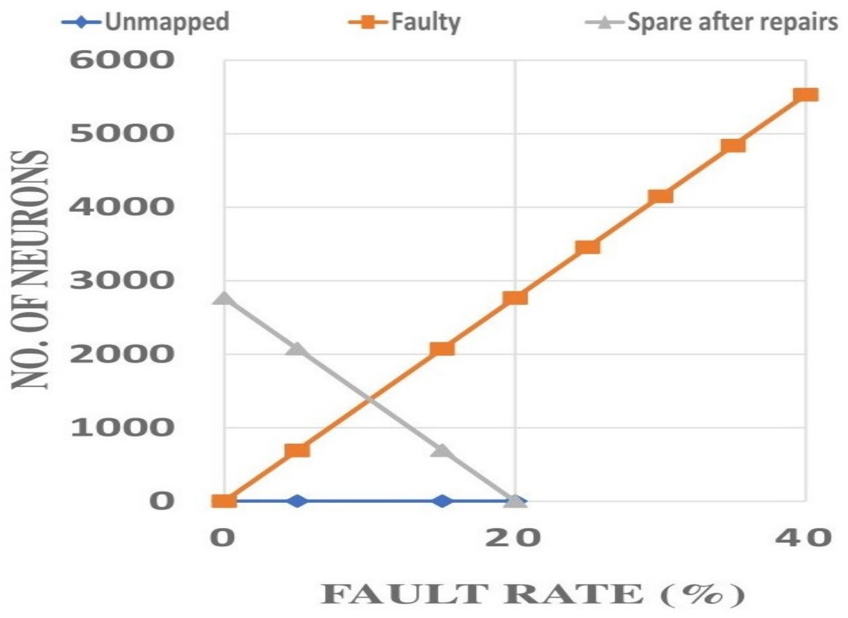


Figure 5.9: Output mapping behavior without the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 64 neurons per cluster.

5.8.2 MAPPING WITH RSM

MAPPING EFFICIENCY

In this assessment, we specifically addressed the scenario where k surpasses R. Figures 5.10 to 5.21 show the mapping efficiency evaluation for neuromorphic systems with 256 and 64 neurons per cluster, respectively. As observed in Figures 5.10 to 5.15, when the neuromorphic system size includes 256 neurons per cluster, there are no remaining unmapped faulty neurons, regardless of variations in their ranks due to contribution levels. This noteworthy accomplishment is attributed to the RSM's capability to prioritize a significant number of higher-ranked neurons for remapping. A similar trend is seen in Figures 5.16 to 5.21, where various neuromorphic system sizes with 64 neurons per cluster exhibit comparable results.

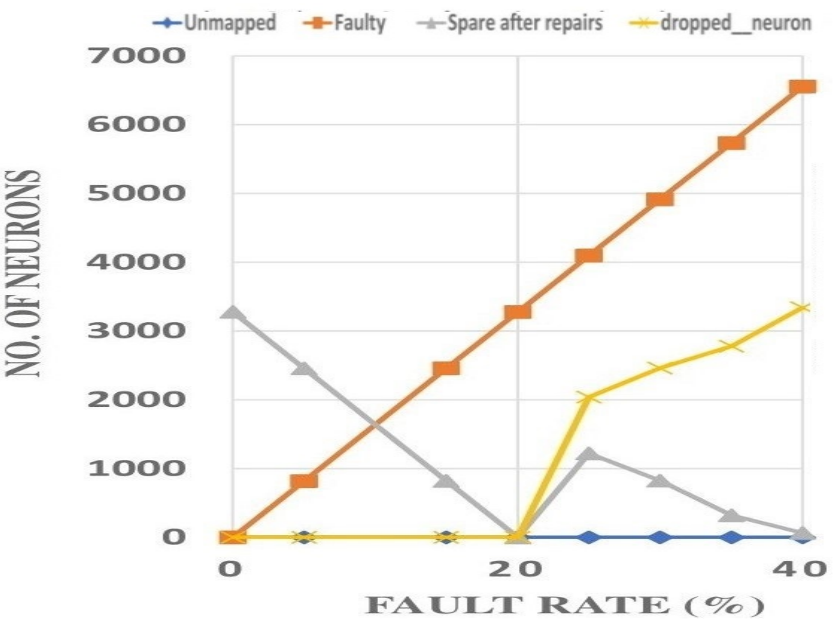


Figure 5.10: Output mapping behavior with the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks different).

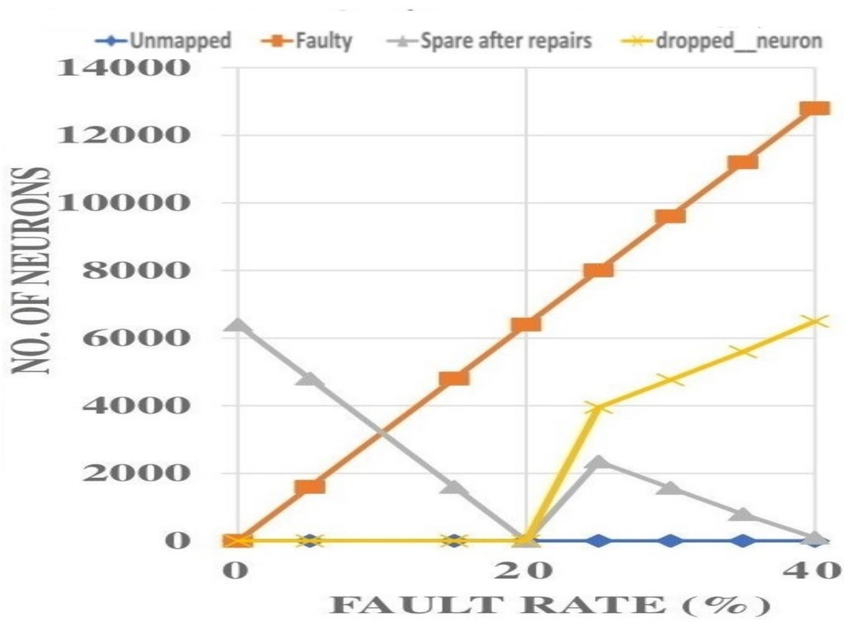


Figure 5.11: Output mapping behavior with the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks different).

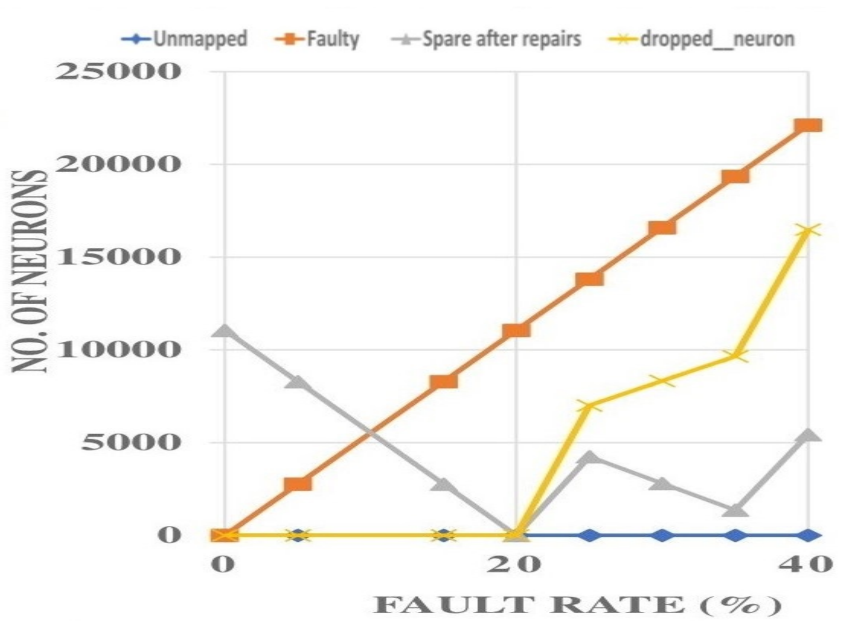


Figure 5.12: Output mapping behavior with the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks different).

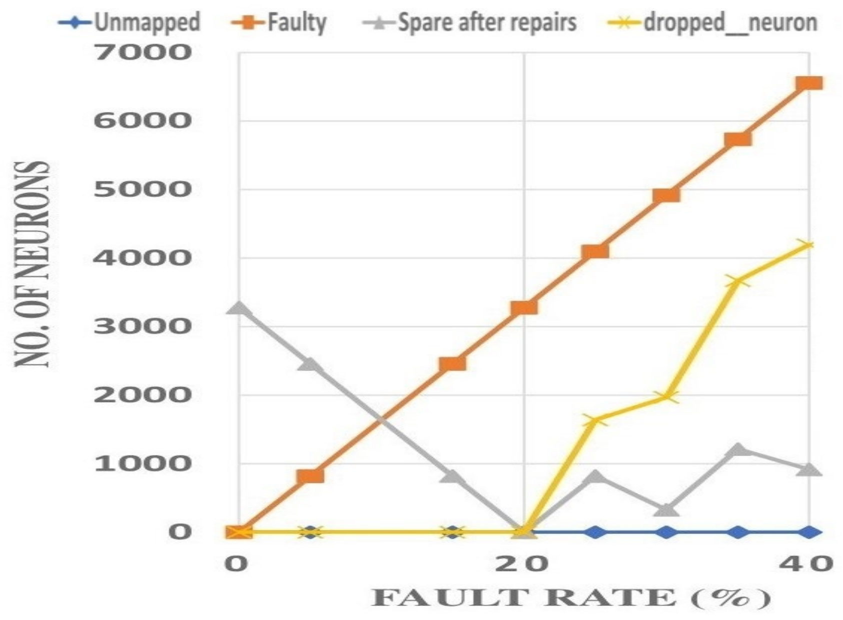


Figure 5.13: Output mapping behavior with the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks the same).

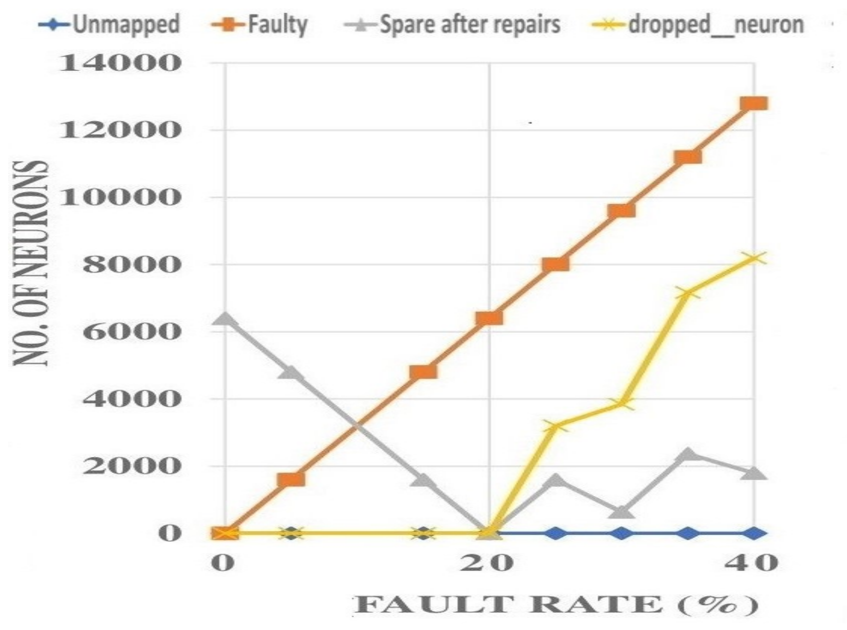


Figure 5.14: Output mapping behavior with the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks the same).

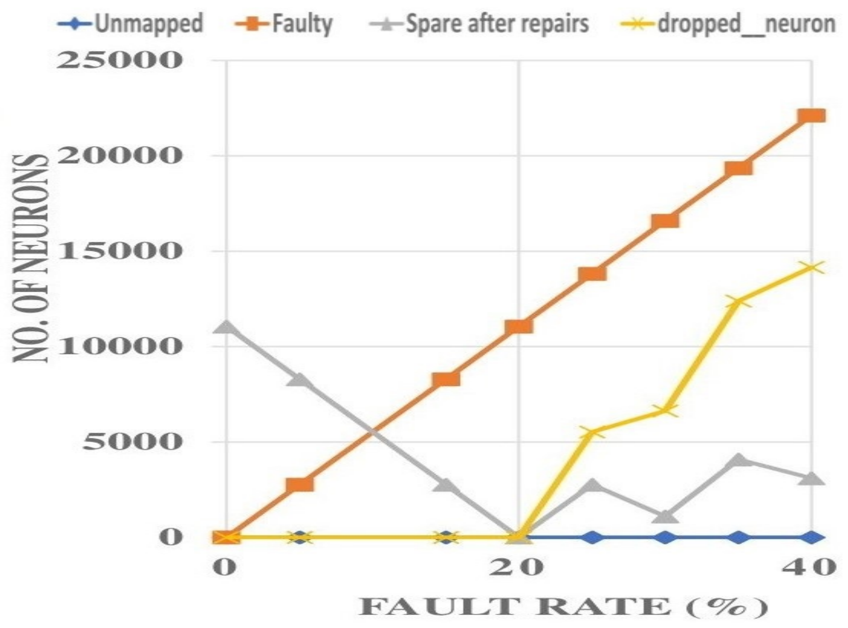


Figure 5.15: Output mapping behavior with the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 256 neurons per cluster(F_N with ranks the same).

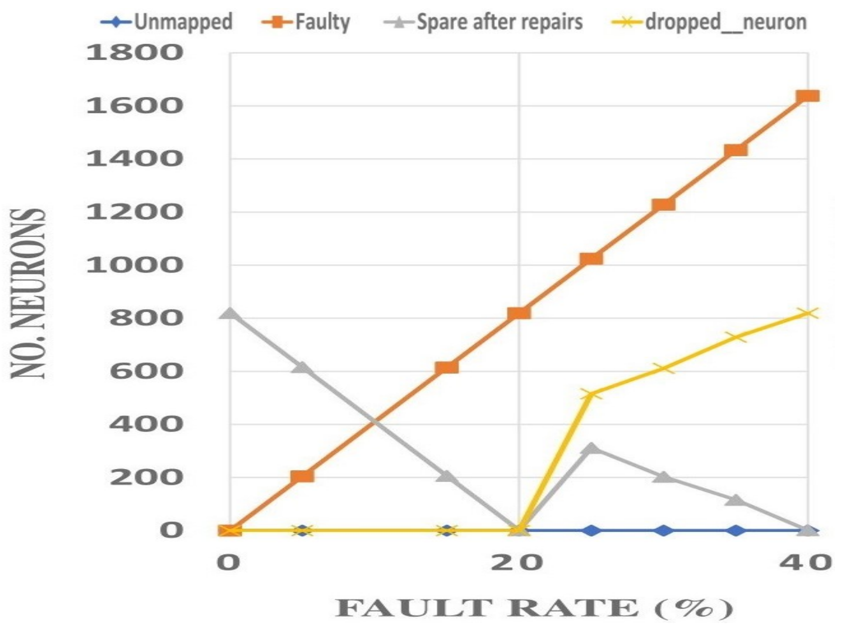


Figure 5.16: Output mapping behavior with the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks different).

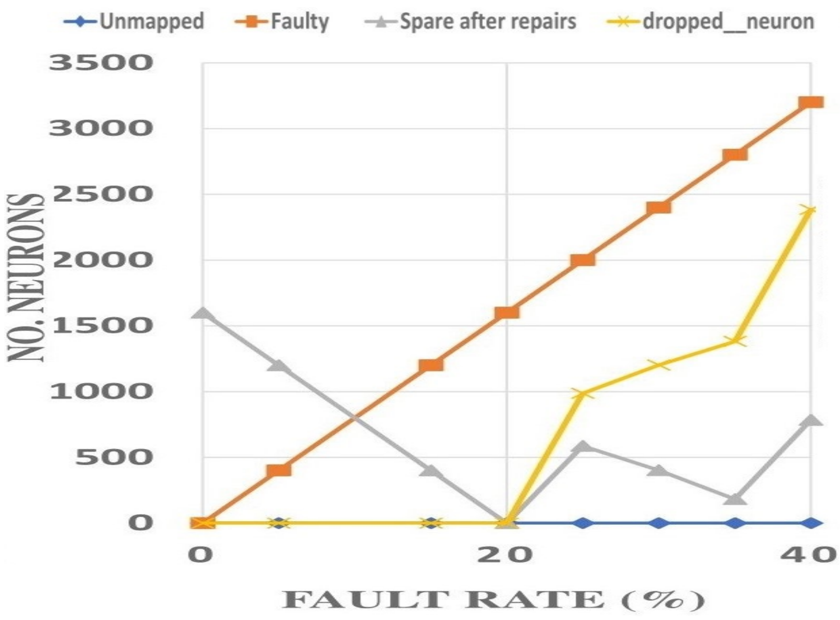


Figure 5.17: Output mapping behavior with the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks different).

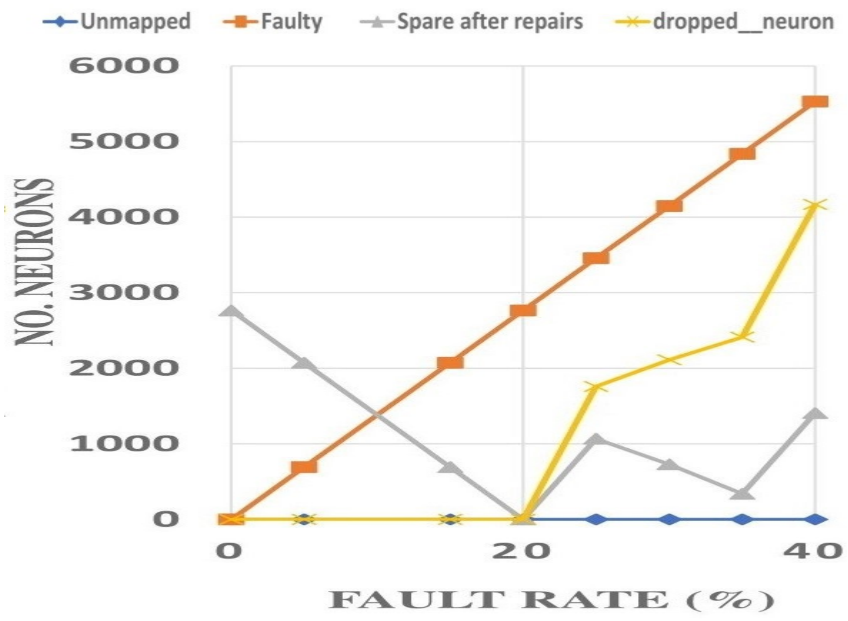


Figure 5.18: Output mapping behavior with the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks different).

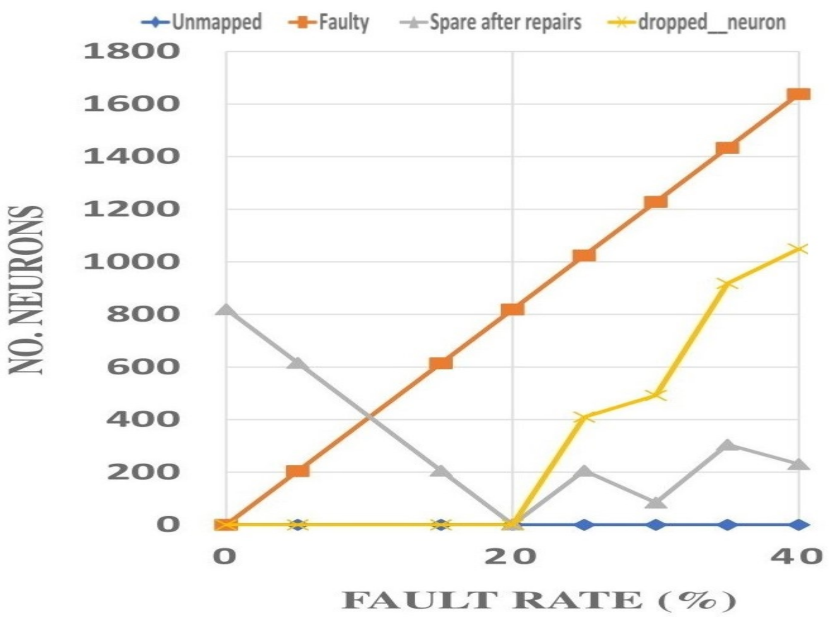


Figure 5.19: Output mapping behavior with the proposed RSM at various fault rates for a $4 \times 4 \times 4$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks the same).

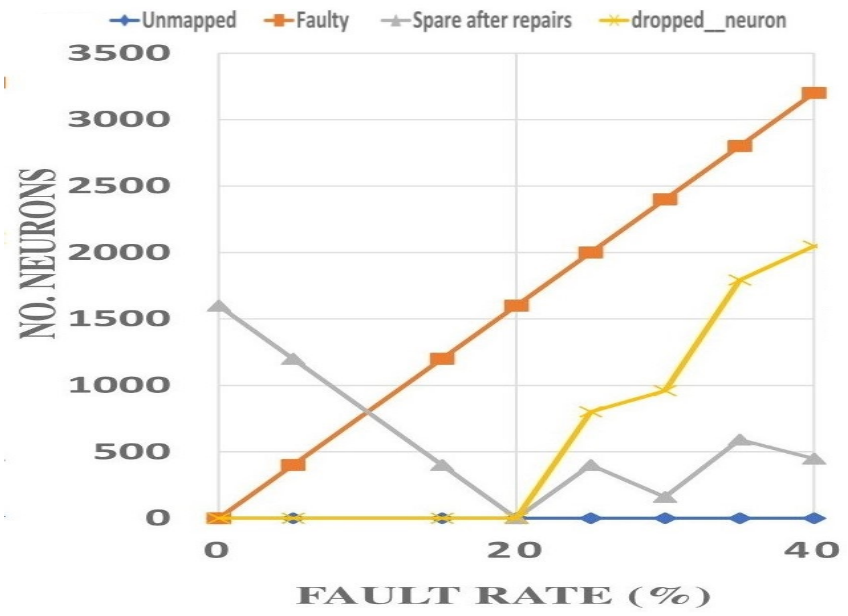


Figure 5.20: Output mapping behavior with the proposed RSM at various fault rates for a $5 \times 5 \times 5$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks the same).

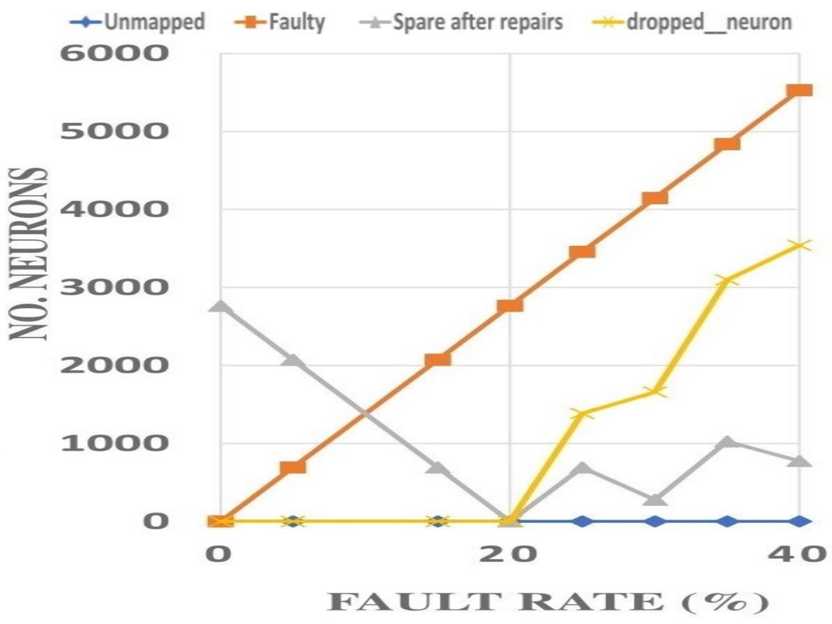


Figure 5.21: Output mapping behavior with the proposed RSM at various fault rates for a $6 \times 6 \times 6$ NoC-based neuromorphic system with 64 neurons per cluster(F_N with ranks the same).

TIME COMPLEXITY

Table 6.3 presents the results of our evaluation regarding the execution time of the RSM across different 3D network sizes. The RSM offers two distinct approaches for ranking and selecting neurons: one that operates throughout the entire system and another that operates on a cluster-by-cluster basis. In the case of ranking and selecting neurons throughout the entire system, the method demonstrates linear time complexity denoted as O(n). This efficiency arises from the algorithm's singular loop which scales linearly with the input size. A notable advantage of this strategy is its concurrent ranking and selection of all faulty neurons contributing to its high efficiency. Conversely, the cluster-by-cluster selection introduces a complexity of O($n \log n$). This complexity emerges from the algorithm's iteration over the cluster size. Consequently, this method may exhibit reduced scalability when applied to very large NoC configurations leading to potential execution time challenges.

Table 5.2: Execution time of RSM for faulty neuron selection in 3D NoCs.

NoC size	RSM selection type	
	Throughout the system	Cluster by cluster
$4 \times 4 \times 4$	21.34 ms	11.16 ms/cluster
$5 \times 5 \times 5$	40.03 ms	21.41 ms/cluster
$6 \times 6 \times 6$	67.81 ms	34.07 ms/cluster

* *The execution time only takes into account the CT for R&S of faulty neurons.*
* *CT: Computation time, R&S: Ranking and selection.*
* *System configuration: N=256, k=40%.*

RELIABILITY ANALYSIS

Reliability analysis is a critical process in assessing a system's capability to fulfill its designated function under specific conditions throughout a defined duration. This systematic evaluation aims to enhance the quality of products, processes, and systems by thoroughly examining key aspects of system performance, including reliability, availability, maintainability, and time to failure. Effectively conducting this analysis often involves making certain assumptions to accurately model an ideal system. These assumptions play a crucial role in establishing a theoretical framework that represents an idealized version of the system. Such a framework serves as a benchmark for evaluating the actual system's performance, facilitating the identification of areas for improvement. In the specific context of evaluating the reliability of the RSM (Reliability, Operational Availability, and Time to Failure) mechanism, two distinct assumptions are considered:

- **Assumption 1:** Neurons can fail independently.

  This assumption is represented using conditional probability. Let $P(F_i)$ be the probability of neuron i failing, and $P(F_i \cap F_j)$ be the probability of both neurons i and j failing simultaneously. The assumption implies that $P(F_i \cap F_j) = P(F_i) \cdot P(F_j)$ for all $i \neq j$.

- **Assumption 2:** The failure rate of higher-ranked neuron selection varies depending on the fault rate in the system.

  Let R_i be the failure rate of higher-ranked neuron selection per year when neuron i has failed. This implies a conditional probability $P(R_i|F_i)$ that depends on the failure of neuron i.

Since our assumptions were based on a probabilistic approach, a probabilistic approach using Monte Carlo reliability simulations supported by the Goldsim simulator [94] [95] [96] is employed. The Monte Carlo method involves running multiple simulations with random inputs to estimate probabilities. In this context, let N be the number of times the model runs (realizations). The reliability of the system can be estimated as the fraction of successful runs. Mathematically, the reliability R is expressed as:

$$R = \frac{\text{Number of successful runs}}{N} \tag{5.6}$$

Tunable parameters include the number of times the model runs (N) and the simulation duration (T) are adjustable, providing flexibility for different experiments while the reliability R is a function of both N and T. To run the Monte Carlo simulation in the Goldsim simulator, our primary parameter is the failure rate when selecting higher-ranked neurons. The failure rate λ over time t can be expressed in Equation 5.7.

$$\lambda = \frac{k_1}{I \times t} \tag{5.7}$$

$$SHR = HR \times (k + Max_{frate}) \tag{5.8}$$

$$k_1 = HR - SHR \tag{5.9}$$

k_1 represents the new count of faulty neurons calculated using Equation 5.9 and I represents the number of neurons utilized by a given application computed using Equation 5.5. Max_{frate} is defined as the maximum fault rate outlined earlier in Table 6.1, HR is defined as the number of higher-ranked neurons, and SHR is defined as the number of selected higher-ranked neurons computed from Equation 5.8. These equations collectively facilitate a comprehensive analysis of the RSM's performance in the context of neuron selection.

Figures 5.22 to 5.24 offer a comprehensive overview of the reliability of the RSM under various scenarios with a 40% fault rate. Notably, in the case of a $4 \times 4 \times 4$ NoC-based neuromorphic system as shown in Figure 5.22, during the fault-tolerant mapping of SNN applications, the RSM mechanism demonstrates an average reliability of 77% over 100 realizations throughout the simulation period. With an upscale to larger system sizes such as the $6 \times 6 \times 6$ configuration illustrated in Figure 5.24, the average reliability experiences a slight decrease to 43%. The operational availability of the RSM stands as a vital metric, representing the probability of the RSM performing its designated function at any given moment. Figures 5.25 to 5.27 illustrate the operational availability of the RSM mechanism during the mapping of SNN applications using the fault-tolerant mapping method with 40% of neurons in a neuromorphic system gone faulty. In the case of a $4 \times 4 \times 4$ NoC-based neuromorphic system, as shown in Figure 5.25, the RSM maintains an average operational availability of 88.2% throughout the simulation, considering 100 realizations. However, for the larger $6 \times 6 \times 6$ NoC-based neuromorphic system shown in Figure 5.27, the operational availability experiences a reduction to 67.8%. For time-related metrics, we consider the TTF and the MTTF of RSM. TTF represents the duration between the initiation of RSM and its eventual failure, while MTTF signifies the average TTF of RSM. In Figure 5.28, the $4 \times 4 \times 4$ NoC-based neuromorphic system demonstrates an impressive TTF of 75.46 years. Additionally, Figure 5.29 reveals that the MTTF for the same system configuration is 16.8 years. This prolonged operational period is attributed to the selective repair of a limited number of higher-ranked neurons. However, in the case of a $6 \times 6 \times 6$ NoC-based neuromorphic system, as observed after 100 realizations, both TTF and MTTF exhibit a slight reduction. These time-related metrics collectively offer valuable insights into the reliability and operational characteristics of the RSM, providing a nuanced understanding of its performance across varying conditions and 3D NoC-based neuromorphic system sizes.

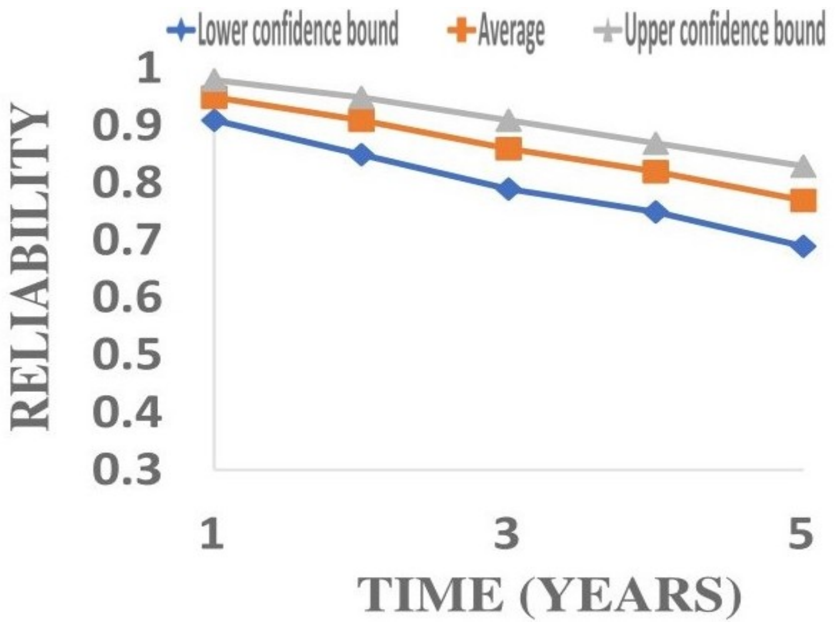


Figure 5.22: Reliability Plot of RSM for a $4 \times 4 \times 4$ NoC-based neuromorphic system throughout the simulation duration.

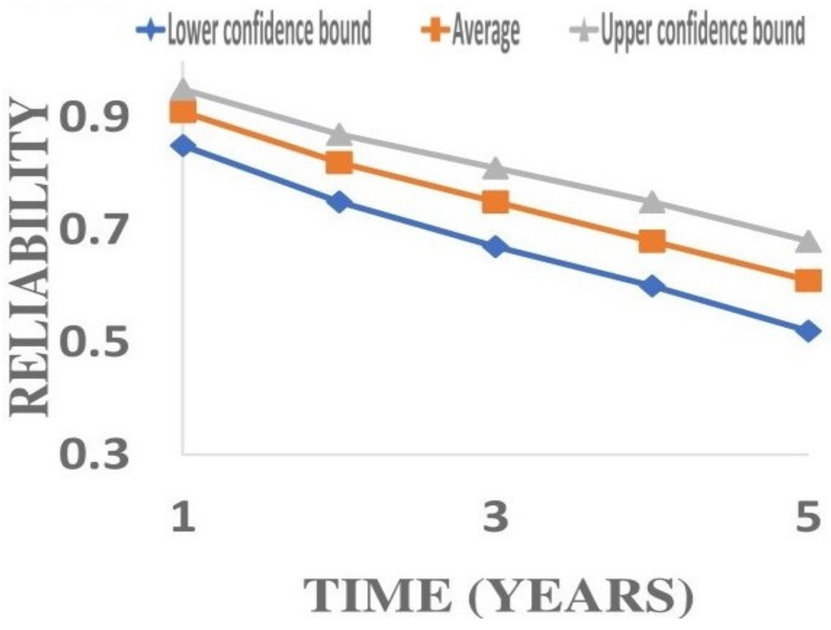


Figure 5.23: Reliability Plot of RSM for a $5 \times 5 \times 5$ NoC-based neuromorphic system throughout the simulation duration.

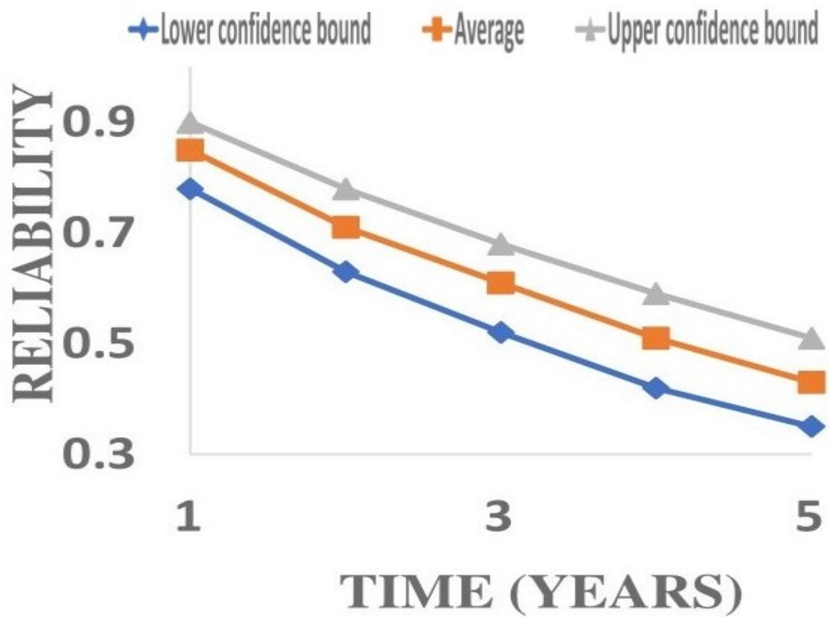


Figure 5.24: Reliability Plot of RSM for a $6 \times 6 \times 6$ NoC-based neuromorphic system throughout the simulation duration.

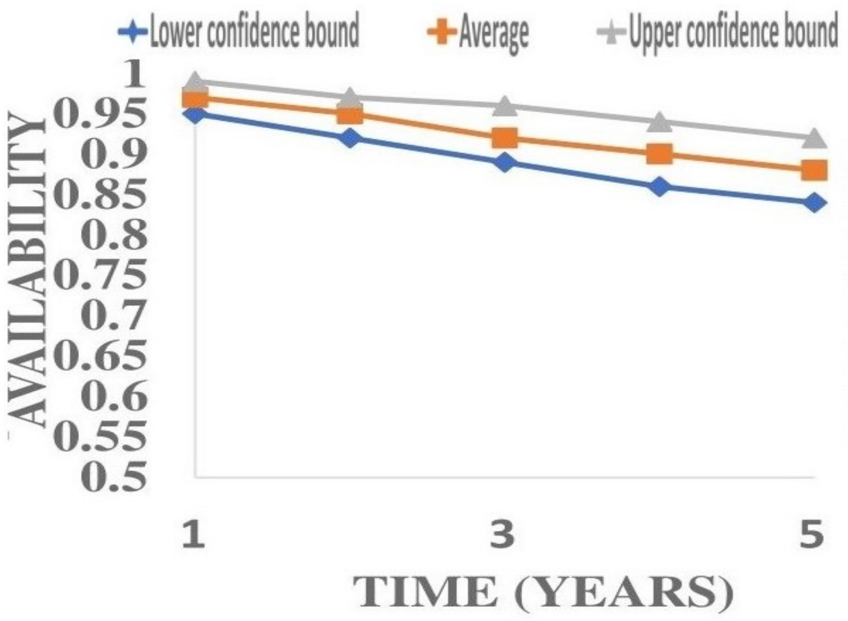


Figure 5.25: Operational availability plot of RSM for a $4 \times 4 \times 4$ NoC-based neuromorphic system throughout the simulation duration.

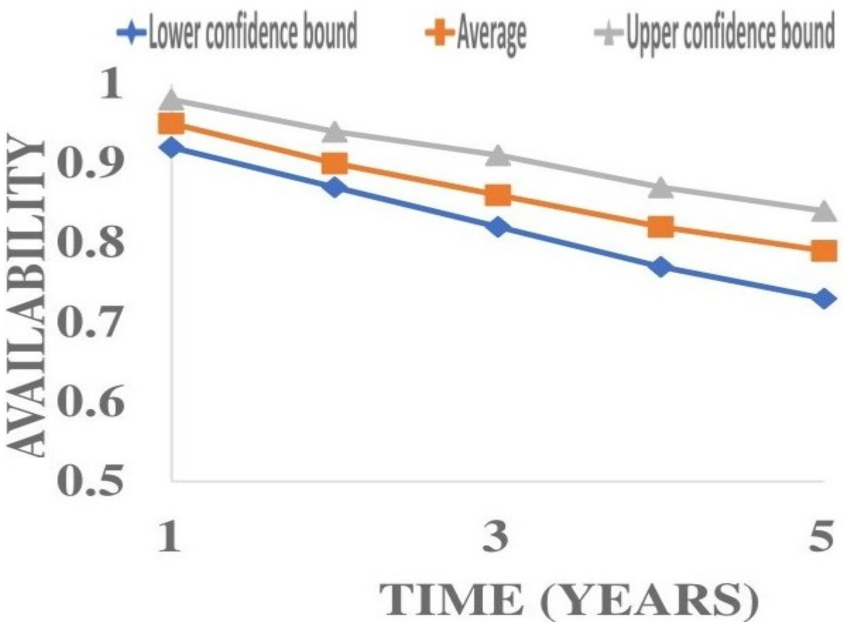


Figure 5.26: Operational availability plot of RSM for a $5 \times 5 \times 5$ NoC-based neuromorphic system throughout the simulation duration.

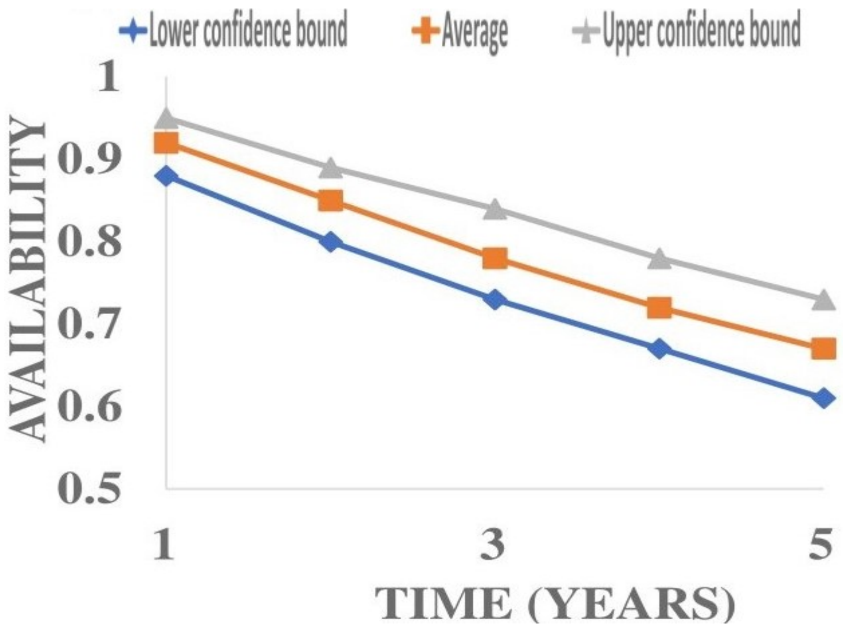


Figure 5.27: Operational availability plot of RSM for a $6 \times 6 \times 6$ NoC-based neuromorphic system throughout the simulation duration.

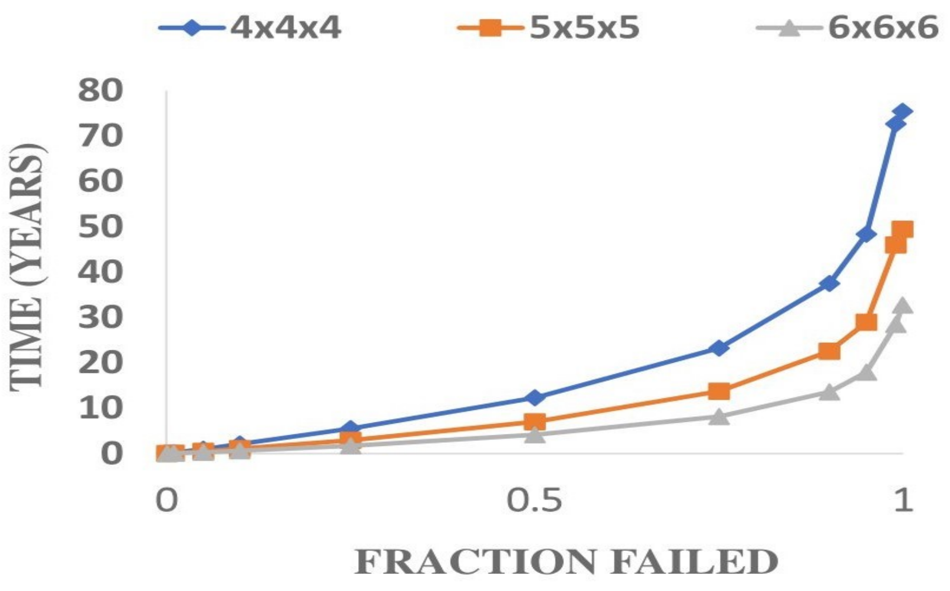


Figure 5.28: Distribution plot of TTF across three different 3D NoC-based neuromorphic system.

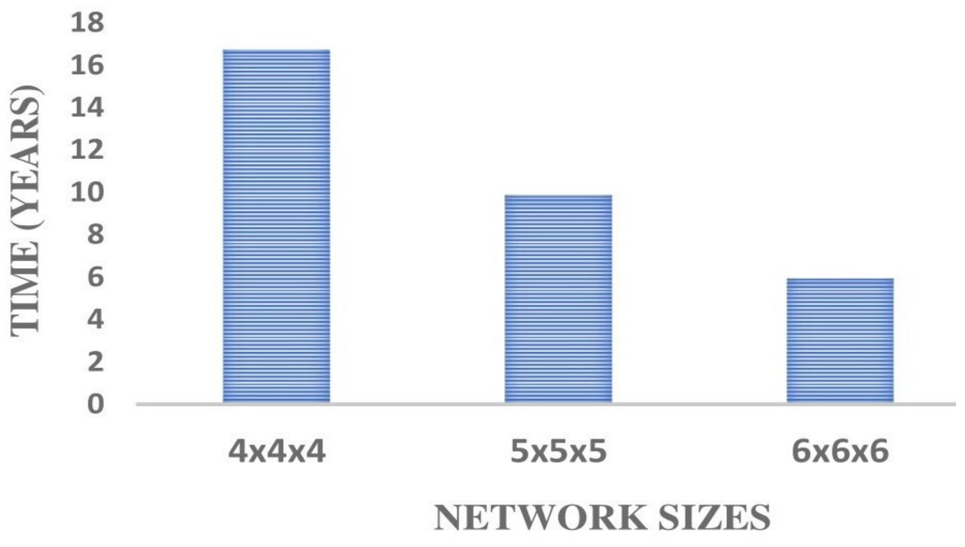


Figure 5.29: Distribution plot of MTTF across three different 3D NoC-based neuromorphic system sizes.

5.9 Chapter Summary

In this chapter, we introduced a fault-tolerant mapping architecture and algorithm integrated with a novel ranking and selection mechanism. The objective of this integration is to ensure continuous mapping even in the presence of high-level faults while minimizing associated cost implications. We conducted a comprehensive evaluation of the fault-tolerant mapping method both before and after its integration with the novel selection mechanism and assessed its efficiency. It is important to acknowledge that the fault-tolerant method relies on added redundancies for recovery presenting certain challenges and drawbacks. The next chapter will present a novel neuromorphic applications mapping approach featuring a novel fault-tolerant mechanism, specifically designed to address challenges associated with the utilization of redundancy as a fault-tolerant mechanism.

6 ROBUST MAPPING TO NEUROMORPHIC SYSTEMS

THE mapping of SNN applications onto neuromorphic systems poses a complex challenge, particularly when attempting to strike a balance between system efficiency and reliability. In our previous efforts to address reliability concerns within neuromorphic systems, we proposed a fault-tolerant mapping approach centered on redundancy implementation and integrated a ranking and selection mechanism. The primary objective was to attain thorough system recovery to overcome the challenge of performance degradation and diminished system reliability particularly in the presence of high-level faults all while carefully managing cost considerations. However, limitations became evident using redundant elements as fault-tolerant mechanisms including significant resource costs and the potential depletion of those (redundant) resources over time. Moreover, alternative fault-tolerant mechanisms such as dynamic resource reconfiguration and periodic redundancy introduction proved to be impractical. This chapter introduces a mapping approach with a novel fault-tolerance mechanism, inspired by the theory of neural reuse. The objective is to address the constraints and drawbacks associated with the utilization of additional redundancies, as well as those related to alternative fault-tolerant mechanisms discussed earlier. By repurposing existing neurons within the system, we demonstrate the adaptability of this brain-inspired approach by effectively reutilizing neurons in specific regions, mitigating errors stemming from faulty neurons. The novel mapping approach provides scalability advantages allowing the scaling of the size of a NoC-based neuromorphic system without necessitating a corresponding increase in fault-tolerant neurons. By removing the constraints of finite fault-tolerant resources, our mapping approach paves the way for a more sustainable, cost-effective, and efficient method of application mapping within the realm of neuromorphic computing.

6.1 The Proposed Mapping Method (R-MaS3N)

The authors in [97] have illustrated the concept of re-purposing existing components to enhance performance. They introduced a novel approach for reusing convolution layers without introducing new ones. While their primary objective was performance enhancement, our focus lies in improving system reliability through fault tolerance. In our proposed methodology, the mapping process initiates with the initial mapping of neurons from an SNN model to clusters within a 3D NoC-based neuromorphic system. Once this mapping process is completed, a subset of neurons in the 3D NoC-based neuromorphic system is intentionally designated as faulty to simulate faults in the system. To achieve fault tolerance and improve system reliability, our new mapping approach known as R-MaS3N employs a two-step process. Initially, unmapped layer neurons are categorized into two partitions based on their spiking patterns: the most active and the least active. In the subsequent step, neurons within the 3D NoC-based neuromorphic system are organized into high and less-active regions. Within the less active region, neurons are further sorted based on their spiking activities. The partitioning process ensures that only neurons in the less active region of the 3D NoC-based neuromorphic system are employed for mapping to achieve fault tolerance and enhance reliability. In R-MaS3N, high-activity neurons are constrained from being mapped to high-spiking neurons to avoid the potential overloading of high-spiking neurons ensuring a balanced and efficient distribution of neural activity. One key aspect of the R-MaS3N lies in its efficient reuse of existing neurons which significantly reduces resource wastage. The following subsections will provide a comprehensive description of both the initial mapping and fault-tolerant mapping steps of R-MaS3N. Subsequently, a detailed design explanation of two critical procedures in the remapping phase of R-MaS3N will be presented. It is essential to note that the terms "clustering" and "partitioning" are used interchangeably in this context.

6.1.1 Initial Mapping to Neuron Clusters

In the initial mapping of neurons to clusters, R-MaS3N follows a systematic layer-to-layer process, as outlined in our previous work [5]. This mapping procedure entails the sequential allocation of application neurons to clusters within each layer of the 3D NoC-based neuromorphic system. Starting with an initial cluster, application neurons are mapped progressively until all neurons within that cluster are efficiently utilized. The process then seamlessly transitions to the next cluster within the same layer iterating until all clusters containing available neurons in the specific layer have been successfully mapped with application neurons. This structured methodology enables the mapping of the neural network application illustrated in Figure 6.1a, onto the 3D NoC-based neuromorphic system presented in Figure 6.1b. The detailed illustration of the mapping sequence is depicted in Figure 6.2.

From Figure 6.1b, each cluster in the 3D NoC-based neuromorphic system contains four neurons. In NL_1 of the 3D NoC-based neuromorphic system, there are a total of 36 neurons distributed with 1, 3, 2, 4, 3, and 3 faulty neurons in clusters C_{11}, C_{12}, C_{13}, C_{14}, C_{15}, and C_{16}, respectively. Similarly, L_1 of the neural network application comprises 36 neurons. After the initial mapping, 16 application neurons remain unmapped. The lack of availability of all required application neurons on the hardware may result in reduced reliability or even the inability to perform desired tasks due to degraded or incomplete functionality.

6.1.2 Robust Mapping to Neuron Clusters

Figure 6.3 illustrates how R-MAS3N tackles the challenge presented in Figure6.1 that leads to degraded system performance and reduced system reliability using the novel fault-tolerant

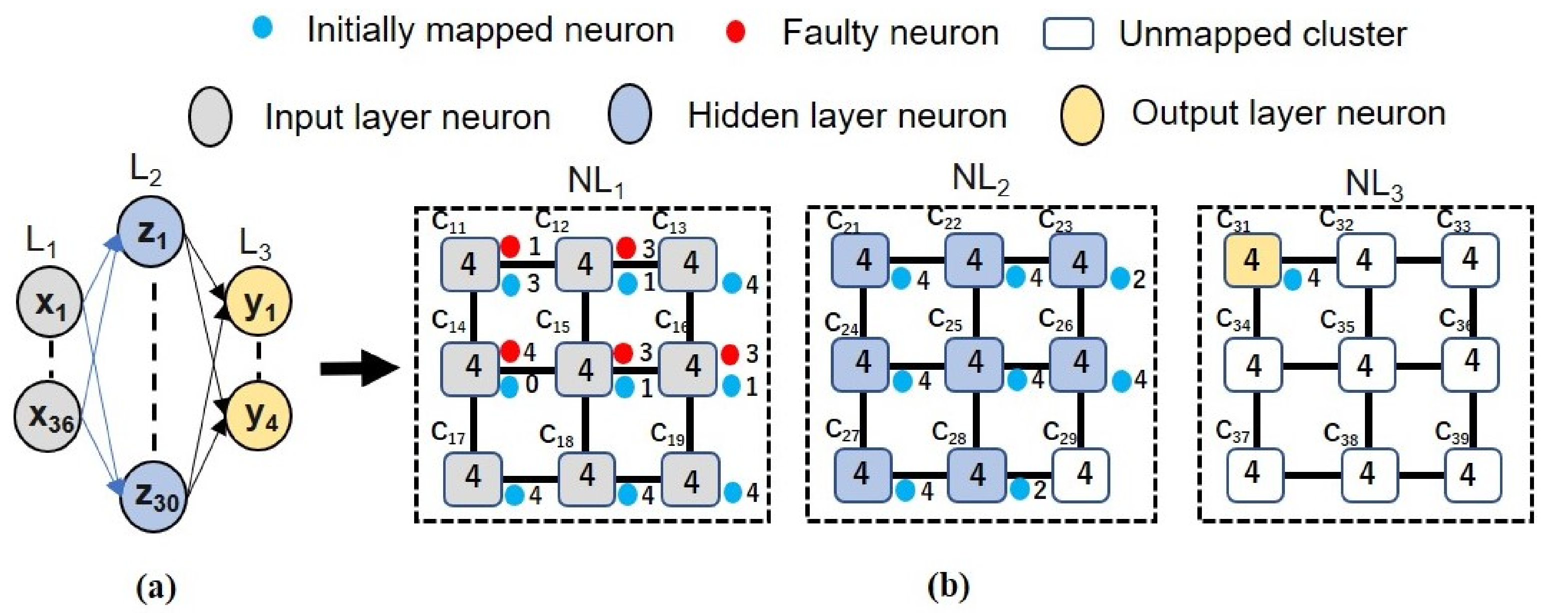
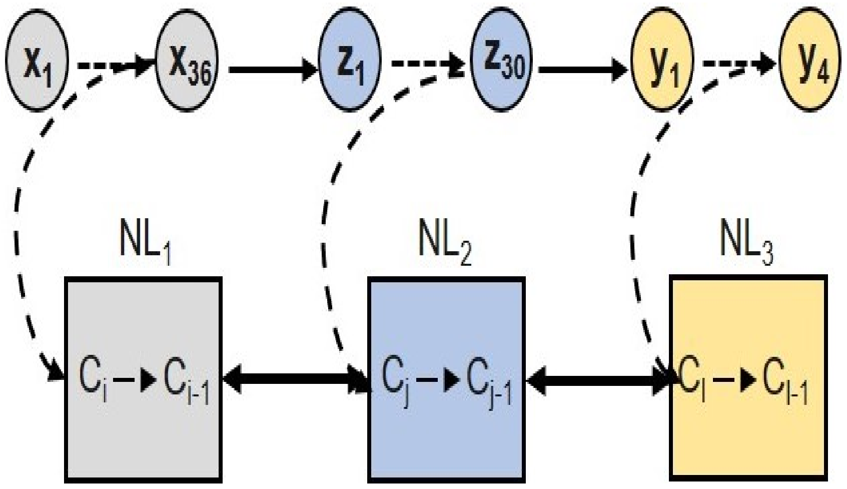


Figure 6.1: An illustration of the initial SNN mapping process on a $3 \times 3 \times 3$ NoC-based neuromorphic system: (a) Represents the neural network application, while (b) showcases the neural network application mapped onto the neuro cores of the 3D NoC-based neuromorphic system. Our mapping method, as introduced in [5], is employed for this mapping. It's noteworthy that certain clusters, specifically C_{11}, C_{12}, C_{14}, C_{15}, and C_{16} within NL_1, contain faulty neurons. Consequently, the mapped neurons in these clusters are fewer than the expected maximum capacity.



x_1
x_{36}
z_1
z_{30}
y_1
y_4
NL_1
NL_2
NL_3
$C_i \rightarrow C_{i-1}$
$C_j \rightarrow C_{j-1}$
$C_l \rightarrow C_{l-1}$

Figure 6.2: Descriptive overview of the mapping sequence executed on the 3D NoC-based neuromorphic hardware, as detailed in Figure 6.1. The process involves mapping neurons from each layer of the network application to their corresponding layers within the neuromorphic system.

mechanism. Building upon the initial mapping shown in Figure6.1b, there are a total of 16 unmapped neurons from L_1 of the neural network application. Based on their spiking behavior, these neurons are categorized into two partitions: the most active and the least active. Subsequently, as detailed in Figure6.3b, 10 neurons are assigned to the most active partition, while 6 neurons are allocated to the least active partition.

In the subsequent step, all the neurons already mapped in NL_1 of the 3D NoC-based neuromorphic system are divided into high and less-active regions. As shown in Figure 6.3c, clusters C_{15}, C_{18}, and C_{19} are classified within the less-active region. Within these clusters, neurons are further sorted into underutilized and highly utilized ones through rank sorting. Following the sorting process, C_{18} and C_{19} have 2 and 3 underutilized neurons respectively while clusters C_{15}, C_{18}, and C_{19} contain 1, 2, and 1 highly utilized neurons respectively. Given that our primary objective is to achieve fault tolerance, R-MAS3N first allocates 10 neurons from the most active partition to underutilized neurons within clusters C_{18} and C_{19}. Similarly, 6 neurons from the least active partition are mapped to the highly utilized neurons in clusters C_{15}, C_{18}, and C_{19}.

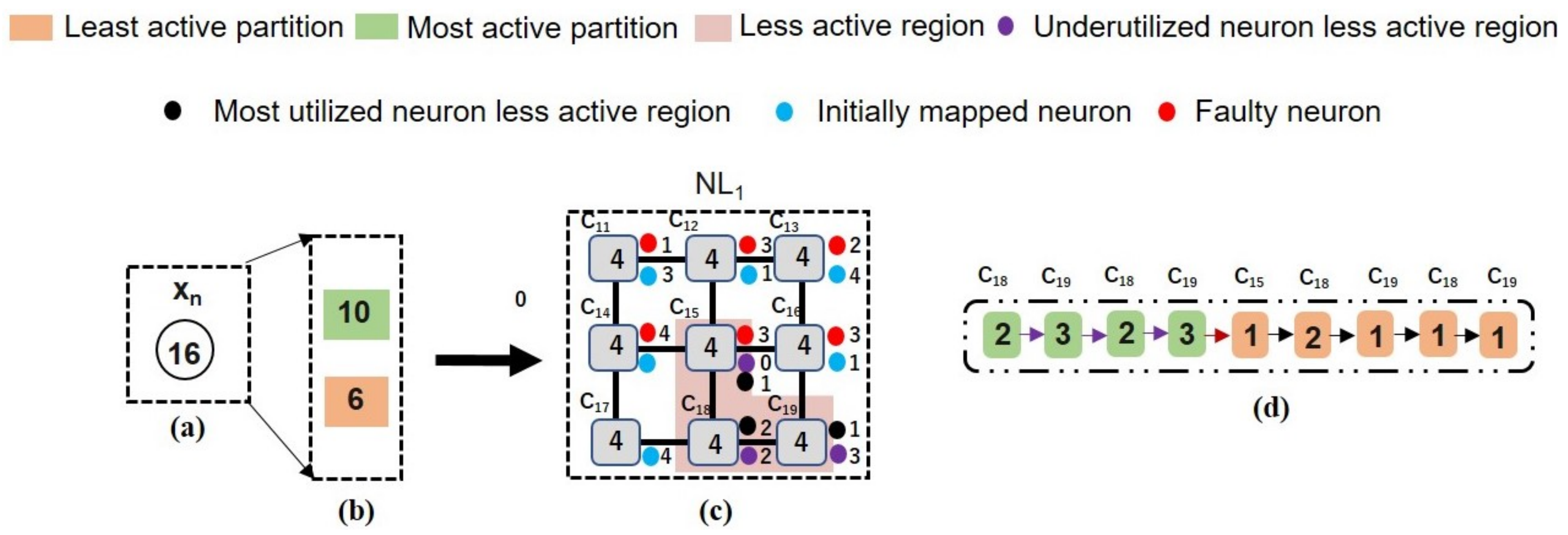


Figure 6.3: An illustrative representation of the proposed solution within a $3 \times 3 \times 1$ NoC-based neuromorphic system, addressing the mapping challenge detailed in Figure 6.1. The sequence unfolds as follows: (a) Depicts the unmapped neurons originating from L_1 of the neural network application after the initial mapping. (b) Illustrates the partitioning of these unmapped neurons into two distinct groups: the most active and least active partitions. (c) Demonstrates the subsequent remapping process where partitioned, unmapped neurons from both the most active and least active partitions are reassigned to neurons located within the less-active regions of the respective layer in the 3D NoC-based neuromorphic system. (d) Offers a visual representation of the sequential steps involved in the remapping process within NL_1.

SNN Layer Partitioning (SLP) Algorithm

The fault-tolerant process in the R-MaS3N commences the remapping process following the initial mapping which leaves certain application neurons unmapped due to faults in the 3D NoC-based neuromorphic system. The initial step in the remapping process involves the partitioning of the application model neurons into two distinct groups based on their firing patterns. This process is formalized in Algorithm 6.1.5. To determine these partitions, we utilize metrics such as spike counts (S_c) and consecutive spike frequency (F_{cs}).
For the partitioning process, we calculate the average spike count (Avg_{S_c}) and the average frequency of consecutive spike counts ($Avg_{f_{cs}}$) across all neurons within the system as defined by Equations 6.1 and 6.2. Neurons having spike counts and consecutive spike frequencies above the respective averages are grouped as the most active neurons (C_{most}) partition per Equation 6.3. Conversely, neurons with spike counts and consecutive spike frequencies below or equal to the averages are grouped as the least active neurons (C_{least}) partition based on Equation 6.4.

$$\text{Avg}_{S_c} = \frac{1}{m}\sum S_c(n_i) \quad \forall n_i \in N \tag{6.1}$$

$$\text{Avg}_{f_{cs}} = \frac{1}{m}\sum F_{cs}(n_i) \quad \forall n_i \in N \tag{6.2}$$

$$C_{\text{most}} = \{n_i \in N \mid S_c(n_i) > \text{Avg}_{S_c} \text{ and } F_{cs}(n_i) > \text{Avg}_{f_{cs}}\} \tag{6.3}$$

$$C_{\text{least}} = \{n_i \in N \mid S_c(n_i) \leq \text{Avg}_{S_c} \text{ or } F_{cs}(n_i) \leq \text{Avg}_{S_c}\} \tag{6.4}$$

When addressing the mapping challenge presented in Figure 6.1, neurons within the C_{most} partition are mapped initially to underutilized neurons and subsequently to the most utilized neurons from the C_{less} partition within the 3D NoC-based neuromorphic system. The same mapping strategy applies to unmapped application neurons within the C_{least} partition. It's important to note that while there are similarities, the SNN layer partitioning problem differs from the classical graph partitioning problem [77] [78] [80]. The classical graph partitioning problem aims to divide a graph into subsets to optimize objectives such as minimizing edge connections between partitions and achieving balanced partition sizes. However, both problems share the commonality of dividing elements be it neurons or graph nodes into groups based on specific criteria. How the SNN layer neuron partitioning method partitions unmapped application neurons is illustrated in Figure 6.4. To partition neurons of layer 2 from Figure 6.4a into C_{most} and C_{least} partitions, assuming these neurons have F_{cs} values of 7, 6, 10, 11, 4, and 5, and S_c values of 200, 300, 1500, 100, 1000, and 700, respectively. The $Avg_{f_{cs}}$ calculated using Equation 6.1 is 7 and the Avg_{S_c} computed using Equation 6.2 is 633 for all the neurons. Based on these results, the neurons are partitioned into two distinct partitions, C_{most} and C_{least} following the criteria outlined in Equations 6.3 and 6.4. Specifically, neurons X_4, X_5, X_7, X_8, and X_9 are placed in the C_{least} partition, while neuron X_6 is assigned to the C_{most} partition. The resulting partitions are shown in Figure 6.4c.

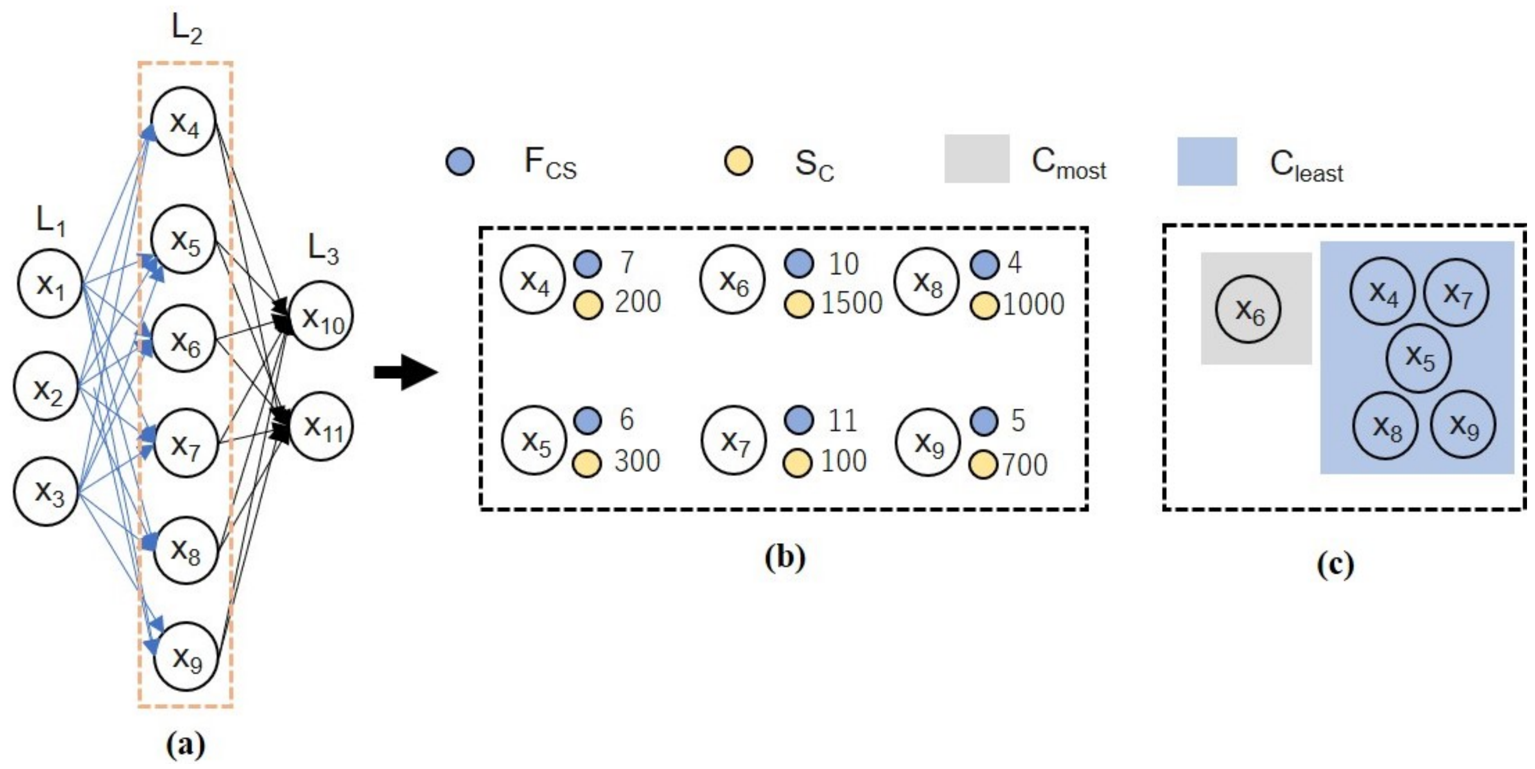


Figure 6.4: An illustration of an SNN layer divided into two distinct neuron partitions using the SLP algorithm: (a) The SNN application (b) Sample layout of the characterized neurons (c) Layout of the clustered neurons.

Algorithm 6.1.5 SNN Layer Partitioning Algorithm

```
1: procedure LAYER PARTITIONING($N$) /* Input: Set of neurons $N$ */
2:     Calculate $Avg_{S_c}$ and $Avg_{f_{cs}}$
3:     Initialize empty sets $C_{\text{most}}$ and $C_{\text{least}}$
4:     for $n_i \in N$ do
5:         if $S_c(n_i) > Avg_{S_c}$ and $F_{cs}(n_i) > Avg_{f_{cs}}$ then
6:             Move $n_i$ to $C_{\text{most}}$
7:         else
8:             Move $n_i$ to $C_{\text{least}}$
9:         end if
10:    end for
11:    Return: $C_{\text{most}}$, $C_{\text{least}}$ /* Output: Neurons in $C_{\text{most}}$ and $C_{\text{least}}$ regions */
12: end procedure
```

NEURON PARTITIONING (NP) ALGORITHM

After partitioning the neurons in the layer(s) of the SNN application, the next step in the fault-tolerant process of the R-MaS3N involves clustering neurons on the 3D NoC-based neuromorphic hardware into high-active and less-active regions. Algorithm 6.1.6 formalizes this step. To classify neurons as highly active ($n_i \in C_{\text{high}}$) region, two conditions must be met: they must have a higher number of positive synaptic connections (*PC*) than negative synaptic connections (*NC*), and their S_c must exceed the Avg_{S_c}. We assume that these parameters are derived from the configuration of the 3D NoC-based neuromorphic hardware. The criteria for clustering neurons into either C_{high} or C_{less} regions are as follows: A neuron belongs to the highly active region ($n_i \in C_{\text{high}}$) if both Equations 6.5 and 6.6 hold true:

$$PC(n_i) > NC(n_i) \tag{6.5}$$

$$S_c(n_i) > \text{Avg}_{S_c} \tag{6.6}$$

Neurons that do not meet these criteria are classified in the less active ($n_i \in C_{\text{less}}$) region. Once the partitioning is complete, the next step is to identify underutilized and most utilized neurons within the C_{less} region by performing rank-order sorting using Algorithm 6.1.7. This sorting primarily considers the spike count (S_c) as the main criterion. First, the S_c is calculated for each neuron $n_i \in C_{\text{less}}$ region. Then, a rank score (RS) is assigned to each neuron based on its S_c. The rank score is determined by ranking S_c in ascending order and assigning lower scores to neurons with fewer S_c. Neurons are sorted based on their rank scores in ascending order with those having the lowest rank scores (indicating fewer S_c) placed at the top of the sorted list. Neurons in the first half of the sorted list are termed "underutilized," while neurons in the second half are termed "most utilized". How the neuron partitioning method works is illustrated

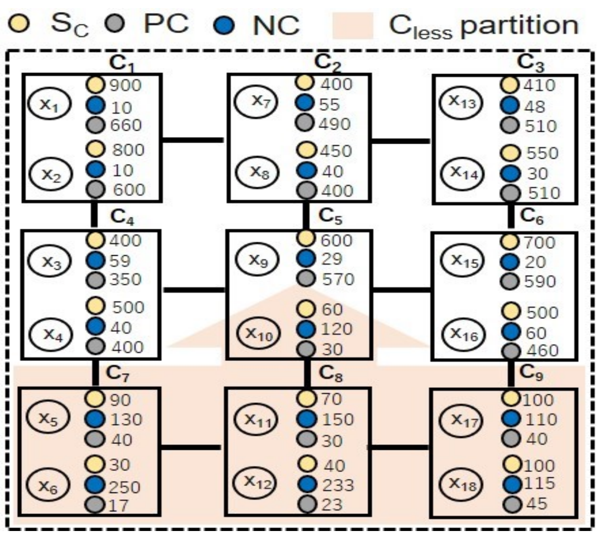


Figure 6.5: An illustrative representation of neurons in clusters within a $3 \times 3 \times 1$ NoC-based neuromorphic system, divided into C_{high} and C_{less} partitions utilizing the NP algorithm.

in Fig. 6.5. It is assumed that neurons within each cluster exhibit distinct S_c values, PC counts, and NC counts, derived from the neuromorphic system configuration. The AvgS_c is calculated as 369 using Equation 6.1. To partition neurons into the C_{high} and C_{less} regions, Equations 6.5 and 6.6 are employed. Evidently, neurons in C_1 to C_4, C_6 clusters, and neuron X_9 in cluster C_5 have S_c values surpassing the computed AvgS_c and exhibit a higher count of PC than NC. Consequently, these neurons are designated to the C_{high} partition. In contrast, neurons in C_7, C_8, C_9 clusters, and neuron X_{10} in cluster C_5 showcase more NC than PC and S_c values below the calculated AvgS_c. Therefore, these neurons are allocated to the C_{less} partition.

Algorithm 6.1.6 Neuronal Partitioning

```
1: procedure NEURON PARTITIONING($M_n$) /* Input: Set of mapped neurons $M_n$ */
2:     Calculate $Avg_{S_c}$ for all neurons $n_i \in M_n$.
3:     Initialize empty sets $C_{\text{high}}$ and $C_{\text{less}}$.
4:     for each neuron $n_i \in M_n$ do
5:         Calculate $PC$, $NC$, and $S_c$.
6:         if $PC > NC$ and $S_c > Avg_{S_c}$ then
7:             Move $n_i$ to $C_{\text{high}}$.
8:         else
9:             Move $n_i$ to $C_{\text{less}}$.
10:        end if
11:    end for
12:    Return $C_{\text{high}}$, $C_{\text{less}}$ /* Output: Neurons in $C_{\text{high}}$ and $C_{\text{less}}$ regions */
13: end procedure
```

Algorithm 6.1.7 Rank-Based Sorting

```
1: procedure RANKSORT($C_{\text{less}}$) /* Input: Set of neurons $C_{\text{less}}$ */
2:     Initialize an empty list rankedList
3:     for $n_i \in C_{\text{less}}$ do
4:         Calculate the rank score $R(n_i)$ based on $S_c$
5:         Append $(n_i, R(n_i))$ to rankedList
6:     end for
7:     Sort rankedList in ascending order of rank scores
8:     Return rankedList /* Output: Sorted list of neurons and their rank scores */
9:
10: end procedure
```

6.3 Evaluation

In this section, we present the results of our evaluation of R-MaS3N providing insights into its effectiveness, efficiency, and reliability. The evaluation encompasses SNN application mapping experiments to a 3D NoC-based neuromorphic system, an assessment of our mapping strategy's performance in the presence of randomly introduced faulty cluster neurons, and an analysis of the method's reliability across different NoC sizes. These evaluations collectively demonstrate R-MaS3N's robustness and its ability to handle fault scenarios within 3D NoC-based neuromorphic systems.

6.3.1 Evaluation Methodology

In the initial phase of our evaluation, we proposed a set of SNN applications each prefixed with '*MLP*' followed by a number (e.g., MLP_{1794}), denoting the total number of neurons as detailed in Table 6.2. In Table 6.2, columns 2 and 3, provide insights into the topology and neuron counts for these applications. We then map these proposed applications introducing random neuron faults at rates of 10%, 20%, 30%, and 40% per layer within the 3D NoC-based neuromorphic system. This mapping was executed across various NoC sizes ranging from $3 \times 3 \times 3$ to $5 \times 5 \times 5$ as specified in Table 6.1. The determination of the system size is contingent upon the size of the application. Additionally, we intend to assess the effectiveness of our proposed method in larger systems where we have identified a higher probability of faults occurring. Unlike our previous methodologies in [19] and [62] where fault insertion is applied uniformly to the entire 3D NoC-based neuromorphic system, the novel approach here ensures an even distribution of faults across individual layers. In the subsequent phase of our evaluation, we assessed the efficiency of R-MaS3N's output neuron mapping and neuron utilization behaviors across various fault rates. This assessment encompassed NoC sizes ranging from $3 \times 3 \times 3$ to $5 \times 5 \times 5$ within the 3D NoC-based neuromorphic system having neuron cluster sizes of 128 and 256 neurons as outlined in Table 6.1. Furthermore, we assessed the computational cost incurred by the R-MaS3N mapping strategy in terms of execution time. This evaluation spanned critical stages of the mapping process, including the initial mapping, the introduction of neuron faults, fault detection, and the subsequent remapping of previously unmapped neurons. We also analyzed the time complexity associated with the neuron remapping phase within R-MaS3N. This is to enable a fair comparison of remapping times against our prior fault-tolerant mapping methodologies. To underscore the enhanced reliability offered by R-MaS3N in mitigating faults within a 3D NoC-based neuromorphic system, even in the presence of faults, we conducted a thorough reliability analysis. In this comprehensive assessment, our focal point was the Mean Time To Failure (MTTF), which represents the average duration until system failure, particularly under the specified fault rates detailed in Table 6.1.

Table 6.1: Configuration used for evaluating R-MaS3N.

Parameter	Value
Neurons per cluster	128 and 256
NoC sizes	$3 \times 3 \times 3$, $4 \times 4 \times 4$ and $5 \times 5 \times 5$
Fault rates (%)	10, 20, 30, and 40

Table 6.2: Applications used for evaluating R-MaS3N

App.	Topology	Neurons	App. ID
MLP	`784_1000_10`	1,794	`MLP_1794`
	`784_2000_10`	2,794	`MLP_2794`
	`784_2000_2000_10`	4,794	`MLP_4794`
	`784_4000_4000_10`	8,794	`MLP_8794`
	`784_3000_3000_3000_10`	9,794	`MLP_9794`
	`784_6000_6000_6000_10`	18,794	`MLP_18794`

**All the applications are feedforward connections.*
**The total number of neurons for any topology is the sum of all neurons from the input layer to the last layer in the topology.*

6.3.2 Evaluation Results

Mapping efficiency

In the evaluation of R-MaS3N's mapping efficiency as demonstrated in Figures 6.6 - 6.11, it becomes evident that with an increasing rate of designated faulty neurons, there is a corresponding rise in the count of unmapped application neurons before any repair actions (*UA_BR*). This trend persists as long as the fault rate continues to escalate. However, following the remapping process, which targets underutilized and most utilized neurons within a designated region of the 3D NoC-based neuromorphic system (*UN_NS*) during the fault-tolerant phase of R-MaS3N, the count of unmapped application neurons after repairs (*UA_AR*) reaches zero. This outcome holds for all the SNN applications proposed in Table 6.2.

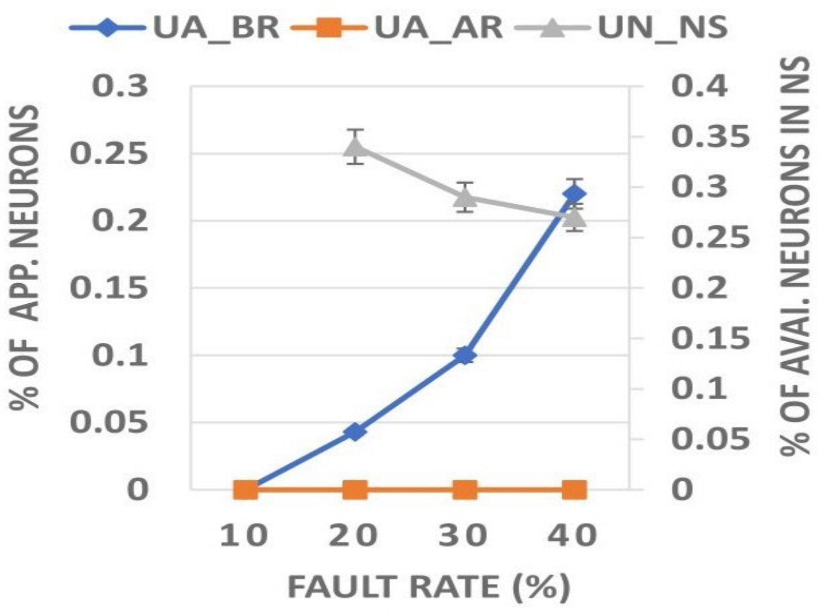


Figure 6.6: Behavior of output neuron remapping across different fault rates within a $3 \times 3 \times 3$ NoC-based neuromorphic system for *MLP*_1794.

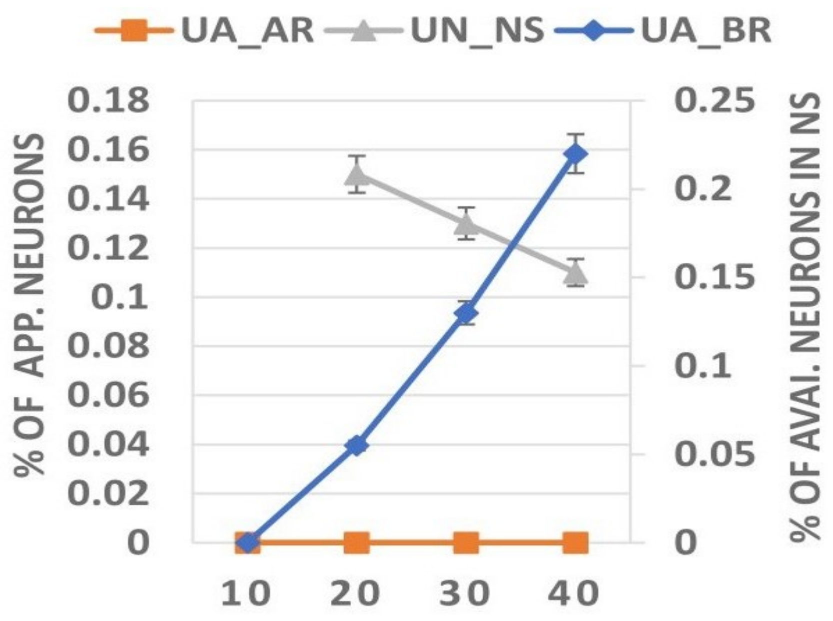


Figure 6.7: Behavior of output neuron remapping across different fault rates within a $3 \times 3 \times 3$ NoC-based neuromorphic system for *MLP*_2794.

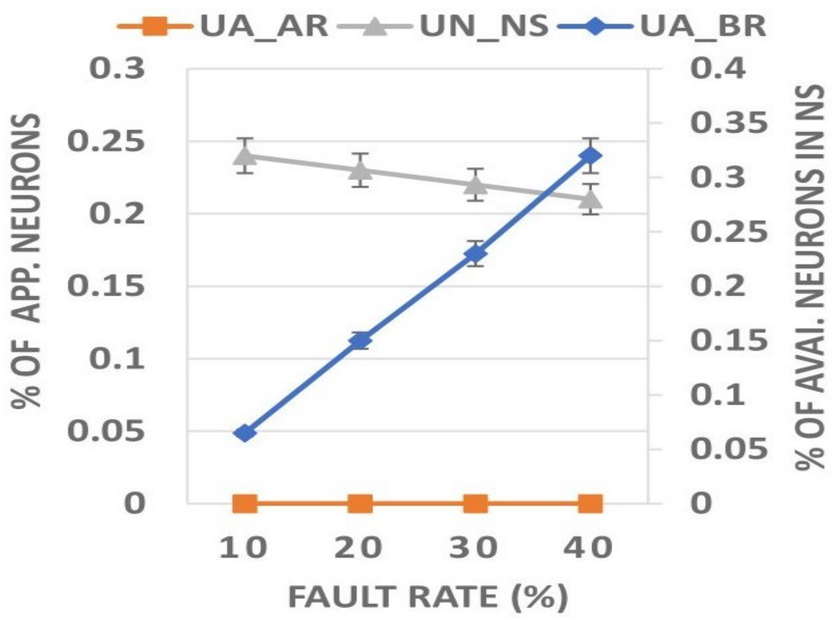


Figure 6.8: Behavior of output neuron remapping across different fault rates within a $4 \times 4 \times 4$ NoC-based neuromorphic system for *MLP*_4794.

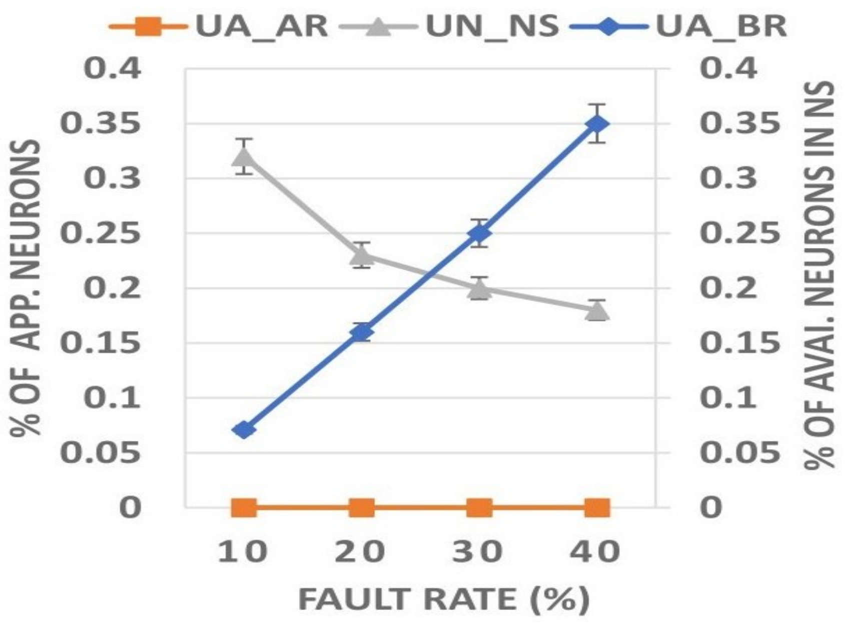


Figure 6.9: Behavior of output neuron remapping across different fault rates within a $4 \times 4 \times 4$ NoC-based neuromorphic system for *MLP*_8794.

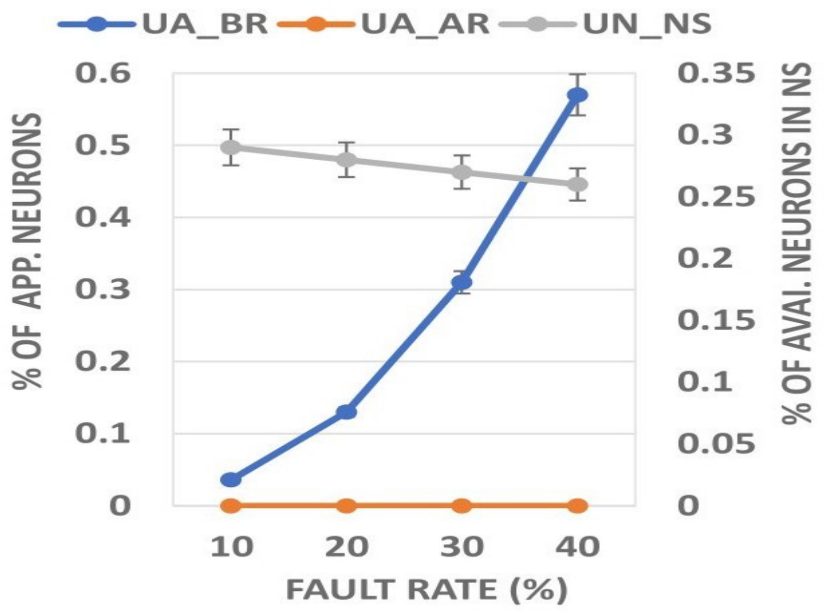


Figure 6.10: Behavior of output neuron remapping across different fault rates within a $5 \times 5 \times 5$ NoC-based neuromorphic system for *MLP*_9794.

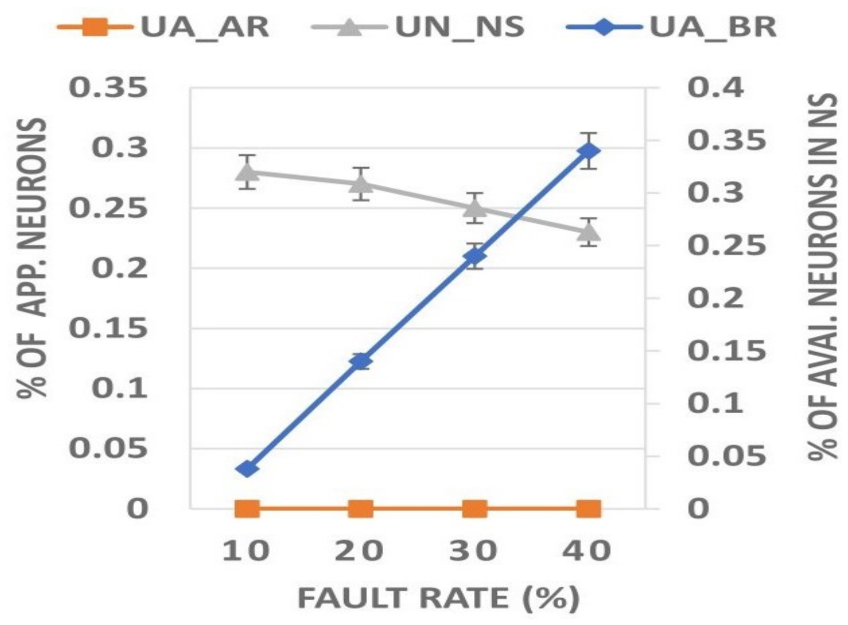


Figure 6.11: Behavior of output neuron remapping across different fault rates within a $5 \times 5 \times 5$ NoC-based neuromorphic system for *MLP*_18794.

Mapping cost

The evaluation of R-MaS3N mapping cost focused on measuring the time required for key stages during the mapping process of SNN applications onto a 3D NoC-based neuromorphic system. This evaluation considered NoC sizes ranging from $3 \times 3 \times 3$ to $5 \times 5 \times 5$. The key stages encompass initial mapping, repeated initial mapping after introducing faults into some neurons, and subsequent remapping of the unmapped neurons. We quantified the computational effort involved in the robust mapping of SNN application neurons to clusters within the 3D NoC-based neuromorphic system. As shown in Figures 6.12 - 6.14, the R-MaS3N computational cost consistently remained below 10 seconds even when dealing with the mapping of a large-scale SNN such as MLP_{18794} onto a $5 \times 5 \times 5$ NoC-based neuromorphic system with 40% of its neurons designated as faulty. This analysis demonstrates the efficiency of R-MaS3N in managing computational costs across various stages of the mapping process.

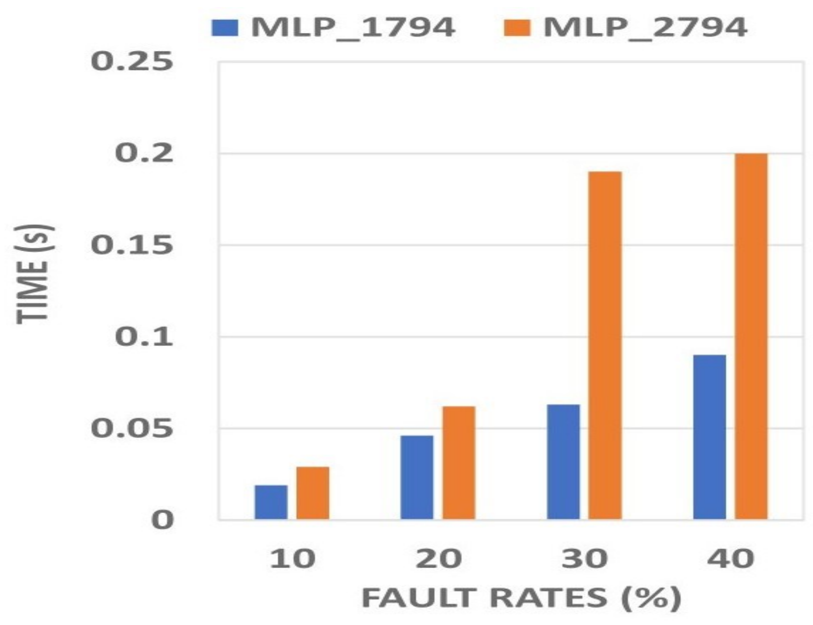


Figure 6.12: Plot illustrating applications mapping cost for a $3 \times 3 \times 3$ NoC-based neuromorphic system under various fault rates.

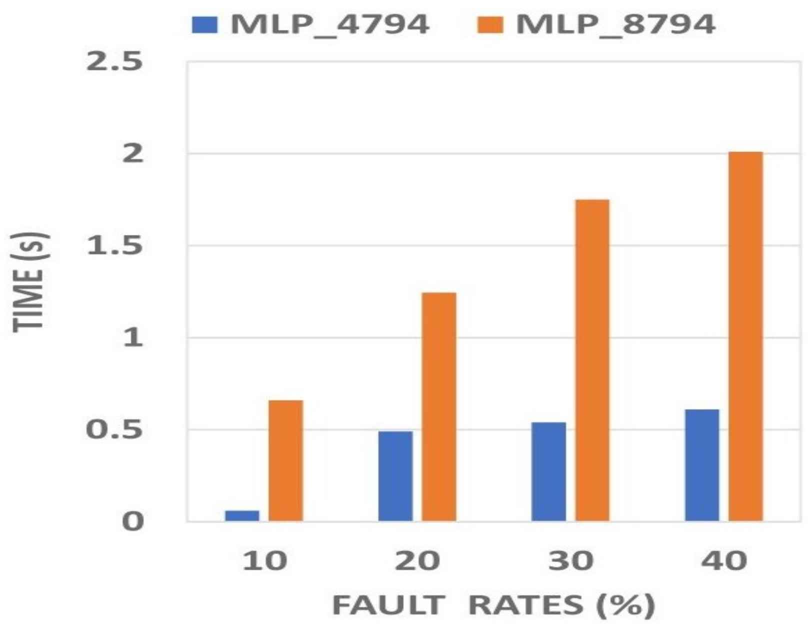


Figure 6.13: Plot illustrating applications mapping cost for a $4 \times 4 \times 4$ NoC-based neuromorphic system under various fault rates.

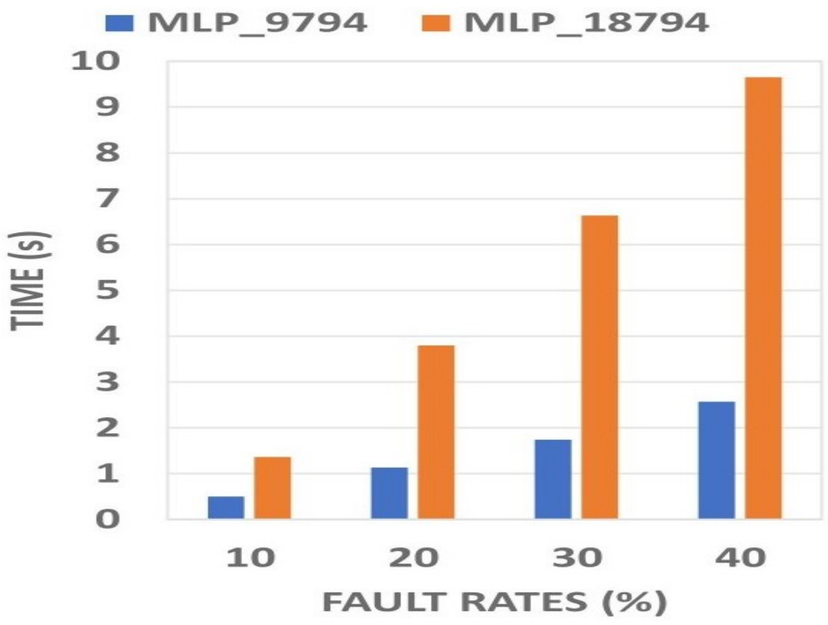


Figure 6.14: Plot illustrating applications mapping cost for a $5 \times 5 \times 5$ NoC-based neuromorphic system under various fault rates.

Neuron utilization behavior

Another crucial aspect of analyzing SNN applications mapped with R-MaS3N is to evaluate whether neuron utilization can be effectively restored in the presence of faults impacting the neuromorphic system during computations. Figures 6.15 - 6.17 depict the utilization dynamics before, during, and after the introduction of faults. Initially, without faults after the initial mapping (IM), the 3D NoC-based neuromorphic system attains optimal utilization of application neurons. However, as the number of faulty neurons increases ($F\_10 - F\_40$), neuron utilization experiences a gradual decline. Encouragingly, the system regains its full operational capacity after remapping (AR). This analysis sheds light on the adaptability and resilience of the system, showcasing its ability to recover optimal neuron utilization even in the presence of faults.

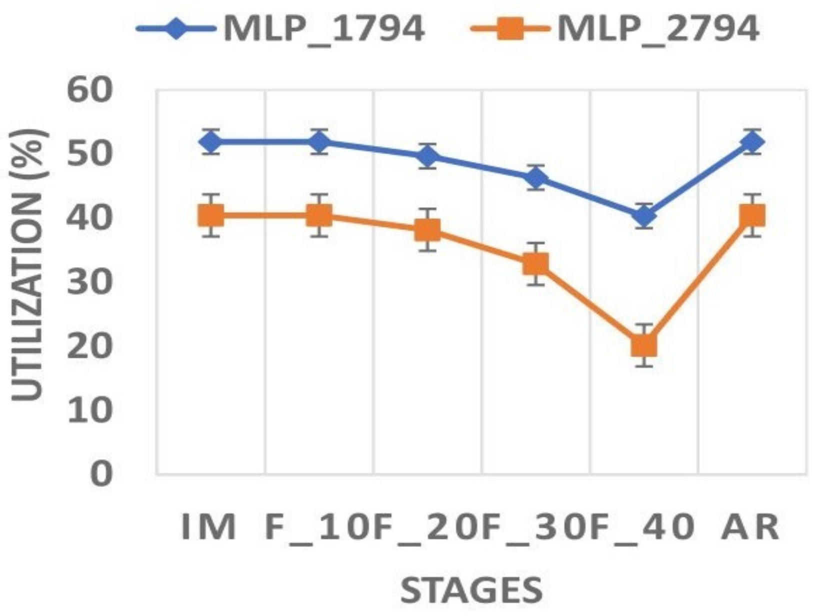


Figure 6.15: Plot illustrating neuron utilization across various stages of the mapping method for a $3 \times 3 \times 3$ 3D NoC-based neuromorphic system.

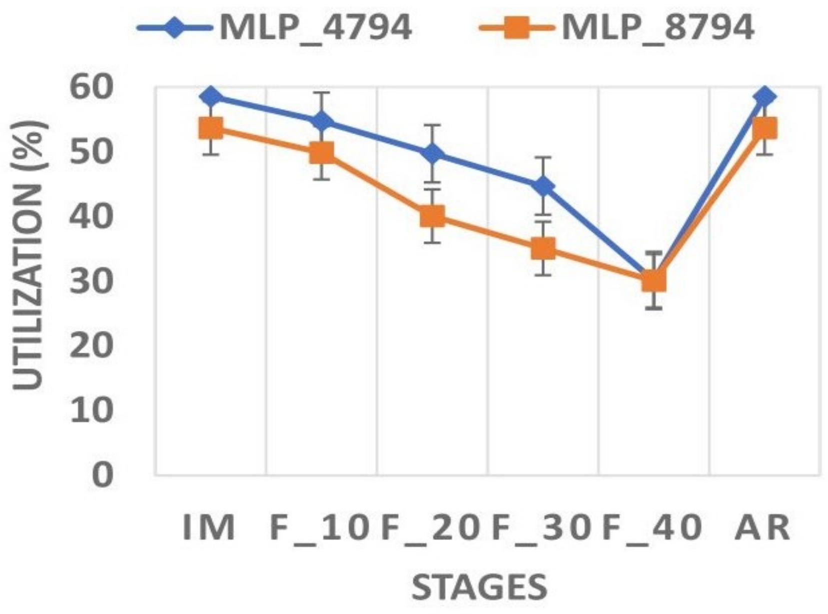


Figure 6.16: Plot illustrating neuron utilization across various stages of the mapping method for a $4 \times 4 \times 4$ 3D NoC-based neuromorphic system.

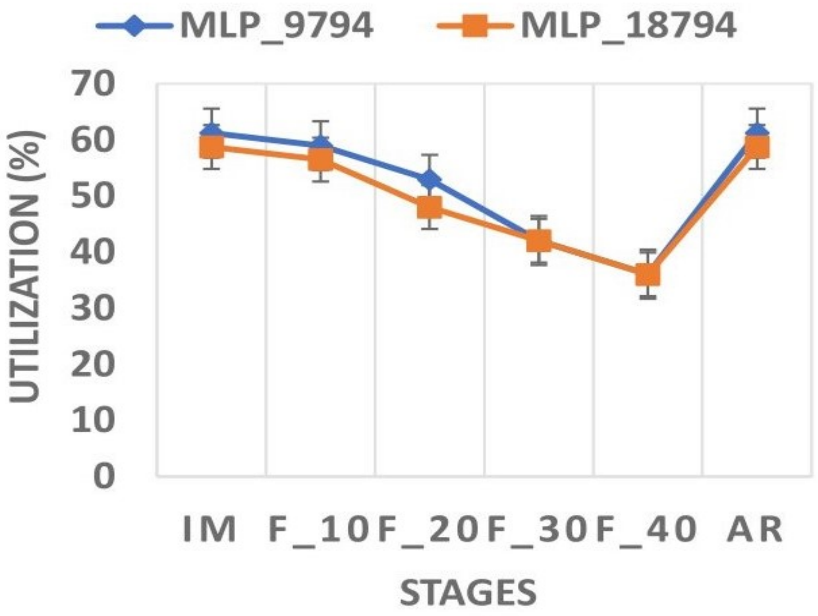


Figure 6.17: Plot illustrating neuron utilization across various stages of the mapping method for a $5 \times 5 \times 5$ 3D NoC-based neuromorphic system.

Time complexity

For the time complexity analysis, the SLP algorithm in Algorithm takes a set of unmapped layer neurons, denoted as N, as input and categorizes them into two partitions: C_{most} and C_{least}. The SLP algorithm operates in linear time as it iterates over each neuron $n_i \in N$ and performs averaging and comparisons to determine the appropriate partition placement for each neuron. Consequently, the time complexity of the SLP algorithm is $O(N)$, where N represents the number of neurons. Next, we consider the NP algorithm in Algorithm which partitions the set of mapped neurons, denoted as M_n, within the 3D NoC-based neuromorphic system. Similar to the SLP algorithm, the NP algorithm exhibits linear time complexity. It involves iterating through each neuron $n_i \in M_n$, where each iteration includes calculations to determine each neuron's connectivity and activity pattern. Thus, the time complexity for this process is $O(M_n)$, where M_n represents the number of mapped neurons. The efficiency of the partitioning process is evident in its linear time complexity, making it suitable for handling a large set of neurons. To assess the time complexity of remapping unmapped layer neurons (N) to existing neurons layer by layer, we observe that it incurs a time complexity of $O(N^2)$. Consequently, the overall execution time (E_t) for neuron remapping is determined by combining the SLP and NP algorithms' time complexities and the layer-to-layer remapping process. This relationship can be expressed mathematically as shown in Equation 6.7.

When determining the execution time for remapping or repairing unmapped application neurons in a neuromorphic system, the time needed to remap these neurons to existing ones is considered. Table 6.3 provides an evaluation of the remapping time for the R-MaS3N remapping step, considering the SNN applications outlined in Table 6.2 with 40% of the neurons in a neuromorphic system gone faulty. For the 3D-NoC-based neuromorphic system with the largest NoC size ($5 \times 5 \times 5$), the remapping process completes in under 10 seconds. A significant comparison arises with the GA-based remapping method used for the 3D-NoC-based neuromorphic system with the smallest NoC size ($4 \times 4 \times 4$), as discussed in [19]. In this comparison, the R-MaS3N remapping algorithm exhibits a remarkable 71× reduction in remapping time, showcasing its scalability and efficiency in handling larger NoC sizes.

$$E_{mt} = O(N) + O(M_n) + O(N^2) \tag{6.7}$$

Table 6.3: R-MaS3N remapping time in the 3D-NoC-based neuromorphic system for different SNN applications.

Network size	SNN Configuration	Time (s)
$3 \times 3 \times 3$	`784_1000_10`	0.06
	`784_2000_10`	0.18
$4 \times 4 \times 4$	`784_2000_2000_10`	0.58
	`784_4000_4000_10`	1.97
$5 \times 5 \times 5$	`784_3000_3000_3000_10`	2.54
	`784_6000_6000_6000_10`	9.60

**The mapping time takes into account the time:*

- *To perform layer partitioning of unmapped neurons in the layer(s) of an SNN application.*
- *To perform neuron partitioning of neurons of the neuromorphic system.*
- *To perform neuron remapping for the highest fault rate.*

RELIABILITY ASSESSMENT: MEAN TIME TO FAILURE (MTTF)

The Mean Time To Failure (MTTF) is a crucial metric representing the expected duration until a system fails. In the case of a k-fault-tolerant system capable of withstanding up to k faults, the MTTF computation involves the failure rates of individual components and the inherent fault-tolerant structure of the system. Let $MTTF_s$ denote the MTTF for the entire system and considering the reciprocal of the overall system failure rate, $MTTF_s$ is defined by the equation:

$$MTTF_s = \frac{1}{\lambda_s} \tag{6.8}$$

where λ_s is the overall failure rate of the k-fault-tolerant system. The overall failure rate is the sum of the failure rates of the individual components taking into account the fault tolerance. Assuming that the failures are independent, we can express λ_s as:

$$\lambda_s = \sum_{i=1}^{k} \lambda_i \tag{6.9}$$

Here, λ_i is the failure rate of the i-th component. Therefore, the MTTF for a k-fault-tolerant system is the reciprocal of λ_s:

$$MTTF_s = \frac{1}{\sum_{i=1}^{k} \lambda_i} \tag{6.10}$$

Equation 6.10 therefore provides the mean time to failure for the entire system considering the fault-tolerant structure. Incorporating time duration (t), the equation becomes:

$$MTTF_s(t) = \frac{1}{\sum_{i=1}^{k} \lambda_i} \tag{6.11}$$

The expression Equation 6.11 therefore represents the mean time to failure of the k-fault-tolerant system over a specified time duration t. However, R-MaS3N has two distinct failure rates: the initial mapping failure rate (IFR) and the remap failure rate (RFR). If the remapping process encounters failure, the RFR is set to 1, as expressed in Equation 6.12.

$$RFR = \begin{cases} 1, & \text{if } U > 2 \times M_{C_{less}} \\ 0 & \text{otherwise} \end{cases} \tag{6.12}$$

Here, U signifies the count of unmapped application neurons, and $M_{C_{less}}$ denotes the number of mapped neurons within the C_{less} region of the 3D NoC-based neuromorphic system. Given that our mapping method adheres to the specified constraints during mapping, it can effectively handle faults while remaining operational. The Mean Time To Failure (MTTF) for a k fault-tolerant system from Equation 6.11 and as in [19] for the R-MaS3N is described by Equation 6.13:

$$\text{MTTF}_{RS} = \left(\frac{1}{\lambda \cdot \left(\frac{k}{p}\right)} \right) \cdot t \tag{6.13}$$

In Equation 6.13 above, λ represents the initial mapping failure rate per neuron (in failures per hour), k denotes the number of neurons to be mapped, p signifies the number of available neurons for remapping, and t represents the time unit.

Figures 6.18 - 6.20 illustrate the Mean Time to Failure (MTTF) of R-MaS3N in mapping the SNN applications outlined in Table 2 onto a 3D-NoC-based neuromorphic system with varying NoC sizes. With 40% of neurons of a neuromorphic system gone faulty, R-MaS3N achieves MTTF values of 50.73 years, 35.36 years, and 26.54 years for NoC sizes of $3 \times 3 \times 3$, $4 \times 4 \times 4$, and $5 \times 5 \times 5$, respectively. Notably, the MTTF of R-MaS3N for the $5 \times 5 \times 5$ NoC configuration at a 40% fault rate surpasses the MTTF of the previous method at a 20% fault rate for a $4 \times 4 \times 4$ NoC size by 16%. This significant improvement highlights the enhanced reliability of R-MaS3N compared to our previous fault-tolerant mapping approach.

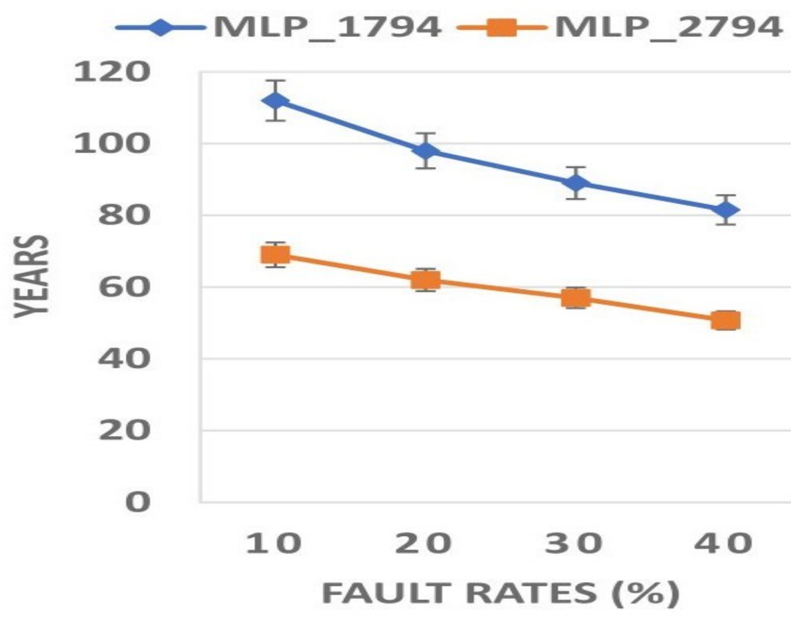


Figure 6.18: The mean time to failure (MTTF) of R-MaS3N when MLP_{1794} and MLP_{2794} applications are mapped onto a $3 \times 3 \times 3$ NoC-based neuromorphic system.

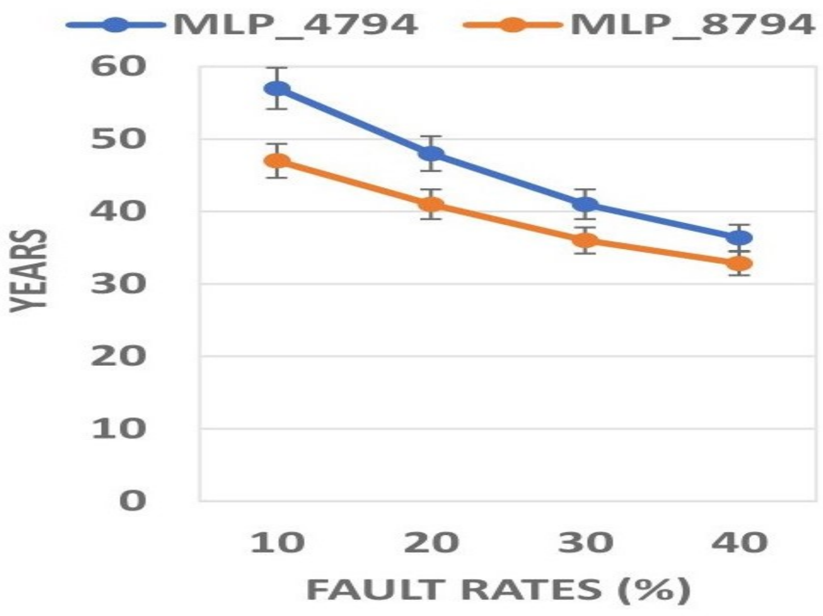


Figure 6.19: The mean time to failure (MTTF) of R-MaS3N when MLP_{4794} and MLP_{8794} applications are mapped onto a $3 \times 3 \times 3$ NoC-based neuromorphic system.

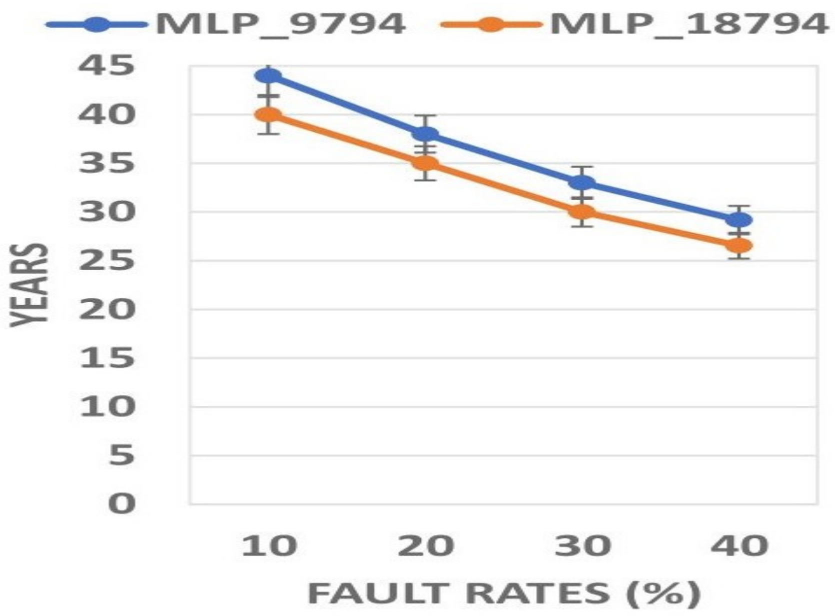


Figure 6.20: The mean time to failure (MTTF) of R-MaS3N when MLP_{9794} and MLP_{18794} applications are mapped onto a $3 \times 3 \times 3$ NoC-based neuromorphic system.

6.4 Chapter Summary

In this chapter, we introduced a novel mapping approach, R-MaS3N designed to map SNNs onto a 3D-NoC-based neuromorphic system and remap if there exist faulty components in the hardware by re-purposing existing neurons to achieve fault tolerance. Our evaluation involved assessing its performance under various scenarios including situations where a portion of cluster neurons was randomly designated as faulty. We conducted a thorough analysis of our mapping method's reliability highlighting its significantly improved reliability compared to our previous fault-tolerant mapping algorithm. The next chapter presents the implementation of the proposed algorithms and architectures focusing on enhancing robustness in the realization of a reconfigurable neuromorphic architecture.

7

TOWARDS A ROBUST RECONFIGURABLE NEUROMORPHIC ARCHITECTURE

This chapter details the implementation of the proposed algorithms and schemes aimed at enhancing robustness in the realization of a reconfigurable neuromorphic architecture. The neuromorphic chip is structured as an interconnection of neural tiles through a NoC interconnect. Each neural tile consists of two primary components: the processing unit and the information communication unit. Focusing on robustness within the processing unit, we provide a comprehensive description of this unit, highlighting a key area that supports our proposed robust methodologies. Additionally, we introduce a sample layout for a 2×2 NoC-based neuromorphic system, currently supporting the migration-based mapping method with the novel RSM mechanism.

7.1 System Architecture

A high-level view of the robust neuromorphic architecture can be seen in Figure 7.1 having three layers. It is based on several layers of 2D spiking neural tiles building upon our earlier 3D-NoC design in [5]. In these interconnected stacked layers, each neural tile employs an array of spiked neurons to process incoming spikes. In a broader system, these 2D neural tiles are often arranged in a hierarchical and scalable manner enabling the implementation of large-scale complex AI applications.

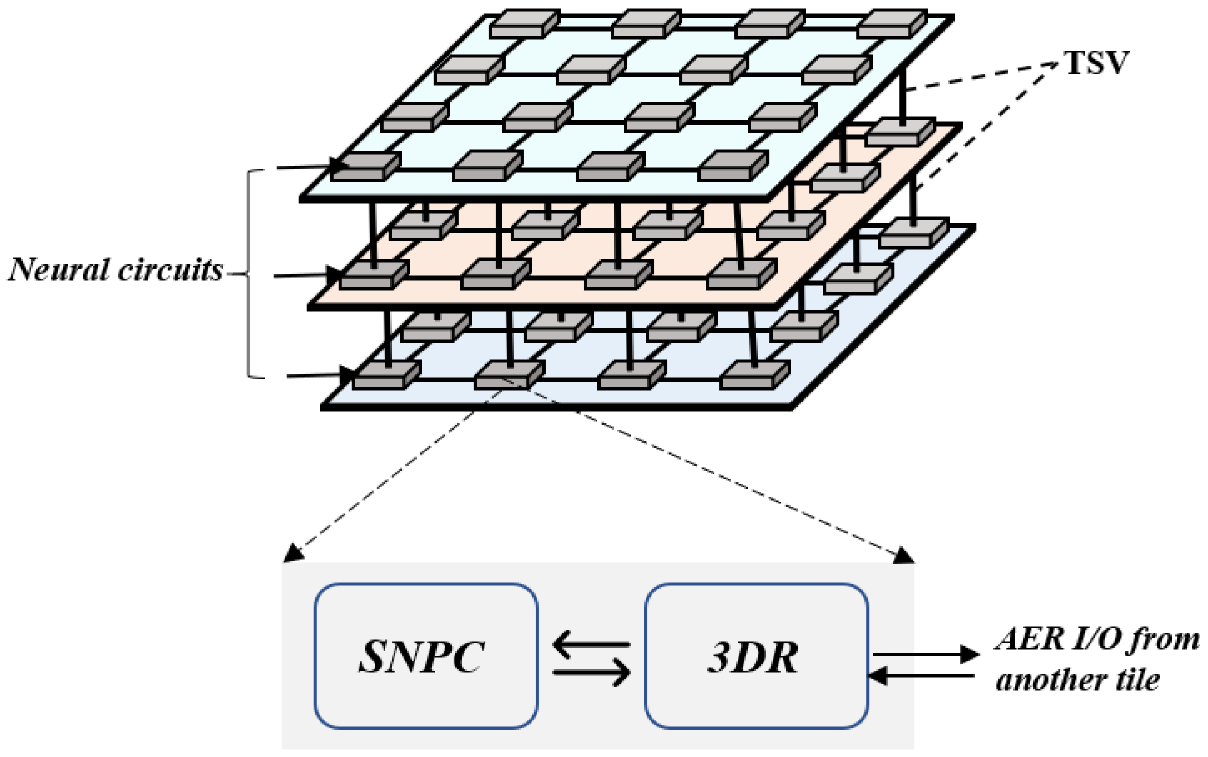


Figure 7.1: High-level view of the neuromorphic architecture.

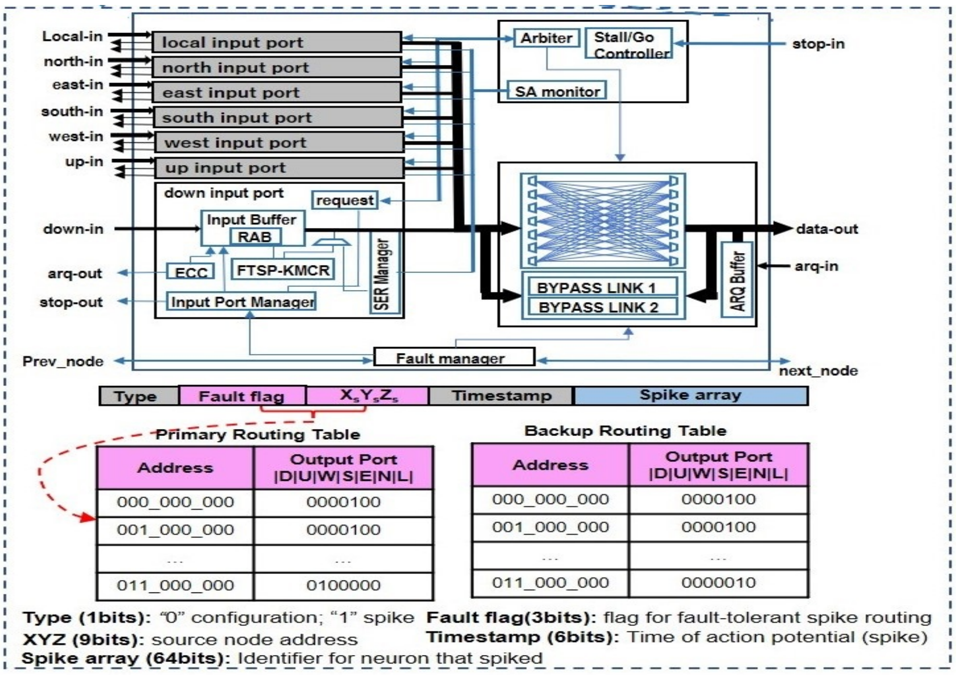
External data is input into the system through a host PC. Since the system employs a 3D-mesh topology with each neural tile equipped with a node (i.e. processing core) and a 3D router as shown in Figure 7.1 that facilitates communication in six directions and a processing core, communication is managed by a 3D-mesh NoC while computation is executed by a node having a processing core controller overseeing all operational processes.

7.2 Neural Tile

There are two main components of the neural tile: a spiking neural processing core (SNPC) and a multicast 3D router. In SNNs, each spiking neuron typically corresponds to a single SNPC and inter-neuron connectivity is established by transmitting spikes (packets) over the on-chip interconnection. The 3D router, through its routing mechanism, facilitates effective communication among the spiking neurons in the SNPC, enabling the exchange of neural information. Interaction between the SNPC and multicast 3D router is instrumental in the functioning of neural tiles, playing a crucial role in the transmission of packets for neural computations.

7.2.1 3D Multi-Cast Router

The 3D FT-router shown in Figure 7.2, utilizes the K-means multicast routing (KMCR) algorithm for spike distribution [22]. To address congestion, a variant called shortest path k-means clustering routing (SP-KMCR) has been implemented [22]. Introducing fault tolerance, the fault-tolerant shortest path k-means clustering routing (FTSP-KMCR) ensures reliable spike data delivery even in the presence of route faults. For fault handling in input buffers and crossbars during packet forwarding, the router integrates a random-access buffer (RAB) and a Bypass-on-demand link [5] [22]. With seven input and output ports, four designated for intra-layer connections, the router's routing process involves buffer writing (BW), routing calculation (RC), switch allocation (SA), and crossbar traversal (CT) [5] [19] [22] [6].



Primary Routing Table

Address	Output Port |D|U|W|S|E|N|L|
000_000_000	0000100
001_000_000	0000100
...	...
011_000_000	0100000

Backup Routing Table

Address	Output Port |D|U|W|S|E|N|L|
000_000_000	0000100
001_000_000	0000100
...	...
011_000_000	0000010

Type (1bits): "0" configuration; "1" spike **Fault flag(3bits):** flag for fault-tolerant spike routing
XYZ (9bits): source node address **Timestamp (6bits):** Time of action potential (spike)
Spike array (64bits): Identifier for neuron that spiked

Figure 7.2: High level view of the 3D multicast router [6].

7.2.2 Spiking Neuro Processing Core Architecture

Figure 7.3 shows the SNPC high level view with crucial components including the LIF array, synapse memory, synaptic crossbar, network interface (NI), control unit, and STDP learning module. The LIF array computes a neuron's membrane voltage by accumulating synapse values with a leak value for synaptic decay. Output spikes are fired if the accumulated value surpasses a voltage threshold; otherwise, no spike is generated. The synaptic crossbar, using a crossbar architecture, represents synaptic connections with 1-bit values, and weights are stored in synapse memory. Presynaptic spikes reach the SNPC and postsynaptic spikes are identified based on synapses [5]. Weights from synapse memory are transmitted to the LIF neuron for accumulation. For learning, the STDP module updates synaptic weights via trace-based STDP learning using 16 presynaptic spikes grouped by their arrival time relative to a postsynaptic spike [5] [22]. During learning, presynaptic spikes arriving before the postsynaptic spike are incremented, while those arriving afterward are decremented. The control unit manages SNPC operations through six states: "idle," "download," "accumulation," "leak," "threshold," and "STDP." In the "idle" state, the SNPC awaits presynaptic spikes; upon arrival, the "download" state initiates download spikes. Subsequently, the "accumulation" state weights and sends spikes to LIF neurons. The "leak" state allows membrane potential leakage, followed by a comparison against a threshold in the "threshold" state. If the threshold is exceeded, an output spike is fired. The "STDP" state activates the learning module. Learning occurs if conditions are met, prompting a reset to the idle state; otherwise, the SNPC returns to "idle.

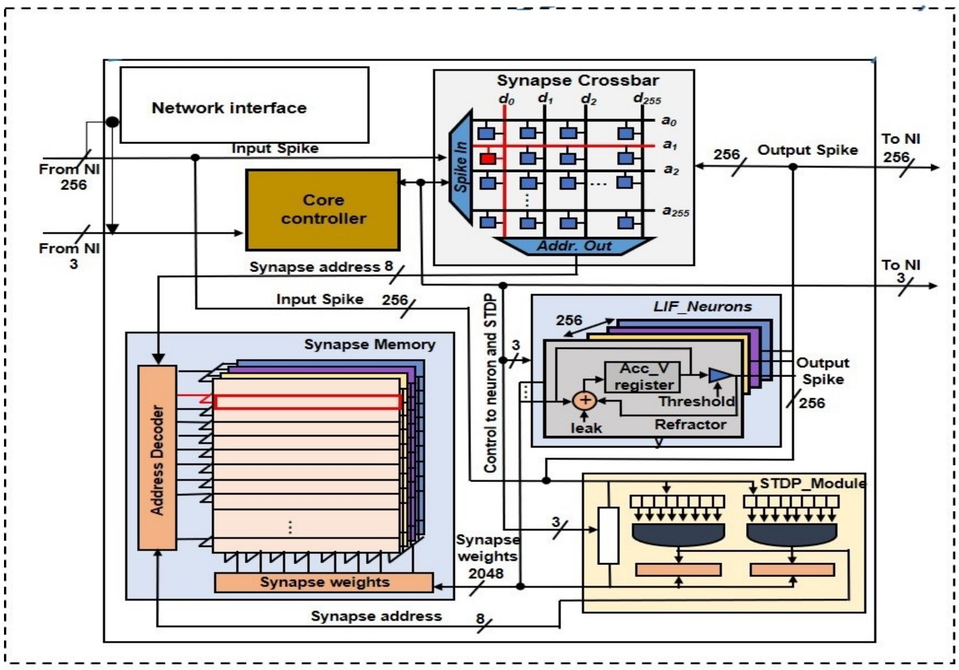


Figure 7.3: High-level view of the spiking neuro processing core [6].

Network Interface (NI)

The NI is a crucial component of the SNPC, incorporating the mapping method. Furthermore, the NI supports both single and burst transaction modes for the efficient reading and writing of weight memory and neuron parameters [6]. Additionally, the NI facilitates communication between neurons using the on-chip network framework, employing an encoder and decoder. On the algorithm side, the NI aligns with the mapping method specified in [5] and [22]. This integration is extended with the introduction of two fault-tolerant mapping methods illustrated in Figure 7.4 and Figure 7.5. These methods are carefully designed considering the routing mechanism within the router while preserving the mapping methodology proposed in our previous works. The system in [6] and [19] supports the MigSpike mapping method, lever-

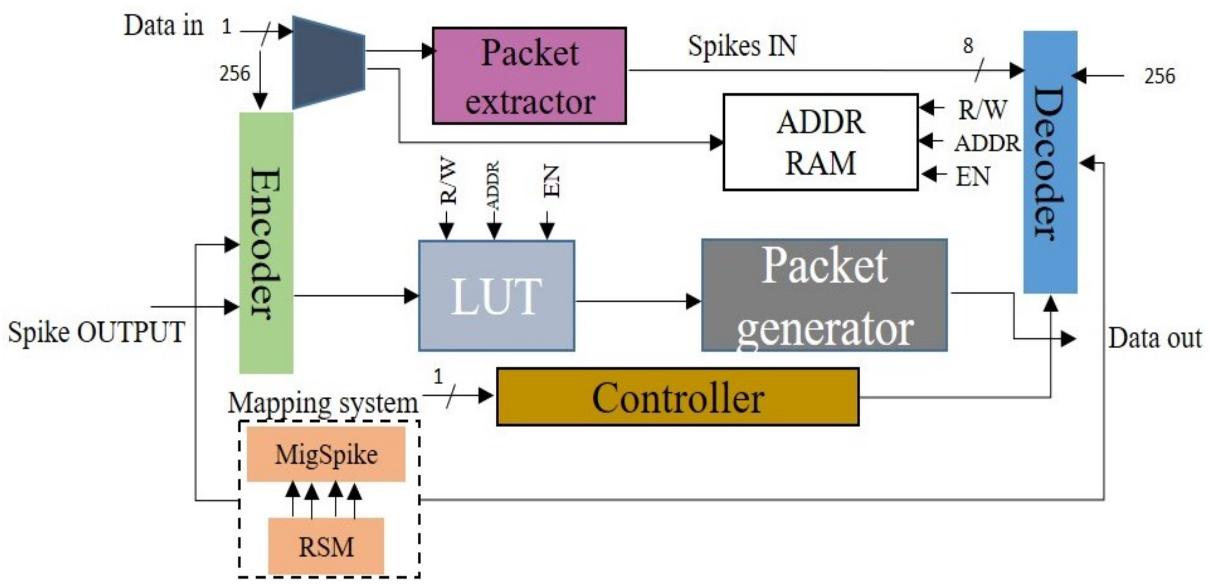


Figure 7.4: A block diagram of the network interface supporting the MigSpike + RSM mapping method.

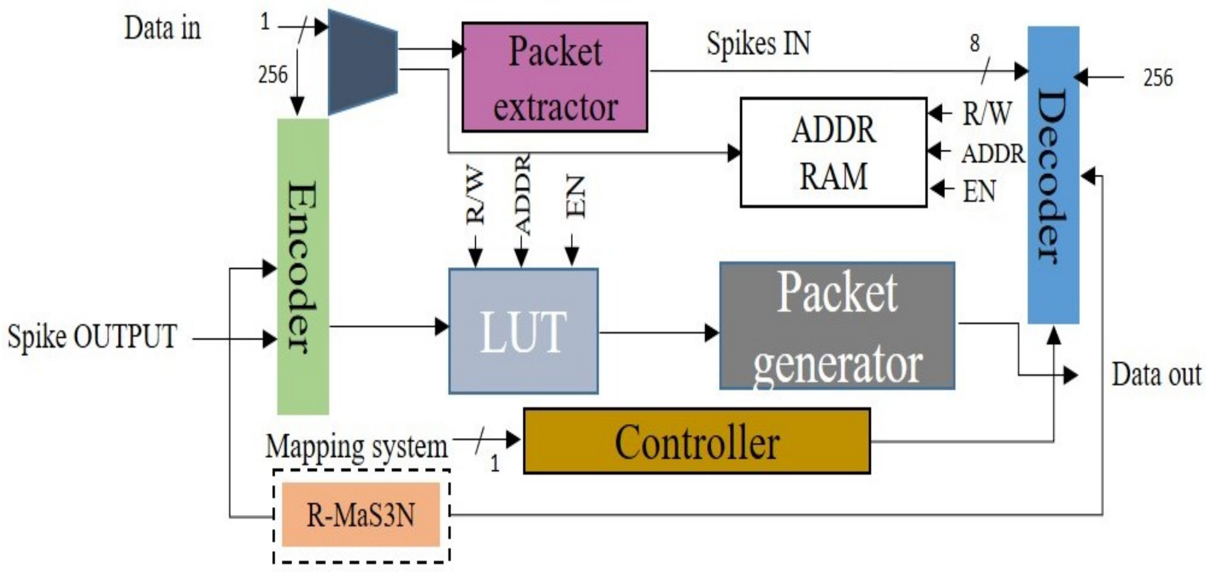


Figure 7.5: A block diagram of the network interface supporting the R-MaS3N mapping method.

aging enhanced migration methods and hardware mechanisms for fault-tolerant task migration. When a fault occurs in a neuron, MigSpike migrates the task to a spare neuron by creating chains of migrations within a cluster or across the entire system. Despite its effectiveness, MigSpike has drawbacks, prompting the introduction of the RSM mechanism in the NI ash shown in Figure 7.4. RSM ensures efficient task migration from faulty neurons to spares, albeit relying on redundant finite elements. To address this limitation, R-MaS3N was later proposed and introduced into the NI as shown in Figure 7.5, eliminating the finite resource bottleneck associated with MigSpike and enhancing the efficiency of task mapping.

7.3 Evaluation methodology

The hardware implementation involved designing the proposed system using Verilog-HDL. Functional simulation was carried out through ModelSim, Synopsys Design Compiler facilitated synthesis, and Cadence Innovus was employed for layout design, utilizing the OpenRAM [5] and NANGATE 45*nm* [19] library. The FreePDK3D [6] played a crucial role in incorporating vertical Through-Silicon-Vias.

7.4 Evaluation results

Table 7.1 presents the hardware complexity of the proposed robust reconfigurable architecture, featuring a network interface currently supporting the MigSpike mapping method integrated with RSM. As indicated in Table 7.1, the NI incorporating the mapping technique occupies 15.36% of the tile area and consumes 7% of the proposed system's power. Figure 7.6 illustrates a sample layout for a 2×2 NoC-based SNN layer with mapping framework support. Each cluster within the layout comprises 256 spike inputs in AER format, and the crossbar is implemented using a 256-bank 8-bit dual-port SRAM through OpenRAM. The clock frequency employed in the design layout is 142MHz. We have kept this value consistent with our previous assumption in [5] that all 256 neurons in the spiking core spike at an identical rate. This assumption leads to a scenario where 256 synaptic operations can occur within a single clock cycle.

Table 7.1: Hardware complexity of the proposed robust reconfigurable system.

Module	Parameters	
	Total area ($\mu\ m^2$)	Power (mW)
Network interface (EN & DE)	15,445	0.93
Neuron cores(s)	64,576	10.64
3D NoC router	16,786	1.27

**EN: Encoder, DE: Decoder*

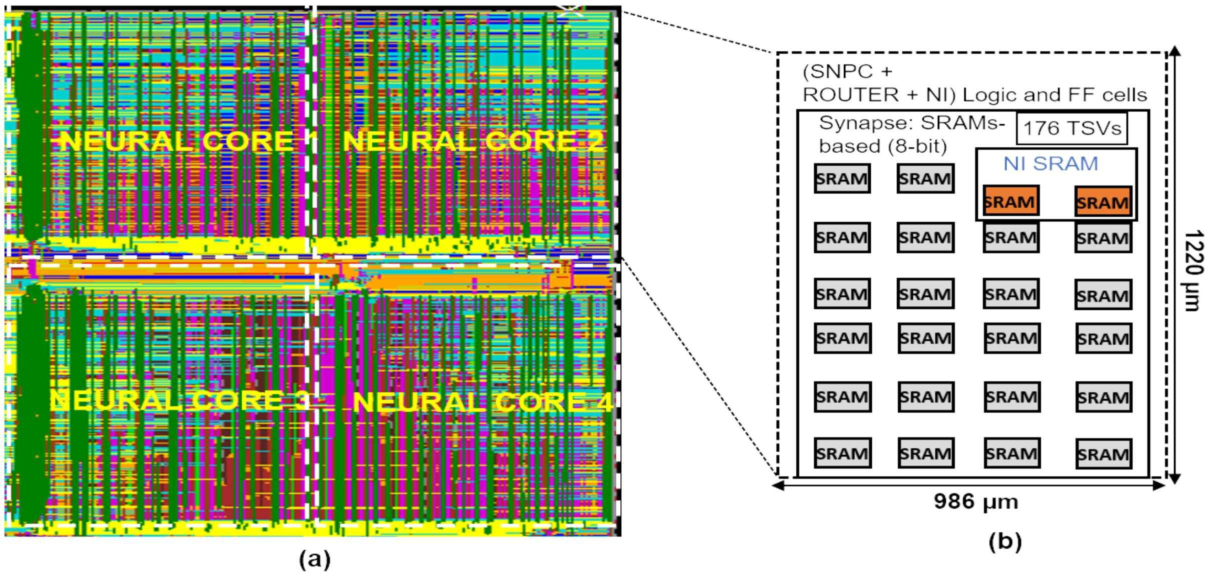


Figure 7.6: Hardware physical design layout of the proposed robust reconfigurable neuromorphic system for a 2×2 NoC size: (a) Layout (b) Schematic layout comprising 256 neuron logic cells and 65K synapses for the crossbar.

7.5 Conclusion

In conclusion, this chapter provides an overview of the architecture, hardware design, and assessment of the proposed robust reconfigurable 3D-NoC-based neuromorphic system. The system capitalizes on fault-tolerant mapping methods inspired by the fault-tolerant techniques found in the human brain. These methods enhance the efficiency of application mapping, enabling effective adaptation to system-wide faults thereby ensuring robust performance even in the presence of significant faults in the system.

8

DISSERTATION SUMMARY AND FUTURE OUTLOOK

8.1 Contributions Summary

Neuromorphic systems have demonstrated exceptional performance across various domains. However, ensuring their fault tolerance and ability to recover from errors has been a significant area of interest. Furthermore, neglecting the reliability concerns can result in a range of consequences from unreliable neural computation outputs to system-wide failures thereby posing significant challenges to their practical deployment. To address these concerns, we have developed neuromorphic algorithms serving two key purposes:

1. **Recovery from fault impact in neural computation:**

   Our first set of algorithms is dedicated to identifying, isolating, and recovering from faults after neural computation processes. Taking inspiration from the brain's remarkable adaptability and recovery capabilities, our proposed methods explore innovative techniques for fault recovery in spiking neural networks. The first method shares common goals with established concepts like neuron dropout; however, it distinguishes itself by adopting a pruning-based approach within an augmented framework. This novel approach selectively identifies and removes only faulty neurons to facilitate fault recovery. However, this initial method proved computationally intensive and the classification accuracy of a faulty model after recovery declines as a result of removing multiple faulty neurons some of which were erroneously identified as faulty. To address these drawbacks, we introduced a target and selection method that ensures only genuinely faulty neurons are chosen for removal. This method achieves nearly pre-fault-level classification accuracy after recovery in faulty models.

2. **Fault-tolerant mapping of neural applications to neuromorphic systems:**

   Mapping applications onto neuromorphic hardware in neuromorphic computing remains a challenging task with non-trivial solutions especially when neural circuits within neuromorphic systems may suffer from potential faults including high-rate faults. Our second set of algorithms focuses on ensuring fault-tolerant mapping of neural applications onto neuromorphic hardware. Building on our prior work of migration-based mapping for fault recovery, we incorporate a novel ranking and selection mechanism during the mapping process to dynamically select faulty neurons. This method ensures the overall mapping process is effective even in the presence of high-rate faults. Furthermore, this enhancement has increased the MTTF of the migration-based mapping on different NoC sizes by an average of 43%. However, this method relies on redundancies a concept inspired by the robust and fault-tolerant nature of the human brain. It is essential to acknowledge that when translated into a physical system implementation, these redundancies become a finite resource. Furthermore, as neuromorphic systems continue to scale to meet growing demands for edge applications, fault tolerance using redundancies necessitates a growing number of redundancies. Furthermore, if redundant resources are exhausted due to simultaneous fault occurrences, reliability could potentially drop to zero. To address these challenges, we've proposed a robust mapping scheme that relies on reusing existing neurons based on neural reuse theory for fault tolerance. This approach not only significantly reduces fault repair or recovery time (71$\times$ less compared to the previous mapping) but also provides better MTTF than the previous mapping method.

8.2 FUTURE OUTLOOK

This dissertation makes significant contributions that can extend across various facets of society. In the realm of neural computations, particularly in applications such as autonomous vehicles and medical devices, the development of fault recovery methods is indispensable. Given the potential serious consequences of errors in these domains, ensuring the resilience of neural computations is paramount. Moreover, the application of fault-tolerant methods emerges as a critical factor in guaranteeing stability and reliability in the context of industrial automation, especially within complex manufacturing environments. As edge computing continues to gain prevalence, the imperative for fault-tolerant methods intensifies, particularly in scenarios with constrained computational resources. The emphasis on robustness in these settings underscores the dissertation's relevance and practical implications in addressing the evolving landscape of technological applications. While this dissertation has made significant strides in developing algorithms for robust and reliable neuromorphic computing, several promising avenues for further research and development can be explored. One noteworthy area of future research involves the establishment of standardized benchmarks tailored specifically to assess the robustness and reliability of neuromorphic computing systems. These benchmarks would enable researchers to rigorously evaluate the effectiveness of fault-tolerant algorithms across diverse hardware and software configurations. Furthermore, to enhance the credibility and comparability of proposed fault-tolerant algorithms, future work can focus on quantitative evaluations. This involves assessing the algorithms' performance against established benchmarks and metrics, providing valuable insights into their efficacy in mitigating faults. A critical aspect yet to be explored is the energy efficiency of fault-tolerant mechanisms in neuromorphic systems. Investigating the energy overhead associated with fault detection, isolation, and recovery can inform designers about trade-offs between robustness and energy consumption, paving the way for more sustainable implementations.

List of Publications

Major Journals

1. **W. Y. Yerima**, O. M. Ikechukwu, K. N. Dang and A. Ben Abdallah, "Fault-Tolerant Spiking Neural Network Mapping Algorithm and Architecture to 3D-NoC-Based Neuromorphic Systems," in *IEEE Access*, vol. 11, pp. 52429-52443, 2023, doi: 10.1109/ACCESS.2023.3278802.

2. **W. Y. Yerima**, K. N. Dang and A. B. Abdallah, "R-MaS3N: Robust Mapping of Spiking Neural Networks to 3D-NoC-Based Neuromorphic Systems for Enhanced Reliability," in *IEEE Access*, vol. 11, pp. 94664-94678, 2023, doi: 10.1109/ACCESS.2023.3311031.

Major Conference(s)

1. **W. Y. Yerima**, K. N. Dang and A. B. Abdallah, "Fault Recovery in Spiking Neural Networks Through Target and Selection of Faulty Neurons for 3D Spiking Neuromorphic Processors," *IEEE 6th International Conference on Knowledge Innovation and Invention (ICKII)*, Sapporo, Japan, 2023, pp. 136-141, doi: 10.1109/ICKII58656.2023.10332575.

Bibliography

[1] A. Mehonic and A. J. Kenyon, “Brain-inspired computing needs a master plan,” *Nature*, vol. 604, no. 7905, pp. 255–260, apr 2022. [Online]. Available: https://doi.org/10.1038%2Fs41586-021-04362-w

[2] Z. Yi, J. Lian, Q. Liu, H. Zhu, D. Liang, and J. Liu, “Learning rules in spiking neural networks: A survey,” *Neurocomputing*, vol. 531, pp. 163–179, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231223001662

[3] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, “Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations,” *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[4] N. Upadhyay, S. Joshi, and J. J. Yang, “Synaptic electronics and neuromorphic computing,” *Science China Information Sciences*, vol. 59, 06 2016.

[5] O. M. Ikechukwu, K. N. Dang, and A. B. Abdallah, “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.

[6] A. Ben Abdallah and K. N. Dang, “Toward robust cognitive 3d brain-inspired cross-paradigm system,” *Frontiers in Neuroscience*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2021.690208

[7] C. Schuman, S. Kulkarni, M. Parsa, J. Mitchell, P. Date, and B. Kay, “Opportunities for neuromorphic computing algorithms and applications,” *Nature Computational Science*, vol. 2, pp. 10–19, 01 2022.

[8] Z. Li, W. Tang, B. Zhang, Y. Rui, and X. Miao, “Emerging memristive neurons for neuromorphic computing and sensing,” *Science and Technology of Advanced Materials*, vol. 24, 03 2023.

[9] Y. Chen, H. H. Li, C. Wu, C. Song, S. Li, C. Min, H.-P. Cheng, W. Wen, and X. Liu, “Neuromorphic computing’s yesterday, today, and tomorrow – an evolutional view,” *Integration*, vol. 61, pp. 49–61, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926017304674

[10] J. Kim, J. Koo, T. Kim, and J.-J. Kim, “Efficient synapse memory structure for reconfigurable digital neuromorphic hardware,” *Frontiers in Neuroscience*, vol. 12, 2018. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2018.00829

[11] P. T. Pfeiffer Michael, “Deep learning with spiking neurons: Opportunities and challenges,” *Frontiers in Neuroscience*, vol. 12, 2018. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2018.00774

[12] W. Guo, M. E. Fouda, A. M. Eltawil, and K. N. Salama, “Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems,” *Frontiers in Neuroscience*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2021.638474

[13] K. Yamazaki, V.-K. Vo-Ho, D. Bulsara, and N. Le, “Spiking neural networks and their applications: A review,” *Brain Sciences*, vol. 12, no. 7, 2022. [Online]. Available: https://www.mdpi.com/2076-3425/12/7/863

[14] A. Taherkhani, A. Belatreche, Y. Li, G. Cosma, L. P. Maguire, and T. McGinnity, “A review of learning in biologically plausible spiking neural networks,” *Neural Networks*, vol. 122, pp. 253–272, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608019303181

[15] Y. Guo, X. Huang, and Z. Ma, “Direct learning-based deep spiking neural networks: a review,” *Frontiers in Neuroscience*, vol. 17, 2023. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2023.1209795

[16] S. Dora and N. Kasabov, “Spiking neural networks for computational intelligence: An overview,” *Big Data and Cognitive Computing*, vol. 5, no. 4, 2021. [Online]. Available: https://www.mdpi.com/2504-2289/5/4/67

[17] C. D. Schuman, J. Parker Mitchell, J. T. Johnston, M. Parsa, B. Kay, P. Date, and R. M. Patton, “Resilience and robustness of spiking neural networks for neuromorphic systems,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–10.

[18] K. Patel and C. Schuman, “Impact of noisy input on evolved spiking neural networks for neuromorphic systems,” 04 2023, pp. 52–56.

[19] K. N. Dang, N. A. V. Doan, and A. B. Abdallah, “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.

[20] J. Timcheck, J. Kadmon, K. Boahen, and S. Ganguli, “Optimal noise level for coding with tightly balanced networks of spiking neurons in the presence of transmission delays,” *PLoS computational biology*, vol. 18, p. e1010593, 10 2022.

[21] C. Torres-Huitzil and B. Girau, “Fault and error tolerance in neural networks: A review,” *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017.

[22] T. H. Vu, O. M. Ikechukwu, and A. Ben Abdallah, “Fault-tolerant spike routing algorithm and architecture for three dimensional noc-based neuromorphic systems,” *IEEE Access*, vol. 7, pp. 90 436–90 452, 2019.

[23] M. Wolfgang, “Networks of spiking neurons: The third generation of neural network models,” *Neural Networks*, vol. 10, no. 9, pp. 1659–1671, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608097000117

[24] S. Thorpe, D. Fize, and C. Marlot, “Speed of processing in the human visual system,” *Nature*, vol. 381, pp. 520–2, 07 1996.

[25] E. T. Rolls and M. J. Tovee, “Processing speed in the cerebral cortex and the neurophysiology of visual masking,” *Proceedings of the Royal Society B: Biological Sciences*, vol. 257, no. 1348, pp. 9–15, Jan. 1994.

[26] W. Gerstner, “Time structure of the activity in neural network models.” *Physical review. E, Statistical physics, plasmas, fluids, and related interdisciplinary topics*, vol. 51 1, pp. 738–758, 1995. [Online]. Available: https://api.semanticscholar.org/CorpusID:45908006

[27] N. S.R., S. Kulkarni, A. V B, and B. Rajendran, “Building brain-inspired computing systems: Examining the role of nanoscale devices,” *IEEE Nanotechnology Magazine*, vol. 12, pp. 19–35, 09 2018.

[28] S. Ghosh-Dastidar and H. Adeli, “Third generation neural networks: Spiking neural networks,” in *Advances in Computational Intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 167–178.

[29] Y. Kim, Y. Li, H. Park, Y. Venkatesha, and P. Panda, “Neural architecture search for spiking neural networks,” 01 2022.

[30] H. Jang, O. Simeone, B. Gardner, and A. Gruning, “An introduction to probabilistic spiking neural networks: Probabilistic models, learning rules, and applications,” *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 64–77, 2019.

[31] S. M. Bohte, J. N. Kok, and H. La Poutré, “Error-backpropagation in temporally encoded networks of spiking neurons,” *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231201006580

[32] R. Gütig and H. Sompolinsky, “The tempotron: a neuron that learns spike-timing based decisions,” *Reviews in the neurosciences*, vol. 16, pp. S27–S27, 01 2005.

[33] Z. Bing, I. Baumann, Z. Jiang, K. Huang, C. Cai, and A. Knoll, “Supervised learning in snn via reward-modulated spike-timing-dependent plasticity for a target reaching vehicle,” *Frontiers in Neurorobotics*, vol. 13, 2019. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbot.2019.00018

[34] T. Wang, Y. Wang, J. Shen, L. Wang, and L. Cao, “Predicting spike features of hodgkin-huxley-type neurons with simple artificial neural network,” *Frontiers in Computational Neuroscience*, vol. 15, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fncom.2021.800875

[35] X. Fang, S. Duan, and L. Wang, “Memristive hodgkin-huxley spiking neuron model for reproducing neuron behaviors,” *Frontiers in Neuroscience*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2021.730566

[36] M.-O. Gewaltig, *Spiking Network Models and Theory: Overview*. New York, NY: Springer New York, 2022, pp. 109–118. [Online]. Available: https://doi.org/10.1007/978-1-0716-1006-0_792

[37] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, “Deep learning in spiking neural networks,” *Neural Networks*, vol. 111, pp. 47–63, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608018303332

[38] M. Johnson and S. Chartier, “Spike neural models (part i): The hodgkin-huxley model,” *The Quantitative Methods for Psychology*, vol. 13, pp. 105–119, 02 2017.

[39] C. Tan, M. Šarlija, and N. Kasabov, “Spiking neural networks: Background, recent development and the neucube architecture,” *Neural Processing Letters*, vol. 52, 10 2020.

[40] W. Guo, M. E. Fouda, A. M. Eltawil, and K. N. Salama, “Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems,” *Frontiers in Neuroscience*, vol. 15, 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2021.638474

[41] S. Bohte, “The evidence for neural information processing with precise spike-times: A survey,” *Nat. Comput.*, vol. 3, 06 2004.

[42] S. Park, S. Kim, B. Na, and S. Yoon, “T2fsnn: Deep spiking neural networks with time-to-first-spike coding,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[43] J. Kim, H. Kim, S. Huh, J. Lee, and K. Choi, “Deep neural networks with weighted spikes,” *Neurocomputing*, vol. 311, pp. 373–386, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231218306726

[44] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[45] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, “Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[46] C. Pehle, S. Billaudelle, B. Cramer, J. Kaiser, K. Schreiber, Y. Stradmann, J. Weis, A. Leibfried, E. Müller, and J. Schemmel, “The brainscales-2 accelerated neuromorphic system with hybrid plasticity,” 2022.

[47] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, “Overview of the spinnaker system architecture,” *IEEE Transactions on Computers*, vol. 62, no. 12, pp. 2454–2467, 2013.

[48] S. Moradi and R. Manohar, “The impact of on-chip communication on memory technologies for neuromorphic systems,” *Journal of Physics D: Applied Physics*, vol. 52, 10 2018.

[49] G. Indiveri and S.-C. Liu, “Memory and information processing in neuromorphic systems,” *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1379–1397, 2015.

[50] S. Yu, Y. Wu, R. Jeyasingh, D. Kuzum, and H.-S. P. Wong, “An electronic synapse device based on metal oxide resistive switching memory for neuromorphic computation,” *IEEE Transactions on Electron Devices*, vol. 58, no. 8, pp. 2729–2737, 2011.

[51] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, “Complementary resistive switches for passive nanocrossbar memories,” *Nature materials*, vol. 9, pp. 403–6, 05 2010.

[52] B. Rajendran and F. Alibart, “Neuromorphic computing based on emerging memory technologies,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 6, no. 2, pp. 198–211, 2016.

[53] J. Ding, Z. Yu, Y. Tian, and T. Huang, “Optimal ann-snn conversion for fast and accurate inference in deep spiking neural networks,” 2021.

[54] H. Gao, J. He, H. Wang, T. Wang, Z. Zhong, J. Yu, Y. Wang, M. Tian, and C. Shi, “High-accuracy deep ann-to-snn conversion using quantization-aware training framework and calcium-gated bipolar leaky integrate and fire neuron,” *Frontiers in Neuroscience*, vol. 17, 2023. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2023.1141701

[55] N.-D. Ho and I.-J. Chang, “TCL: an ANN-to-SNN conversion with trainable clipping layers,” in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, dec 2021. [Online]. Available: https://doi.org/10.1109%2Fdac18074.2021.9586266

[56] K. S. Ahmed and F. F. Shereif, “Neuromorphic computing between reality and future needs,” in *Neuromorphic Computing*, P. Y. Yi and D. H. An, Eds. Rijeka: IntechOpen, 2023, ch. 5. [Online]. Available: https://doi.org/10.5772/intechopen.110097

[57] M. Mahowald, *An Analog VLSI System for Stereoscopic Vision*. USA: Kluwer Academic Publishers, 1994.

[58] K. Boahen, “Point-to-point connectivity between neuromorphic chips using address events,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 5, pp. 416–434, 2000.

[59] P. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, vol. 345, pp. 668 – 673, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:12706847

[60] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, “Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations,” *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[61] V. N. Balaji, P. B. Srinivas, and M. K. Singh, “Neuromorphic advancements architecture design and its implementations technique,” *Materials Today: Proceedings*, vol. 51, pp. 850–853, 2022, cMAE’21. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785321046587

[62] W. Y. Yerima, O. M. Ikechukwu, K. N. Dang, and A. Ben Abdallah, “Fault-tolerant spiking neural network mapping algorithm and architecture to 3d-noc-based neuromorphic systems,” *IEEE Access*, vol. 11, pp. 52 429–52 443, 2023.

[63] W. Y. Yerima, K. N. Dang, and A. B. Abdallah, “R-mas3n: Robust mapping of spiking neural networks to 3d-noc-based neuromorphic systems for enhanced reliability,” *IEEE Access*, pp. 1–1, 2023.

[64] V. Duddu, D. V. Rao, and V. Balas, “Towards enhancing fault tolerance in neural networks,” in *MobiQuitous 2020 - 17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, ser. MobiQuitous ’20. New York, NY, USA: Association for Computing Machinery, 2021, p. 59–68. [Online]. Available: https://doi.org/10.1145/3448891.3448936

[65] L.-C. Chu and B. Wah, “Fault tolerant neural networks with hybrid redundancy,” in *1990 IJCNN International Joint Conference on Neural Networks*, 1990, pp. 639–649 vol.2.

[66] H. Takase, M. Masahiko, K. Hidehiko, and H. Terumine, “Enhancing both generalization and fault tolerance of multilayer neural networks,” 09 2007, pp. 1429 – 1433.

[67] M. Rastogi, S. Lu, N. Islam, and A. Sengupta, "On the self-repair role of astrocytes in stdp enabled unsupervised snns," *Frontiers in Neuroscience*, vol. 14, 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2020.603796

[68] K. Rhazali, B. Lussier, W. Schön, and S. Geronimi, "Fault tolerant deep neural networks for detection of unrecognizable situations," *IFAC-PapersOnLine*, vol. 51, no. 24, pp. 31–37, 2018, 10th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S240589631832216X

[69] V. Sridharan, N. DeBardeleben, S. Blanchard, K. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," *ACM SIGPLAN Notices*, vol. 50, pp. 297–310, 05 2015.

[70] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 415–426.

[71] P. V. Bhanu, P. V. Kulkarni, and S. J, "Fault-tolerant network-on-chip design with flexible spare core placement," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 1, jan 2019. [Online]. Available: https://doi.org/10.1145/3269983

[72] Y. Nong, H. Cai, P. Ye, L. Li, and F. Chen, "Evaluating and comparing memory error vulnerability detectors," *Information and Software Technology*, vol. 137, p. 106614, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584921000896

[73] S. Mitra, P. Sanda, and N. Seifert, "Soft errors: Technology trends, system effects, and protection techniques," in *13th IEEE International On-Line Testing Symposium (IOLTS 2007)*, 2007, pp. 4–4.

[74] L. Xiong, Q. Tan, and J. Xu, "Effects of soft error to system reliability," in *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*, 2011, pp. 204–209.

[75] T. Titirsha, S. Song, A. Das, J. Krichmar, N. Dutt, N. Kandasamy, and F. Catthoor, "Endurance-aware mapping of spiking neural networks to neuromorphic hardware," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 2, pp. 288–301, 2022.

[76] C. Xiao, Y. Wang, J. Chen, and L. Wang, "Topology-aware mapping of spiking neural network to neuromorphic processor," *Electronics*, vol. 11, no. 18, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/18/2867

[77] A. Balaji, A. Das, Y. Wu, K. Huynh, F. G. Dell'Anna, G. Indiveri, J. L. Krichmar, N. D. Dutt, S. Schaafsma, and F. Catthoor, "Mapping spiking neural networks to neuromorphic hardware," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 76–86, 2020.

[78] O. Jin, Q. Xing, Y. Li, S. Deng, S. He, and G. Pan, "Mapping very large scale spiking neuron network to neuromorphic hardware," ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 419–432. [Online]. Available: https://doi.org/10.1145/3582016.3582038

[79] L. Wang, P. Lv, L. Liu, J. Han, H.-F. Leung, X. Wang, S. Yin, S. Wei, and T. Mak, "A lifetime reliability-constrained runtime mapping for throughput optimization in many-core systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1771–1784, 2019.

[80] A. Balaji, T. Marty, A. Das, and F. Catthoor, "Run-time mapping of spiking neural networks to neuromorphic hardware," *Journal of Signal Processing Systems*, vol. 92, 11 2020.

[81] Q. Xu, S. Chen, H. Geng, B. Yuan, B. Yu, F. Wu, and Z. Huang, "Fault tolerance in memristive crossbar-based neuromorphic computing systems," *Integr. VLSI J.*, vol. 70, no. C, p. 70–79, jan 2020. [Online]. Available: https://doi.org/10.1016/j.vlsi.2019.09.008

[82] A. Gebregirogis and M. Tahoori, "Approximate learning and fault-tolerant mapping for energy-efficient neuromorphic systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 3, dec 2021. [Online]. Available: https://doi.org/10.1145/3436491

[83] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, and A. C. Knoll, "A survey of robotics control based on learning-inspired spiking neural networks," *Frontiers in Neurorobotics*, vol. 12, 2018. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbot.2018.00035

[84] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, apr 2019. [Online]. Available: https://doi.org/10.1145/3304103

[85] E. Ledinauskas, J. Ruseckas, A. Juršėnas, and G. Buračas, "Training deep spiking neural networks," 2020.

[86] J. K. Eshraghian, M. Ward, E. O. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *Proceedings of the IEEE*, pp. 1–39, 2023.

[87] A. Siddique and K. A. Hoque, "Improving reliability of spiking neural networks through fault aware threshold voltage optimization," 2023.

[88] G. Yuan, Z. Liao, X. Ma, Y. Cai, Z. Kong, X. Shen, J. Fu, Z. Li, C. Zhang, H. Peng, N. Liu, A. Ren, J. Wang, and Y. Wang, "Improving dnn fault tolerance using weight pruning and differential crossbar mapping for reram-based edge ai," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 135–141.

[89] T. Horita, I. Takanami, and M. Mori, “Learning algorithms which make multilayer neural networks multiple-weight-and-neuron-fault tolerant,” *IEICE - Trans. Inf. Syst.*, vol. E91-D, no. 4, p. 1168–1175, apr 2008. [Online]. Available: https://doi.org/10.1093/ietisy/e91-d.4.1168

[90] W. Guo, H. E. Yantır, M. E. Fouda, A. M. Eltawil, and K. N. Salama, “Towards efficient neuromorphic hardware: Unsupervised adaptive neuron pruning,” *Electronics*, vol. 9, no. 7, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/7/1059

[91] E. Rahiminejad, F. Azad, A. Parvizi-Fard, M. Amiri, and B. Linares-Barranco, “A neuromorphic cmos circuit with self-repairing capability,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 2246–2258, 2022.

[92] J. Seo, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, J. A. Tierno, L. Chang, D. S. Modha, and D. J. Friedman, “A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons,” in *2011 IEEE Custom Integrated Circuits Conference (CICC)*, Sep. 2011, pp. 1–4.

[93] D. J. Eckman and S. G. Henderson, “Reusing search data in ranking and selection: What could possibly go wrong?” *ACM Trans. Model. Comput. Simul.*, vol. 28, no. 3, jul 2018. [Online]. Available: https://doi.org/10.1145/3170503

[94] T. Zheng and N. Beier, “Increasing transparency and accessibility of a slurry consolidation model in goldsim,” 10 2022.

[95] A. Komey, Q. Deng, G. Baecher, P. Zielinski, and T. Atkinson, “Systems reliability of flow control in dam safety,” 07 2015.

[96] . H. Y.-S. Lee, Youn-Myoung, “A comparative study between goldsim and amber based biosphere assessment models for an hlw repository.” Proceedings of the KNS autumn meeting, (pp. 1CD-ROM). Korea, Republic of KNS, 07 2007.

[97] O. Köpüklü, M. Babaee, S. Hörmann, and G. Rigoll, “Convolutional neural networks with layer reuse,” in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 345–349.