Functional safety and reliability of neuromorphic computing systems

Theofilos Spyrou

THÈSE DE DOCTORAT
DE SORBONNE UNIVERSITÉ

FUNCTIONAL SAFETY & RELIABILITY OF NEUROMORPHIC COMPUTING SYSTEMS

présentée par

THEOFILOS SPYROU

École Doctorale Informatique, Télécommunications et Électronique

réalisée au

Laboratoire d'Informatique de Paris 6

soutenue le 7 juin 2023

devant le jury composé de :

M.	Bertrand GRANADO	Prof., Sorbonne Univ., CNRS, LIP6, Paris	Président
Mme.	Lorena ANGHEL	Prof., Grenoble INP, Spintec, Grenoble	Rapporteure
M.	Paolo RECH	Assoc. Prof., Univ. Degli Studi di Trento, Italie	Rapporteur
M.	Said HAMDIOUI	Prof., Univ. Technologique de Delft, Pays-Bas	Examinateur
M.	Alkiviadis HATZOPOULOS	Prof., Univ. Aristote de Thessalonique, Grèce	Examinateur
M.	Haralampos STRATIGOPOULOS	DR, Sorbonne Univ., CNRS, LIP6, Paris	Directeur de Thèse

Functional Safety & Reliability of Neuromorphic Computing Systems

Theofilos Spyrou
Paris, 2023

SUPERVISOR:
Haralampos Stratigopoulos

To my parents
who always supported and believed in me,

my sister
who is always there for me and stands by my side for life,

my canine brother
who taught me how to be a child again...

ABSTRACT

The recent rise of Artificial Intelligence (AI) has found a wide range of applications essentially integrating it. The goal is the improvement of the provided services and ultimately a more convenient life for everyone. Although AI has had some bad days in the past, it is now more mature than ever and it is gaining more and more ground in almost any field of our lives. However, there is a question that still persists and remains unanswered: how trustworthy AI really is?

The answer to this question is a combination of multiple factors such as the quality of the AI application, the ethics of the people who implemented it and those using it, etc. Among these factors, there is one standing out and needs to be thoroughly considered before the employment of AI in the field, especially in mission- and safety-critical applications like autonomous vehicles. This is no other than the dependability of the systems hosting the operation of AI applications, or the AI hardware accelerators.

At first sight, there might seem that there is no problem, as Artificial Neural Networks (ANNs), and particularly the biology-inspired Spiking Neural Networks (SNNs) are believed to be resilient structures just like their biological counterparts. The human brain for instance is known to be remarkably capable of tolerating faults that may occur in the neurons or synapses, retaining its functionality intact. However, although SNNs become heir of this property to some extent, an assumption of inherent fault tolerance is rather naive when considering that electronics do not operate the same way as biology. Hence, a defect of an electronic component or a fault occurring either during the fabrication of an Integrated Circuit (IC) or after its deployment in the field can have a disastrous effect on the performance of the executed AI application, threatening in this way the safety of the surrounding people and the environment.

Because of these, it is made evident that a methodological exploration of the dependability characteristics of AI hardware accelerators and neuromorphic platforms, i.e., accelerators hosting the training and/or inference of SNNs, is of utmost importance. First, a resilience analysis of the SNN and its neuromorphic chip against hardware-level faults helps in the study of the system's reliability by pinpointing the critical parts. Next, these parts need to be protected with a fault tolerance strategy that allows the network to tolerate some of the faults proactively and the rest reactively after testing and mitigating the effects of the detected faults.

This thesis tackles the subjects of testing and fault tolerance in SNNs and their neuromorphic implementations on hardware. It starts with a defect-oriented taxonomy of faulty behaviors of a spiking neuron at transistor level, which forms the basis of a behavioral-level fault model specific to SNNs, yet agnostic to the circuit design and architecture. Based on this, a series of large-scale

fault injection experiments is conducted through a hardware-accelerated fault injection framework designed for SNNs, aiming at analyzing their resilience. Leveraging the results of these experiments, a cost-effective fault tolerance strategy for SNNs is proposed. Also, a neuromorphic hardware experimentation platform is presented, on which a reliability assessment of SNNs running on actual neuromorphic hardware is performed. After the assessment, the platform is equipped with an on-line on-chip testing mechanism that detects faults in real time. Finally, a compact functional test-set generation technique is demonstrated to address the problem of testing neuromorphic hardware in a generalized way.

RÉSUMÉ

L'essor récent de l'Intelligence Artificielle (IA) a trouvé un large éventail d'applications cibles. L'objectif est l'amélioration des services fournis au final une vie plus simple pour tous. Après avoir connu de mauvais jours, l'IA revient sur le devant de la scène, plus mature que jamais, et gagne de plus en plus de terrain dans presque tous les domaines de notre vie. Cependant, une question persiste et reste sans réponse : dans quelle mesure l'IA est-elle vraiment digne de confiance ?

La réponse à cette question est une combinaison de multiples facteurs tels que la qualité de l'IA, l'éthique des personnes qui l'ont mise en œuvre et de celles qui l'utilisent, etc. Parmi ces facteurs, il en est un qui se démarque et qui doit être examiné de manière approfondie, en particulier dans les applications critiques pour la mission et la sûreté telle que les véhicules autonomes. Il s'agit de la fiabilité des systèmes hébergeant l'IA, ou de leurs accélérateurs matériels.

On peut croire, à première vue, qu'il n'y a pas de problème, car les réseaux neuronaux artificiels (ANNs : Artificial Neural Networks en anglais), et en particulier les réseaux neuronaux à impulsions (SNNs : Spiking Neural Networks en anglais) d'inspiration biologique, sont considérés comme des structures résilientes, tout comme leurs homologues biologiques. Le cerveau humain, par exemple, est connu pour être remarquablement capable de tolérer les défauts qui peuvent se produire dans les neurones ou les synapses, en conservant sa fonctionnalité intacte. Cependant, bien que les SNNs héritent de cette propriété dans une certaine mesure, l'hypothèse d'une tolérance inhérente aux fautes est plutôt naïve si l'on considère que l'électronique ne fonctionne pas de la même manière que la biologie. Ainsi, le défaut d'un composant électronique ou une faute survenant pendant la fabrication d'un circuit intégré ou après son déploiement peut avoir un effet désastreux sur les performances de l'application exécutée, menaçant ainsi la sécurité des personnes environnantes et de l'environnement.

Pour ces raisons, il est évident qu'une exploration méthodologique des caractéristiques de fiabilité des accélérateurs matériels d'IA et des plateformes neuromorphiques, c'est-à-dire des accélérateurs hébergeant l'entraînement et/ou l'inférence de SNNs, est de la plus haute importance. Tout d'abord, une analyse de la résilience du SNN et de sa puce neuromorphique contre les fautes au niveau matériel aide à l'étude de la fiabilité du système en identifiant les parties critiques. Ensuite, ces parties doivent être protégées par une stratégie de tolérance aux fautes qui permet au réseau de tolérer certaines des fautes de manière proactive et le reste de manière réactive après avoir testé et atténué les effets des fautes détectées.

Cette thèse aborde les sujets du test et de la tolérance aux fautes dans les SNNs et leurs implémentations neuromorphiques matérielles. Elle commence

par une taxonomie des comportements défectueux d'un neurone à impulsions au niveau des transistors, qui forme la base d'un modèle de fautes au niveau comportemental spécifique aux SNNs, mais agnostique à la conception et à l'architecture du circuit. Sur cette base, une série d'expériences d'injection de fautes à grande échelle est menée par le biais d'un *framework* d'injection de fautes accéléré matériellement conçu pour les SNNs, dans le but d'analyser leur résilience. En s'appuyant sur les résultats de ces expériences, une stratégie de tolérance aux fautes efficace pour les SNNs est proposée. Une plateforme d'expérimentation de matériel neuromorphique est également présentée, sur laquelle une évaluation de la fiabilité des SNNs fonctionnant sur du matériel neuromorphique réel est effectuée. Après l'évaluation, la plateforme est équipée d'un mécanisme de test sur puce et en ligne qui détecte les défauts en temps réel. Enfin, une technique compacte de génération de test-set fonctionnel est démontrée pour répondre au problème du test de matériel neuromorphique d'une manière généralisée.

PUBLICATIONS

As a result of this thesis the following publications have appeared:

[1] S. A. El-Sayed, T. Spyrou, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Spiking neuron hardware-level fault modeling," in *Proc. 26th IEEE Int. Symp. On-Line Test. Robust Syst. Des. (IOLTS)*, Jul. 2020.

[2] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Neuron fault tolerance in spiking neural networks," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Feb. 2021.

[3] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Reliability analysis of a spiking neural network hardware accelerator," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2022.

[4] S. A. El-Sayed, T. Spyrou, L. A. Camuñas-Mesa, and H.-G. Stratigopoulos, "Compact functional testing for neuromorphic computing circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2022.

[5] T. Spyrou and H.-G. Stratigopoulos, "On-line testing of neuromorphic hardware," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2023.

[6] T. Spyrou, S. A. El-Sayed, and H.-G. Stratigopoulos, "SpikeFI: a fault injection framework for spiking neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2023 (submitted).

ACKNOWLEDGMENTS

"As you set out for Ithaka
hope the voyage is a long one,
full of adventure, full of discovery."

— Konstantinos Kavafis

Pursuing a PhD degree is not only about research; it's a true voyage in the world of knowledge. And as in every journey, discovery is its basis and is essential in order to learn and understand the purpose. Besides, it is not the destination that matters but the journey itself, the adventure that is unrolled while heading toward the destination. During this, a lot of people come and go, each one contributing in their own way to the arrival at the final destination. Acknowledging these people's contribution and thanking them for that is the least act of appreciation someone can show in return. Before that though, let me start with how this wonderful philosophical voyage began.

It is June 2019 when a senior student at the Electrical and Computer Engineering department of the Polytechnic School of Aristotle University of Thessaloniki is finalizing his engineering thesis and preparing for the approaching exams of the final semester. And then, prof. Alkis Hatzopoulos, forwards to his students an email of prof. Haralampos Stratigopoulos, with the latter announcing the opening of a PhD position at Sorbonne Univerisity for which he was seeking a student. His proposal had just been accepted by Sorbonne Center for Artificial Intelligence (SCAI) and advanced to the final round before approval. So, the selected for the position student had to compete through a short interview with other 19 Artificial Intelligence (AI)-related topics and convince the jury that the topic was worth one of the ten granted fellowships.

Without spending a single moment, I directly replied to the email and a few days later I'm in prof. Hatzopoulos office discussing with prof. Stratigopoulos about the PhD. I remember the two first things I said: (i) I haven't got my diploma yet and (ii) I speak no French. Thankfully, none of these was a constraining factor and after a closer discussion with Haralampos, I was his choice for the position. Having no time to lose as the final interview was only two weeks ahead, we started preparing while in parallel the exams in Aristotle University were in progress. Eventually, everything goes fine with the exams and shortly after I find myself in Paris before the jury of SCAI. Let me mention at this point that the interview took place in the 20^{th}+ floor of Tour Zamansky, France's third highest building, which means that the view was better than any existing carte postale of the city of lights... Trying not to get distracted by the beauty of Paris, I turn my head the other way and start the presentation. Minutes passed like seconds and now we're waiting for the results. Of course, I had planned a few

days of vacation after the interview, since this was my first time in the city. After a lunch with Haralampos, I calmly head back to my hotel not expecting to have the results soon as it was a summer Friday afternoon at the end of June. But no! The results were ready a couple of hours later and before I make it back to the hotel, Haralampos calls me to announce the great news. I can't hide that this was the best day of my (professional) life and the weekend that followed the most relaxing and peaceful I ever had.

First and before all, I would like to warmly thank from the deepest of my heart **Haralampos Stratigopoulos** for trusting me and giving me this amazing opportunity. Honestly, having discussed with many other PhD students, it is not often the case to hear the best about student-supervisor relationships. Maybe my case was an outlier as I believe Haralampos was the best supervisor someone can have. He was always there during my PhD willing to help, explain, and teach. His guidance was essential and his involvement to the right extend: neither too much being intrusive, nor too little to feel abandoned. As his student, I learnt a lot of useful things by his side that I believe will shape my future career and contribute to my professional development. But beyond all lessons, Haralampos showed me what being a good professor is about, not only professionally but also humanly by respecting and treating everyone nicely, patiently, and selflessly. For all these, I am more than grateful to have had Haralampos as my mentor during this voyage.

A special thank you to prof. **Alkis Hatzopoulos** as well, who trusted me from the first moment and recommended me to Haralampos without any hesitations. It is more than fascinating the fact that the academic community can be so bonded with mutual cooperation and communication among universities worldwide. Prof. Hatzopoulos was my engineering thesis supervisor, so in a way, he prepared me for moving forward toward pursuing a higher academic degree with all the difficulties it may hide.

Another vital contribution to the realization of this PhD was the one of SCAI and the colleagues who work there. SCAI was founded right before my voyage started and as one of its very first ten students, I feel but proud if I have contributed the slightest at its scientific development and recognition through my work and effort. I would also like to thank Prof. **Gérard Biau** (director) and Dr **Xavier Fresquet** (deputy director) for all their help throughout these years and for their educational and scientific initiatives, which made the presence of SCAI strong in the scientific community. Also, the whole team of SCAI was very warm and I really enjoyed every meeting and every moment with my fellow students there, whose work is always admiring to hear about.

Luis A. Camuñas Mesa played a major role in the realization of this PhD with his essential contributions. Thank you Luis for the circuits you provided us with, which enabled the experiments performed as part of the work of this thesis; the multiple meetings to explain us the details of the various functionalities of the circuits; and your patience to answer all our questions. Once again, you proved the nice values of cooperation beyond the borders of a university or a country. I

can also say that the Spanish comments in the designs taught some basics of the language and made me eager to actively start learning it!

When I started my voyage I saw it as something ordinary rather than a adventure full of surprises. Yet, there were many hidden surprises waiting to be unravelled in the way. The most important of them was when I met **Sarah A. ElSayed**, my twin sister albeit a few years older, as she used to say. Sarah was undeniably a source of inspiration for me, who literally made me view the world through a new pair of eyes. I will never forget the endless conversations, debates, cultural exchanges, etc., which could last until late at the office. It was my honor to work with you Sarah and definitely I hope that we will have this opportunity again at some point in time and space. Beyond all, I feel more than happy to have met such a great friend for life.

Sharing an office with other students and researchers is a nice experience. Either working on the same project, or on a similar research field, it is always interesting to learn about the work of colleagues and hear about their successes. The office 415 was one of these places where I made amazing friends and I am grateful to have shared it with them. **Antonis** the Grand Maître who turned chess into the sport of the lab and made us only dreaming a victory against him; **Julian** who was taking care to make our day with his endless positive energy; **Gabriel** who was the "serious" of the office and who helped me meet closer the French culture; **Spyros** who I know since high school and we had once again the opportunity to work together. Of course I cannot skip all these amazing colleagues that I worked with during these years: **Engin**, **Mohamed**, **Paul**, and **Shaima**. Thank you all for sharing the journey with me.

However, the borders of an office were not enough to keep me from making more friends. **Ilias** with whom we explored every corner of Paris and travelled in almost every place of France; **Alan** who was always in to every plan and showed me the amazing Mexican culture; **Maxime** who was the "teaser" of the lab (in a good way) and the most cheerful person I have ever met; **Clara** and **Mathuran** whose good vibes were brightening our days. **Ning**, **Andrien**, **Rieul**, **Thomas**, **Nathan**, **Jonathan**, **Baptiste**, **Aymeric**, and **Abdelrahman** were also some of the amazing friendships I built during my journey.

Outside the professional environment, the journey continued and actually was augmented as I lived for three years in the *Cité Internationale Universitaire de Paris* (CIUP); a multicultural campus with tens of residences of countries from all around the world, endless green spaces, and full of sport and cultural events. I had the opportunity to stay in the Portuguese and the Dutch houses where I learnt a lot not only about the cultures of these two countries but also for many more, since there was a huge variety in the nationalities of the residents. It was more than fascinating to meet so many people coming from different parts of the world and studying on various disciplines. Some of the greatest friendships that were born during my stay were these of **Tasos**, who is coming from a neighbor city of my hometown in Greece but we met so far away, **Dimitris**, and **Anastasia**, with all of whom I had an awesome time and enjoyed our endless conversations and adventures.

Nonetheless, the largest of my gratitude is reserved for my family. My parents **Anna** and **Dimitris** never stopped being by my side supporting each of my decisions and steps in life. They are the ones who helped me reach this point by teaching me how to dream and follow my ambitions. Thank you mom and dad for believing in me. My sister **Vasilina** is the only person in the world that was, is, and will be standing by my side for life. Although younger, I have to admit that I admire her for her willingness to achieve her goals being strong and independent, always ready to define her own paths. Thank you for helping me become a better person. Last but not least, I cannot leave aside the youngest member of the family, my brother **Faidon**. Despite the fact that he is a dog, the word ”brother“ describes our bond pretty accurately. He is the only one reminding me what is really important in life and what it means to be truly selfless. Thank you for being the most vivid example of life and the kindest creature I have ever met.

Theofilos

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AER	Address Event Representation
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASIC	Application Specific Integrated Circuit
BER	Bit-Error Rate
BIST	Built-In Self-Test
CMOS	Complementary Metal-Oxide-Semiconductor
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DfT	Design-for-Test
DNN	Deep Neural Network
DVS	Dynamic Vision Sensor
ECC	Error Correction Code
FAM	Fault-Aware Mapping
FATM	Fault-Aware Training and Mapping
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
I&F	Integrate-and-Fire
IC	Integrated Circuit
I/O	Input/Output
IoT	Internet-of-Things
IP	Intellectual Property
LSB	Least Significant Bit
MAC	Multiply-and-Accumulate
MBIST	Memory Built-In Self-Test
MC	Monte Carlo
ML	Machine Learning
MSB	Most Significant Bit
PDK	Process Design Kit
PE	Process Element

PPA Power-Performance-Area

RBF Radial Basis Function

R-STDP Reward-modulated Spike-Timing-Dependent Plasticity

SEU Single-Event Upset

SLAYER Spike LAYer Error Reassignment

SNN Spiking Neural Network

SPI Serial Peripheral Interface

SRM Spike Response Model

STDP Spike-Timing-Dependent Plasticity

STL Software Test Libraries

SVM Support Vector Machine

TMR Triple Modular Redundancy

TPU Tensor Processing Unit

VHDL Very High-Speed Integrated Circuit Hardware Description Language

VLSI Very Large Scale Integration

1 INTRODUCTION

"Artificial Intelligence is the new electricity."

— Andrew Ng

The curiosity to explore, discover, and comprehend has been an integral part of human nature since the very beginning of its existence. It is nothing more than the desire deriving from deep within us to evolve; to go up; to reach further. And there is no better way to keep the hungriness of this unending desire fed than observation. Nature offers endless sources for inspiration, amongst which the most intriguing and challenging so far being the brain. That same thing defining our consciousness and logic, we use it in an effort to understand and replicate its own functionality in order to create a new form of intelligence, an artificial one.

And now that human evolution has eventually reached this point, new questions arise as we keep exploring the uncharted waters brought by Artificial Intelligence (AI) such as can we actually rely on it and trust it?

1.1 ARTIFICIAL INTELLIGENCE: INDUSTRY V4.0

Human history has repeatedly shown that advancing is the cornerstone to a thriving society. Throughout evolution, humans have never seized to seek for improvements in all the aspects of their lives. A proof to this are the numerous breakthroughs whose outcome has contributed in redefining society in order for it to move forward.

Out of the plethora of technological advancements, there are three of them that left an enormous footprint on the shaping of the modern world. During less than three centuries, three revolutionary steps were taken and brought society into the future. Namely, the so-called *industrial revolutions* are composed of (i) the mechanization of production; (ii) the emergence of new sources of energy, i.e., electricity and fossil fuels; and (iii) nuclear energy.

What is arguably the forth component of the equation, is AI, although yet in its infant steps. With its full potential waiting to be unrolled, AI has met a rapid growth over the past years and can already find applications anywhere in almost any field, ranging from entertainment to medicine.

1.1.1 *The Chronology of Thinking Machines*

Since the dawn of computers, a "thinking" machine has been many people's dream. Alan Turing was the first to substantially pioneer in the field of AI in the mid-20th century. During his public lecture in London in 1947, he referred to computer intelligence as a machine that can learn from experience and that the mechanism to achieve this is the possibility of letting the machine alter its own instructions. His later work was a source of inspiration to many researchers in the AI community.

In 1950 Turing defined intelligence through his homonymous test [2], illustrated in Fig. 1.1. According to the Turing Test, or the Imitation Game as he named it, interrogators pose questions to a computer and a human and receive a written response. The goal is to determine which is the computer. The interrogators are free to ask anything as penetrating and wide-ranging as they like, while the computer is permitted to do anything possible to force a wrong identification. At the end of the experiment, if a sufficient proportion of the interrogators are not in position to differentiate the computer from the human being, then the computer is considered intelligent. Ever since there have been endless trials of AI algorithms to pass the Turing Test but none of them have yet actually succeeded.

The first successful AI programs were structured around mastering "mind sports", like chess and checkers, already from the 1950s [3]. Chess for example has always been a challenging game with clearly defined problems, allowing for reasoning and testing problem solving methods. A naive chess-playing computer would play by foreseeing all the available moves up to a depth that the game is over. However, this approach is practically impossible as this would involve examining an astronomically large number of moves. Therefore, developing a

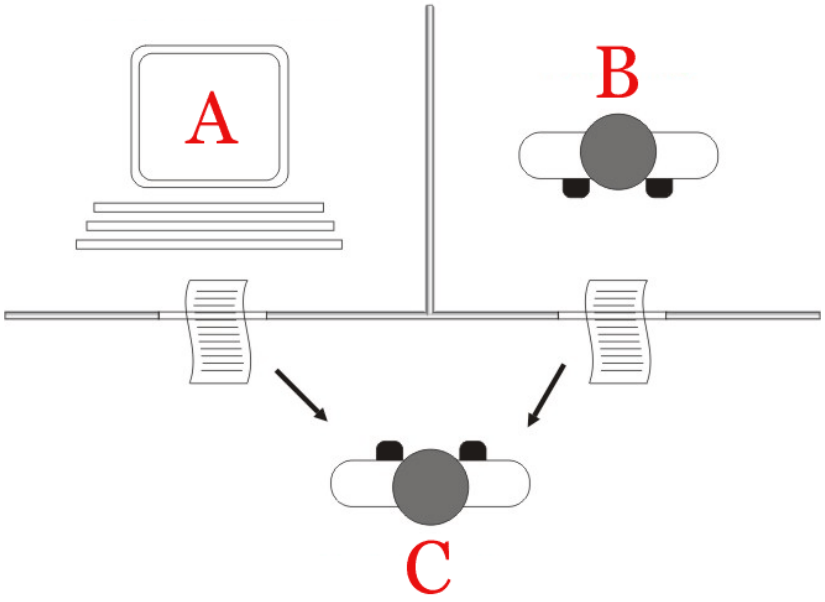


Figure 1.1: The Turing Test. Player C, the interrogator, is given the task of trying to determine which player – A or B – is a computer and which is a human [1].

heuristic strategy for playing this kind of games would indeed require some intelligence to win. Having said that, chess does not make the best fit in the search of an intelligent system nowadays, since technology advanced and already in 1997 the reigning chess world champion, Garry Kasparov, was defeated by the Deep Blue computer of IBM, a massively parallel system with multiple levels of parallelism which could examine millions of possible moves per second and thus look ahead as many as 14 turns of play [4]. However, although fascinating, this cannot be considered an advancement in AI.

Another great milestone, which also shaped the modern form of AI, was in 1943 when the neurophysiologist Warren McCulloch and the mathematician Walter Pitts, published their work on *neural nets and automatons* [5]. According to it, each neuron in the brain is a simple digital processor and the brain as a whole is a form of computing machine. They proposed the "binary neuron" model, which, a decade later, was employed to bring the first Artificial Neural Network (ANN) to life by Belmont Farley and Wesley Clark [6]. Albeit limited by the early stage of the technology at the time, they managed to train a two-layer network consisting of 128 neurons to recognize simple patterns.

A few years later, Frank Rosenblatt's research on *percerptrons* [7], as he used to call neural networks, contributed to form the base of modern ANNs. More specifically, he aimed at generalizing the training procedure of neural networks, so that it could be applied to multi-layer networks, as well. The *back-propagating error correction*, or *back-propagation* as it is known today, is the most dominant training technique and is now in everyday use in the world of ANNs.

During the years that followed, the AI community never rested and contributions continued to arise in theoretical, philosophical, and applied levels. There was made progress on the domains of natural language processing and robotics,

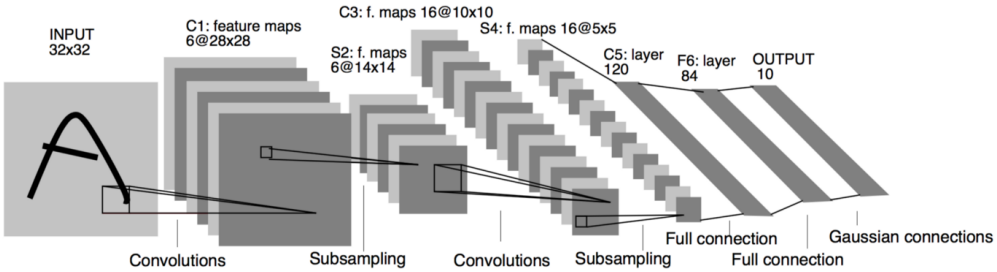


Figure 1.2: The architecture of LeNet-5 [8].

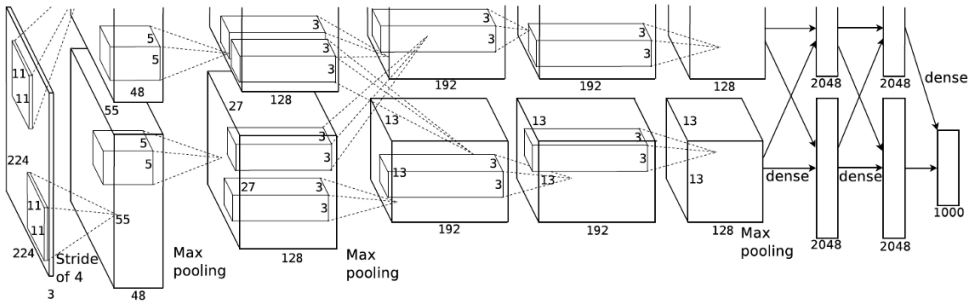


Figure 1.3: The architecture of AlexNet [10].

while also a lot of programming languages were designed and suited well the needs of AI applications, like LISP and PROLOG.

In the 1990s, a refreshing wind was blowing for AI that was about to drastically change it. Yann LeCun et al. created the LeNet network [9], one of the earliest Convolutional Neural Networks (CNNs), and promoted the development of deep learning. It was the first time that the back-propagation algorithm was applied on a practical application to recognize handwritten arithmetic digits. The final result of LeNet-5 was published in 1998 [8] and was shown to outperform all the other methods of handwritten character recognition in paper. Fig. 1.2 presents the architecture of LeNet-5. The success of this research was great making a big impact on the field of AI that initiated and inspired a wave of people to study neural networks.

LeNet was the starting point for a large number of neural network architectures that followed up. In 2012, AlexNet [10], a deep CNN shown in Fig. 1.3, was the winner of the ImageNet Large Scale Visual Recognition Challenge of that year with a clear difference over the runner-up. ImageNet is a huge dataset consisting of more than 14 million images containing objects of more than 20 thousand categories. The results showed that the depth of the model, i.e., the number of layers consisting it, was essential for its high performance, leading the way towards Deep Neural Networks (DNNs).

Nowadays, deep architectures have become the mainstream of neural networks with depths reaching hundreds of layers of various types, e.g., convolutional, dense, pooling, etc. Some examples of the most famous and widely-used DNN models include the VGGNet [12] and ResNet [13] variations, and the GoogLeNet,

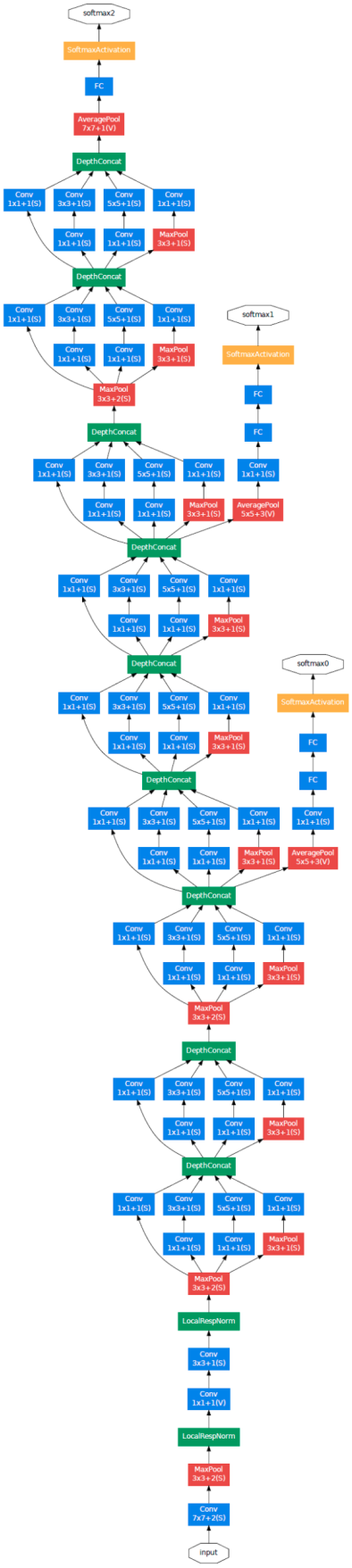


Figure 1.4: The architecture of GoogLeNet, or Inception Network [11].

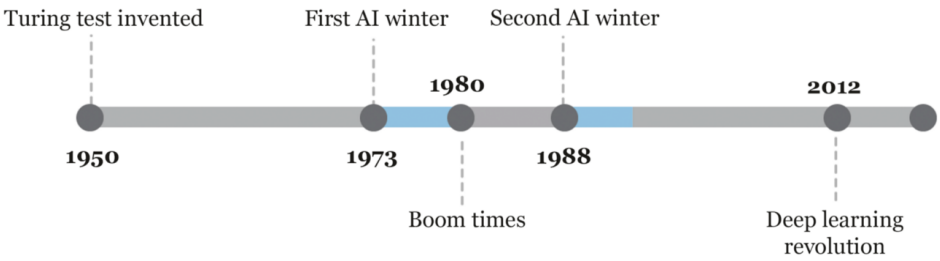


Figure 1.5: Timeline of AI winters.

or Inception Network [11], amongst many others just to name a few. However, deeper does not necessarily mean better. Thus, concurrently to the improvement of neural network architectures, machine learning techniques keep improving as well with more and more concepts being proposed constantly.

AI continues to emerge in very high speeds with more and more applications being adopted everywhere. A typical example is *ChatGPT*, an intelligent conversational agent that can answer followup questions, admit its mistakes, challenge incorrect premises, and reject inappropriate requests in a dialogue format. Since the application was released in November, 2022, and until the moment of writing this thesis, it has gained more than 100 million users in two months becoming the all-time fastest growing app.

It is evident that people rely more and more on AI which continues to grow and spreads in tremendous speed. The future of AI looks bright and we cannot but await to discover what lies within it. Just like every trend though, it must be treated with caution before it is ready to be used widely in a reliable way.

1.1.2 *The AI Winters & the Transition towards an AI Spring*

It is common for emerging technologies to be surrounded by hype. Over-inflated promises by developers, unnaturally high expectations from end-users, and extensive promotion in the media [14] are the leading factors to cause a bubble. Soon the situation explodes and hype is replaced by pessimism in the community, which extends to the media, causing disappointment and criticism that result in funding cuts and people's interest moving away.

A similar situation happened with AI in the second half of the 20th century, which was described as the *AI winter* as an analogy to the nuclear winter. By taking a closer look at the timeline of AI (Fig. 1.5), we can distinguish two such major periods followed by a calmer post-AI winter that prepared the current *AI spring*.

1.1.2.1 *AI Winter I, 1973-1980*

Already from the early 1970s, people have started questioning AI and its capabilities. The recent failure of AI research to effectively respond to community's expectations in domains such as robotics and natural language processing, led

to a disappointment and abandonment of AI. Lighthill's report in 1973 [15] depicted these opinions in a rather pessimistic view of AI, causing funding cuts, the most severe of which being the British government's decision to end support for relevant research in most universities of the country. According to the report, AI researchers had failed to address the combinatorial explosion when solving problems within real-world domains. That is, the report states that AI techniques may work within the scope of small problem domains, but the techniques would not scale up well to solve more realistic problems.

1.1.2.2 *AI Winter II, 1988-1993*

After the exaggeration of pessimism that caused the first AI winter, AI makes a come back with more enthusiasm in the community than ever. This is mainly attributed to the rise of *expert systems*, which were very rapidly adopted by corporations around the world in the early 1980s. In an effort to replace human experts, expert systems are computers emulating the human decision-making abilities [16]. They do so by reasoning through *knowledge bases* that represent facts and rules, mainly in an if–then structure, rather than through conventional procedural code.

Already by the first half of the decade, the use of expert systems was calculated to have saved tens of millions of dollars to the companies who employed them, causing more and more to start developing and deploying them. A new industry grew up to support expert systems, including both software and hardware companies, who built specialized computers optimized for the LISP programming language. Soon, the AI industry boomed from a few million dollars in 1980 to billions of dollars at the end of the decade [17].

Though, the extremely high cost and complicated architectures required by the expert systems led some companies to start seeking for alternative solutions. As a result, simpler and cheaper workstations appeared and LISP was soon adjusted in order to be portable to all UNIX systems allowing desktop computers to offer a simpler and more popular architecture to run LISP applications on. This in combination with the very expensive maintenance of expert systems made many companies to abandon the field, leading the billion-dollar AI industry to a new collapse. Once again, the hype of AI had taken over and was succeeded by disappointment.

1.1.2.3 *Post-AI Winter, 1994-2011*

After the successive AI winters, the field kept moving forward with small but steady steps. AI found purpose as part of larger systems, assisting in sub-tasks necessary to carry out the overall operation. A lot of applications had silently embedded cutting edge - for the time - AI or other technologies that have developed from subdivisions of it, like fuzzy logic controllers that were used in automobiles.

The "cold" past of AI though is still haunting it and its reputation remains a midsummer night's dream. It is very often for the researchers in 1990s and

2000s to deliberately call their work by other names, such as cognitive systems, computational intelligence, etc., in order to avoid the stigma that was still present in the field. This was also an effort to emphasize that their work aimed at a particular sub-problem and not a global intelligent solution, as AI had already failed to reach these false promises of the past.

In general, during the post-winter period, AI did not stop to progress and was integrated in real-world applications, breaking down the myth that AI had failed. Computer scientists and companies may have avoided the term in an effort to seek for funding and not be viewed as dreamers but this let huge investments to take place at both state and private level, which by the end of the decade had reached again an amount of magnitude of billions in the USA and the European Union. People have also started to find again meaning in AI after the success of practical applications in fields like language translation and image recognition, which prove that AI can become a powerful tool.

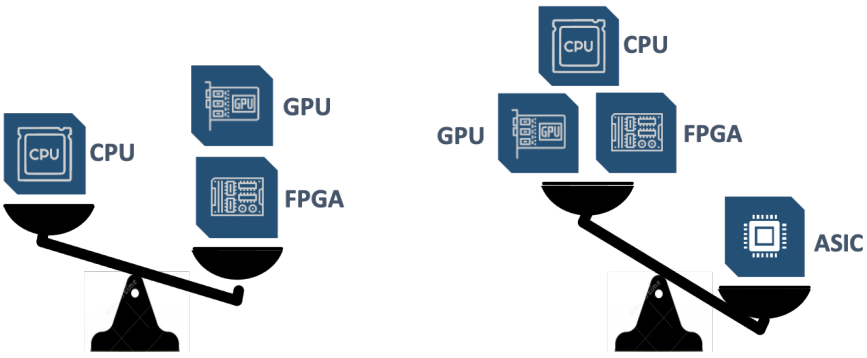
1.1.2.4 *AI Spring, 2012-now*

Already by the mid-2000s the scene of AI has started to heat up, getting ready for the upcoming "AI spring". Starting from 2012, the interest in the field of machine learning is substantially increased from both the research community and the corporate world, especially thanks to the revolution of deep learning. AI starts to develop more rapidly than ever establishing itself in general everyday-life applications, like language translation and search engines, as well as giving other sectors new aspects, like the car industry where the deployment of AI gave birth to the autonomous cars.

Although the advances occurred since the beginning of the AI spring seem to live the AI winter far behind in the past, concerns are still raised occasionally that a new winter could await due to overly ambitious or unrealistic promises given by scientists and commercial vendors. On the contrary, AI has strongly rooted in society and has found itself a purpose. Having said that, new hypes may arrive causing ups and downs in the interest to AI and its reputation, however, a collapse of the field seems now quite an impossible future for the technology. Of course, an "AI summer", where AI is deeply established in a way that people depend on it and is really intelligent, may still be very far away and the path towards it looks challenging and tough but at the same time it is more than fascinating to experience its true potentials being unfolded.

1.1.3 *Breathing New Life into AI*

Reviewing the history of AI shows that its potential was understood very early. Besides, this is why it soon became a hype and over-enthusiasm was formed around it. But what were the reasons behind the failure of AI to fulfill people's expectations and in fact twice? There are many factors that played a major role and led to this but there are some that are attributed to the technology's immaturity and inefficacy to serve the needs of AI. Amongst them, the following two can be distinguished:



(a) CPU vs. GPU/FPGA (b) CPU/GPU/FPGA vs. ASIC

Figure 1.6: AI hardware accelerators comparison.

- insufficient computing capability;
- inadequately large data and wide access to it.

1.1.3.1 *AI Hardware Accelerators*

The immense complexity that lies within the highly distributed processing of information required by AI algorithms, makes traditional computer hardware, such a Central Processing Unit (CPU), look totally powerless. It is true after all that fairly simple neural networks require significant computing capacity even by today's standards.

The AI community is forced to look for and employ other means that would allow for an effective realization of ANNs. In an effort to do so, a new purpose is given to an already used computer component, the Graphics Processing Unit (GPU). It is made obvious that if the parallelization characteristics of GPUs are exploited, then the processing of an ANN, and more particularly the training procedure, would be significantly accelerated.

AlexNet [10] proves this theory in practice and in great extend when the DNN utilizes a GPU to achieve the high performance that was a prerequisite because of the network's great depth. However, although AlexNet is considered the precursor of deep learning, it was not the first neural network to employ a GPU. Already since mid-2000s this technique has been applied [18]–[20] to speed up the calculations of ANNs, mainly CNNs, outperforming conventional processors.

There are many strong points of GPUs that make them are the perfect candidate for AI processing. One of them is their ability to access memory in a 2-D manner, i.e., row access and column access to memory take the same amount of time, even with very large matrices containing several thousand rows and columns [19]. Another advantage is that GPUs contain a huge amount of processing cores that allows for a very large scale of process distribution, which is also a common characteristic of ANNs where each neuron is a separate processing element and completely independent from the rest of the neurons in the same layer. In other words, the enormous matrix Multiply-and-Accumulate (MAC) operations needed for the calculation of a layer's activation, can be carried out in parallel if we exploit the full capabilities of GPUs.

Despite their advantages, GPUs remain a general purpose processing unit. Therefore, a logical question that is raised in the effort to accelerate AI computations, is why not design a custom hardware architecture to perform the processing in the optimal way possible. The answer is simple and is no other than complexity of realization. As noted in [18] in 2005, using dedicated hardware to do machine learning typically ends up in disaster because of cost, obsolescence, and poor software.

A few years later this belief is reassessed as the maturation of Field Programmable Gate Arrays (FPGAs) during the 2010s transcends the aforementioned difficulties and makes dedicated AI hardware feasible in an elegant way. FPGAs soon begin to be used more and more often for AI acceleration until the problem of hosting the training and the inference of a DNN has become as simple as flushing a pre-designed hardware Intellectual Property (IP), many of which are offered as a publicly available open-source implementation.

FPGAs, as re-configurable devices, are by definition generic, so that their hardware components can be rearranged to implement a given circuit. This holds them back from fully leveraging the potentials of dedicated AI hardware. Considering also the advancements in Very Large Scale Integration (VLSI) Integrated Circuits (ICs), where integration has reached the scale of one-digit nanometer architectures, the way is paved for the implementation of Application Specific Integrated Circuit (ASIC) AI accelerators. The cost-energy-performance trade-off is increased over one magnitude with custom chip solutions, which is made evident by the relevant innovation proposed by leading companies and universities in the field in the late 2010s, such as Google's Tensor Processing Unit (TPU) [21], Apple's Neural Engine, and MIT's Eyeriss [22].

1.1.3.2 *AI at the Edge*

Beyond speeding up training and inference on complex DNN models, there is also an incentive to design ASIC AI hardware accelerators that can fit inside the resource-constrained Internet-of-Things (IoT) edge devices. IoT is an example of edge computing, which is to push the computation, or specifically for AI the execution of AI algorithms, from the cloud closer to the user, i.e., to the edge.

AI at the edge holds so much promise because it can be applied to practically any electronic device, from self-driving cars that see pedestrians in the road to coffee makers that respond to voice commands. Motivated by concerns of availability, privacy, latency, network bandwidth, low power, and low cost, applications that require any combination of the above will eventually migrate to AI inference at the edge.

Opposed to the mainstream approach that wants AI to run in the cloud on giant server farms of GPUs and FPGAs, or clusters of them, the key point of AI at the edge is that all the processing is restricted locally to the hosting device, which allows for plenty of improvements. The most important of them are listed below:

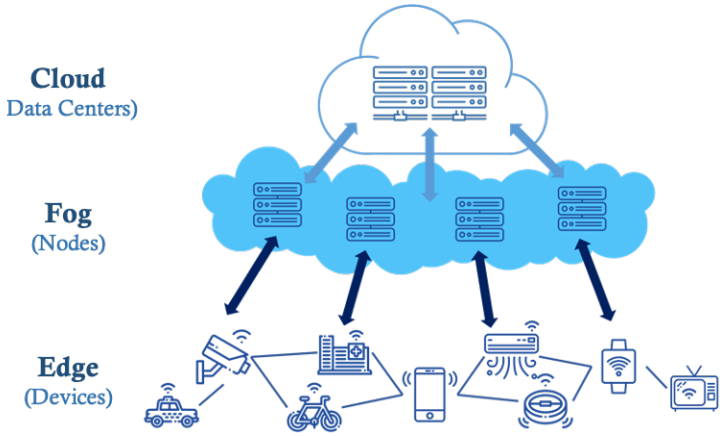


Figure 1.7: The edge computing infrastructure.

- ***Availability***: The system is continuously available since no external factor, such as an internet connection interruption, can affect it and therefore undermine its operation.
- ***Privacy & Security***: Sensitive data are persisted at a local level and thus they do not have to be sent over the internet, ensuring the privacy and enhancing the security of the application.
- ***Speed***: The overall application time is shorter as there is no need to wait for the data to be sent in the cloud and then receive back the results, resulting in low latency.
- ***Bandwidth***: The off-line processing does not occupy any bandwidth, allowing this way for more devices to be simultaneously connected.
- ***Energy consumption***: A significant amount of energy is saved after reducing the power-hungry data transfers that are needed.

The large and power-hungry GPUs and FPGAs are the worst match for the small battery-powered IoT devices. On the contrary, ASICs can become very small and achieve very low energy consumption, which makes them the ideal choice if the aforementioned advantages are to be achieved.

1.1.3.3 *Big Data*

The bloom of the digital era, brought new perspectives for AI into light. The information storage switched from analog to digital, with the latter growing exponentially in capacity, which by the end of 2000s had almost completely dominated the global market. Not only this but the generated data met an exponential growth as well. As of 2012, 2.5 exabytes (2.5×2^{60} bytes) of data were generated daily. This number continued only to grow, and as the International Data Group (IDC) reported, the global data volume grew from 4.4 zettabytes (4.4×2^{70} bytes) to ten times higher between 2013 and 2020. By 2025, IDC predicts that there will be 163 zettabytes of data around the world. This explosion

is mainly a result of data collected by devices such as mobile phones and information-sensing devices, which already count tens of billions worldwide, far outnumbering the human population of the earth.

Of course the size of the data alone is of no importance. What differentiates the *big data* is a combination of characteristics, or the "5 Vs":

- ***Volume***: The quantity of generated and stored data. It determines the value and potential insight.
- ***Variety***: The type and nature of the data, i.e., structured, semi-structured, unstructured.
- ***Velocity***: The speed at which the data is generated and processed.
- ***Veracity***: The truthfulness or reliability of the data, which refers to the data quality and the data value [23].
- ***Value***: The worth in information that can be achieved by the processing and analysis of large datasets.

Big data is comprised by data sets that are too large and complex to be dealt with traditional software tools within a tolerable elapsed time. The insights embedded in such data sets, that are diverse, complex, and of a massive scale, can become a source of knowledge if extracted properly. A technique that helped in this is machine learning.

It is no coincidence the fact that the first large data sets, like the ImageNet, begin to pop up from the late 2000s. It was not until then that the digital world was ready enough to qualitatively capture, widely distribute, and efficiently process big data. The great statistical power offered by the huge number of fields (rows) of big data sets, enabled DNN architectures to be trained efficiently, leaving behind problems caused by inadequate data sets, like *over-fitting*. In fact, it would never have been possible for deep architectures to work without setting their depth according to the size of data from which they extract information.

1.1.4 *Neuromorphic Engineering: An Artificial Brain?*

It has always been a challenge for technology the way in which animal brains engage with their world effortlessly. Despite the technological advancements in computer science, electronics, machine learning, and other relevant fields, a system with skills comparable to those of the simplest intelligent being still seems unreachable. It is true that brains far outperform computers across a wide spectrum of tasks, particularly from the aspect of power consumption. A bee, for instance, demonstrates remarkable navigational and social capabilities at the same time that searches for nectar, and all this is the result of the power of less than a million neurons, burning less than a milliwatt. This performance is many orders of magnitude more task-competent and power-efficient than current neuronal simulations or autonomous robots [24].

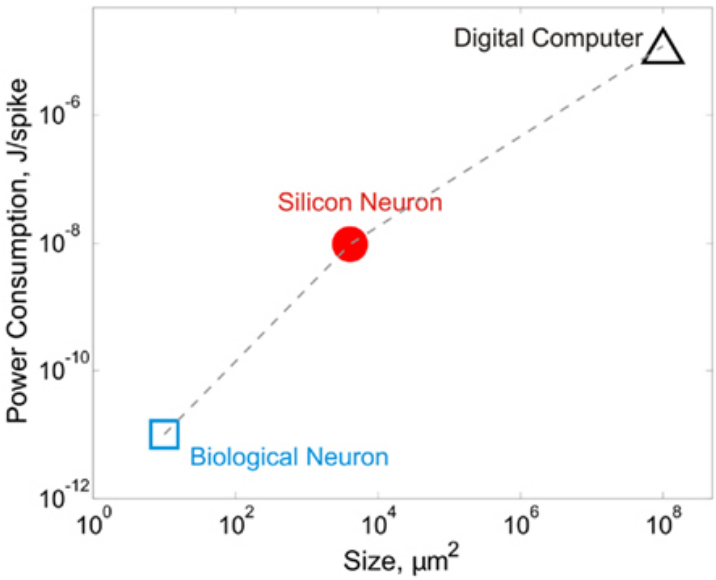


Figure 1.8: Comparison of power-area performance among biological neurons, silicon neurons, and digital computers [29].

In an attempt to mimic the operation of neural computing systems, neuromorphic engineering was introduced by Carver Mead in the late 1980s [25]. Neuromorphic engineering studies the use of electronic neural networks whose architectures and operations are based on those of biological nervous systems [26] and, unlike in the classic von Neumann architectures, the computational load is distributed among a multitude of artificial neurons.

Similar to biological cognitive systems, a real-time response is a fundamental requirement in neuromorphic engineering. Even with the current supercomputers though, simulating a large network of a neurobiological system is unrealistic in terms of time [27]. For example, a 2009 "cat-scale" neural simulation of 1 billion neurons and 10 trillion synaptic connections ran 700 times slower than real time, while burning about 2 megawatts [28]. Hence, the inefficiency of digital computers to compete with neurobiological systems, led to the advent of analog VLSI implementations of silicon neurons. As shown in Fig. 1.8, the power-area performance of silicon neurons lies between this of biological neurons and digital computers [29].

In his original contribution to the evolution of neuromorphic engineering, Mead emphasized that the physics of neural computation remain analog, rather than digital. Thus, instead of operating the transistors as on-off switches, he exploited their analog properties when operating in the sub-threshold region. This way, Complementary Metal-Oxide-Semiconductor (CMOS) transistors consume only a low level of power and also their output current vs. input voltage characteristic is similar to the sigmoid input-output characteristic of the neuronal ion channels [29].

Today's neuromorphic ICs include various digital and mixed-signal designs, too, rather than purely analog ones as in the beginning of neuromorphic engineering. Some of the most famous chips include IBM's TrueNorth [30], Intel's

Loihi [31], and ETH's ROLLS neuromorphic processor [32]. These chips offer compact designs that incorporate large numbers of neurons and synapses to implement Spiking Neural Networks (SNNs) with a very low power consumption. For example, TrueNorth offers up to 1 million neurons and 256 million synapses with a power density of $20mW/cm^2$. However, their user base still consists mostly of universities and industrial research groups [33]. It is also worth mentioning SpiNNaker [34], a massively parallel, multi-processor architecture for modeling and simulating large-scale SNNs.

Although many improvements are offered by neuromorphic engineering, it is not yet as mature as the AI hardware accelerators for conventional ANNs. Before industry is ready to adopt neuromorphic technology and employ it in the field, many obstacles would have to be overcome. That involves progress in the field of SNNs as well as the capture of data in neuromorphic, or spiking form. The specific advantages of SNNs, along with the challenges they face, will be further discussed in Chapter 2. The one thing that is for sure though is that the arrival of an *artificial brain* is not foreseen in the close future.

1.2 CRITICALITY OF AI APPLICATIONS

With AI expanding its applications in increasing speed, more and more fields employ it in order to improve the quality of the provided services and consecutively the quality of life. These services can be categorized in a wide range, sometimes involving entertainment purposes or simplification of a fairly simple task. In this case the outcome of an AI application can be described as "innocent", since it can have no impact on the users and their environment. If someone in France, for example, asks the AI assistant of their phone for an Italian restaurant and the latter replies with a restaurant located in Italy, then no one would take the next plane to go there. Even at the extreme scenario that this happened, no harm no foul.

On the contrary, if the AI-powered autopilot driving an autonomous car misinterprets a pedestrian with a dove, then this is serious. It could harm irreversibly not only the passengers of the car but also the people nearby, resulting in an accident with human and material damages. Hence, the evident need of being well prepared for critical situations is an integral parameter of an AI system in order to avoid the worst ethical consequences and ensure safety.

1.2.1 *AI Hardware in Safety-Critical Applications*

In the near future, many safety- and mission-critical applications are expected to materialize and exploit the advantages of AI hardware. In this context, the killer application is autonomous-driving vehicles, whose safety needs to be ensured before anything, even in the case where a system has broken down. There are already standards that take care of that, which also evolve and get adjusted in order to suit the technological advancements of the market. In the case of automotive there is the ISO 26262, titled "Road vehicles – Functional safety"

[35], which was proposed as an adaptation of the functional safety standard IEC 61508 [36] in 2011 and was revised in 2018 to include all type of vehicles.

As made evident by the title of the standard, its goal is to set the metrics for *functional safety* in automotive applications. To do so, it provides a vocabulary of definitions, with particular emphasis on the terms of *fault*, *error*, and *failure*. It considers that "A fault can manifest itself as an error ... and the error can ultimately cause a failure" [35]. As demanded by the standard, there must be no more than 0.2 failures per billion operating hours. This might sound strict but realizing the extend of the number of cars being driven every second in the streets, then it is easy to reconsider the strictness of the rule.

More specifically, *functional safety* is defined as the absence of unreasonable risk due to *hazards*, which are potential sources of harm, i.e., physical injury or health damage, caused by a *malfunctioning behaviour* of the system. The term *malfunction* corresponds to a terminated or unintended behaviour with respect to the system's design intent. In the case of termination of an intended behaviour of a component or system due to a *fault* manifestation, there is a *failure*. A *fault* is any abnormal condition that can cause a component or system to fail and a system is *fault tolerant* if it is able to deliver a specified functionality in the presence of one or more specified *faults*. Finally, an *error* is simply a discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition.

In other words, *functional safety* is reduced by a resulting *malfunction* that has a hazardous effect. Keeping that in mind, equipping AI hardware with Built-In Self-Test (BIST) mechanisms to regularly monitor the system during its normal operation becomes essential in order to detect *faults* in real time before they cause a *failure* or a *malfunction*, so that any consequent *hazard* is avoided.

1.2.2 *Reliability Concerns over AI Hardware*

The obscurity in the operation of the majority of AI systems has raised concerns over its integration into our lives, questioning this way its trustworthiness. Numerous are the examples that amplify the major preoccupation nowadays about whether AI systems are to be trusted. Since the beginning of allowance of tests involving autonomous vehicles on public roads in the mid-2010s, there have been counted plenty of incidents.

The high peak of accidents was a fatal one that costed the life of Elaine Herzberg - the first person to die by a self-driving car - in 2018. The woman was pushing a bicycle across a four-lane road in Tempe, Arizona, United States, when she was struck by an Uber test vehicle. The AI system of the car spotted and recognized successfully that someone was crossing the road but the low confidence of the system for such an unlike event was not enough to trigger the decision threshold and thus it was discarded. The human safety backup driver sitting in the driver's seat was also incapable of avoiding the collision from happening and was later found guilty for negligent homicide.

Considering all the possible things that can go wrong during the operation of any AI system, no matter the field of application, it is easy to see that incidents like the aforementioned one are only the tip of the iceberg. The concerns cannot but increase when taking into account other potential safety threats that can potentially result in a malfunctioning system, like unfair bias and adversarial attacks. But first and beyond all, there is an urgent need to focus on the *dependability* of AI hardware, in a sense that the system constantly operates as it was indented to by its design, which has been customarily overlooked so far.

A first logical thought that comes to mind is that SNNs are already fault tolerant, as they derive from biological neural networks. The brain is known to be capable of tolerating a finite number of faults in the neurons and synapses and even able to regenerate or rewire network elements to make up for a larger damage. Modeled after the immensely parallel architecture and operation principles of biological neural networks, SNNs indeed inherit some of the remarkable fault-tolerance capabilities of their biological counterparts. Moreover, it is frequent to contain more computational units than the minimum requirements of a certain cognitive task, a property known as over-provisioning [37], which adds a certain degree of robustness [38].

However, arguing that this assumption is true based only on architecture resemblance to biology or over-provisioning is somewhat imprudent. In fact, electronics do not share the same principles of operation as biology and hence hardware-accelerated neural networks are vulnerable to hardware-level faults, which may result from manufacturing defects, process variations, aging, and Single-Event Upsets (SEUs). To this end, the cognitive capabilities of a neural network can become quite fragile, risking the degradation of the network's performance due to constraints and imperfections of the VLSI technologies.

1.3 MOTIVATION

With the foreseen industrialization and high-volume production of AI hardware in the coming years, special attention must be attributed to the reliability and functional safety of neuromorphic systems, particularly when safety- and mission-critical applications are concerned. The safety standards declared by the industry, e.g., ISO 26262, help in regulating the requirements for avoiding unreasonable risks but putting them into effect is a challenging process and not so straightforward.

To do so, there is a need for exploration of the yet overlooked domains of reliability and functional safety of AI hardware [39]. There are many reasons that can cause faults to occur in a hardware accelerator. Thus, knowing how these faults are manifested abstractly on behavioral level of the circuit, it is an essential tool for understanding how the performance is affected. Then, measuring the performance degradation allows for a quantification of the inherent fault tolerance capabilities of ANNs, which also enhances the explainability of the system.

Hardware testing is a well studied domain of electronics. Particularly for digital designs, BIST mechanisms are standardized with many online and offline schemes proposed over the years to test a circuit. The error-correction part is also well covered by algorithms and mechanisms that compensate for the errors caused by faults, usually transient. However, testing efficiency and fault tolerance can be largely improved if the architectural particularities of hardware neural networks are exploited, targeting only those fault scenarios that have a measurable effect on performance [40].

Concerning SNNs, which are the focus of this thesis, there is even a larger gap in the literature. Mainly because of the difficulties that the networks of this type express in their training, coding schemes, data representation, etc., they have met slower progress although they exist for many decades. It is usual to make adaptations of architectures and algorithms meant for conventional ANNs in order to aid in the faster integration of SNNs. However, once again it is not solid to assume that the specific characteristics, and particularly the robustness, of a technique that works for a given type of ANNs will be guaranteed if generalized for SNNs.

To this end, the motivation behind this thesis is first to explore the reliability and functional safety of AI hardware, and also propose fault tolerance strategies that can boost the dependability of the related systems. With the focus set on SNN applications, this thesis aims at unraveling the vulnerabilities of this type of neural networks and the corresponding neuromorphic hardware designed to host their computations, in order to contribute in the better understanding of their robustness and provide a know-how for augmenting it.

1.4 TOWARD TRUSTWORTHY AI

Trust in development, deployment and use of AI systems is a prerequisite for people and societies, without which unwanted consequences may ensue and the uptake of AI might be hindered. Hence, to truly enable the vast socioeconomic benefits that these systems can bring, it is essential for them to be trustworthy. In an effort to shape the foundations of trustworthy AI, the European Union proposed the *Ethics Guidelines for Trustworthy AI* in 2019 [41].

The key point toward this direction is to think of AI as a human-centric means to increase human flourishing. Its only and one purpose is to serve humanity and the common good, with the goal of improving human and society welfare through progress and innovation. Without this principle in mind, AI systems could give rise to certain risks, threatening in this way fundamental values and rights, like individual freedom, personal integrity, and equal access.

1.4.1 *A Multi-Dimensional Problem*

A balanced and prosperous assimilation of AI entails seeking to maximize the benefits of AI systems while at the same time preventing and minimizing their risks. The latter depends on various factors and therefore it consists a multi-

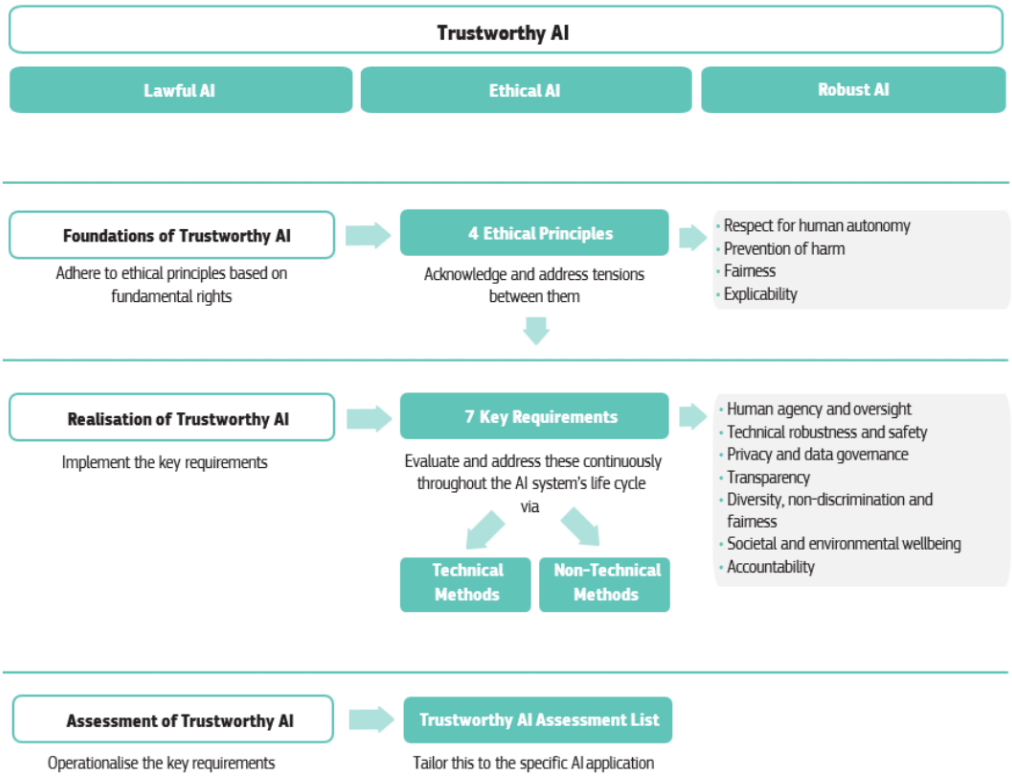


Figure 1.9: The framework for trustworthy AI as proposed by the EU [41].

dimensional problem. The achievement of trustworthiness in AI can be broken down to three components, which should be met throughout the system's entire life cycle, and each of which is necessary but not sufficient in itself. More specifically, an AI system should be [41]:

1. ***lawful***, complying with all applicable laws and regulations;
2. ***ethical***, ensuring adherence to ethical principles and values;
3. ***robust***, both from a technical and social perspective since, even with good intentions, AI systems can cause unintentional harm.

Fig. 1.9 demonstrates the above as part of the framework proposed by the EU that consists of the foundations, the realization, and the assessment of trustworthy AI.

1.4.1.1 *Lawful AI*

Operating in a lawful world, AI systems cannot be the exception to any law and thus they need to abide by all the laws of the society that surrounds them. If the rules are respected, the first step is set in order to gain the trust of the system's users, as any malignant action against persons, their well-being, or their interests is ensured to be avoided. AI should not in any way be used as a means to undermine people's rights, harm them, or profit against them.

The law provides obligations not only with reference to what cannot be done, but also with reference to what should be done and what may be done. Apart from prohibiting certain actions, it designates the good practices to address areas like data protection, non-discrimination, freedom of the arts and sciences, etc.

1.4.1.2 *Ethical AI*

It is quite common that ethics fall within the jurisdiction of law, as many ethical principles are already to a large extend reflected in existing legal requirements. However, there is not always a clear line separating what *should* be done from what *can* be done with technology, allowing for cases where ethical norms are not necessarily legally binding yet crucial to ensure trustworthiness.

As with any powerful technology, AI poses ethical challenges, since it inevitably makes an impact on people and society. Concerns are raised on whether the good life of individuals is threatened in terms of quality of life or human autonomy and freedom. And this is where AI ethics enters the equation so that to identify how AI can halt these concerns and ensure the fairness of AI systems, which need to be in line with ethical values. Of course, this cannot function as a substitute for ethical reasoning and so an ethical mind-set is still something we need to build and maintain through public debate, education and practical learning.

Having as a base the human dignity, a human-centric approach of AI should respect human fundamental rights with goal to improve individual and collective well-being. Fig. 1.9 presents four ethical principles that, if adhered by AI systems, can set the foundations for a trustworthy AI. Non-hierarchically, these principles are:

- ***Respect for human autonomy***: Human choice must be firmly preserved, allowing the humans interacting with AI systems to keep full and effective self-determination over themselves and be able to partake in the democratic process.
- ***Prevention of harm***: Human dignity as well as mental and physical integrity must be protected, so that AI systems never adversely affect in any way human beings, the natural environment, and all living beings, e.g., neither cause nor exacerbate harm.
- ***Fairness***: Individuals and groups should be free from unfair bias, discrimination and stigmatization. AI systems should ensure equal and just distribution of both benefits and costs, while balancing competing interests and objectives. Fairness also entails the ability to contest and seek effective redress against decisions made by AI systems.
- ***Explicability***: Processes need to be transparent, the capabilities and purpose of AI systems openly communicated, and decisions explainable. This is also essential to prevent an AI system performing a critical application

from outputting an erroneous or inaccurate result that may lead to severe consequences.

It is possible that some of the above principles can be in conflict with some others. For example, AI systems used for *predictive policing*, they can help reduce crime (principle of prevention of harm), but in ways that entail surveillance activities that impinge on individual liberty and privacy (principle of human autonomy). Hence, there is not always a right solution based on these principles. Because of this, ethical dilemmas should be approached via reasoned reflection based on evidence rather than intuition.

The principles outlined previously must be translated into concrete requirements to achieve trustworthy AI, whose implementation should occur throughout an AI system's entire life cycle and depends on the specific application. Building on the four ethical principles, the realisation of trustworthy AI entails the following non-exhaustive list of key requirements without imposing any hierarchy as defined by the EU [41] and depicted on Fig. 1.9:

- ***Human agency and oversight***, including fundamental rights, human agency and human oversight;
- ***Technical robustness and safety***, including resilience to attack and security, fall back plan and general safety, accuracy, reliability and reproducibility;
- ***Privacy and data governance***, including respect for privacy, quality and integrity of data, and access to data;
- ***Transparency***, including traceability, explainability and communication;
- ***Diversity, non-discrimination and fairness***, including the avoidance of unfair bias, accessibility and universal design, and stakeholder participation;
- ***Societal and environmental well-being***, including, sustainability and environmental friendliness, social impact, society and democracy;
- ***Accountability***, including auditability, minimization and reporting of negative impact, trade-offs and redress.

To implement the above requirements, both technical, e.g., proposed architectures, testing and validating, and non-technical methods, e.g., regulation and standardization, can be employed during all stages of an AI system's life cycle, accompanied by an evaluation of these methods on an ongoing basis in order to update the implementation processes accordingly. Finally, the realization of trustworthy AI is a continuous process, as AI systems are evolving and acting in a dynamic environment.

1.4.1.3 *Robust AI*

Complying with the law and being in line with the ethical principles are not enough for an AI system to be worthy of trust by individuals and society. It

is also necessary that AI systems are robust, or in other words designed in a way such that they reliably behave as intended while minimizing unintentional and unexpected harm and preventing unacceptable harm. Technical robustness and safety are closely linked to the principle of prevention of harm, thus, it is evident that ethical and robust AI are intertwined and complement each other.

A reliable AI system needs to work properly with a range of inputs and in a range of situations. It is also critical that the results of the system are reproducible, so that to verify whether an AI experiment exhibits the same behavior when repeated under the same conditions. This helps in understanding and describing what an AI system does.

Another aspect of robustness is accuracy, which is the AI system's ability to make correct judgements depending the form of its application, e.g., correctly classifying information, making correct predictions, decisions, etc. A well-formed evaluation process can mitigate the unintended risks deriving from potential inaccurate predictions.

Vulnerabilities are innate to AI systems, like in all software and hardware systems. Therefore, AI systems should be protected against adversaries trying to exploit vulnerabilities with an ultimate goal of undermining the right operation of the system causing corruptions in the data, model or infrastructure. For an AI system to be considered secure, steps should be taken to prevent and mitigate any malicious intentions for erroneous decisions or even physical harm.

Finally, AI systems should be equipped with safeguards that enable a fallback plan in case the system is facing a problem, so that the system will continue operating properly without putting at risk living beings or the environment. In the scenario that this cannot be ensured by a predefined procedure, the system should halt its operation and ask for a human intervention. It is also important to clarify and assess potential risks a priori, so that the AI system is prepared for various scenarios.

1.4.2 *Robustness through Dependability*

In the previous section, robust AI was described in an abstract level. Ensuring the robustness of an AI system can become a complicated procedure, thereby a more solid definition is necessary in order to accurately measure it. For this reason, *dependability* is employed as a means to define and measure the robustness of a system in a more accurate way. Dependability is a broad term used to define the ability of a system to deliver its intended service.

As described in Section 1.2.1, the service of the system is subject to intrinsic, e.g., faults, or extrinsic effects, e.g., adversary attacks, that can lead to its malfunction or failure. Dependability of a system is the ability to avoid such service deviations from the intended one that exceed an acceptance level throughout the life cycle of deployment.

There is a wide spectrum of attributes encompassed within dependability, some of which were described in Section 1.4.1.3. Here, a more thorough defini-

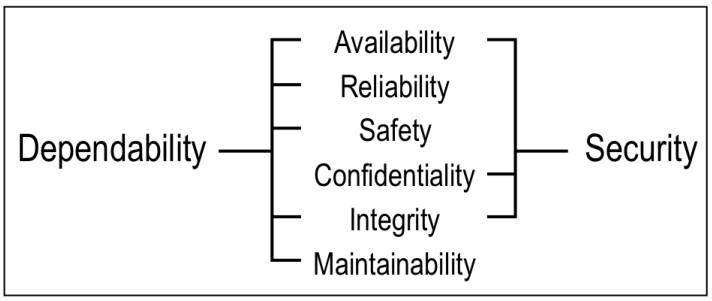


Figure 1.10: Dependability and security attributes of a system [42].

tion is given with quantities to measure dependability from various perspectives. A list of these attributes can be found below [39], [42]:

- ***Reliability*** denotes the continuity of the correct service and its level is commonly specified in terms of *mean time to failure*, MTTF;
- ***Maintainability*** denotes the ability to repair a given service failure and is. commonly specified in terms of *mean time to repair*, MTTR;
- ***Availability*** denotes the readiness for correct service and it can be expressed as a function of reliability and maintainability as $\frac{\text{MTTF}}{\text{MTTF}+\text{MTTR}}$;
- ***Functional safety*** denotes the absence of unreasonable risk due to hazards caused by malfunctions or failures;
- ***Integrity*** denotes the absence of improper system alterations;
- ***Security*** denotes the ability to prevent risks related to malicious intrusions into a system.

When addressing security, an additional attribute has great prominence, *confidentiality*, i.e., the absence of unauthorized disclosure of information. While highly related with dependability, security is often not characterized as a single attribute of dependability. Instead security is considered a composite notion combining the attributes of confidentiality, integrity and availability [42]. Fig. 1.10 summarizes the relationship between dependability and security in terms of their principal attributes.

It is not unlikely that the aforementioned attributes cause some confusion, since they are quite close to each other and all contribute to the dependability of the system. Also, the broadness that characterizes dependability is not easily covered in all its attributes, no matter how extensive this thesis can be. Therefore, the focus, as the title suggests, is on functional safety and reliability of AI systems. In an effort to simplify the above definitions, following are some differences among them [39].

FUNCTIONAL SAFETY VS. (TRADITIONAL) SAFETY

Functional safety is part of the overall safety of a system. The key difference between the two is that functional safety depends on automatic protection

triggered in the event of a failure. This automatic protection needs to respond correctly to its inputs and have predictable responses to failure in a predictable manner (fail-safe). This includes human errors, hardware failures, and operational/environmental stress.

For example, let us consider the protection of a building against fire. Traditional safety would be simply using fireproof materials to construct the building. On the other hand, if a fire protection system is installed, which is an intended function designed for fighting fire, then this is functional safety.

For the sake of completeness, another part of safety is the *safety of the intended functionality*, i.e., the absence of risks caused by performance limitations of the intended behaviors or by reasonably foreseeable misuse by the user.

FUNCTIONAL SAFETY VS. RELIABILITY

Target of reliability is to decrease fault occurrences as much as possible, while functional safety aims at ensuring correct/acceptable behavior of the system despite the presence of a fault. A higher level of reliability implies an increase in the quality of design/production process, e.g., by selecting higher quality components. Functional safety faces "unknown" and "unpredictable" situations, thereby it must be conservative, e.g., functional safety standards require less strict failure rates with respect to the numbers used for reliability computation.

FUNCTIONAL SAFETY VS. AVAILABILITY

It is no surprise that functional safety is in contradiction with availability, since the two can have different targets. For instance, in the extreme case, the safest car is the one that always stays in the garage and is never driven. Theoretically, this way the safety is "maximized" but availability is absent as the "correct service" implied by the latter is never in use, accepting even a broken, unusable car.

It is evident that usually there is a trade-off between the two attributes, which should be carefully decided in order not to affect availability but at the same time not undermine the safety of the system. An over-sensitive safety device could cause *nuisance trips*, which is when no real danger exists but the safety mechanism is triggered.

FUNCTIONAL SAFETY VS. SECURITY

While security aims at protecting the system from unintended or unauthorized access, change or destruction, such an unwanted scenario could seriously jeopardize the operation of the system and consequently lead to a potential cause of danger, risk or injury to people or the environment by the system, which is what functional safety aims at preventing. Therefore, it is clear that safety and security are interlinked; there is no safety without security and vice-versa.

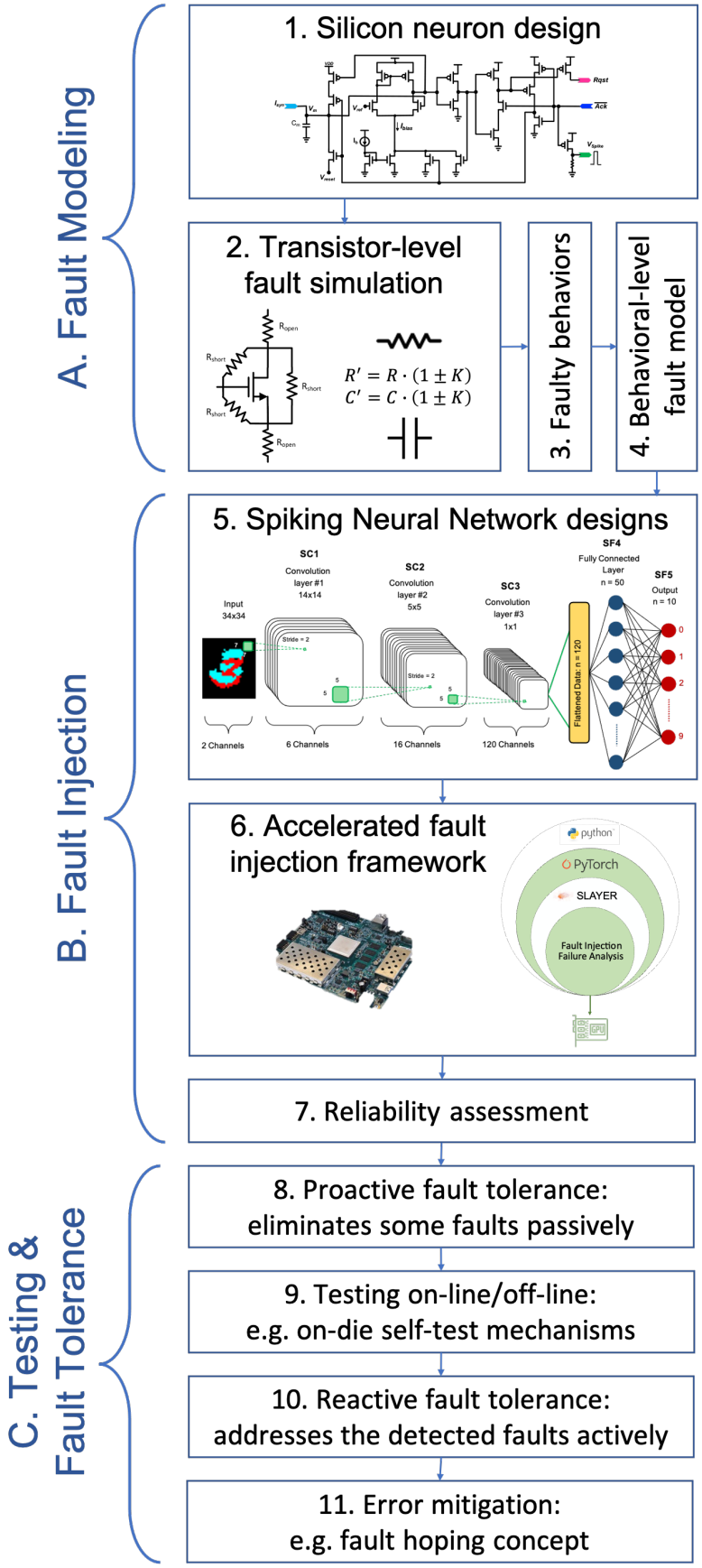


Figure 1.11: Proposed methodology to ensure the reliability of SNNs.

1.5 RESEARCH METHODOLOGY

The reliability and the functional safety of a system are implicitly linked with the capability of the system to withstand faults, i.e., the *fault tolerance* of the system. As mentioned in section 1.2.1, for a system to be fault tolerant, it must be able to deliver its functionality no matter the presence of faults. In this way, both the continuity of the correct service (reliability) and the avoidance of unreasonable risk (functional safety) are ensured.

Throughout this thesis, the work has been focused on building a framework that will allow SNNs to be fault tolerant. Fig. 1.11 summarizes our proposed methodology, which has been validated on SNNs being simulated in software and running on actual neuromorphic hardware. The methodology is separated in three main parts as shown in Fig. 1.11, namely *fault modeling*, *fault injection*, and *testing and fault tolerance*.

1.5.1 *Fault Modeling*

The first step is to know what can go wrong within the neuromorphic processor. To do so, we perform detailed fault simulations at transistor-level on the components consisting it, e.g., on the spiking neuron design. We then exploit the faulty behaviors of the system, that resulted from the fault simulations, in order to produce a behavioral-level fault model and a fault taxonomy specific to SNNs.

For the derived fault model to be efficient, there are some requirements needed to be satisfied, so that the fault model is:

- ***consistent*** with manufacturing defects and faults that can happen in the field such as aging effects and soft errors;
- at ***behavioral level*** to allow for large-scale fault injections in deep networks and simplify their fault simulations;
- ***abstract***, meaning it can be used to test any SNN regardless of its architecture or implementation.

1.5.2 *Fault Injection*

Using the derived behavioral-level fault model, faults are injected into SNN designs across all layers of the networks. The fault injection experiments are automated by simulating the faulty instances of the networks in our accelerated fault injection framework, or by running them on actual hardware in our neuromorphic experimentation platform. The goal is to monitor the effects of the injected faults on the networks under study, allowing for a reliability assessment that exposes the vulnerabilities of the networks and provides knowledge of how the system will operate when being faulty.

The reliability assessment of a SNN can be a helpful tool so as to:

- determine the ***severity*** of the fault types, e.g., catastrophic, benign;

- pinpoint the ***critical parts*** of the network to allow for a more targeted hardening;
- understand the ***propagation*** of faults across the layers of the network.

1.5.3 *Testing & Fault Tolerance*

The findings of the reliability assessment are used to develop cost-effective fault tolerance strategies that focus on the critical parts of the design. The fault tolerance strategies are composed of an initial passive part and a second active part. The purpose of the passive fault tolerance is to eliminate and nullify the effect of some of the faults proactively. The rest of the faults that persist after the passive part are addressed reactively by an active fault tolerance technique. This requires the presence of on-die BIST mechanisms that can operate either on-line by monitoring the operation of the system's components, or off-line by performing periodical tests. If a malfunction is detected, the error mitigation mechanisms are triggered in order to deal with the corresponding fault and avoid any undesired effects on the operation of the system.

1.6 THESIS STRUCTURE

The work carried out as part of this thesis concerns the reliability and the functional safety of SNNs and the neuromorphic hardware used to host their operations. After studying the resilience of SNNs and neuromorphic systems, the goal is to harden them against faults, hence enhancing their fault tolerance. The outcome of the thesis is the methodology described in 1.5 and the implemented work follows the corresponding flow. Before that though, there precedes an introduction to the principles of SNNs in Chapter 2. Also, each chapter is accompanied by a corresponding literature review on the related state-of-the-art contributions where applicable.

Chapter 3 presents a behavioral-level fault model of a spiking silicon neuron design after Monte Carlo (MC) and defect simulations at transistor level. The derived taxonomy of faulty behaviors leads to a fault model that is independent to the used hardware accelerator, thus allowing for large-scale fault injection experiments network-wise. Chapter 4 exposes the critical parts of deep convolutional SNNs by forming a resilience analysis based on the results of the fault injection experiments that were automated and accelerated by our fault injection framework for SNNs in Python.

Despite the fault model being pragmatic in terms of hardware, the simulation so far was performed in software, thus inevitably leaving outside some blocks of a realistic neuromorphic design, e.g., the inter-neuron communication modules. To address this, Chapter 5 introduces a neuromorphic hardware experimentation platform designed for SNNs, which is then used in Chapter 6 in order to conduct fault injection experiments on actual neuromorphic hardware, allowing to shape a more concrete reliability analysis of the respective systems.

Continuing with the testing of neuromorphic circuits, Chapter 7 shows a testing methodology to find a compact and high-coverage functional test-set based on a metric that evaluates the proneness of each sample in the dataset to misclassification.

Chapter 8 leverages the fault resilience results of Chapter 4 to propose a cost-effective fault tolerance strategy consisting of a proactive part, which passively nullifies the effect of some critical faults, and a reactive one, which tests for the rest of the faults with on-line and off-line schemes and then activates the error mitigation mechanism if any faults were detected.

Chapter 9 exploits the results of the reliability assessment of Chapter 6 to propose an on-line testing mechanism that detects in real time abnormal operation of the neuromorphic design.

Finally, Chapter 10 concludes the thesis providing also perspectives and directions for future work.

2 A BRIEF INTRODUCTION TO SPIKING NEURAL NETWORKS

While brain outperforms the most sophisticated architectures of ANNs, it remains a challenge to keep improving AI algorithms. In an effort to bridge the gap between biology and Machine Learning (ML), SNNs emerged bringing the principles of operation of neurobiological systems to AI. SNNs, also known as the *third generation* of neural networks [43], are a special type of neural networks in a sense of information encoding and processing. The neurons comprising a SNN use unit instantaneous signals, called *spikes*, in order to transmit information to their neighbor neurons. Sequences of spikes, or *spike trains*, are then propagated to the network through layers of neurons linked via weighted connections, called *synapses*, in order to encode and process information.

Though the full potentials of SNNs are yet to be exploited, mainly due to the difficulties with training and representing data, AI could reach new high levels. This chapter makes a brief introduction to the basics of SNNs, concerning information processing, learning, and implementation on neuromorphic hardware. At the end of the chapter, a concise comparison between conventional ANNs and SNNs is reported, as well.

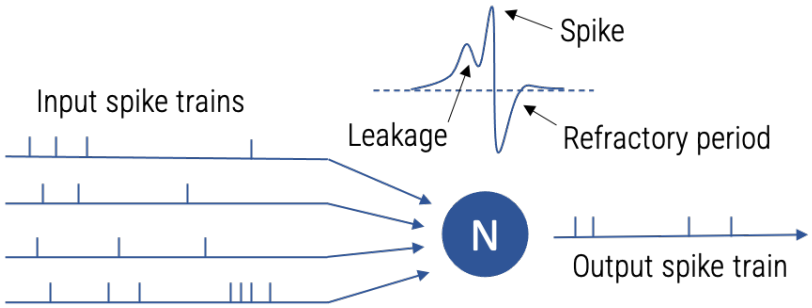


Figure 2.1: Operation of the leaky I&F spiking neuron.

2.1 INFORMATION PROCESSING

The primary means of communication among spiking neurons is a *spike*. Spikes are modeled after the electrical pulses, called *action potentials*, generated and propagated by biological neurons. They are abrupt momentary (~ 1ms) and short (~ 100mV) changes of the state of the neuron, hence expressed as a sum of delta functions, called *spike trains*, $S(t) = \sum_f \delta(t - t^f)$, where t^f represents the f^{th} firing time of the neuron. From this, it is evident that SNNs incorporate the concept of time within their computations, allowing to exploit the temporal characteristics of the data as well. Another powerful feature of spikes is their shape which holds no information at all but instead it is the timing of spikes that carries all the neural information.

Spiking neurons can be seen as simple computing elements which accumulate the incoming spike activity at their input when triggered accordingly. The basic theory wants the inner state, called *membrane potential*, of a spiking neuron to be affected (positively or negatively) when such an event occurs. When a certain *threshold* is reached, the neuron *fires* a spike at its output, which, after weighted by the neuron's post-synaptic connections, or *synapses*, is used to stimulate other neurons.

Throughout the history of neural networks there have been many neuron models proposed after their biological counterparts. The first objective was an accurate simulation of biological neural systems, which led to the creation of biologically plausible models. These models were usually very complex in order to cover all the neural activities. For example, the famous and classic Hodgkin-Huxley model [44] uses four-dimensional nonlinear differential equations to describe neural behavior.

Because of this complexity and the expensive implementation it requires, biologically plausible models gave their place to biologically inspired models, whose target is to replicate neural behaviors without the obligation of emulating the physical activity of biological systems. Therefore, the needed computations are simpler making these models more efficient in modeling large-scale systems. Some examples are the FitzHugh–Nagumo [45], [46] and the Izhikevich models [47].

Yet, the purpose of SNNs as computational units is usually not to simulate the brain activity. Thus, simpler models emerged in order to minimize the

computational complexity but incorporate the neuronal dynamics. The prevalent spiking neuron model and one of the least computationally complex, hence hardware-friendly, is the Integrate-and-Fire (I&F) model [48]. Initially, the neuron is at a *resting state*, where the membrane potential is set to a low value. The neuron integrates the incoming spikes from the synapses at its input and the membrane potential is increased correspondingly. Once the potential exceeds the specified threshold, the neuron fires a spike, which is propagated to the next layer of neurons via the synapses connected to its output, and the neuron is reset to its resting state again. The refractory period is the time in-between successive spikes regulating the maximum possible spiking frequency of the neuron. An extension to the aforementioned model is the leaky I&F neuron, where the potential is periodically brought closer to the resting state during the idle time of the neuron, so if there are no incoming spikes, the neuron is gradually reset. The operation of the leaky I&F neuron is demonstrated in Fig. 2.1. There exist also generalized versions of the I&F neuron, like the Spike Response Model (SRM) [49] which performs the neural behavior in the form of response kernels, so that the model is adjustable to accommodate for computational needs and precision depending on the application.

Similarly to the conventional ANNs, both feed-forward and recurrent topologies of SNNs exist. In a feed-forward SNN, spikes propagate in only one direction, from input to output. Addition of feedback loops allows the spikes to flow in both directions, giving new properties in recurrent neural networks such as associative memory and context-dependent pattern classification like speech recognition [27]. Hybrid networks containing both strictly feed-forward sub-populations along with recurrent ones are also possible. Interactions between the sub-populations may be one-directional or reciprocal [50]. One of the most popular hybrid SNNs is the *synfire chain*, a mechanism for representing relationships between delayed events [51], so as to mimic the way humans learn by linking a signal with a subsequent action.

Regardless the utilized neuron model and the topology of the network, questions still persist about how information is encoded in the neuron signals and processed by the neurons. Once again, biology holds the key to answer these questions and already in 1926, Edgar Andrian illustrated the idea that the neural information is encoded in the firing rate [52]. Experiments he performed on frogs, showcased that neurons responded with more spikes when the force of the applied stimuli increased. This type of neural code consists of the *rate coding schemes*, where the activity level of neurons is converted into a firing rate. A frequently used scheme that falls within this category is the *spike count scheme*, where the greater number of spikes averaged over time produced by a neuron, the more "benefitted" the neuron is.

Recent neurophysiological results, though, suggest that efficient processing of information is more likely to be based on the precise timing of action potentials rather that on their firing rate [50]. The human ability for instance, to recognize visual scenes in just a few hundred milliseconds [53], pinpoints that the timing of individual spikes carries important information in order to allow for such a

high-speed neural processing. This led to the creation of *spike coding schemes*, like the *time-to-first-spike* where information is encoded in the latency between the beginning of stimulus and the time of the first spike response, enabling ultra-fast information processing.

2.2 LEARNING

Electrophysiological experiments have shown that the synaptic weights characterizing each synapse are not constant parameters. In neuroscience, *synaptic plasticity* is a term employed to express such changes of the synaptic weights that consequently alter the postsynaptic response of a neuron to an arrival of a spike. With the appropriate stimulation, these changes can last for days. This effect is called long-term *potentiation* or *depression* of synapses depending on whether the stimulation paradigm leads to a persistent increase or decrease of the synaptic weight, correspondingly. In the formal theory of neural networks, the synaptic weights are considered adjustable parameters so as to optimize the performance of a network for a given task. The process of parameter adaptation is called *learning* and the procedure for adjusting the weights is referred to as a *learning rule* [54].

In 1949, D. Hebb postulated that there is a correlation between two neurons that are simultaneous active, hence their coupling weight is strengthened. Simply put, "neurons that fire together, wire together" [55]. In general, correlation-based learning is now known as *Hebbian learning*. Although Hebb formulated his principle on purely theoretical grounds, later studies on synaptic plasticity [56], [57] proved that the resulting change in the synaptic weight is a function of the difference between the firing times of the pre- and postsynaptic neurons. This observation has given rise to the term Spike-Timing-Dependent Plasticity (STDP), which, particularly for SNNs, constitutes the most intensively studied learning mechanism. More specifically, the synapse is strengthened if the presynaptic spike occurs shortly before the postsynaptic neuron fires, but the synapse is weakened if the sequence of spikes is reversed [54]. This is in accordance with the Hebbian learning rule, and the vice versa, called *anti-Hebbian plasticity*, is also possible.

One of the limits of Hebbian learning is that it is unsupervised, as there is no notion of "good" or "bad" changes of a synapse. There is no feedback that could allow to distinguish between actions that do and those that do not lead to a successful outcome. To accommodate for this limiting factor, *reward-based learning* or *reinforcement learning* is employed, similarly to what happens with animals that are in position to learn complex action sequences if the desired behavior is awarded. In order to solve this problem, Reward-modulated Spike-Timing-Dependent Plasticity (R-STDP) takes into account two important aspects. First, rules of synaptic plasticity must consider the success of an action, and second, the synapses need to play a role of a short-term memory that stores past actions, since success often comes with a delay of a few seconds after an action has been taken [54].

Since the early years of machine learning, and particularly ANNs, supervised learning techniques, like the *back-propagation algorithm*, have been a very successful concept and consist the mainstream in the training of conventional ANNs. However, the discontinuity and non-differentiability characterizing the information processing within SNNs creates incompatibility with the standard error back-propagation algorithm. To compensate these issues, there have been various techniques proposed over the recent years.

One of the most classic adaptations of supervised learning for SNNs is to use a conventional ANN to train an equivalent *shadow network*. Technically, this is a form of converting an ANN trained with state-of-the-art learning methods to a SNN with the cost of some loss of accuracy. There have been proposed different approaches to overcome the loss of accuracy such as scaling the weights [58], [59] and constraining the network parameters [60].

Other proposed approaches train directly on the SNN by approximating the derivative of the spike function in various ways. For example, SpikeProp [61] keeps track of the membrane potential of spiking neurons only at spike times and back-propagates errors based on that but is prone to end up with a "dead network" meaning that no learning occurs when none of the neurons spike. To avoid such scenarios, other methods [62], [63] suggest to back-propagate errors based on the membrane potential at a single time step only. The problem in this case is that the temporal dependency between spikes is ignored, since the error is credited to the input signals at the given time step only, thus neglecting the effect of earlier spike inputs [64].

Another supervised learning technique is Spike LAYer Error Reassignment (SLAYER); a general method of error back-propagation for SNNs [64]. SLAYER distributes the credit of error back through the network's layers, like in the traditional back-propagation algorithm for ANNs. Unlike back-propagation, SLAYER takes into account the fact that a spiking neuron's current state depends on its previous states and those of its presynaptic neurons, as well. Thus, it also distributes the credit of error back in time, allowing to simultaneously learn both synaptic weights and axonal delays. To estimate the derivative of the spiking function, it uses the probability of a change in the spiking state of a neuron, i.e., spiking or non-spiking state.

For methods that train directly a SNN, like SLAYER, to work, the input to the network needs to be in a neuromorphic form, too, i.e., in a spike train representation. To achieve this, the most precise and loss-less method is to create a neuromorphic dataset using a neuromorphic sensor, like a Dynamic Vision Sensor (DVS) for visual recognition applications. A DVS resembles the retina of the human eye and is composed by pixels that behave similarly to a spiking neuron and respond to changes in the brightness. If the brightness of a pixel has changed sufficiently, a spike is generated. Spikes can have a positive or negative polarity corresponding to changes from low to high brightness and vice versa, respectively. Each pixel operates independently and reports the brightness changes as they occur. Sensors like this provide a timing resolution in the scale of microseconds and consume the minimal power of some tens of milliwatts.

Table 2.1: Main characteristics of four large neuromorphic platforms [27].

Architecture	IBM TrueNorth	Intel Loihi	ROLLS	SpiNNaker
Type	Digital	Digital	Analog	ARM CPU
Neurons	10^6	2^{17}	$1 - 2^8$	10^3 per core
Synapses per neuron	2^8	2^{10}	$2^8 - 2^{17}$	10^3
On-chip learning rules	None	STDP, R-STDP	STDP, short-term plasticity	Any
Power[1](mW)	63	110	4	10^3 per CPU
Size (mm^2)	430	60	51.4	102
CMOS process (nm)	28	14	180	130

Because though producing new recordings and creating new datasets is a very challenging procedure, methods to convert existing static image datasets to neuromorphic ones have been proposed. According to [65], the images of the dataset are successively presented on the screen of a computer for a short period of time and a DVS performs saccades, i.e., quick, simultaneous movements like the ones of an eye, in order to capture changes in the pixel intensities. Using an actual sensor, instead of approximating its operation, is more credible as noise is inherently included in the result. Also, by selecting to move the sensor rather than the image is more biologically realistic and eliminates timing artifacts introduced by monitor updates. The resulted datasets, e.g., N-MNIST and N-Caltech101, are neuromorphic versions of classic datasets in the literature, allowing this way for a more direct and valid performance comparison between an ANN and a SNN.

2.3 NEUROMORPHIC IMPLEMENTATION

The first implementation of a silicon neuron dates back in the 1940s, soon after the first artificial neuron model proposed by McCulloch and Pitts [5]. Since then, many other designs have followed both in digital and analog implementations [48]. As mentioned in Section 1.1.4, such neurons consist the fundamental elements of neuromorphic hardware accelerators on ASICs [30]–[32], [34] or FPGAs [66]–[68]. The main characteristics of four large neuromoprhic platforms are elaborated below and summarized in Table 2.1.

IBM TrueNorth [30] is a fully digital neuromorphic processor consisting of 1 million leaky I&F spiking neurons interconnected with up to 256 synapses each by an event-driven routing infrastructure. The neurons are organized in a mesh

1 Measurements related to power are not from the same benchmark so they cannot be compared directly.

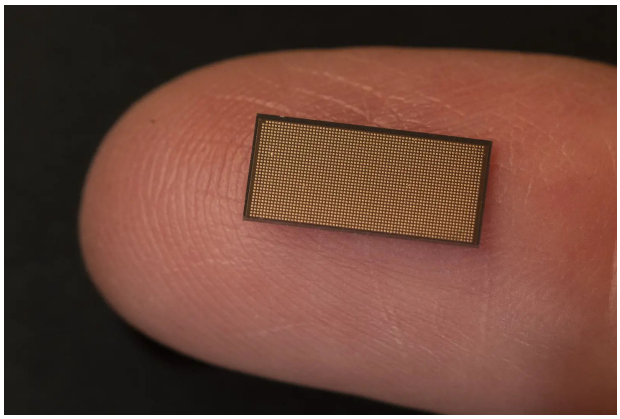


Figure 2.2: The Intel Loihi 2 die.

of 4096 parallel and distributed neurosynaptic cores. The chip is well-suited for applications that use complex low-power neural networks in real time and was tested on an multi-object detection and classification application, during which the chip consumed 63mW.

Intel Loihi [31] is also a digital neuromorphic processor based on a many-core mesh arrangement comprising 128 neuromorphic cores, each of which implements 1024 leaky I&F spiking neurons, named *spiking neural units*. A great advantage of the chip is its capability to scale-up thanks to its off-chip communication interfaces that hierarchically extend the mesh in four planar directions to other chips. The mesh communication protocol supports scaling to 4096 on-chip cores and up to 16384 chips, which allows for the training and inference of deep SNN architectures. [69] uses a two layer neural network keyword spotter trained to recognize a single phrase as a benchmark to calculate the mean power consumption of the chip, which is 110mW. Four years after Loihi's release in 2017, a second generation chip succeed it with 8× more neurons (a total of 2^{20} neurons) and 3-D mesh scaling. The die of Intel Loihi 2 is shown in Fig. 2.2.

The ROLLS neuromorphic processor [32] is a mixed-signal VLSI device. Its purpose varies from exploring the properties of computational neuroscience to building brain-inspired computing systems. The 256 I&F spiking neurons with biologically plausible dynamics and the 131072 synapses comprising the processor are implemented on the analog domain and are combined with digital circuits, mainly for handling the inter-neuron communication. The synapses are organized in a 256×512 array, half of which are learning synapses modeling the *long-term potentiation/depression* mechanisms and the other half are *short-term plasticity* synapses with programmable synaptic weights. By default, each neuron is connected to a specific set of 512 synapses but it is possible to reallocate synapses, with the extreme case of connecting all of them to a single neuron and leaving the remaining neurons unused. Typically, the chip consumes approximately 4mW.

SpiNNaker [34] is a massively parallel multiprocessor architecture for modeling large-scale SNNs in real time. Unlike the previously mentioned neuromorphic

chips, SpiNNaker is structured upon *processing nodes* where each node incorporates 18 ARM microprocessor cores and two routers. One router is responsible for the communication among the node's microprocessor cores while the second handles the communications with other nodes and the peripherals. The communications infrastructure is also the key innovation of the architecture, as it is optimized to carry very large numbers of very small packets, in contrast to the conventional cluster communications system. Each of the cores is capable of simulating up to 1000 neurons and around 1000 synapses while consuming 1W. The limit of integration comes up to 65536 nodes, resulting in a total of more than 1 million cores. Because of its CPU-based structure, the platform provides a lot of flexibility in terms of spiking neuron models and learning rules.

From a hardware perspective, there is a belief that SNNs offer faster inference and lower energy consumption compared to ANNs thanks to their asynchronous operation which results in a sparser activity. The above statement is verified by the aforementioned state-of-the-art neuromoprhic architectures and is the reason that neuromorphic computing is an appealing and promising technology. Its advantages can be summed up as follows:

- ***Power efficiency***: The sparse spiking activity of the neurons can lead to huge power savings in neuromorphic ICs. As opposed to the per layer activation-based operation of conventional ANNs, SNNs process spikes as they come, thus the neurons are working only when there is something to be processed, otherwise they are idle. This event-driven operation allows for very low power consumption in neuromorphic architectures at a magnitude of some tens of milliwatts, even for ASICs made up of billions of transistors.

- ***Computation speed***: A great advantage of SNNs is their ability to process substantial amount of data using a relatively small number of spikes. This property is mainly attributed to the asynchronous information processing performed by the spiking neurons. The first output spikes in a SNN start to emerge as the input events still flow in, already after the first few input events. The event-driven sensing and processing allows for the input and output event flows of a processing stage to be (in practice) simultaneous or coincident in time, which is called the *pseudo-simultaneity* property [70].

- ***Noise robustness***: As discussed in Section 2.1, the shape of a spike carries no information, while the only important aspect is the presence of a spike. Moreover, the scope of a spike is only within the postsynaptic neurons, since the latter will generate their own, hence leading to a kind of regeneration of signals at every neuron [71]. This gives SNNs the advantage of noise robustness, similar to the one observed in digital systems.

Neuromorphic chips can incorporate millions of neurons which operate and fire spikes asynchronously to each other. It is evident that in such a highly complex system, connectivity and communication challenges are posed, acting restrictively for the implemented network to some extend, e.g., allowing a finite

Table 2.2: Comparison of features between conventional ANNs and SNNs [72].

Feature	ANN	SNN
Data processing	Frame-based	Spike-based
Time processing	Sampled	Continuous
Time resolution	Low	High
Latency	High	Low
Recognition speed	Low	High
Recognition accuracy	Higher	Lower
Neuron model complexity	Low	High
Hardware multiplexing	Possible	Not possible
System scale-up	Ad hoc	Adding modules
Power consumption	High	Ultra low

number of synapses per neuron. Unlike the brain which facilitates its 3-D volume to easily build synaptic connections, a neuromorphic architecture handles the inter-neuron transmission of information via representation and communication protocols, like the Address Event Representation (AER).

The AER protocol makes use of the fact that spikes carry no information other than the fact that neuron n fired at time t_f. Each neuron is assigned an address which is unique either globally in the chip, or locally in a sub-cluster of neurons. According to the protocol, a spike is represented under the form of an event containing (i) the address of either the neuron that generated it, or the ones that the spike is destined to, and (ii) the exact timestamp of the firing. This allows the neurons to be triggered only when there is an event associated to them and eliminates the need for huge synaptic connections, thus efficiently allocating hardware resources where and when needed.

2.4 COMPARISON WITH CONVENTIONAL ANNS

The discussion on the relative performance between ANNs and SNNs is not trivial due to fundamental differences characterizing them, e.g., input of sequence of static frames versus continuous-time event flow. A comparison of the main distinctive features between traditional ANNs and SNNs is summarized in Table 2.2.

The primary structural difference between conventional and spiking networks concerns the way the *processing of data* is carried out. ANNs operate upon a frame-based logic whereas SNNs are spike-based. The capture and processing of framed input data premises a segmentation of the computation time steps using samples of the incoming information, which reduces significantly the *time resolution* in temporal or spatiotemporal applications. On the other hand, spikes arrive whenever a change occurs providing a continuous *time processing* with very detailed resolution in time in microsecond scale.

Based on the above, *latency*, i.e., the time needed by the neural network to make a decision, is a distinguishing feature. In ANNs the computation of each stage, i.e., the activation of each layer, must be completed before propagating the results to the next one and so forth, resulting in a sequential computation between successive layers. The asynchronous nature of SNNs removes this limit allowing for massively parallel architectures. More specifically, computation is performed spike by spike, so that output spikes in a computational layer are generated as soon as enough spikes evidencing the existence of a certain feature have been collected [72]. The latency between the input and output spike flows of a processing SNN convolution layer has been measured to be as low as 155 ns [73].

The latency is also of high importance when considering the *recognition speed* of a neural network. Hence, as discussed in Section 2.3, SNNs are characterized by a high computation speed, if the appropriate neural coding is used, as they process each input spike in (almost) real time, which, in cooperation with the low latency, leads to a very fast recognition, or decision making. ANNs, on the contrary are strongly dependent on the computation capabilities of the hardware and the network complexity, as it regulates the number of total operations to be carried out. However, quicker recognition speed does not imply higher *recognition accuracy* and this is a basic point where conventional ANNs outperform SNNs over a given task, thereby making them the dominant neural network type. This is not a discouraging factor though as training techniques advance for SNNs as well, bringing the error increment for the same deep architecture down to only 0.15% for the ImageNet dataset and 0.38% for the CIFAR10 dataset [74].

Continuing with the downsides of SNNs, the addition of the time variable leads to a higher *neuron model complexity* than the one of the level-based ANN neurons. Also, the time-sampled computation of ANNs, allows for time *multiplexing* of the available hardware resources by fetching data and storing intermediate variables [72]. In SNNs, where spikes need to be processed in real time, there is no room for multiplexing. If though a *system scale-up* is needed, because of the highly parallel structure of the neuromorphic chips, a modular expansion of the hardware resources is possible, e.g., by adding more computation units or by combining multiple chips.

Finally, as it has been already pointed out, the *power consumption* of SNN architectures implemented on neuromorphic ICs can achieve ultra low levels benefiting from the power efficiency of sparse spike representations and the use of efficient coding strategies. Conversely, the determinant factor of ANNs power is the consumption of the hardware accelerator hosting the network's calculations and the memory read/write operations.

3 HARDWARE-LEVEL FAULT MODELING

Biological neural networks are a remarkably resilient structure. Their performance remains exemplary even under the worst circumstances, while even in the case of a damage, operation can continue uninterrupted and unabated. In a trial to imitate nature, SNNs were inspired to mimic the behavior of their biological counterparts. This leads to the reasonable assumption that SNNs are inherently fault-tolerant; a fact that is true up to a certain extend. The circuit though that hosts the SNN calculations, i.e., the neuromorphic processor, is susceptible to transistor-level faults, which consequently may lead to a degradation of the performance of the network and its cognition ability.

Such scenarios make evident the need to assess the reliability of neuromorphic processors to hardware-level faults in a variety of applications, including safety-critical ones, before their deployment in the field. This is a first and mandatory step for developing cost-effective fault tolerance techniques and entails performing large-scale fault simulation experiments. However, transistor-level fault simulation is prohibitive and fault simulation should be carried out at a higher abstraction level.

This chapter presents a bottom-up approach starting from transistor-level simulations for developing a neuron behavioral-level fault model that can be readily employed for performing behavioral-level fault simulation of deep SNNs [75].

3.1 THE SPIKING NEURON

3.1.1 *Behavioral Model*

The I&F model is nowadays one of the most dominant and widely used for describing spiking neurons [76]. It offers enough complexity to capture the characteristics of biological neural processing, while being simple enough for analysis and intuitive understanding of its dynamics.

The I&F model explains the dynamics of a neuron through its membrane potential, V_m:

$$C_m \cdot \frac{dV_m}{dt} = I_{syn} + I_{inj}, \tag{3.1}$$

where C_m is the membrane capacitance, I_{syn} is the post-synaptic current fed to the neuron, and I_{inj} is the current injected into the neuron either externally or through a positive feedback path.

The simplicity of the I&F model comes from the separation of the sub-threshold integration dynamics from the spike generation mechanism. Since a spike is a momentary surge in voltage whose form holds no information, the shape of the spike is not formally stated in the model. Instead, the spike generation behavior is characterized by a firing time t^f and a threshold criterion, i.e., the neuron produces a spike at time t^f when V_m reaches the threshold value, V_{ref}:

$$t^f : V_m(t^f) = V_{ref} \tag{3.2}$$

As soon as the neuron fires, the membrane potential is reset to a value $V_{reset} < V_{ref}$:

$$\lim_{t \to t^f; t > t^f} V_m(t) = V_{reset} \tag{3.3}$$

For $t > t^f$, the neuron dynamics again follow Eq. (3.1) until the next time V_m reaches V_{ref}.

3.1.2 *Transistor-Level Design*

Fig. 3.1 shows the transistor-level design of the I&F neuron used in this thesis. It is designed in the ams 0.35µm technology, and was originally part of a neuromorphic cortical-layer processing chip for spike-based processing systems [77]. The microchip performs 2-D convolutions on video inputs decoded using the AER protocol in real time.

The neuron takes the input current spikes I_{syn} coming from the synapses, integrates them on capacitor C_m, and fires a spike at the output V_{spike} when the capacitor voltage V_m reaches a certain threshold V_{ref}. The circuit has an

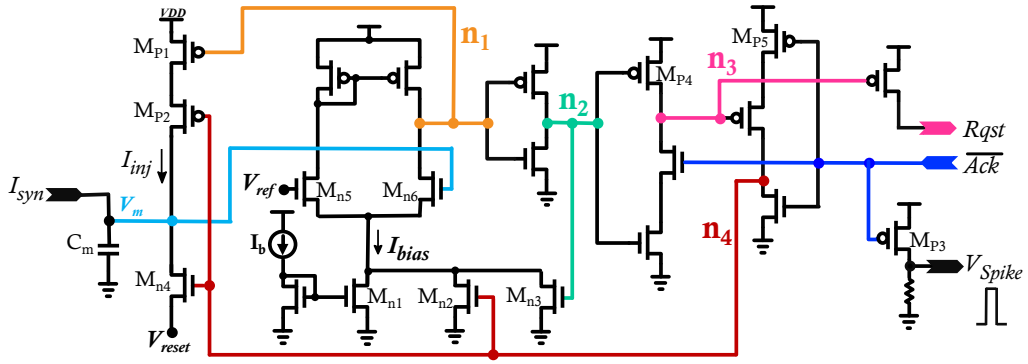


Figure 3.1: I&F neuron circuit.

extra set of input/output nodes, namely the $\overline{\text{Ack}}$ and *Rqst* nodes, which are used by the AER communication protocol.

The main blocks are a comparator and a set of inverters that control the signal flow. During the charging time of the capacitor the circuit is inactive, transistors M_{p1} and M_{n4} are off, and transistor M_{p2} is on. Since the comparator is constantly following V_m and comparing it to V_{ref}, its bias current is kept low through transistor M_{n1} to minimize power consumption. As V_m increases towards V_{ref}, node n_1 starts changing state and switches on two transistors:

- Transistor M_{p1}, which slowly introduces a positive feedback current that accelerates the charging of the capacitor.
- Transistor M_{n3} through node n_2, which offers a brief surge in the comparator bias current.

Combined, these actions speed up the transition time of the comparator output.

Once the transition is complete, i.e. node n_1 is low and node n_2 is high, node n_3 goes low and an output request signal is sent to the AER communication block by pulling up line *Rqst*. After a few nanoseconds, the AER block acknowledges back the request and the $\overline{\text{Ack}}$ input pulls node n_4 up and turns on transistor M_{p3} which produces the output spike of the neuron. Node n_4 has three main effects on the neuron circuit:

1. It turns transistor M_{n2} on to keep the comparator bias current high during the back transitioning.
2. It turns off transistor M_{p2} which cuts off the positive feedback path to the capacitor.
3. It turns transistor M_{n4} on to reset V_m to V_{reset} so the capacitor is able to charge again.

3.2 FAULT SIMULATION SETUP

In order to build a taxonomy of neuron faulty behaviors, which will allow for a behavioral-level fault model for the spiking neuron, we perform:

- MC simulation with 1000 runs using the technology Process Design Kit (PDK), considering both global and local process variations.
- Structural defect simulation in an automated workflow using the mixed-signal defect simulator Tessent®DefectSim by Mentor®, A Siemens Business [78].

In analog VLSI circuits, hard faults refer to physical defects caused by foreign particles on the wafer surface, wafer mishandling (e.g., scratching, and over- or under-itching), mask misalignment, etc. Soft faults, on the other hand, happen due to the inherent variability of the VLSI manufacturing process, e.g., local geometric deformations (i.e., variation in the effective channel length and width), doping concentration variations, etc. To enable the efficient simulation, detection, and effective mitigation of these faults, they need to be modeled into the transistor-level. The fault model should be able to sufficiently quantify the dominant faults that affect the circuit.

We consider a standard defect model for the transistors that includes stuck-on and stuck-off behaviors. Stuck-on is modeled with a short-circuit across the drain and source terminals implemented with a default small resistance of 10Ω. Stuck-off is modeled with an open-circuit in the gate terminal. Since the simulator cannot handle ideal opens and since a very high series-resistance would have no effect, a gate open is implemented with a weak pull-up or pull-down gate voltage. In particular the gate-to-source voltage is controlled by the drain-to-source voltage with a gain coefficient set to a default value of 0.5 [79]. Finally, for passive elements, i.e., resistors and capacitors, the defect model includes large variations of ±50%. For our neuron, the defect model size is $N_{defects} = 46$.

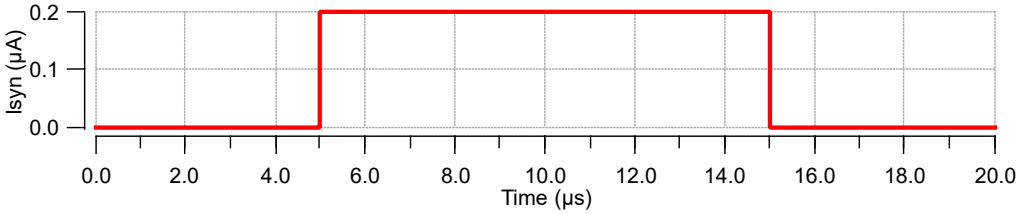
3.3 SPIKING NEURON FAULTY BEHAVIORS

To stimulate the neuron, an input current pulse of 10µs width was used, shown in Fig. 3.2a. In a fault-free scenario, the neuron should start spiking at regular intervals after the input stimulus begins and stop spiking once the input stimulus is over, as shown in Fig. 3.2b.

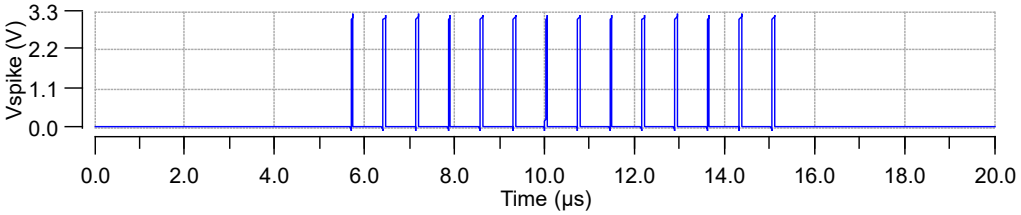
Simulation experiments revealed different types of faulty behaviours ranging from catastrophic, i.e. the neuron is clearly non-functional, to parametric, i.e. the neuron still produces an output spike train but it shows variations in timing parameters with respect to the nominal response. Catastrophic faulty behaviors were observed in 31 defect simulations, while parametric faulty behaviors were observed across the 1000 MC runs and in the rest 15 defect simulations.

3.3.1 *Catastrophic Faults*

We observe six different catastrophic faulty behaviors. These faults are considered fatal to the circuit operation. The neuron is not spiking correctly in response to the input stimulus, and it is considered defective. These faults are observed



(a) Post-synaptic input current stimulus.



(b) Nominal neuron output.

Figure 3.2: Nominal operation of I&F neuron.

only as a result of physical defects in the circuit elements, and they are listed next, along with an example of a root-cause defect.

1. ***Saturated Output***
   A state where the neuron is constantly firing regardless of the presence of an input stimulus. Fig. 3.3 shows a saturated output caused by a stuck-on transistor M_{p1}. This defect triggers a constant high feedback current to the capacitor, so the capacitor is always charging even without a current from the synapse.

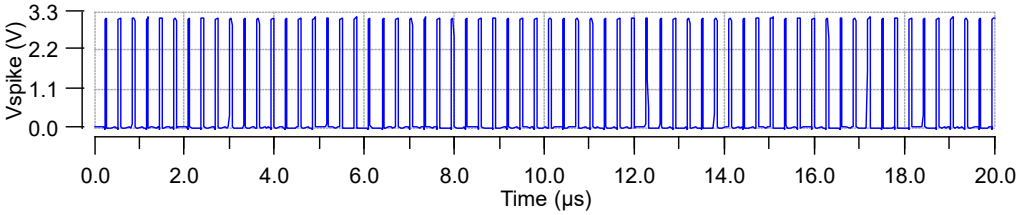


Figure 3.3: Saturated output caused by a stuck-on M_{p1} transistor.

2. ***Dead Output***
   A state where the neuron output is stuck-at-0 when it should be spiking. The red curve in Fig. 3.4 shows a dead output caused by a stuck-on transistor M_{n4}. The capacitor cannot charge since it is constantly held at its reset value and, thereby, the neuron is incapable of spiking.

3. ***Stuck-at-X output***
   A state where the neuron output gets stuck at an arbitrary DC value between the supply voltage V_{dd} and ground. The blue curve in Fig. 3.4 shows such a faulty behavior caused by a stuck-off transistor M_{p3}. This defect isolates the neuron output from the $\overline{Ack}$ signal and, in the case of an ideal stuck-off, it turns the output node into a floating node which can

settle to any DC value. Given our modeling of stuck-off transistor, the neuron node ends up settling at 1.1 V.

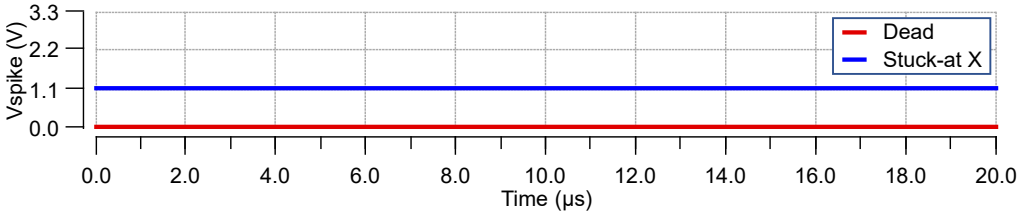


Figure 3.4: Dead (or stuck-at-0) output caused by a stuck-on M_{n4} transistor and stuck-at-X output caused by a stuck-off M_{p3} transistor.

4. ***Stuck-at-1 output***
   A state where the neuron output gets stuck at V_{dd}. This faulty behavior can be produced even in the absence of an input stimulus, or it gets triggered once an input stimulus comes along. An example root-cause defect for the former case is a stuck-on transistor M_{n6}, as shown with the dotted red curve in Fig. 3.5, while an example root-cause defect for the latter case is a stuck-off transistor M_{n5}, as shown with the blue curve in Fig. 3.5. A stuck-on transistor M_{n6} forces node n_1 to be permanently low and, thereby, node n_2 to be permanently high. In the start-up, $\overline{Ack}$ is high, thus n_3 enables the *Rqst* and $\overline{Ack}$ goes low producing a spike. When $\overline{Ack}$ goes low, node n_3 is floating but retains its low value, thus $\overline{Ack}$ is permanently set low and the output gets stuck-at-1. On the other hand, a stuck-off transistor M_{n5} cuts off V_{ref} from the comparator input. According to our modeling of stuck-off transistor, the gate voltage of M_{n5} varies with time and is set equal to $V_{G,M_{n5}}(t) = (V_{D,M_{n5}}(t) + V_{S,M_{n5}}(t))/2$, where $V_{D,M_{n5}}(t)$ and $V_{S,M_{n5}}(t)$ are the drain and source voltages of M_{n5}, respectively. Initially the comparator output is high, and the capacitor keeps charging. At some point $t = t_s$, V_m exceeds $V_{G,M_{n5}}$ and the neuron eventually spikes. At the time of spiking $V_{G,M_{n5}}(t_s)$ is lower than V_{reset}, thus node n_1 gets permanently stuck at a low value and the output gets stuck-at-1 as explained above for the stuck-on transistor M_{n6}.

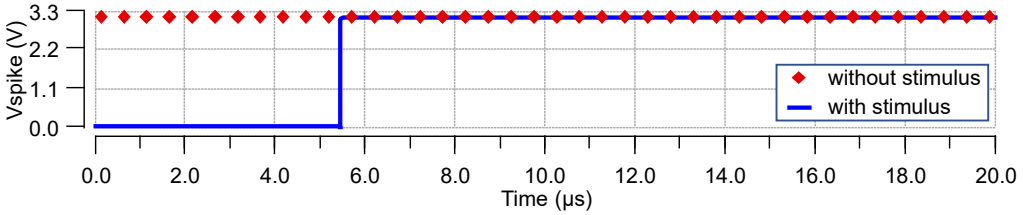


Figure 3.5: Stuck-at-1 output caused by a stuck-on M_{n6} transistor (without requiring input stimulus) and by a stuck-off M_{n5} transistor (triggered with input stimulus).

5. ***Ghost-spike firing***
   A state where the neuron generates extra spike(s) that is(are) not a result of the membrane potential exceeding the reference voltage. We refer to these

spikes as "ghost" spikes. Fig. 3.6 shows such a faulty behavior caused by a stuck-off transistor M_{p4}. When the neuron spikes, the path from node n_3 to ground gets cut-off. Node n_3 is floating since the defect isolates node n_3 from node n_2. Because of our defect model, node n_3 will eventually be weakly pulled up to V_{dd}. Simulations show that it first gets weakly pulled up to V_{dd} stopping spiking and then again it is weakly pulled down to ground producing a second ghost spike before it is finally stabilized bringing the neuron to its resting state.

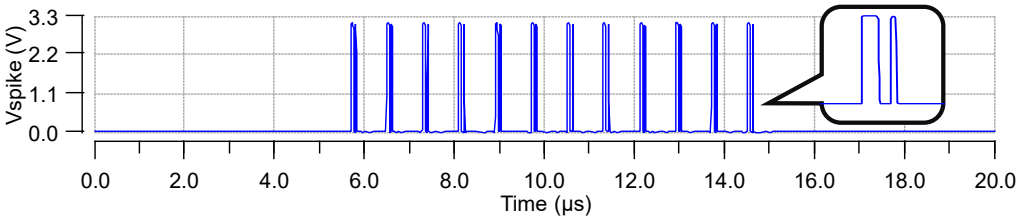


Figure 3.6: Output with ghost spikes caused by a stuck-off M_{p4} transistor.

6. ***Long-duration spike firing***
   A state where the neuron produces spikes of longer duration. Fig. 3.7 shows such a faulty behavior caused by a stuck-off transistor M_{p5}. When signal $\overline{Ack}$ goes low and the neuron spikes, node n_4 does not go immediately high to instantaneously reset the membrane potential and restart the integration. Instead, node n_4 is initially weakly pulled up to V_{dd} and gradually increases. As a result, the capacitor starts resetting but at a slow rate, thus extending the duration of the output spike.

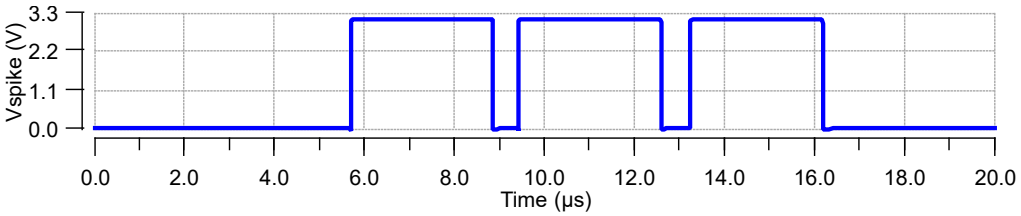


Figure 3.7: Long-duration spikes caused by a stuck-off M_{p5} transistor.

Table 3.1 provides a summary of catastrophic faulty behaviors and shows the number of defects that produce them.

3.3.2 *Parametric Faults*

As for the parametric faulty behaviors, we consider two types of timing parameters, namely the time-to-first-spike and the firing rate. It should be noticed that such timing variations may not be problematic at network-level, i.e., they may be accommodated during training.

Fig. 3.8 shows the histograms of the time-to-first-spike and the firing rates observed across the MC runs and the defect simulations, excluding the simulations that led to a catastrophic faulty behavior as discussed above. Results suggest

Table 3.1: Catastrophic faulty behaviors resulting from defect simulation.

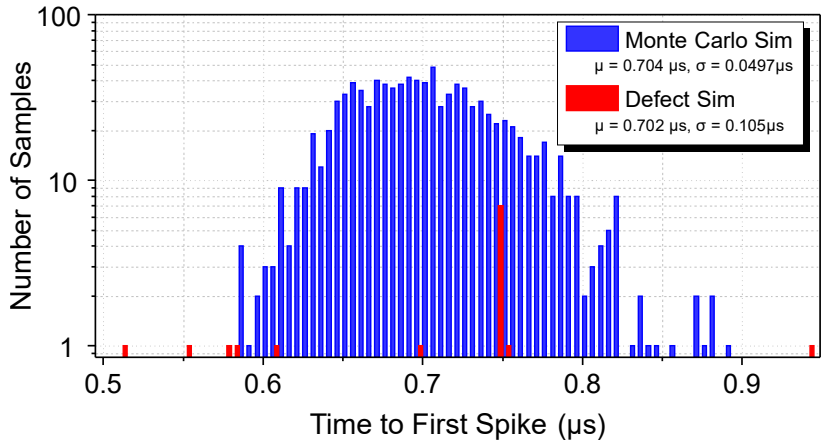
Catastrophic faulty behavior	Number of defect simulations producing it ($N_{defects} = 46$)	Example neuron response
Saturated	5	Fig. 3.3
Dead	12	Fig. 3.4
Stuck-at-X	1	Fig. 3.4
Stuck-at-1	10	Fig. 3.5
Ghost-spike firing	1	Fig. 3.6
Long-duration spike firing	2	Fig. 3.7

a correlation between the time it takes for the neuron to fire its first spike and the firing rate, i.e., a neuron that produces a first spike faster has a higher firing rate, and vice versa. This neuron is implemented with no adaptation mechanism and simulated with the initial condition for the capacitor voltage set equal to the reset value, hence the time-to-first-spike is equal to the inter-spike interval, which is the inverse of the firing frequency.

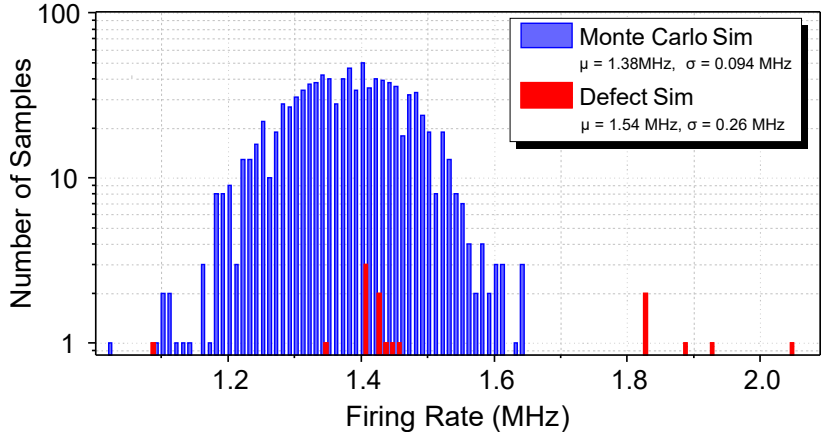
This data was collected from 1000 MC runs and 15 defects that result in timing variations. As evident from the figure, the timing parameters of the circuit are very sensitive to process variations and mismatch. Time-to-first-spike values are distributed around a mean value of 0.702µs with a standard deviation of 0.1µs, Fig. 3.8a. Similarly, firing frequencies are also normally distributed around a mean value of 1.38MHz and vary with a standard deviation of 94kHz, Fig. 3.8b.

On the other hand, out of 46 physical defects, only 15 result in timing variations, as shown in red in both parts of Fig. 3.8. Some of these variations are barely noticeable, i.e., they cause a change in the time-to-spike that is so small that the firing frequency is not affected. An example is a stuck-off transistor M_{p2}. This transistor is on in the idle state of the circuit, and once the neuron spikes, it is responsible for cutting off the positive feedback path to the capacitor. When it is stuck-off, the feedback path is cut from the start and the capacitor charges only through the synaptic input. Since this feedback current is applied to accelerate the charging rate of the capacitor, its absence has very small effect on the actual circuit operation, and the neuron spikes with a frequency almost equal to the nominal value.

Other defects can lead to a clear change in the firing frequency, albeit without affecting the functionality of the neuron. For example, a 50% decrease in the membrane capacitance results in a similar decrease of the integration time constant, i.e., the charging speed of the capacitor. This entails that the capacitor reaches the reference voltage faster than the nominal case, thus producing an earlier first spike and by definition, spike at a higher firing rate of over 2 MHz. Other defects that can lead to an apparent variation in the circuit timing are stuck-off defects in transistors M_{n1}, M_{n2} and M_{n3}. As explained in section 3.1.2,



(a)



(b)

Figure 3.8: Histograms of timing variations.

these three transistors form a dynamic biasing circuit for the comparator that momentarily change its bias current to control the transition rate. Consequently, when one of them gets stuck-off, the transition rate of the comparator is affected and ends up altering the firing rate of the neuron.

3.4 BEHAVIORAL-LEVEL FAULT MODEL

Based on the experiments conducted in this work, we propose a fault model according to the different faulty behaviors observed in Section 3.3. This model can be used to test spiking neurons based on the I&F model in a complete network.

At the behavioral level, these faults can be recreated depending on their types. Parametric faulty behaviors are emulated by manipulating the model parameters described in Section 3.1.1. For example, timing variations can be represented as changes in either the membrane capacitance C_m in Eq. (3.1), the reference voltage V_{ref} in Eq. (3.2) or the reset voltage V_{reset} in Eq. (3.3). Catastrophic faults on the other hand are modeled a little differently. For example, a dead output or an output that is stuck-at-1 or stuck-at-X can be simulated just by

forcing the neuron output to take the respective value. A saturated output is obtained by forcing a high constant input current applied from the start, so the neuron is spiking all the time. A delay in the resetting mechanism of the neuron would produce output spikes with long duration. Finally, the ghost-spike firing can be recreated by simply adding ghost spikes to the nominal spike train after decreasing their widths.

The above idea is illustrated in more detail in Chapter 4, where the faulty behaviors are simulated by faulty instances of a network's elements, i.e., neurons and synapses, located anywhere across its boundaries. The faults are injected in the network with the fault injection framework that we developed for SNNs by customizing the computational flow of the targeted elements, causing the corresponding outputs to be faulty in a similar manner to the one previously described. At the end of the experiments, the impact of each fault is measured in order to assess the resilience of the underlying network.

4 SNN FAULT INJECTION FRAMEWORK

The derived behavioral-level fault model of the I&F neuron from Chapter 3 is a realistic approach of how a spiking neuron can come to a failure. The neurons of a network along with the other components, i.e., synapses, can fail and therefore affect the network's operation and interfere with its performance.

In order to form a complete picture of a network's vulnerabilities, a series of large-scale fault injection experiments is conducted network-wise across the layers of neurons and the synapses connecting them. Each fault injection experiment pinpoints the critical fault types and locations throughout the network. After all the experiments are performed, a complete resilience analysis of the network can be shaped by leveraging the experimental results. To automate the fault injection procedure, we have developed a fault injection framework specifically for SNNs, which is accelerated on a GPU [80].

This chapter describes the SNN fault injection framework and demonstrates it on two custom-designed deep convolutional SNNs that solve the recognition tasks of (i) handwritten arithmetic digits and (ii) hand/arm gestures, such as hand waiving.

4.1 RELATED WORK ON FAULT INJECTION EXPERIMENTS AND FRAMEWORKS

An important step towards reliable AI design is assessing the effect of hardware-level faults on the network performance. Hardware-level faults include process variations, manufacturing defects, aging phenomena, soft errors, etc. Recent fault injection experiments in AI hardware accelerators have shown that they can be highly vulnerable to hardware-level faults especially when those are happening after training the neural network [81]–[88], since such faults can have detrimental effect on the inference when occurring in the field of application and can seriously jeopardize it.

Several fault injection experiments have been reported in the literature for various DNN models running on different types of AI hardware accelerators. Transistor-level fault simulations can be performed only at neuron-level [85] or for small-size networks [89]. In general, performing large-scale fault injection experiments necessitates the use of a fault model of higher abstraction in order to make simulation traceable. This also enables a reliability analysis at higher-level independent of the specific hardware implementation. To this end, fault injection experiments have been performed using behavioral-level faults at the synapse and neuron level [84], static and transient bit flips in data-paths and memories [81], [82], [87], and stuck-at faults at gate-level [83], at quantizing activations [90], or at the conductance of memristors in memristor crossbars [91].

To perform fault injection experiments, there exist several works that use customized fault injection frameworks for SNNs [84], [92]–[94] but they support a specific fault model and none of them has been made public until today. Software-level fault injection frameworks have also been developed for ANNs, ranging from basic ones [81], [82], [95], [96] to more elaborate ones aiming at improving the one-to-one mapping between hardware and software fault injection [97], reproducing more complex fault models, i.e., extracted from radiation tests [98], or speeding up the analysis by reducing the fault injection space [99], [100]. Another possibility is to use generic fault injection tools [101], [102] to emulate fault effects in the hardware platform, i.e., GPU, running the application. Such fault injection frameworks are crucial towards the testability and dependability of AI hardware accelerators [39].

4.2 DESCRIPTION OF THE FAULT INJECTION FRAMEWORK

The growth of machine learning during the last decades had a great impact on programming languages as well, leading to many new frameworks designed for easily developing AI applications. One of the most established languages in the scientific community for this task is Python, which already contains a plethora of frameworks, like TensorFlow [103], Keras [104], Theano [105], Pytorch [106], and more. Each one of them implements its own approach on solving AI problems, emphasizing on different aspects each time and trading off between others, e.g.

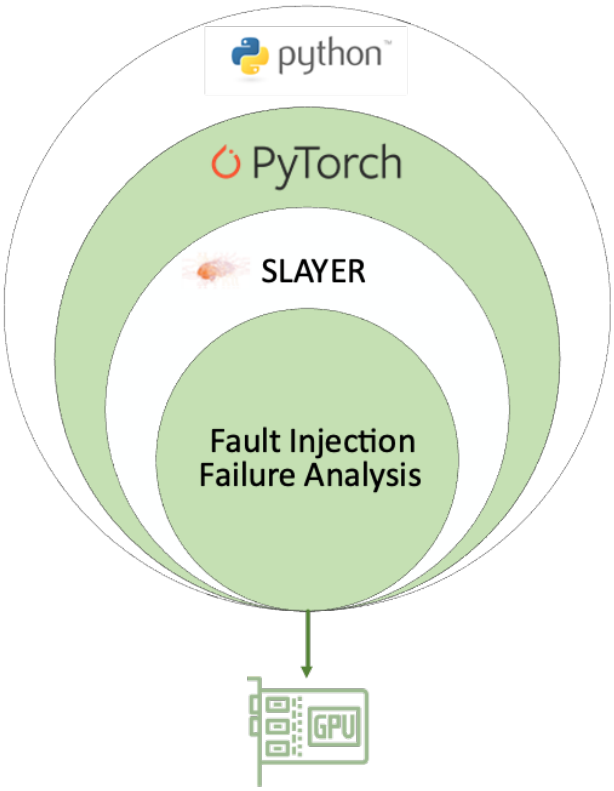


Figure 4.1: The fault injection framework built on top of the SLAYER and PyTorch frameworks accelerated on a GPU.

simplicity and modularity. This makes it clear that the selection is left to the designer based on their preferences and the needs of the application.

Therefore, we selected PyTorch to work with in the fault injection framework. PyTorch is a machine learning library in Python used for designing DNN architectures. It is optimized to perform dynamic tensor computations with automatic differentiation and GPU acceleration, and does so while maintaining performance comparable to the fastest current libraries for deep learning [106].

Despite the huge variety of AI frameworks, none of them has integrated support for SNNs. To this end, in our fault injection framework, we have chosen to work with the SLAYER framework [64] which is completely built on PyTorch, inheriting all its benefits and capabilities. Moreover, it uses the SRM model for the spiking neurons described in Section 4.2.1 and implements learning with a variation of the back-propagation algorithm. A synopsis of the employed frameworks is illustrated in Fig. 4.1.

Fault injection and simulation are performed by customizing the flow of computations according to the behavioral modeling of the faults described in Section 4.2.2. The fault injection methodology is described in Section 4.2.3. Fault simulation acceleration is achieved first by considering a behavioral model of the SNN and performing fault injection at this level independently of the hardware implementation, and second by running training and inference on a GPU. Note that a traditional CPU-only system is proven to be largely inefficient

for the complexity of a large-scale fault injection campaign even at this abstract representation.

For the execution of the experiments we used the same system setup, consisting of an Intel Xeon W-2133 CPU, a Nvidia Quadro RTX 4000 GPU, and 32 GB of RAM. Inference and training times with injected faults are similar to the average fault-free inference and training times regardless the type of the injected fault.

4.2.1 *The Spike Response Model*

The spiking neurons of the simulated networks are modeled after the SRM; a generalization of the ubiquitous I&F model in a sense that the parameters of the model are replaced by (parametric) functions of time, called filters [54].

In the SRM, the state of the neuron at any given time is determined by the value of its membrane potential, $u(t)$, and this potential must reach a certain threshold value, ϑ, for the neuron to produce an output spike. The membrane potential of a neuron j in layer l is calculated as:

$$u_j^l(t) = \sum_i \omega_{i,j}^{l-1,l}(\varepsilon * s_i^{l-1})(t) + (\nu * s_j^l)(t) \tag{4.1}$$

where $s_i^{l-1}(t)$ is the pre-synaptic spike train coming from neuron i in the previous layer $l-1$, $s_j^l(t)$ is the output spike train of the neuron, $\omega_{i,j}^{l-1,l}$ is the synaptic weight between the neuron and the neuron i in the previous layer $l-1$, $\varepsilon(t)$ is the *synaptic kernel*, and $\nu(t)$ is the *refractory kernel*.[1]

In Eq. (4.1), the spiking action of the neuron is described in terms of the neuron's response to the input pre-synaptic spike train and the neuron's own output spikes. The incoming spikes by the neurons in the previous layer are scaled by their respective synaptic weights and fed into the post-synaptic neuron. The response of the neuron to the input spikes is defined by the synaptic kernel $\varepsilon(t)$ which distributes the effect of the most recent incoming spikes on future output spike values, hence introducing temporal dependency. For our experiments, we use the form [64]:

$$\varepsilon(t) = \frac{t}{\tau_s} \cdot e^{\left(1-\frac{t}{\tau_s}\right)} \cdot H(t) \tag{4.2}$$

where $H(t)$ is the unit step function and τ_s is the time constant of the synaptic kernel. The second term in Eq. (4.1) incorporates the refractory effect of the neuron's own output spike train onto its membrane potential through the refractory kernel. The form used here is:

1 Eq. (4.1) holds for any neuron no matter the type of the layer; however, in convolutional layers neurons are arranged in a 3-D representation, i.e., width × height × channels, and, thereby, we either need to consider a flat indexing or 3-D indexes.

$$\nu(t) = -2\vartheta \frac{t}{\tau_{ref}} \cdot e^{(1-\frac{t}{\tau_{ref}})} \cdot H(t) \tag{4.3}$$

where τ_{ref} is the time constant of the refractory kernel.

When $u(t) > \vartheta$ the neuron fires a spike, e.g. $s_j^l(t) = 1$. If $u(t) < \vartheta$, then the neuron remains silent, e.g. $s_j^l(t) = 0$.

4.2.2 *Fault Modeling*

We treat the SNN as a distributed system where neurons are discrete entities that can fail independently. We use the neuron fault model proposed in Chapter 3 to describe faults at behavioral-level and make fault simulation for deep SNNs traceable. Because this fault model is generated by performing detailed transistor-level simulations at neuron-level and collecting all types of faulty behaviors, it becomes independent of the hardware implementation, which helps us draw general conclusions. The most systematic faulty behaviors are adapted to our neuron model of Section 4.2.1.

4.2.2.1 *Neuron Faults*

We define five fault types for a neuron. The first two types, namely dead and saturated neuron, explicitly act on the output spike train, while the last three types, namely neuron timing variations, threshold perturbation, and refractory period perturbation are *parametric faults* that act on internal parameters of the neuron and implicitly affect the output spike train.

1. ***Dead neuron***: A fault in the neuron that leads to a halt in its computations and a zero-spike output. This fault is modeled by forcing the output spike train to be always low.

2. ***Saturated neuron***: A fault that causes the neuron to be firing all the time, even without any external stimuli. This fault is modeled by skipping the computations and forcing the output to be high at every time step.

3. ***Neuron timing variations***: A fault that results in timing variations of the output spike train, i.e. time-to-first-spike and firing rate. This parametric fault is modeled by varying the value of τ_s in Eq. (4.2).

4. ***Threshold perturbation***: A fault in the value of the threshold at which the neuron spikes, which can change the frequency of spiking or eventually cause the neuron to be stuck either at a saturated or a dead state. We model this fault as a deviation in the value of ϑ.

5. ***Refractory period perturbation***: A fault that can influence the refractory mechanism of a spiking neuron and eventually restrain the output of the neuron or completely stop it from firing. This fault is modeled as a variation in the value of τ_{ref} in Eq. (4.3).

Additional faulty behaviors observed in Chapter 3, i.e., neuron stuck at an intermediate output value, ghost-spike firing, and long-duration spike firing, are specific to the spiking neuron employed and, thereby, are not considered herein.

4.2.2.2 *Synapse Faults*

Synapses transfer signals from neurons in layer *l* to neurons in layer *l+1*, and they determine the significance of each signal on the recipient neuron through the weights. Therefore, weight errors can be used to model faults in the synapses. Here we define three synaptic fault types, namely dead, saturated, and bit-flipped synapse.

1. ***Dead synapse***: A fault interpreted as a cut in the synapse circuit that obstructs the transmission of signal from neuron *i* in layer *l* to neuron *j* in layer *l+1*. This fault is equivalent to -and modeled as- a zero weight.

2. ***Saturated synapse***: A fault that causes a synapse to transfer the signal from neuron *i* in layer *l* to neuron *j* in layer *l+1* with a weight of high absolute value far beyond the tails of the nominal weight distribution resulting from training. A weight can have either positive or negative sign and a fault can push it towards the positive or negative maxima. Therefore, we consider both positive and negative saturation. This fault is modeled by assuming a relatively large absolute weight value.

3. ***Bit-flipped synapse***: A fault where one or more of the q bits of the quantized synaptic weight is flipped. The binary value is then mapped back to a real one. This fault model essentially corresponds to a hardware-aware synapse fault.

4.2.3 *Fault Injection Methodology*

Fault injection is performed by customizing the flow of computations in the SLAYER and PyTorch frameworks. Fig. 4.2 illustrates how neuron faults are injected in layer l and synapse faults are injected in the synaptic matrix connecting layer l with layer $l+1$.

In particular, if neuron n_x in layer l is dead, then its spike train output calculation is bypassed and its output is forced permanently to 0, e.g. $s_x^l = 0$. If neuron n_y in layer l is saturated, then its spike train output calculation is bypassed and its output is forced permanently to 1, e.g. $s_y^l = 1$.

For a fault in the synapse connecting neuron i in layer l to neuron j in layer $l+1$, we modify the synapse weight, e.g. $w_{i,j}^{l,l+1} = \bar{w}_{i,j}^{l,l+1}$, where $\bar{w}_{i,j}^{l,l+1}$ is 0 for a dead synapse fault, has a relatively high positive value for a positive saturation fault, or has a relatively low negative value for a negative saturation fault.

For parametric faults in neurons we cannot simply modify the parameters of a single neuron since the parameters are set initially and shared among all neurons in the network. Our approach, as illustrated in Fig. 4.2, is to create a

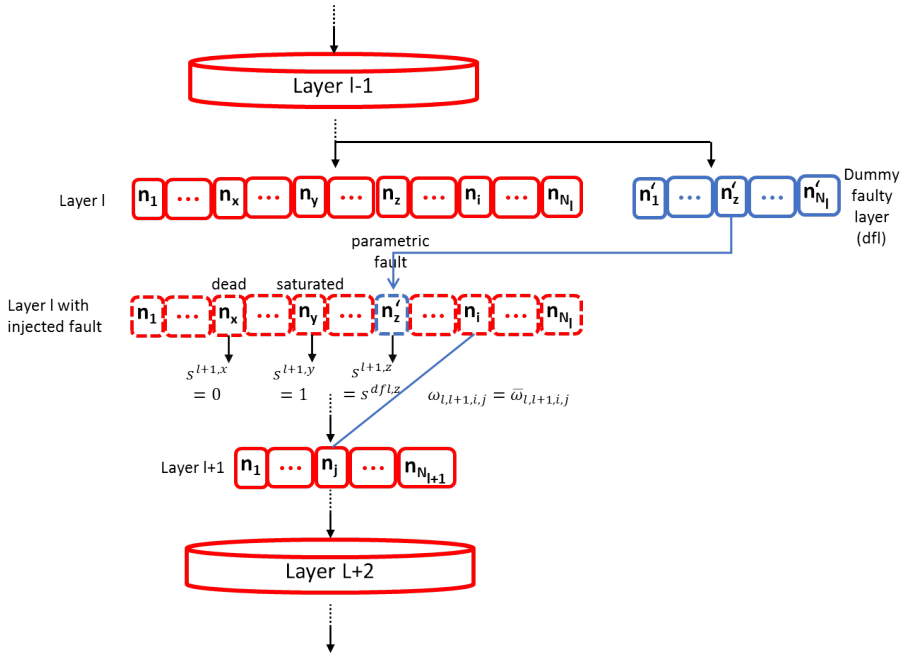


Figure 4.2: The fault injection methodology.

dummy faulty layer (dfl) identical to layer l with the exception that all neurons have the target parametric fault. The neurons in dfl are driven by the incoming spike trains from the neurons in layer $l-1$. Then, the output spike train of the neuron n_z in layer l where the parametric fault is to be injected is replaced by the output spike train of the corresponding neuron n'_z in dfl, e.g. $s^l_z = s^{dfl}_z$.

The fault injection and simulation procedures are non-intrusive to the inner part of the network operations, meaning that the neuron model remains unaffected. It is only the outcome of the calculations that is altered according to the injected fault's desired effect.

Faults can be injected in a flexible way in the network in terms of fault application time, fault position, fault number, and fault effect. To allow this and automate the fault injection process in order to cover as many experiment types as possible, we have equipped the framework with the following tools:

- ***Pre/Post-training fault injection***: Faults can be injected at any point of the SNN simulation either before or during or after the training of the network.
- ***Permanent/Transient fault injection***: Faults can exist permanently in the network, or they can have a transient effect by being deactivated anytime during the simulation.
- ***Multiple fault injection***: Multiple faults can be simultaneously injected at any positions, which may also concern different layers across the network.
- ***Successive fault injections***: A single fault or a set of faults is successively injected to a range of positions across the network. In other words, the fault(s) is(are) injected to the first position of the given ones, the effects are evaluated, the injection continues with the next position, and so on.

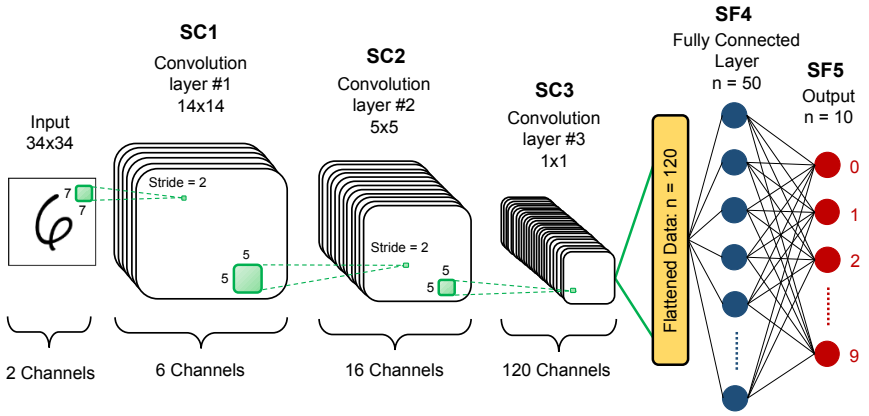


Figure 4.3: Architecture of the SNN for the N-MNIST dataset.

This feature allows for a failure analysis of a network. In this case, the necessary simulation time may be accelerated and the calculations may be reduced by storing the output of the last non-faulty layer and using it as a start for the feed-forward operation in order to avoid a massive repetition of calculations at the non-faulty, unaffected part of the network.

4.3 CASE STUDIES

As case studies, we use two convolutional SNNs performing two different cognitive tasks, namely a SNN trained to classify the N-MNIST dataset [65] and a SNN trained to classify IBM's DVS gesture dataset [107]. Their architectures are shown in Figs. 4.3 and 4.4, respectively. In each case, the winning class is declared based on the most triggered neuron at the output layer.

4.3.1 *N-MNIST SNN*

The N-MNIST dataset is a neuromorphic, i.e., spiking, version of the MNIST dataset [108], which comprises images of handwritten arithmetic digits in gray-scale format [65]. It consists of 70000 sample images that are generated from the saccadic motion of a DVS in front of the original images in the MNIST dataset. The samples in the N-MNIST dataset are not static, i.e. they have a duration in time of 300ms each. The dataset is split into a training set of 60000 samples and a testing set of 10000 samples. The SNN architecture is inspired from the LeNet-5 network [8] and is shown in Fig. 4.3. The classification accuracy on the testing set is 98.08%, which is comparable to the performance of state-of-the-art level-based DNNs.

4.3.2 *Gesture SNN*

The IBM's DVS128 gesture dataset consists of 29 individuals performing 11 hand and arm gestures in front of a DVS, such as hand waving and air guitar, under 3

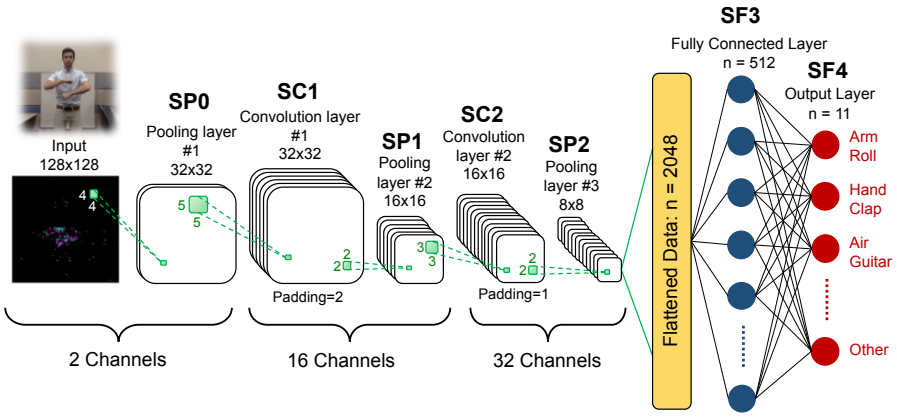


Figure 4.4: Architecture of the SNN for the IBM's DVS128 gesture dataset.

different lighting conditions [107]. Samples from the first 23 subjects are used for training and samples from the last 6 subjects are used for testing. In total, the dataset comprises 1342 samples, each of which lasts about 6s, making the samples 20× longer than of those in N-MNIST. Due to computation limitations of the neuromorphic simulation, we trimmed the length of the samples to about 1.5s. We used the architecture proposed in [64], shown in Fig. 4.4. The network performs with an 82.2% accuracy on the testing set, which is acceptable considering the shortened samples of the dataset and the shallower architecture compared to the architecture in [107].

4.4 LAYER CRITICALITY

By default, the convolution layers are not fully-connected, thus having a reduced number of synaptic connections. For example, in the N-MNIST SNN, if we were to fully connect the input to layer SC1, there would be 4624× more synapses. Instead, kernels are used in order to scan the input image or the neuron outputs of the previous convolution layer. In this way, the synaptic connections are recycled and a fault occurring in a synapse of a kernel will be affecting all neurons of the next layer for the specific channel corresponding to the kernel. Moreover, a synapse fault is of greater impact than a neuron fault in convolution layers. However, although these faults will propagate to the following layers, they do not have a significant effect since there are many kernels used in each layer. If a kernel contains a fault, then the network only loses its ability to recognize a very specific pattern or shape associated with this kernel.

4.5 FAULTS OCCURRING AFTER TRAINING

This case considers faults that occur in an already trained network, thus affecting the network performance during inference. Under this scenario, it is likely that a single fault may have detrimental effect on the performance. In fact, the effect of a fault depends on many factors. For example, considering a neuron, it depends on the location of the neuron in the network hierarchy, as well as on the

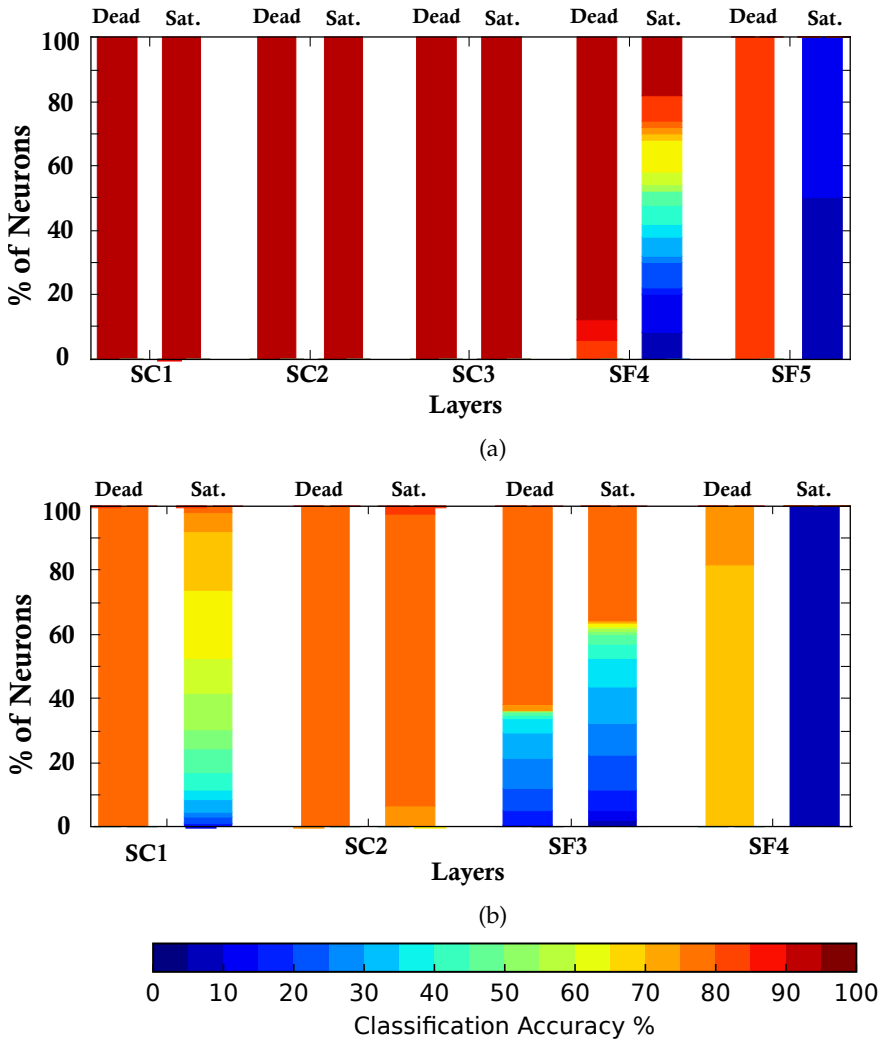


Figure 4.5: Effect of neuron faults on classification accuracy: (a) N-MNIST SNN; (b) Gesture SNN.

weights of subsequent synaptic connections. Therefore, for this part of the study, fault injection is performed on a per-neuron and a per-synapse basis, i.e. we consider a single fault assumption where one fault is injected at a time affecting one element at a time. In Chapter 8 we will show how multiple neuron fault scenarios can be covered as well in a great extend.

The metric used to evaluate the fault severity is the classification accuracy drop of the faulty network with respect to the baseline classification accuracy of the fault-free network, computed on the testing set.

4.5.1 *Neuron Fault Injection Results*

The effect of dead and saturated neuron faults on the classification accuracy is shown in Figs. 4.5a and 4.5b for the N-MNIST and gesture SNNs, respectively. The x-axis shows the different layers and for each layer we show two columns each corresponding to a fault type: dead and saturated neuron. Pooling layers

SPX in the gesture SNN aggregate regions of spikes of their previous convolution layers and do not contain any neurons, thus they are excluded from the analysis. A column is a colored bar possibly separated into chunks of different colors. Each chunk of the bar corresponds to a specific classification accuracy according to the color shading shown at the bottom of Fig. 4.5, and the projection on the y-axis shows the percentage of neurons for which the fault results in this classification accuracy.

The effect of perturbed neurons is shown in Figs. 4.6-4.8. For a given layer, we vary the corresponding parameter of one neuron at a time. We demonstrate the per-layer average, minimum, and maximum classification accuracy observed across all faulty neurons for parametric values expressed in % of the nominal value.

Following are some brief observations that can be made regarding the effect of different neuron fault types. The general conclusions are that saturation neuron faults are the most lethal, and that the impact of all neuron fault types may be severe for the last hidden and output layers.

4.5.1.1 *Dead Neurons*

Dead neuron faults may impact classification accuracy only for the neurons in the last hidden and output layers. In the output layer, a dead neuron implies always misclassifying samples of the class corresponding to the neuron. For example, as it can be seen for the N-MNIST SNN in Fig. 4.5a, a dead neuron in the output layer SF5 directly drops the classification rate to $(1 - \frac{1}{\#classes}) * 100\% = 90\%$.

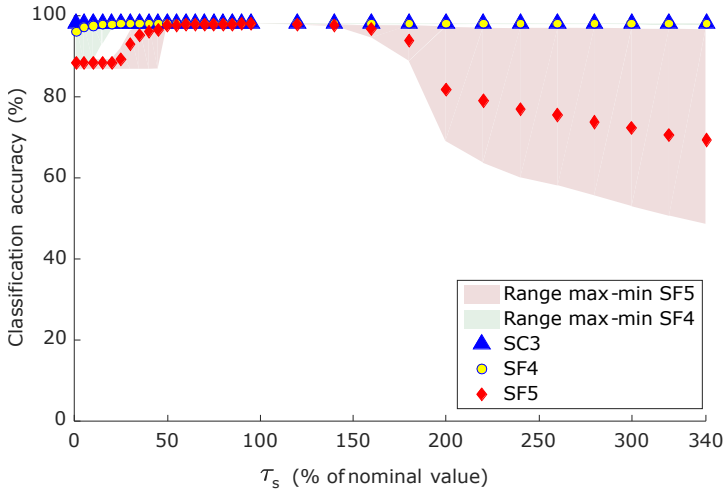
4.5.1.2 *Saturated Neurons*

Saturation neuron faults, on the other hand, may impact classification accuracy for neurons at any layer, as shown in the gesture SNN. In the output layer, a saturated neuron implies always selecting the corresponding class of the saturated neuron, thus samples from all other classes are always misclassified. This means that only one class is correctly predicted at all times regardless of the input. For example, as it can be inferred from Fig. 4.5b the accuracy of the gesture SNN with a saturated neuron fault at the output layer drops to a value of $\frac{1}{\#classes} * 100\% = 9.1\%$.

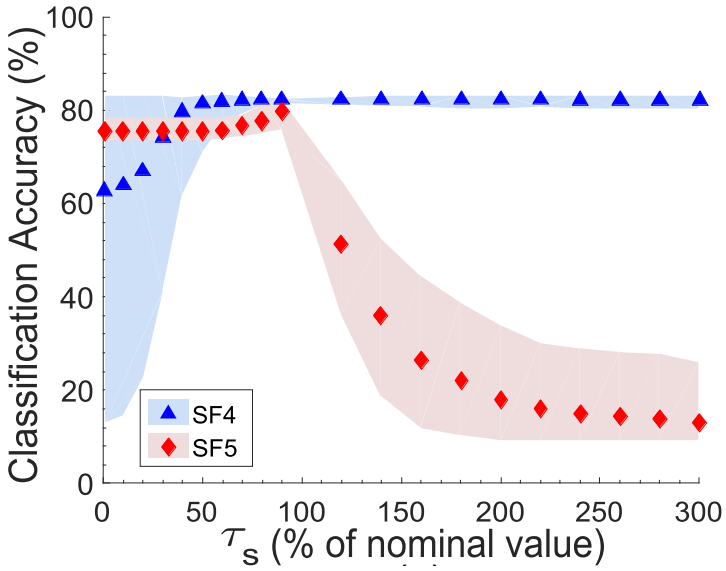
We also observe that the effect of saturated neuron faults is magnified for layers that have a smaller number of outgoing synapses, i.e., compare SC1 and SC2 layers in the gesture SNN, where the synapses connecting SC1 and SC2 are much less than those connecting SC2 and SF3.

4.5.1.3 *Neuron Timing Variations*

Neuron timing variations have an impact only for the last hidden and output layers, thus for simplicity Figs. 4.6a and 4.6b exclude all other layers. For the N-MNIST SNN, large variations in τ_s must occur to observe a drop in the classification accuracy of no more than 10%, i.e. more than 80% reduction for the hidden layer SF4 and more than 50% reduction or 100% increase for the output



(a)



(b)

Figure 4.6: Effect of neuron timing variations: (a) N-MNIST SNN; (b) Gesture SNN.

layer SF5. For the gesture SNN, we observe that this fault type can seriously affect the output layer SF4, while the last hidden layer contributes to significant classification accuracy drop only when τ_s reduces by more than 50%. A smaller τ_s implies a narrower synaptic kernel in Eq. (4.2), i.e., a decreased integration time window, thus reducing the value that the membrane potential can reach. As a result, the spiking probability is reduced and at the extreme the neuron could end up as a dead neuron. Similarly, it can be argued that a higher τ_s increases the spiking probability and at the extreme the neuron could end up as a saturated neuron.

4.5.1.4 *Neuron Threshold Perturbation*

Fig. 4.7 demonstrates the effect of threshold perturbation faults for each of the three layers SC3, SF4, and SF5 of the N-MNIST SNN. Small thresholds trigger spiking at lower values of the membrane potential and the neuron may spike

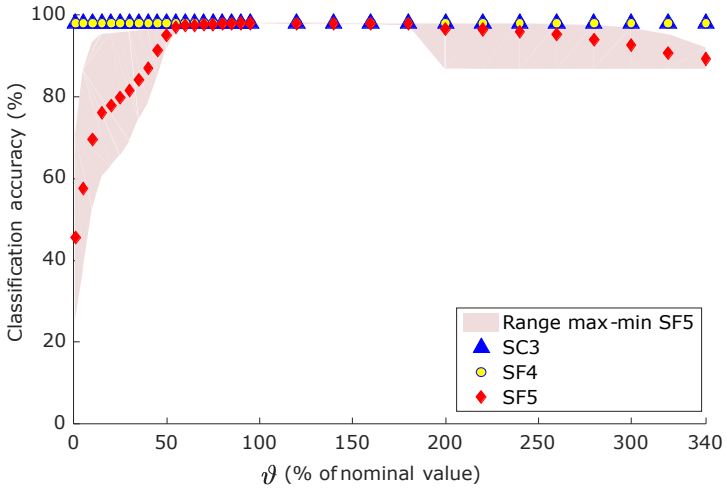


Figure 4.7: Effect of threshold perturbation faults on the N-MNIST SNN.

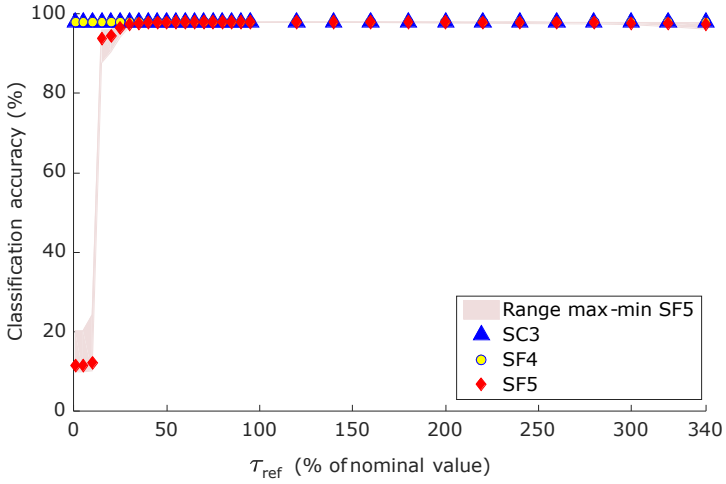


Figure 4.8: Effect of refractory period faults on the N-MNIST SNN.

more than usual. In the extreme, the neuron could end up as a saturated neuron. On the other hand, a high threshold requires higher values of the membrane potential for spiking and in the extreme the neuron could end up as a dead neuron. For layer SF5 it is evident that classification accuracy drops drastically as ϑ starts reducing below 50% of the nominal value, while even very large values of ϑ do not cause the accuracy to fall below 90%. As for layers SC3 and SF4, threshold perturbation faults have no obvious effect.

4.5.1.5 *Neuron Refractory Period Perturbation*

Fig. 4.8 demonstrates the effect of refractory period faults for each of the three layers SC3, SF4, and SF5 of the N-MNIST SNN. In general, the minimum interval between two consecutive output spikes cannot be lower than the refractory period. If the refractory period is too high, the neuron has a hard time spiking and could end up with a dead output. On the other hand, if it gets too short, the neuron will be allowed to spike more than usual and might lead to a saturated output. Results show that for layer SF5 τ_{ref} must drop below 30% of the nominal

value for the classification accuracy to start decreasing and that τ_{ref} values below 15% result in severe classification accuracy drop down to 10%. Similar to the other two parametric faults, layers SC3 and SF4 are hardly affected by refractory period faults.

4.5.2 *Synapse Fault Injection Results*

For synapse faults, fault space reduction becomes of utmost importance since the number of synapses can be in the order of several millions. We performed an analysis for the dead and positively saturated synapses on the N-MNIST SNN to validate the hypothesis that only the last hidden and output layers are affected and the result is presented in Fig. 4.9. Concerning the effect of the bit-flip synapse fault on the classification accuracy, it is shown in Fig. 4.10 for the gesture SNN. Each box in Figs. 4.9 and 4.10 corresponds to one synapse connecting two neurons in two subsequent layers $j-1$ and j, with the neuron numbers for layers $j-1$ and j shown in the row and columns, respectively. The classification accuracy in the presence of a synapse fault is shown with the box color according to the color map at the bottom of the figures.

There follow some observations concerning the effect of the various synapse faults on the two case studies. Similarly to neuron fault types, the saturation synapse faults cause the most serious impact on the networks' performance, while the effect of any synapse fault becomes noticeable only at the last hidden and output layers.

4.5.2.1 *Dead Synapses*

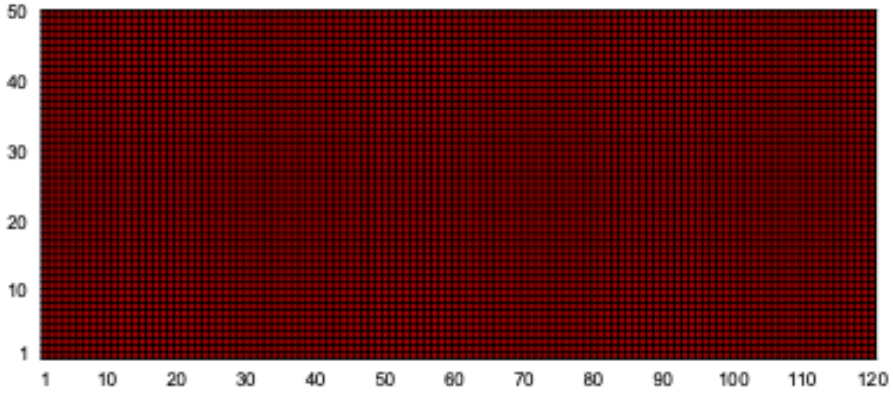
Dead synapses just reduce the firing activity of the post-synaptic neuron. As it can be seen from Figs. 4.9a and 4.9c, they have no impact on the classification accuracy for none of the network's layers. These findings were corroborated on the gesture SNN as well.

4.5.2.2 *Positively Saturated Synapses*

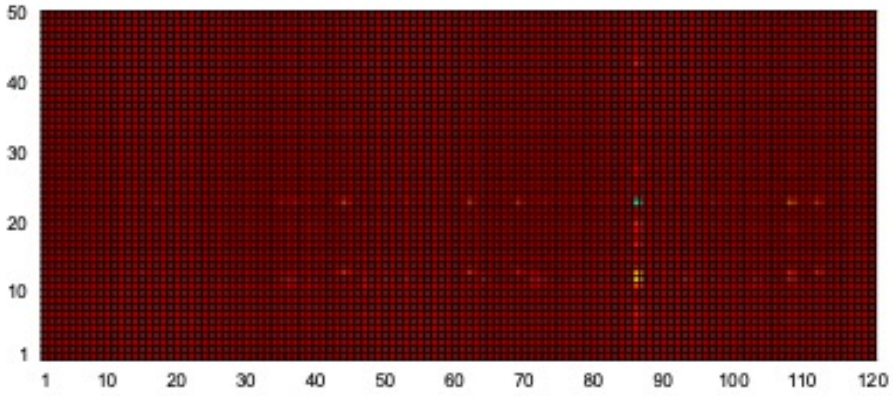
Figs. 4.9b and 4.9d show that positively saturated weights can seriously affect the synapses connecting the last two layers SF4-SF5, while few such critical synapse faults are observed for layers SC3-SF4. Synapse faults in previous layers have no impact and are excluded from Fig. 4.9. The reason behind this observation is that positive saturated weights could cause the post-synaptic neuron to always fire, i.e, saturate.

4.5.2.3 *Bit-Flipped Synapses*

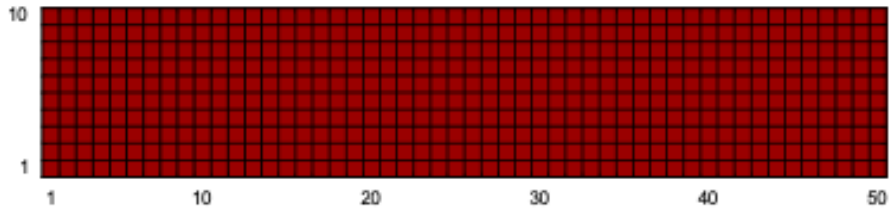
Fig. 4.10 presents the effect of the hardware-aware synapse fault model of bit-flips on the quantized synaptic weights connecting the last two layers SF3-SF4 of the gesture SNN. As shown, a bit-flipped synapse has a smaller impact on the performance of the network. We observe that only bit flips in the first two Most Significant Bits (MSBs) 7 and 6 cause a classification accuracy drop, while for bit



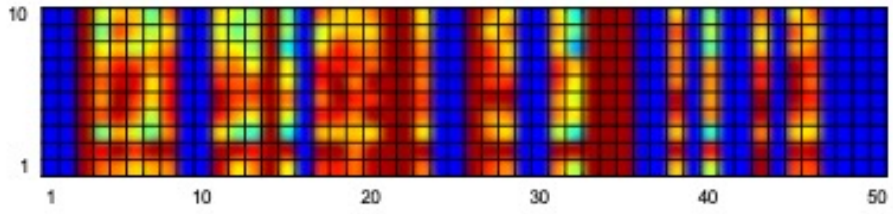
(a) Dead synapses between SC3-SF4.



(b) Positively saturated synapses between SC3-SF4.



(c) Dead synapses between SF4-SF5.



(d) Positively saturated synapses between SF4-SF5.

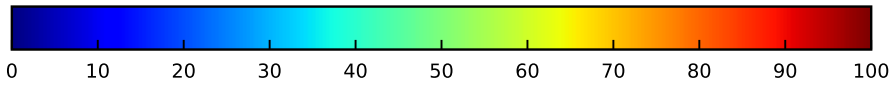
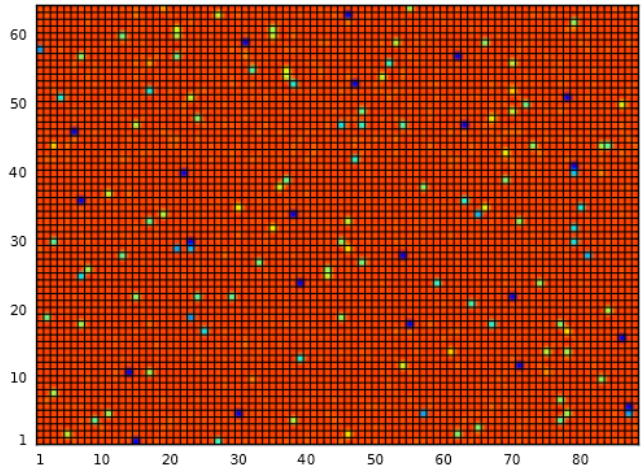
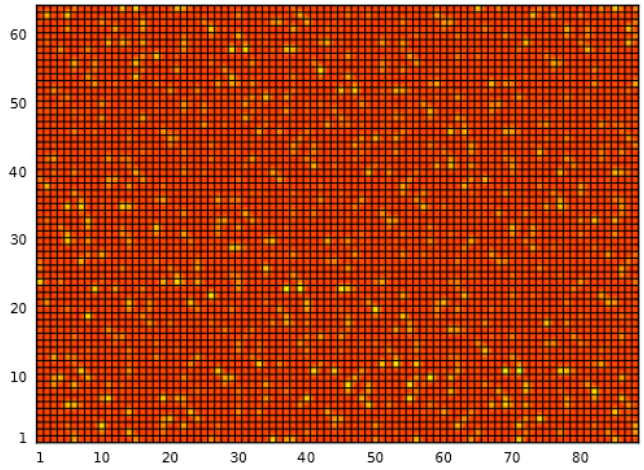


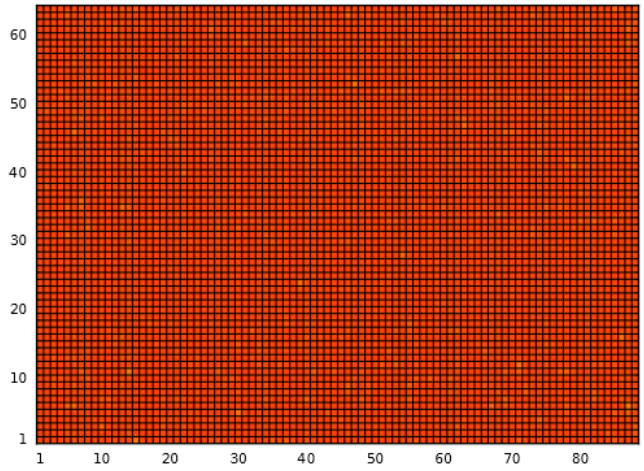
Figure 4.9: Effect of synapse faults on the classification accuracy of the N-MNIST SNN.



(a) Bit 7 (MSB).



(b) Bit 6.



(c) Bit 5.

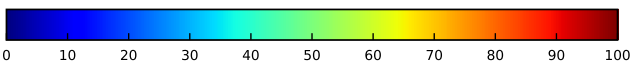


Figure 4.10: Effect of bit-flip synapse faults between SF3-SF4 for the gesture SNN.

5 the baseline accuracy is observed. Bit-flips for bit positions 0-4 have no effect and are not shown in Fig. 4.10.

4.6 FAULTS OCCURRING BEFORE TRAINING

This case considers faults that occur before the network is trained, effectively altering the architecture of the network and its ability to learn the specific task. Under this scenario, it is intuitively expected that for the majority of single faults, the neural network will show great resilience as it has a strong ability to learn around faults, i.e. bypassing the undesired behavior of its faulty part. Therefore, for this part of the study, we need to consider a multiple fault assumption serving as a first order analysis on the most influential faults and setting out to find the fault density beyond which the learning capacity starts degrading.

We confirmed that training the N-MNIST network with one fault or even a few tens of faults injected into neurons of layers SC_3 and SF_4 or into synapses connecting layers SC_3 and SF_4 and layers SF_4 and SF_5 has no effect on the classification accuracy. In contrast, in the case of layer SF_5, neuron faults have a drastic and pre-determined effect, as discussed in Section 4.5, that cannot be masked by training. For these reasons, in this experiment we exclude layer SF_5, and we set out to find the number of fault injections or fault density, i.e. the maximum inherent fault tolerance capability, beyond which classification accuracy starts degrading.

More specifically, we consider only the case of neuron faults in layers SC_3 and SF_4 since, as it has been shown in Section 4.5, faults in neurons are much more likely to cause classification accuracy degradation compared to faults in synapses. Furthermore, among all neuron fault types, we consider only dead and saturated faults since their impact is far greater compared to parametric faults, and when parametric faults become severe, then the neuron approximates either a dead or a saturated neuron.

The fault injection is performed using batches of faults in an accumulative way. More specifically, each batch contains 10 faults that are randomly selected amongst the dead and saturated types and injected across 10 randomly selected neurons in layers SC_3 and SF_4. After that, the network is trained and the classification accuracy is evaluated for the first batch of faults. Then, a second non-overlapping batch of faults is added to the first batch to include 20 faults in total, and so forth. Overall, we performed training with $k * 10$ faults injected at a time, where $k = 1, \cdots, 17$. Each training experiment is repeated 10 times and the average classification accuracy is reported.

The results of this experiment are shown in Fig. 4.11. These results show that the network is capable of masking faults during its training. For up to 100 simultaneous faults, the final classification accuracy on the testing set after 20 training epochs reaches the fault-free value of around 98%. However, as the number of faults increases, we notice a slowing down in the learning curve. For 110 or more faults, we observe a drop in the classification accuracy that worsens with the increase in fault density. For example, for 130 faults the classification

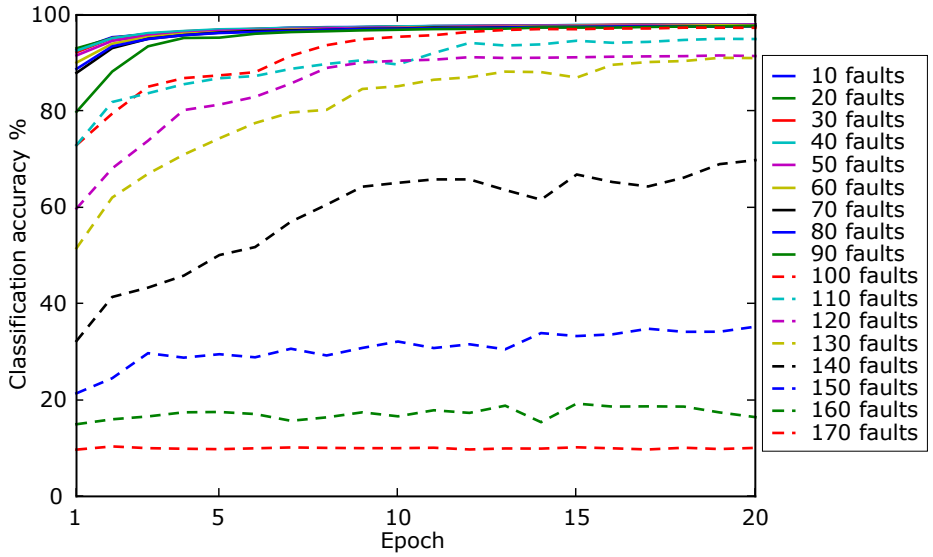


Figure 4.11: Classification accuracy drop as a function of fault density. Only dead and saturated faults in layers SC3 and SF4 of the N-MNIST SNN are considered.

accuracy drops by about 10%, whereas for 170 faults the classification accuracy is stuck at 10% from the first learning epoch, implying that the network performs a random classification. As a conclusion, the SNN can tolerate up to about 100 faulty neurons in layers SC3 and SF4 out of the 170 neurons in these two layers, which is arguably a high defect density. This shows that for the specific cognitive task the network presents a lot of redundancy and can be pruned to save area and power.

5 NEUROMORPHIC HARDWARE EXPERIMENTATION PLATFORM

Chapters 3 and 4 focused on the SNN resilience from a software point of view. Although the injected faults were derived from a hardware design, the neurons and synapses were considered isolated from each other. The use of actual neuromorphic hardware to perform the network's calculations imposes new challenges that cannot be simulated in software. The reason behind this stands in the practical components that build up the system and are used for second-role operations, like the transmission of the spikes among the neurons. In terms of software, tensors are used to hold the synaptic weights and the states of the neurons and are updated after a relevant operation. On the other hand, concerning a hardware approach, neurons of different layers need to communicate with each other in order to pass the generated information, i.e., spike trains, and this subpart of the system may fail equivalently.

This chapter presents a neuromorphic hardware experimentation platform used to accelerate the inference of convolutional SNNs and which constitutes the basis of the experiments held as part of the neuromorphic hardware reliability analysis in Chapter 6, the on-line testing procedure in Chapter 9, and was used to validate the compact functional test generation approach in Chapter 7. The hardware accelerator is designed in Very High-Speed Integrated Circuit Hardware Description Language (VHDL) and is implemented on a FPGA board as part of an ARM-based embedded system that complements the utilization and evaluation of the accelerator.

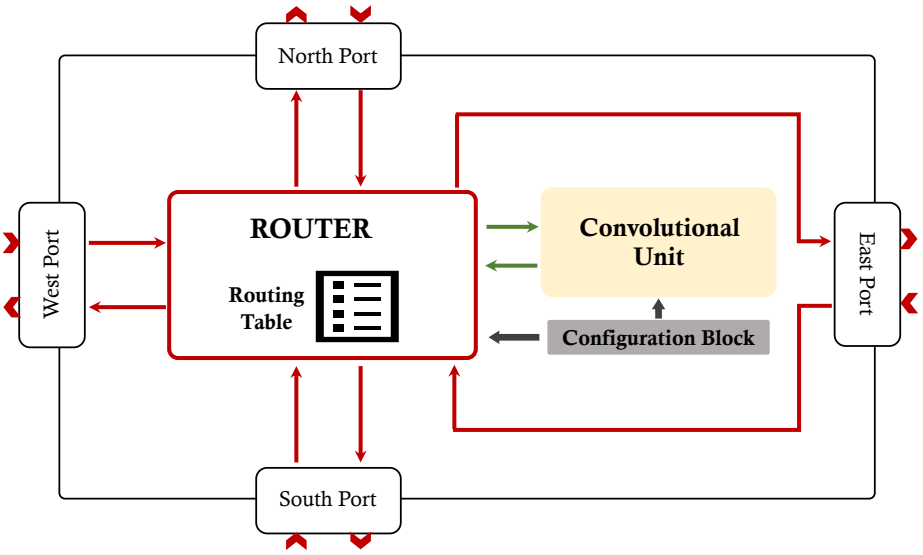


Figure 5.1: The convolutional node.

5.1 NEUROMORPHIC HARDWARE ARCHITECTURE

The building block of the SNN is an event-driven configurable convolutional node proposed in [66] as a generic block that can be used to build multi-layer feature maps for convolutional SNNs communicating through the AER protocol. This section presents the node along with a brief description of its most important features.

5.1.1 *The Convolutional Node*

The node consists of three main blocks, namely a convolutional unit, an internal configuration block, and a router, as shown in Fig. 5.1. The ports of the node are optimized for a 2-D layout, an efficiently adopted structure in hardware CNNs since it optimizes the use of on-chip space. Each node has four bidirectional ports connecting it to its immediate neighbors to the north, south, east, and west.

5.1.1.1 *The Convolutional Unit*

Shown in Fig. 5.2, the convolutional unit consists of an array of I&F neurons representing pixels, three main memory blocks (i.e., kernel memory, neuron memory, and rate-saturation memory), First-In-First-Out (FIFO) input and output registers, a controller block, and a Serial Peripheral Interface (SPI) block.

The convolutional unit is where the convolution of an input flow of events $ev_{in}(t, x, y, p, k)$ and a kernel $w_k(x, y)$ takes place to produce an output flow of events $ev_{out}(t, x, y, p)$, where t is time, x and y are the pixel address coordinates, p is the polarity of the event, and k is the kernel ID in the kernel memory. In addition, the unit has two more important features: *global leakage* and *rate saturation*. Leakage is the decay of the neuron membrane potential in between

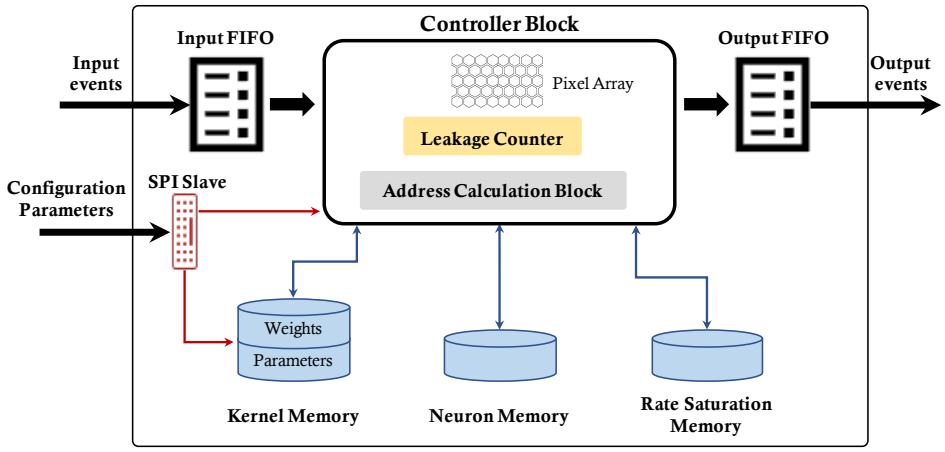


Figure 5.2: The convolutional unit.

incoming spikes and it is implemented by forcing the neuron state to converge towards the reset value after a certain time interval, which is determined by a global counter. The rate saturation feature, on the other hand, is the imposition of the minimum refractory period property found in biological neural networks, i.e., the neuron is not allowed to produce an output spike for a certain period after the last output spike, hence controlling the maximum spiking frequency of a neuron.

With every incoming event read from the input FIFO register, a convolution operation is executed and the values of the corresponding pixels are updated and compared to the positive and negative thresholds. If the value of a threshold is reached by a pixel and the condition imposed by the rate saturation mechanism is fulfilled, the pixel produces an output event with address (x_{out}, y_{out}) and polarity p_{out} and writes it in the output FIFO register.

To control the traffic, a signal is activated when the registers get full, and the incoming events are discarded until there is room for more events in the register. While this implies that the output events would get down-sampled and some information will eventually be lost, the spatio-temporal correlation of the passing events is preserved, keeping the integrity of the carried information.

5.1.1.2 *The Router*

In hardware implementations of neural networks, the highly dense connectivity required between neurons poses a challenge in terms of on-chip area. Routers handle the transmission of events from their origin to their destination, hence providing a practical solution to this problem [109]. In this design, the destination-driven addressing scheme is adopted, which means that for every event, there is a routing header that carries the x and y coordinates of the destination node in the mesh distribution of the network.

5.1.1.3 *The Configuration Block*

The convolutional node has a set of adjustable parameters that need to be configured prior to the inference operation of the network. Some parameters belong to the convolutional unit, such as the neuron threshold and the kernel weights. Others belong to the router, such as the local address of the node and the routing table information necessary for redirecting events through the ports of the node. Each parameter value is sent to the node through a SPI, with an index indicating its identity. The configuration block interprets the parameter identities and handles their allocation in their corresponding locations in the different memory blocks.

5.1.2 *Memory Hierarchy*

As discussed in Section 5.1, the generic convolutional node used as a building block in this hardware implementation is configurable through a set of modifiable parameters. These parameters are stored in mutually exclusive memory blocks inside the node, which are the subject of our reliability experiments. The parameters have an 8-bit representation in hardware and can be categorized into:

1. ***Splitter Parameters:***
   The splitter is parametrized by the number of input event copies to generate and send to each first-layer node, as well as the node addresses. Splitter parameters make up 0.52% of the used memory. Splitter parameters are stored in registers with a total size of 12B, making up about 0.013% of the entire memory.

2. ***Router Parameters:***
   The router needs two important settings, namely the local address of the node and the routing information necessary for redirecting events through its ports, i.e., addresses of next-layer nodes and the direction towards them (down, right). The router also carries the kernel ID information so that the correct kernels corresponding to each event can be retrieved from the memory. Router parameters make up 12.39% of the used memory. These parameters are also stored in registers with a total size of 298B and make up a little over 0.3% of the entire memory.

3. ***Neuron Parameters:***
   Neuron parameters govern the key features of the I&F neurons within the node. They include the neuron threshold, the leakage pulse amplitude and period, and the refractory period. These parameters are set for the whole node, i.e., they are global to the whole array of neurons inside a node. Neuron parameters make up 7.8% of the used memory. These parameters are again stored in registers with a total size of 180B, occupying less than 0.2% of the entire memory.

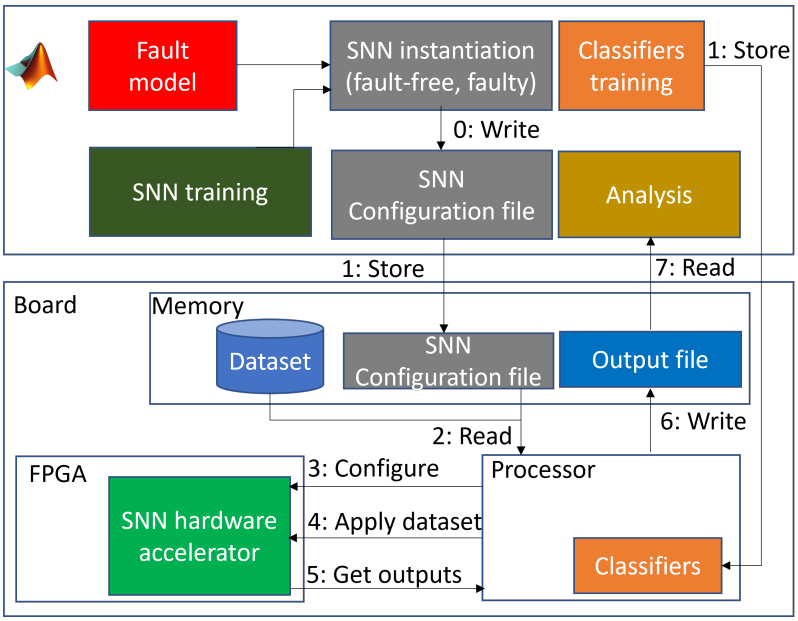


Figure 5.3: Embedded neuromorphic hardware accelerator platform.

4. ***Kernel Parameters:***
   Every node of the network has a specific number of kernels and needs two parameters per kernel, namely the kernel size, which defines the size of the word needed to store each kernel, and the center-shift of the kernel which determines whether the kernel is applied to the pixels in the zone around the one given by the event destination address or it is shifted to another pixel. Kernel parameters make up 10.75% of the used memory. These parameters are stored in a RAM of size 2.25KB, making up around 2.5% of the entire memory.

5. ***Kernel Weights:***
   The number of kernel weights per kernel is determined by the kernel size, i.e., a $n \times k$ kernel has $n * k$ weights. The kernel weights considering all kernels in all nodes of the network make up 68.55% of the used memory. The kernel weights necessary for the convolution process with the input events are stored in a RAM of size 88KB that makes up about 97% of the entire memory. This kernel memory size is determined by the number of kernels and the maximum acceptable size of each kernel.

5.2 EMBEDDED SYSTEM DESIGN

Like the majority of ASICs, our SNN hardware accelerator is a complex circuit that does not stand on its own. Thus, it is necessary to have a controlling system that is in position to communicate with the hardware design via its Input/Output (I/O) pins. For this, we make use of the ARM Cortex A53 on-board processor of the Zynq®UltraScale+™ MPSoC ZCU104 FPGA board, where the accelerator design is flushed. To automate the whole procedure of the configuration, injection, and evaluation of the network, we built a framework to

support these operations. Fig. 5.3 summarizes the embedded system design and how its building components cooperate.

5.2.1 *Controlling Processor*

Controlling a high-performance circuit, like a SNN hardware accelerator, is a very demanding procedure and can become quite challenging. The huge amount of events consisting a neuromorphic dataset creates the need for a high input and output throughput in the circuit. The average time between successive input events in only a few nanoseconds, leaving only tiny time windows for the controller to perform the needed operations to set the next input and store any output that might have be raised asynchronously. It is evident that the communication with the circuit is required to be fast enough, so that it does not interfere with its operation. This way, any time delays are avoided and the temporal dependencies of the spike trains, that play a major role in the processing of SNNs, remain intact.

For the reasons above, and in order to have a more compact solution, we selected to create a standalone system application in C running on the quad-core on-board processor of the platform. We also exploit the benefits of the multiple cores in order to parallelize some of the tasks where possible and thus execute them without affecting each other, e.g., there is no need to wait for a store operation of an output event to finish before sending a new input event request. This becomes more evident for the more demanding operations related to the on-line testing features of the platform, which are explained in details in Chapter 9.

To automate the reliability analysis, as presented in details in Chapter 6, the platform supports a batch mode during which a group of experiments is loaded at once to the SD card. The C application iterates over the experiments and executes them successively, after resetting the SNN accelerator at the end of each experiment in order to generate independent results. For each experiment, the following actions are coordinated by the controlling application:

1. The configuration of the SNN. During this step, the configuration file of the experiment containing the values of all the parameters mentioned in Section 5.1.2 is read from the SD card and the values are written in the corresponding memories sequentially via the SPI of the SNN accelerator.

2. The generation of the input spiking events according to the dataset. The dataset file, stored in AEDAT format, is read from the SD card. When the execution time reaches the timestamp of the next event in the line, a spiking event is generated. The controlling application then sends an input request signal to the SNN accelerator and waits for an input acknowledgment signal before it sends over the event.

3. The monitoring of the output spiking events and their storage to the SD card. The SNN accelerator may produce an output event at any time. Whenever this happens, it sends an output request signal to the controlling

application and the latter acknowledges its successful reading after having stored it.

5.2.2 *Support Framework*

The setup of a SNN and its necessary components before its deployment on the platform, as well as the post-analysis of the experimental results, do not run concurrently to the inference of the network, and therefore these operations are handled by an external support framework. The support framework is designed in MATLAB and is responsible for all these operations taking place before and after the execution of an experiment. The major tasks handled by the framework are listed below:

- Generation of the configuration data for the nominal SNN. The configuration file containing all the information to be stored on the accelerator's memories is exported in a binary format. The configuration file can be then stored to the SD card for a fault-free experiment, or be passed to the next script for a fault injection experiment.
- Injection of one or multiple faults into the configuration data to generate faulty versions of the SNN. The resulted files are stored to the SD card. A detailed description of the fault model is presented in Section 6.2.
- Training of the classifiers used for the on-line testing of the hardware accelerator (see Ch. 9).
- Processing of the output spiking events written to the SD card in order to classify the results. The winning class is the one whose node produces the largest sum of spikes. Note that spikes have either a positive, or a negative polarity, thus the absolute number of spikes is different than the sum of the spikes' polarities. This step also calculates the SNN recognition accuracy over the testing set.

6 RELIABILITY ASSESSMENT OF NEUROMORPHIC HARDWARE

The transfer of neural networks into hardware unavoidably makes them susceptible to hardware-level faults, despite the parallelism and sparsity that defines them. Hardware-level faults can occur either during manufacturing, such as physical defects and process-induced variations, or in the field due to environmental factors and aging. The performance under fault scenarios needs to be assessed so as to develop cost-effective fault-tolerance schemes.

In Chapters 3 and 4, resilience characteristics of SNNs have been studied by performing fault injection at transistor-level for single neurons and in a behavioral-level model for entire networks, respectively. Although experimenting on higher abstraction models allows flexibility, the particularities of a hardware implementation are not taken into consideration.

In this chapter, we assess the resilience characteristics of actual neuromorphic hardware, and particularly of the neuromoprhic hardware accelerator for SNNs that was presented in Chapter 5. The fault injection experiments pinpoint the parts of the design that need to be protected against faults, as well as the parts that are inherently fault-tolerant [110].

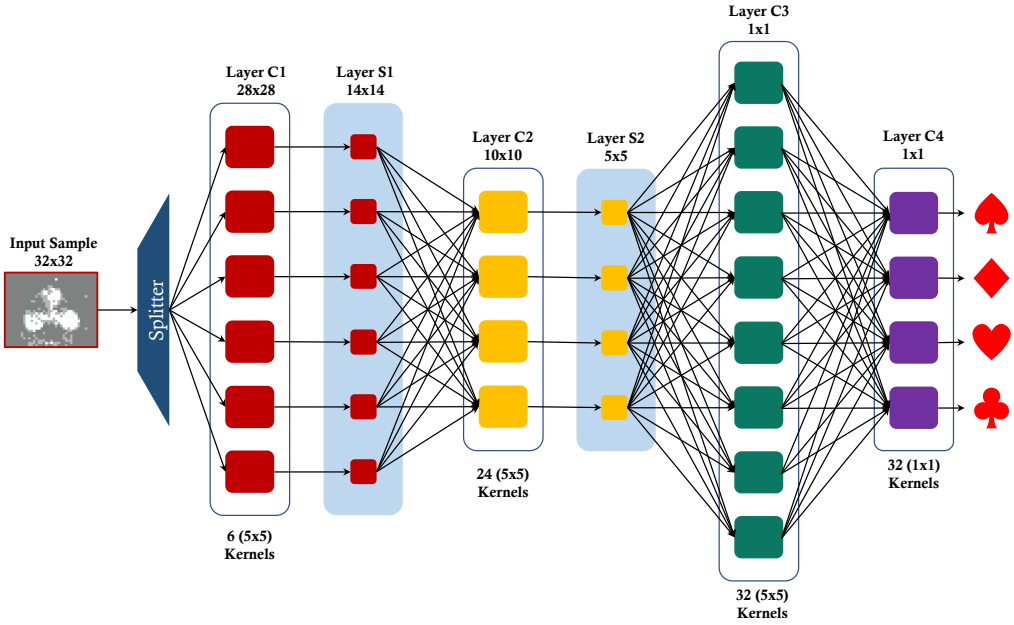


Figure 6.1: Convolutional SNN for poker card symbol recognition.

6.1 CASE STUDY

The SNN used in this work is built to classify a dataset representing the 4 poker card symbols [111]. A deck of 40 poker cards was presented in front of a DVS for a period of around 1 s. The events were recorded and processed in order to generate 40 samples of 32×32 pixel windows showing only the centered symbols. The resulting stimulus has a total of 174644 events, a duration of 950 ms, and an average speed of 184K events per second. In our experiments, we use a version of the dataset slowed down to 1% of the original speed in order to ensure a scenario where no input events are discarded.

The convolutional SNN is designed and trained in software in a frame-based format using backpropagation, and then transformed into the equivalent spiking form [111]. Afterwards, the weights and parameters are scaled, rounded, and then tuned to make up for the discrepancies between hardware and software implementations using simulated annealing as an optimization algorithm [66].

As shown in Fig. 6.1, the SNN consists of 4 convolutional layers (C1, C2, C3, and C4) made up of 22 convolutional nodes. The first 2 layers are followed by 2 sub-sampling layers (S1 and S2). The network has 94 kernels in total, where layer C1 has 1 kernel per node, layer C2 has 6 kernels per node, layer C3 has 4 kernels per node, and layer C4 has 8 kernels per node.

The 2-D hardware layout of the FPGA implementation is shown in Fig. 6.2, where the nodes are arranged in a 6×4 mesh with bidirectional connections between the routers of each block and its immediate neighbors. Every node carries an identification corresponding to its x and y address in the mesh, and its color indicates the respective layer in the network. Nodes (5,4) and (6,4) are extra nodes added for routing purposes but do not perform any processing.

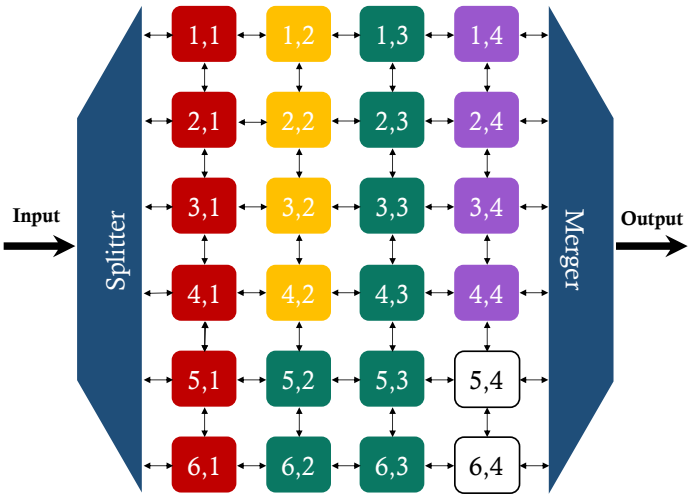


Figure 6.2: 2-D mesh SNN implementation on the FPGA.

Input events do not have any specified destination address and they need to be sent to all nodes of the first layer. Therefore, there is an extra *splitter* block at the input side, which creates 6 copies of every incoming event, adds the address of a node in the input layer to each copy, and delivers them to the corresponding nodes. At the output side, there is a *merger* block which simply forwards events from the 4 nodes of the output layer to the output of the network without altering them.

6.2 FAULT MODEL

The fault model consists of permanent bit-flips in the memories mentioned in 5.1.2. Bit-flips are injected in two different ways, in particular with a Bit-Error Rate (BER) probability, leading to a multiple-bit fault scenario with uniform random distribution of bit-flips, or considering a single-bit fault scenario. In the former scenario, assigning different BER probabilities helps assessing the BER that can be tolerated by the SNN. We consider BER probabilities up to 10^{-1}. This value is justified for memristor-based implementations since memristors have low yield and endurance, reduced-voltage memory operation, and harsh environments. The maximum tolerated BER depends on the criticality of the application. In the latter scenario, we study the effect of single bit-flips parameter-by-parameter, layer-by-layer, and for different bit positions. The goal is to identify critical parts of the design, as well as critical bit positions.

6.3 RELIABILITY ANALYSIS

For each fault injection experiment we evaluate the SNN recognition rate and compare it with the baseline value for the nominal design which is $85 \pm 2.5\%$. We consider that an accuracy drop to 82.5% is still acceptable.

Each fault injection experiment takes approximately 2 minutes including (i) the configuration time, (ii) the execution time, and (iii) the storing time. In the

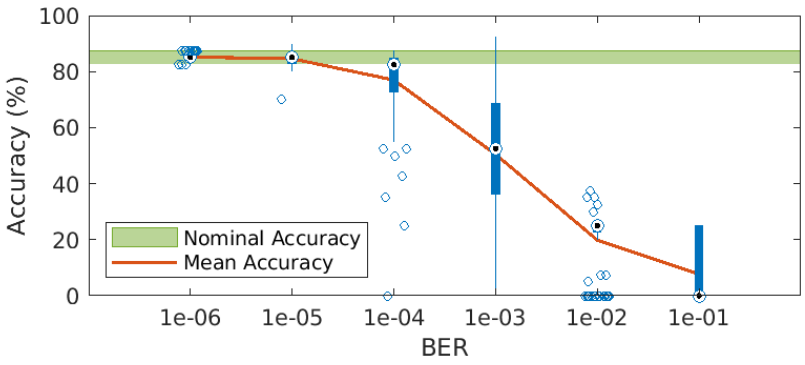


Figure 6.3: Network accuracy for different BER values.

multiple-fault scenario, for a given BER value, we perform 100 repetitions. As the total memory size is 18464 bits, it is very time-consuming to perform all single bit-flip scenarios even in hardware where run-time is accelerated. Thus, for the kernel weights in the first three layers C1, C2, and C3 that occupy the largest fraction of the memory, we perform fault sampling, randomly selecting 20%, 10%, 10% fault locations, respectively. For the rest of the parameters we perform exhaustive fault injection. In total, we performed 11925 fault injections which took approximately 16.5 days of simulation time.

We visualize summary statistics using box plots of the network accuracy versus BER (Figs. 6.3 and 6.5), or bit position (Fig. 6.4). The bottom and top edges of the box indicate the 25th and 75th percentile, respectively, the whiskers extend to the most extreme data points not considered outliers, and the outliers are plotted individually using the 'o' symbol and are not always aligned vertically for illustration purpose. We also report the median shown with a dotted circle and the average accuracy across repetitions of the same experiment. Experiments with 0% accuracy correspond to an application crash as the result of fatal errors which made the system unable to respond to any incoming event activity, i.e., the impact on the network is catastrophic.

6.3.1 *Global Fault Injection Results*

Fig. 6.3 shows the accuracy versus BER in the case where bit-flips are injected uniformly at random across the entire network. As it can be seen, the accuracy drops with increasing BER and beyond 10^{-4} the drop is below the tolerated zone shown with green color. This shows that the maximum tolerated BER is 10^{-5} or less. We also observe that for moderate BER values the variance of the accuracy increases as BER increases, which shows that accuracy drop is largely dependent on the combinations of faulty bits and their locations. Another observation is that for BER=10^{-5} there is a network instance that performs with accuracy higher than 95%. In essence, performing random multiple bit-flips is equivalent to a random brute-force training. Interestingly, a combination of bit-flips resulted in small perturbations of network parameters that improved the result of the original training algorithm.

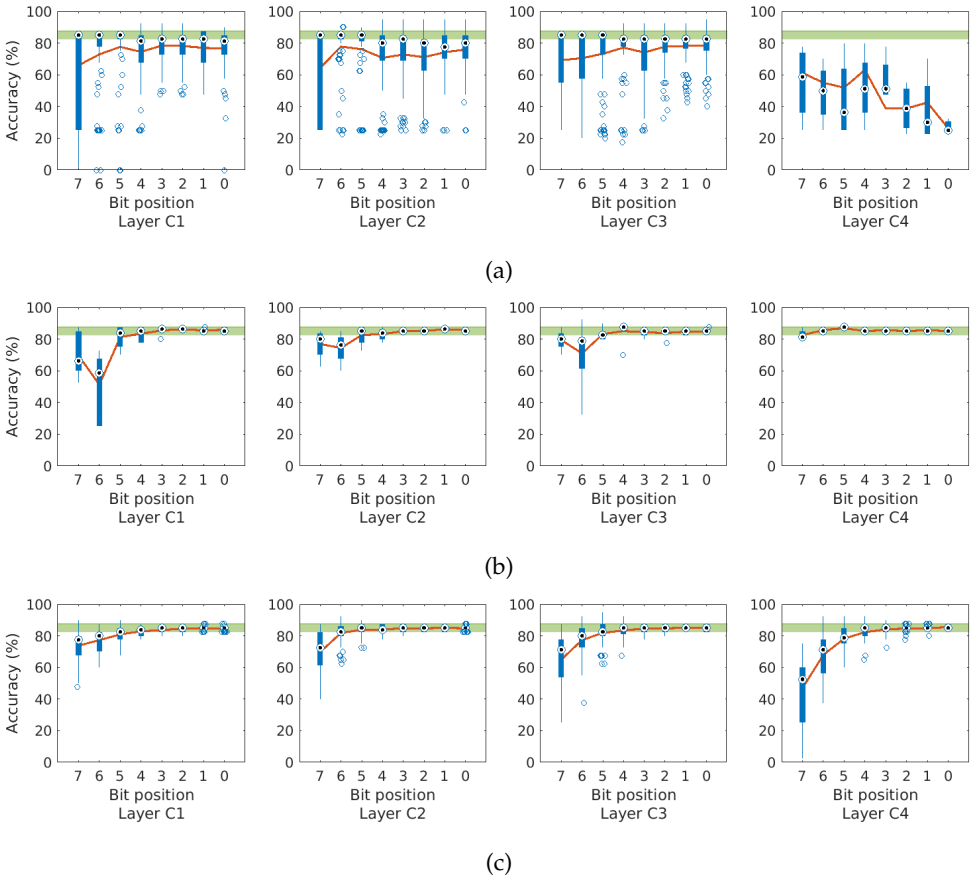


Figure 6.4: Single bit-flips layer-by-layer: (a) Router Parameters; (b) Neuron Threshold; (c) Kernel Weights.

6.3.2 *Targetted Fault Injection Results*

The outliers for moderate BER values of 10^{-4} and 10^{-5} in Fig. 6.3 are due to faults occurring is the smaller yet far more critical memory blocks storing splitter, router, and kernel parameters. In fact, the network is susceptible even to single bit-flips affecting these parameters. Faults in the splitter may lead to generated addresses that do not correspond to a valid node inside the mesh, and this leads to input events not reaching any destination address and, thereby, not being processed. For several experiments, the network was unresponsive, i.e, there were no output spiking events. Faults in the kernel parameters change the kernel's size and center-shift and faults in the router parameters change the routing of the spikes. Thus, such faults essentially result in a structurally different network architecture. For example, Fig. 6.4a shows single bit-flips in the router parameters layer-by-layer and across different bit positions. The network is very sensitive and the bit position where the flip occurs is irrelevant. Thus, we conclude that splitter, router, and kernel parameters are critical and protecting them is of utmost importance.

Figs. 6.4b and 6.4c show results for single bit-flips layer-by-layer for the neuron threshold and kernel weights, respectively. For the single-bit fault scenario, among the different neuron parameters, accuracy drop was observed only for

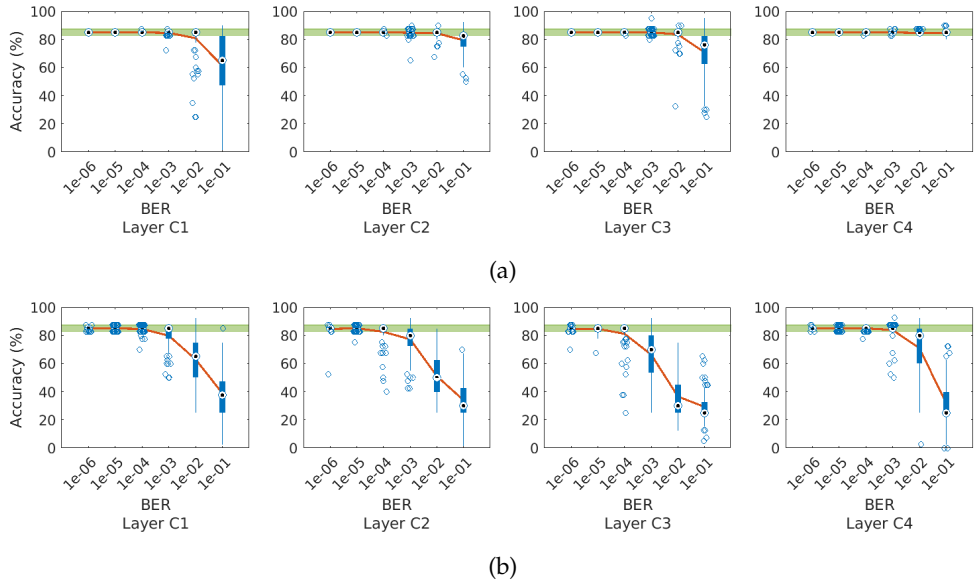


Figure 6.5: Multiple bit-flips for different BER values layer-by-layer: (a) Neuron Parameters; (b) Kernel Weights.

the neuron threshold, thus Fig. 6.4b shows results only for the neuron threshold. Figs. 6.5a and 6.5b show results for multiple bit-flips with different BER values layer-by-layer for the neuron parameters and kernel weights, respectively.

From Fig. 6.4b we observe that the network performance is sensitive to single bit-flips in the neuron threshold only if these occur in the 3 MSBs, while the last layer C4 shows no vulnerability. From Fig. 6.5a we observe that layers C1 to C3 start showing vulnerability for BER values larger than 10^{-2}. Thus, if such high failure rates are expected, neuron parameters must be protected too.

Regarding kernel weights, from Fig. 6.4c we observe that the network sensitivity increases with the layer number. Thus, some faults in the beginning of the network tend not to propagate. Another observation is that single faults affecting Least Significant Bits (LSBs) can be tolerated. For the first layer C1 the first 3 MSBs are critical, while for the last layer C4 the first 4 MSBs are critical. From Fig. 6.5b we observe that the network can tolerate up to a BER of 10^{-5}. These results show that leaving unprotected the 4 LSBs of the kernel weights, which always occupy the largest fraction of the memory, is feasible, leading to significant cost reduction in fault tolerance schemes.

7 COMPACT FUNCTIONAL TESTING FOR NEUROMORPHIC CIRCUITS

In this chapter, the problem of testing AI hardware accelerators implementing SNNs is addressed in a generalized way [112]. We define a metric to quickly rank available samples for training and testing based on their fault detection capability. The metric measures the inter-class spike count difference of a sample for the fault-free design. In particular, each sample is assigned a score equal to the spike count difference between the first two top classes. The hypothesis is that samples with small scores achieve high fault coverage because they are prone to misclassification, i.e., a small perturbation in the network due to a fault will result in these samples being misclassififed with high probability. We show that the proposed metric correlates with the per-sample fault coverage and that retaining a set of high-ranked samples in the order of ten achieves near perfect fault coverage for critical faults that affect the SNN accuracy. The proposed test generation approach is demonstrated on the N-MNIST and IBM's DVS128 Gesture SNNs (see Chapter 4) and on the SNN for the poker card-symbol recognition (see Chapter 6) using the hardware experimented platform described in Chapter 5. Finally, the fault space is reduced so as to offer a faster test generation time.

7.1 RELATED WORK ON TESTING AI HARDWARE ACCELERATORS

High-volume manufacturing of ASIC AI hardware accelerators is foreseen in the near future. On-die neural networks have been explored in the past for building an on-die "test brain" that classifies chips as functional or faulty [113]. The "inverse" problem, i.e., how to efficiently test AI hardware accelerators, however, is an emerging problem [114]–[116].

In general, existing and proven test methods for traditional computing devices can be portable to AI hardware accelerators. Nevertheless, the unique architectural features of AI hardware accelerators make these test methods less efficient and give rise to new test challenges. For instance, AI hardware accelerators usually consist of multiple identical cores, e.g., the MAC units (also referred to as Process Elements (PEs)), which are too small to implement traditional Design-for-Test (DfT) techniques, e.g., scan test, with reasonable overhead. Another characteristic of AI hardware accelerators is that they are memory-hungry, with the memory storage being dominated by the synapse weights which can be in the order of millions. Testing large embedded memories with today's Memory Built-In Self-Test (MBIST) tools can pose large Power-Performance-Area (PPA) penalties.

Traditional fault models, such as stuck-at, delay, and cell-aware fault models, can be reused as well in the context of AI hardware accelerator testing, but emerging in-memory computing architectures based on memristive crossbars [117] or SNNs [118]–[120] for example, require new fault models. Moreover, fault models for AI hardware accelerators could be defined in software at a higher-abstraction behavioral-level, i.e., variations in neuron outputs and synapse weight values [40], aiming at speeding up test generation (see Chapter 4). This is because software and hardware implementations of neural networks closely match together [97].

Most faults are benign, that is, they affect a component that does not take part in the computation, they are completely masked thanks to the information propagation through the network, they change the order of the top predicted classes but not the top-1 class, or they lead to inaccurate predictions for only a tiny fraction of the inputs and in this sense they can be tolerated. However, some faults remain critical and can lead to a large drop of correct classification percentage, and test efforts can focus on these critical faults to reduce test time [116]. This fault behavior has been demonstrated in several recent fault injection experiments at software level [80], [84], [92]–[94], [121], [122] and in AI hardware accelerators [81], [82], [86], [87], [97], [110], [123], which show that there is some inherent fault tolerance since many faults are benign. Fault tolerance is further discussed in Chapter 8.

Test methods that take into account the architectural particularities of AI hardware accelerators have started surfacing recently. DfT methods suited for AI hardware accelerators based on large arrays of small PEs are proposed in [114], [124]. Test generation algorithms aiming at creating a compact set of functional tests that can detect the presence of faults are proposed in [125], [126],

[93]. Symptom detectors that detect some anomaly in intermediate nodes, i.e., high neuron activation, are proposed in [80], [81], [88], [94]. Selective Triple Modular Redundancy (TMR) applied to the most critical neural network layers is proposed in [80], [90], [127]. Algorithmic-based error detection and correction methods using checksum arithmetic are discussed in [86], [128]–[131]. On-line test methods are proposed in [132] based on Software Test Libraries (STL), in [133] based on a simplified metric of dynamic power consumption, and in [134] based on encrypting weights in the memory with an encryption algorithm that spreads single bit-flips extending them to multiple bit-flips and checking if the padding bytes used for the encryption to work properly are correctly decrypted.

Specifically now for SNNs, in [80], [94], symptom detectors are designed for testing for the two main catastrophic fault mechanisms in SNNs, namely neuron saturation and large synapse weight drifts. The symptom detector that we propose will be described in more detail in Chapter 8. In [85], a BIST for biological spiking neurons is proposed where the neuron is exercised from its bias voltages to span the entire range of operation and output all possible firing patterns. If a pattern is missing, then the neuron is labelled faulty. And finally, a functional test generation method for SNNs is proposed in [93]. Functional test generation methods for AI hardware accelerators, including [93], which are directly related to the work presented herein, will be discussed in greater detail at the end in Chapter 7.7.

7.2 PROPOSED FUNCTIONAL TEST GENERATION ALGORITHM

Functional test generation aims at either identifying or generating new input samples that are capable of sensitizing the fault and propagating its effect to the output, leading to a different prediction with respect to that of the nominal fault-free network. As shown in Fig. 7.1 using as an example an image recognition cognitive task, these samples could be original images from training and testing sets [125], [126], adversarial examples generated from original images [93], or synthetic images generated from original images [135], [136]. The proposed algorithm for SNNs selects tests from the set of available samples in the training and testing sets.

Let us consider an SNN employed for an N-class classification cognitive task. The SNN has N neurons in the output layer each corresponding to one class. We consider that the SNN uses the firing rate as classification criterion, i.e., the winning class is the one whose corresponding neuron produces the largest number of spikes within a given duration interval.

Let us consider also a set of input samples of cardinality M. For a given sample t_i, $i = 1, \cdots, M$, let n_i^j denote the spike count for output neuron j, $j = 1, \cdots N$. We rank the values n_i^j from high to low, resulting in the ordered set $\{n_i^{(1)}, n_i^{(2)}, n_i^{(3)}, \cdots\}$, i.e., the neuron that produces $n_i^{(1)}$ spikes corresponds to the top-1 class, the neuron that produces $n_i^{(2)}$ spikes corresponds to the top-2 class, and so forth.

Then, we define the *margin*

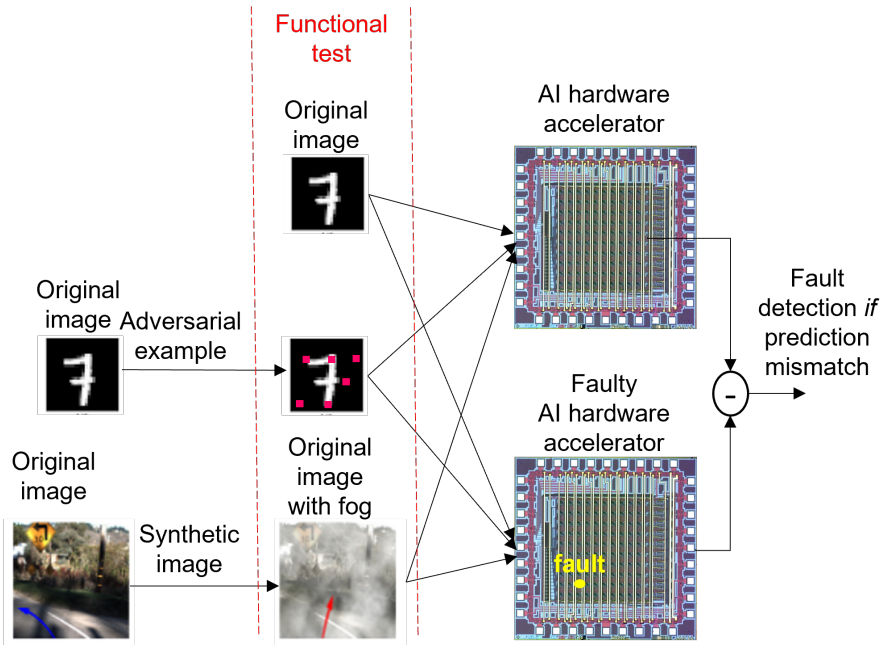


Figure 7.1: Functional test generation. The street images are from [136]. The chip image is from [113].

$$X_i = n_i^{(1)} - n_i^{(2)}, \tag{7.1}$$

i.e., X_i is the difference in spike count between neurons corresponding to the top-1 and top-2 classes. The *quality metric* of sample t_i is defined by the score

$$q_i = \frac{1}{X_i} \tag{7.2}$$

The rationale is that samples with high scores (or, equivalently, small margins) are likely to be *distinguishing samples*, i.e., they are prone to producing different top-1 class predictions for the nominal and faulty networks. This is based on the intuition that when the first top classes are close in terms of firing rate, the network has low confidence in its decision and it is likely that a fault will alter the class ranking for this particular sample. In other words, this sample lies close to the multi-class classification hyper-boundary and is likely to be misclassified when a fault occurs.

Let us consider now the ranking of samples based on their scores from high to low, resulting in the ordered set $\{t^{(1)}, t^{(2)}, \cdots, t^{(i)}, \cdots, t^{(M)}\}$, where $t^{(1)}$ is the sample with the highest score, and so forth. A functional test set of size T can be generated by considering the first T higher-score samples in this ordered set.

As we will see in our experimental results, the score in Eq. (7.2) directly correlates with the per-sample fault coverage, i.e., the samples fault coverage is shown to increase linearly with the samples score. To this end, the proposed

algorithm first performs M inferences, i.e., one inference per available sample, on the nominal network, then ranks the samples according to their score. Next, starting from the top ranked sample and sequentially adding the next top ranked samples, it evaluates the cumulative fault coverage. In each step, the detected faults are dropped from the fault list. Let $N_{uf}(i)$ denote the number of undetected faults at the beginning of iteration i where the i-th ranked sample is examined, i.e., $N_{uf}(1) = K$ for a fault model of size K. The algorithm stops adding samples when the fault coverage saturates. Our experimental results show that the size of the resultant test set needs to be in the order of few tens of samples for reaching 100% fault coverage for critical faults. For a test set of cardinality T, the total number of inferences which dominates the test generation time is

$$N_{inf} = M + \sum_{i=1}^{T} N_{uf}(i) \tag{7.3}$$

where the first term corresponds to test generation and the second term to fault coverage evaluation.

Finally, for a fault model of size K, let F_k denote fault k. We define the following indicator function for test t_i

$$I^{t_i}(F_k) = \begin{cases} 1 & : \ F_k \text{ is detected} \\ 0 & : \ \text{otherwise} \end{cases} \tag{7.4}$$

where detection means that the responses of the nominal fault-free network and the faulty network with fault F_k injected differ, i.e., a different class is predicted.

The fault coverage of test t_i indicates the percentage of faults detected by this particular test. It is defined as

$$FC(t_i) = \frac{\sum_{k=1}^{K} I^{t_i}(F_k)}{K} \tag{7.5}$$

Considering a test set of cardinality T denoted by $\{t_1, \cdots, t_T\}$, its global fault coverage is defined as

$$FC = \frac{\sum_{k=1}^{K} \min(1, \sum_{i=1}^{T} I^{t_i}(F_k))}{K} \tag{7.6}$$

7.3 SNN CASE STUDIES

As case studies, we use three convolutional SNNs performing three different cognitive tasks, namely a SNN trained to classify the N-MNIST dataset [65], an SNN trained to classify IBM's DVS128 Gesture dataset [107], and a SNN trained to classify the 4 poker card symbols [66]. The networks are explained in detail

in Sections 4.3 and 6.1. respectively, and their architectures are shown in Figs. 4.3, 4.4, and 6.1, respectively. In each case, the winning class is declared based on the most triggered neuron at the output layer.

7.4 FAULT SPACE REDUCTION

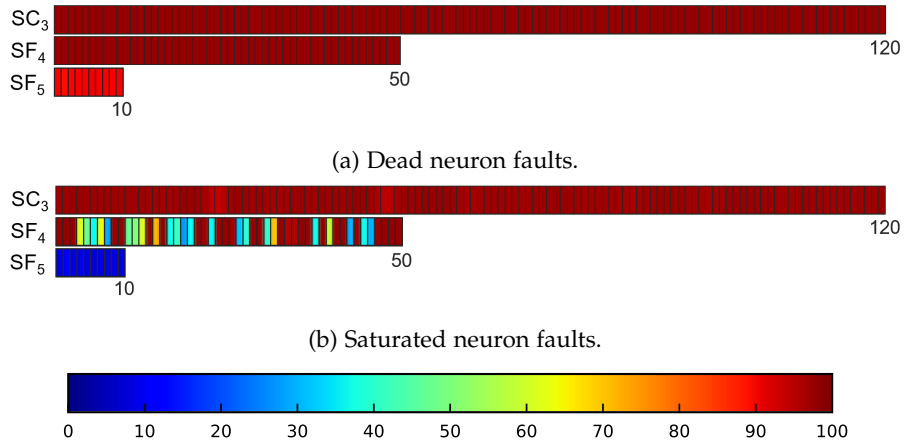
The common conclusion of several published fault injection and reliability experiments for ANNs [81], [82], [86], [87], [97], [123] and SNNs [80], [84], [92]–[94], [110], [122] is that not all faults are equal. A large number of faults are either completely masked or they induce a negligible drop in the network classification accuracy. Such benign faults can be excluded to reduce the fault space and speed up fault simulation that is invoked several times during test generation. Including all faults can quickly make fault simulation intractable, even for small-size networks. For example, for the N-MNIST and IBM's DVS128 Gesture SNNs the number of neurons is 1756 and 25099, respectively, and the number of synapses is 57488 and 1059616, respectively. Consequently, a prudent elimination of benign faults is required so as to avoid inadvertently excluding critical faults.

In the context of this work, we define benign and critical faults in a more strict fashion. A fault is considered benign if the response of the nominal network and the network in the presence of the fault match on a sample-by-sample basis with a certain tolerance, e.g., a 0% tolerance requires an exact match between both responses. If the number of mismatched samples between the nominal network response and that of the faulty network is above the tolerance, the fault is considered critical.

7.4.1 *Behavioral-Level Fault Model*

For SNNs, in Chapter 8, it is shown that training with dropout [137] offers proactive fault resilience against dead neuron faults and neuron timing variations. Dropout was originally proposed to prevent over-fitting and reduce the generalization error on unseen data. It has its roots on the observation that model combination, a.k.a. ensemble learning, nearly always improves performance. In this regard, it temporarily removes neurons and their associated synapse connections during training with some probability that could vary from one layer to another, which is equivalent to combining many "thinned" scaled-down models during one training session. Dropout achieves fault resilience because it equalizes the importance of neurons and distributes the neuron activity across the network. Thus, if a neuron becomes dead or it presents timing variations, the impact is inherently tolerated [80]. In contrast, a neuron saturation fault is not compensated because a neuron constantly firing is likely to severely perturb the propagating spike trains [80], [94].

As an example, Fig. 7.2 shows for the N-MNIST SNN the effect of neuron faults in the last 3 layers on the network classification accuracy. Each row corresponds to a layer and each rectangle within a row corresponds to a neuron in this layer.



(a) Dead neuron faults.

(b) Saturated neuron faults.

Figure 7.2: Effect of neuron faults on the classification accuracy of the N-MNIST SNN.

The color of the rectangle shows the accuracy when this particular neuron is faulty based on the color map shown at the bottom of Fig. 7.2. As it can be seen, dead neuron faults are benign except in the last layer. Layers SC1 and SC2 are not shown due to their high neuron count. Similarly, timing variations are proven benign apart from the last layer [80]. The same findings were obtained for the gesture SNN.

As for synapse faults, fault space reduction becomes of utmost importance since the number of synapses can be in the order of several millions. In [93], only the last layer synapse faults were considered arbitrarily. In Chapter 4 we performed an analysis for the N-MNIST SNN to validate this hypothesis. The result is shown in Fig. 4.9. We assumed extreme synapse faults, namely dead synapses and positively saturated weights. As it can be seen, only positively saturated weights appear to be critical, and this holds primarily for the synapses connecting the last two layers SF4-SF5, while few such critical synapse faults are observed for layers SC3-SF4. Synapse faults in previous layers have no impact and are excluded from Fig. 4.9. The reason behind this observation is that positive saturated weights could cause the post-synaptic neuron to always fire, i.e, saturate. In contrast, dead synapses just reduce the firing activity of the post-synaptic neuron, while we know that with dropout dead neuron faults are benign. These findings were corroborated on the gesture SNN as well. Note that such extreme faults are not realistic from a hardware perspective since real-weighted synapses after model training in software are quantized and stored as digital words in an on-die memory. Using the hardware-aware synapse fault model, i.e., bit-flips on quantized weights resulting in weight perturbation, has a smaller impact as shown in Fig. 4.10 for the gesture SNN considering synapses connecting the last two layers SF3-SF4. We observe that only bit flips in the first two MSBs 7 and 6 can be critical, while for bit 5 the baseline accuracy is observed. Bit-flips for bit positions 0-4 are benign and are not shown in Fig. 4.10.

From these fault injection experiments we can conclude about the criticality of fault types occurring at different locations in the network. Critical faults most likely can only be neuron saturation faults at any point in the network, dead

neuron faults in the last two layers, and synapse faults in the last two layers. Still, many of these faults will end up being benign. The rest of the faults are benign with a very high probability and could be excluded from fault simulation.

7.4.2 *Hardware-Level Fault Model*

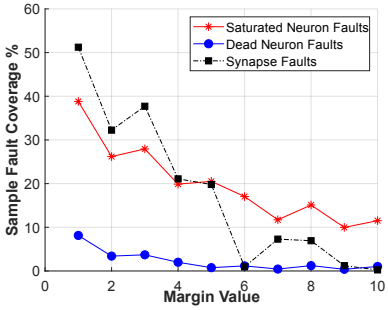
A detailed reliability analysis for the poker card symbols SNN implemented on neuromorphic hardware was presented in Chapter 6. The used fault model is the one described in Section 6.2 and fault injection was performed on the actual hardware presented in Chapter 5. Critical bit-flip faults were located across different network parameters and bit positions. For example, it was shown that bit-flips in the 4 LSBs of synapse weights are benign which can help to significantly reduce the fault space by nearly 50% since synapse weights occupy most of the memory size.

7.5 RESULTS

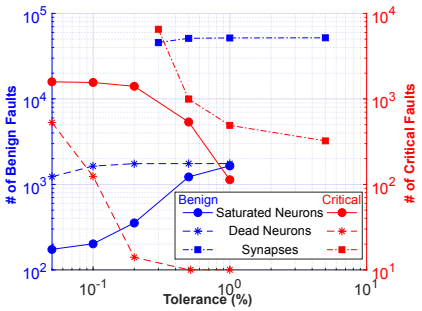
Thanks to the acceleration of fault injection on hardware, i.e., using a GPU for the N-MNIST and gesture SNNs and the FPGA-based SNN hardware accelerator for the poker card symbols SNN, we considered a conservative fault space reduction only for the synaptic faults and only in the N-MNIST and gesture SNNs. In particular, we considered only the synapse faults in the last two layers. The results are grouped per SNN in Figs. 7.3-7.5.

Figs. 7.3a, 7.4a, and 7.5a show the average per-sample fault coverage as a function of the margin value. Samples with identical margin values are grouped. For each sample, the average fault coverage is computed across all faults separately for each fault type, i.e., dead neuron, saturated neuron, and synapse faults for the N-MNIST and gesture SNNs, and single bit-flip and multiple bit-flip faults with BER=10^{-4} for the poker card symbols SNN. Then, fault coverage values are averaged again across all samples within the same group. A clear trend is observed, i.e., samples with low margin, or equivalently with high score, tend to achieve a higher fault coverage. This proves the suitability of the chosen fault-agnostic metric for ranking samples according to their fault detection capability. We also observe that the fault coverage rapidly increases when the margin decreases for small margin values, while for large margin values, fault coverage values are flattened or show a small fluctuation. This means that input samples with relatively large margin values may show small deviation in their fault detection capability, whereas for input samples with small margin values, the deviation can be significant. Note also that a flat fault coverage curve does not necessarily imply that adding more samples with larger margins to the test set is meaningless since they may be achieving the same fault coverage, but at the same time they may be detecting a different set of faults compared to the faults already detected by samples of higher ranking.

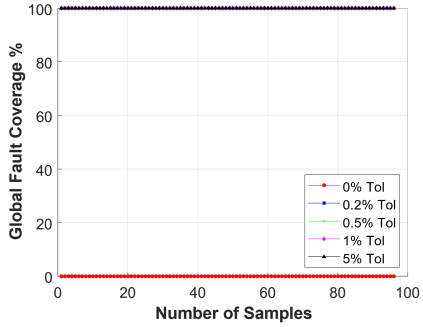
Figs. 7.3b, 7.4b, and 7.5b show for the different fault types the number of benign and critical faults by varying the error tolerance. By increasing the



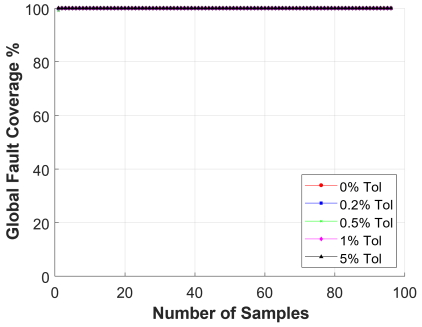
(a) Relationship between the margin of a sample and the percentage of faults it covers.



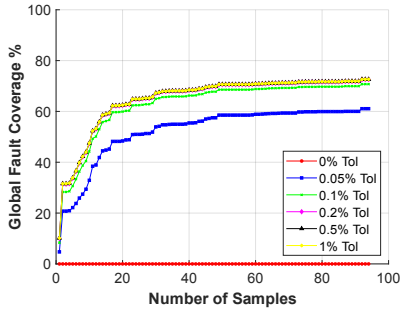
(b) Change in number of critical & benign faults with the tolerance.



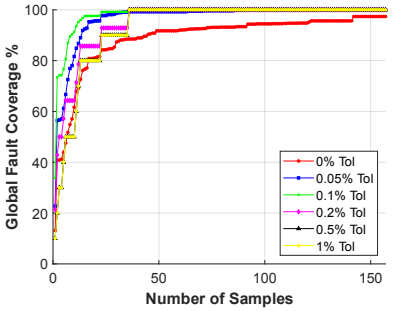
(c) Cumulative fault coverage of benign synaptic faults.



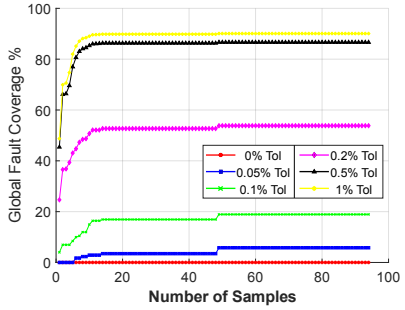
(d) Cumulative fault coverage of critical synaptic faults.



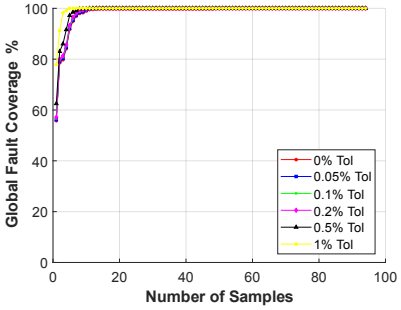
(e) Cumulative fault coverage of benign dead neuron faults.



(f) Cumulative fault coverage of critical dead neuron faults.

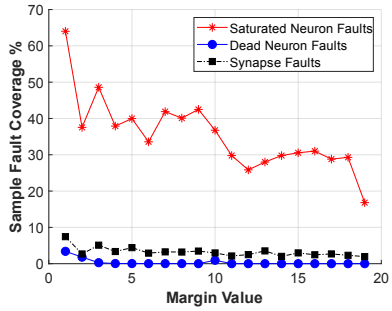


(g) Cumulative fault coverage of benign saturated neuron faults.

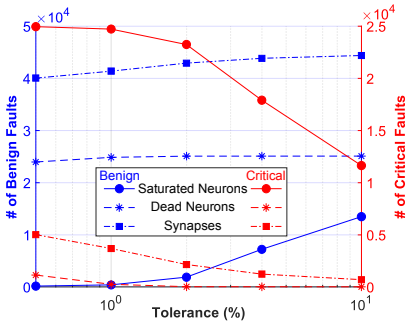


(h) Cumulative fault coverage of critical saturated neuron faults.

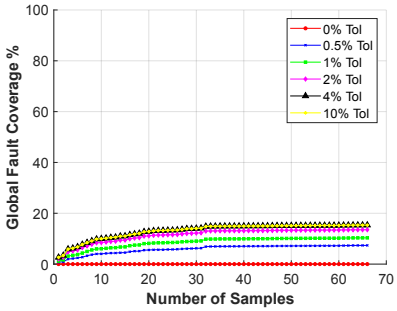
Figure 7.3: N-MNIST SNN.



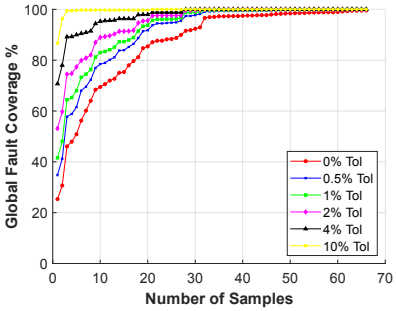
(a) Relationship between the margin of a sample and the percentage of faults it covers.



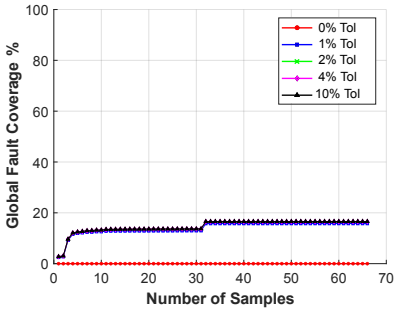
(b) Change in number of critical & benign faults with the tolerance.



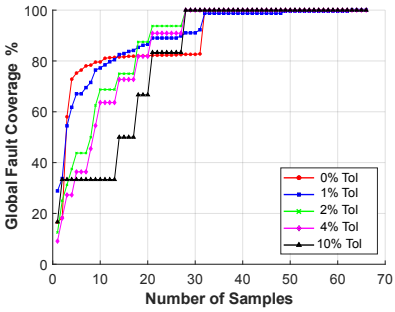
(c) Cumulative fault coverage of benign synaptic faults.



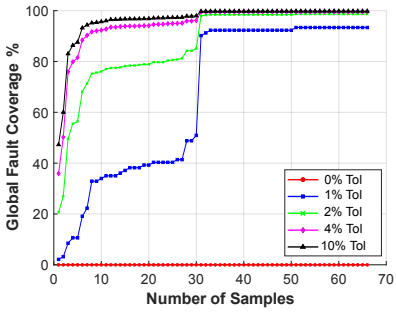
(d) Cumulative fault coverage of critical synaptic faults.



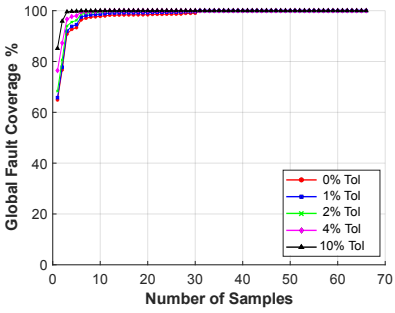
(e) Cumulative fault coverage of benign dead neuron faults.



(f) Cumulative fault coverage of critical dead neuron faults.

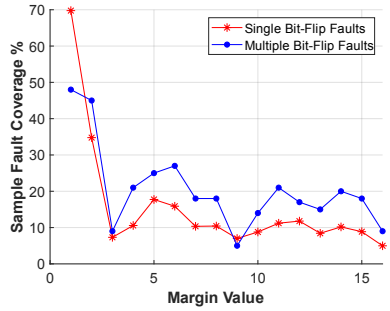


(g) Cumulative fault coverage of benign saturated neuron faults.

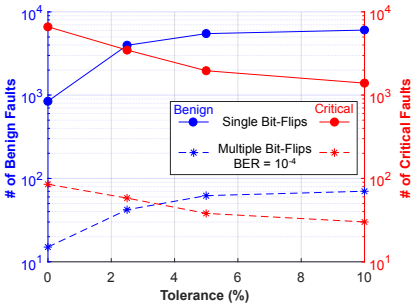


(h) Cumulative fault coverage of critical saturated neuron faults.

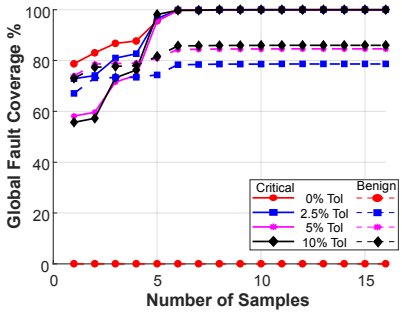
Figure 7.4: Gesture SNN.



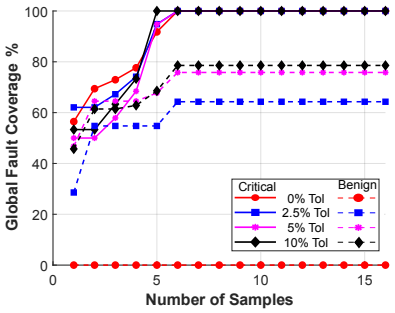
(a) Relationship between the margin of a sample and the percentage of faults it covers.



(b) Change in number of critical & benign faults with the tolerance.



(c) Cumulative fault coverage of single bit-flip faults.



(d) Cumulative fault coverage of multiple bit-flip faults at BER=10^{-4}.

Figure 7.5: Poker card symbols SNN.

tolerance from 0%, some critical faults are labelled as benign, thus the number of benign faults increases and the number of critical faults drops. The number of truly benign faults, i.e., none of the samples is misclassifed when these faults occur, is shown in Figs. 7.3b, 7.4b, and 7.5b for tolerance 0%. Fault type criticality for a given tolerance value can be assessed by examining the percentages of benign and critical faults. For tolerance 0% we observe that for all fault types, faults turn out to be more critical than benign, while for dead neuron faults, as expected, the critical and benign fault populations are more balanced. For all fault types, we observe that there is a tolerance value for which the critical and benign curves cross, which means that there is a certain tolerance value for which benign faults start outnumbering critical faults.

In Figs. 7.3c-7.3h, 7.4c-7.4h, and 7.5c-7.5d we rank the samples in an ascending order according to their scores, and we show the global cumulative fault coverage as we add samples in the test set for the different fault types. Two general conclusions can be drawn here. First, for the critical faults, the global cumulative fault coverage curves quickly reach 100%, while the convergence speed in general increases with the tolerance. This is because higher tolerance means less critical faults and, thereby, smaller test effort to reach 100% fault coverage. The number of samples required to achieve 100% coverage varies from one SNN to another and from one fault type to another. Second, for the benign faults, the global cumulative fault coverage is not expected to reach 100% since many faults are truly benign. For tolerance 0%, the benign fault coverage is by definition zero.

We also observe that the global cumulative fault coverage curves quickly saturate. The number of samples at the saturation point corresponds to the number of samples needed for detecting all benign faults that are not truly benign.

Table 7.1 extracts some representative results from Figs. 7.3c-7.3h, 7.4c-7.4h, and 7.5c-7.5d, summarizing for each SNN and for three representative tolerance values the two most interesting quantities, i.e., the number of samples required to reach 100% fault coverage for critical faults and the number of samples required to detect all benign faults that are not truly benign. For example, for the N-MNIST SNN, we observe that the most highly ranked sample alone can detect all critical synapse faults for any tolerance value. To reach 100% fault coverage for critical dead neuron and saturated neuron faults for 0% tolerance, 150 and 33 samples are required, respectively. The required number of samples drops as the tolerance increases, i.e., for 0.2% tolerance 36 and 33 samples are required and for tolerance 0.5% 36 and 9 samples are required. Also from the cumulative benign fault coverage curves, we observe that, in general, for any tolerance value, the saturation point is reached before the maximum number of 150 samples required for detecting all critical faults at 0% tolerance, i.e., for 0.2% tolerance saturation is observed from 92, 33, and 1 samples onwards for benign dead neuron, saturated neuron, and synapse faults, respectively. For the gesture SNN, the number of samples required to achieve 100% critical fault coverage is 66, 35, and 28, for tolerance values 0%, 2%, and 4%, respectively. For these test sets, maximum coverage for not truly benign faults is achieved, except for tolerance 4% where an additional 31-28=3 samples from the ordered list need to be included. For the poker card symbols SNN, 6 samples suffice to detect all critical and not truly benign faults for any fault type.

Overall, results show that in all cases a compact functional test set is found that is capable of detecting all critical faults for any tolerance value, and as an auxiliary benefit it detects all benign faults that result in even the smallest accuracy drop, i.e. one sample is misclassified. Taking as an example the N-MNIST SNN for which 70,000 samples are available from the training and testing sets, the functional test set required to achieve 100% critical fault tolerance for all fault types is compacted to $150/(7 \cdot 10^4) = 0.214\%$ and $36/(7 \cdot 10^4) = 0.051\%$ of the combined size of the training and testing sets for tolerance 0% and tolerance $> 0\%$, respectively, where the maximum number of required samples across the three fault types is used.

7.6 DISCUSSION

7.6.1 *Generality*

The proposed functional test generation method is generic, treating the SNN architecture as a black-box. In this regard, the method is virtually applicable to any SNN hardware accelerator and neuromorphic computing hardware platform, including for example the SpiNNaker [34], TrueNorth [30], Loihi [31],

Table 7.1: Number of samples needed to reach the maximum fault coverage for different fault types at different tolerance values.

% Tolerance	Neuron Faults				Synapse Faults	
	Dead		Saturated		Benign	Critical
	Benign	Critical	Benign	Critical		
0	-	150	-	33	-	1
0.2	92	36	33	33	1	1
0.5	92	36	17	9	1	1

(a) NMNIST SNN.

% Tolerance	Neuron Faults				Synapse Faults	
	Dead		Saturated		Benign	Critical
	Benign	Critical	Benign	Critical		
0	-	32	-	25	-	66
2	13	28	31	19	35	35
4	13	28	31	15	32	28

(b) Gesture SNN.

% Tolearance	Single Bit-Flips		Multiple Bit-Flips	
	Benign	Critical	Benign	Critical
0	-	6	-	6
2.5	6	6	6	6
5	6	6	6	6

(c) Poker card symbols SNN.

BrainScaleS [138], Neurogrid [139], FPGA-based implementations [66], and application-specific small-scale chips [32], [77], [140]–[143].

7.6.2 *Test Generation Effort*

The proposed test generation algorithm can be decomposed into two steps:

1. The ranking of available input samples according to their fault coverage ability.
2. The fault coverage assessment of a test set composed of highly ranked input samples.

Step (1) is agnostic to the fault model, using only inference data, i.e., number of spikes per output class for each sample in the training and testing sets from a pre-trained SNN model. This information is already available from the training phase, thus step (1) can be completed very fast independently of the SNN and dataset sizes.

Step (2) requires fault simulation, thus the effort is proportional to the SNN size. The number of fault locations increases with the increase in the SNN size and, thereby, the number of functional tests required to achieve the desired fault coverage is likely to increase as well. For large SNNs, fault simulation effort may rapidly explode as explained in Section 7.4. For this reason, the fault space needs to be conservatively reduced by considering the impactful fault locations, i.e.,

neurons in last layer, saturation neuron faults, and synaptic connections in the last layers. Note, however, that the highest ranked input sample will detect a very high number of faults and in each step, by adding the next highest ranked input sample, the cumulative global fault coverage rises quickly. This behavior was observed in the results in Section 7.5. In each step, we exclude from the simulation the faults already detected by previous input samples in the ranked order. This way, fault simulation is not exhaustively repeated for every input sample and it can become tractable even for large size SNNs. Note also that step (2) is an one-time effort and can be significant for traditional computing chips as well. Once the functional test set is generated, it will be used as a fixed test program in high-volume production. As a final observation, the effort in step (2) is not related to the dataset size. A large dataset size means more input samples, thus a larger pool of functional tests to choose from. In fact, a larger dataset size may lead to a more compact functional test set.

7.6.3 *Other Metrics for Grading Functional Tests*

The metric used for grading functional tests is based on spike-count difference for the first two top classes, i.e., the only relevant aspect of a spike train is its total number of spikes. In principle, any metric that quantifies the distance between two spike trains can be used. Several such metrics have been proposed in the past generalizing the distance measure to include the temporal structure in the spike trains [144]–[148]. For example, in the Victor-Purpura metric [144], the distance is defined as the minimum cost required to transform one spike train into the other via a path of elementary steps. The cost equals the sum of the costs assigned to each of the allowed elementary steps. In our context, the cost is inversely proportional to the score of the functional test. There are two kinds of elementary steps: (a) adding or deleting a spike which is assigned a cost of 1; and (b) shifting in time the occurrence of a single spike by an amount Δt which is assigned a cost of $q * |\Delta t|$, where q is a parameter that expresses the relative sensitivity of the metric to precise timing of spikes. The two extreme cases are $q = 0$ and $q = \infty$ (or very large q). By setting $q = 0$, we recover the spike-count difference metric used in this work. This is because for two spike trains with number of spikes n_1 and $n_2 > n_1$, we can align the first n_1 spikes with zero cost, then transform the second spike train to the first by deleting its last $n_2 - n_1$ spikes with cost $n_2 - n_1$. In the limit $q = \infty$, it is less costly to add or delete spikes than to shift a spike and so the distance between two spike trains becomes the total number of non-synchronous spikes.

7.7 RELATED WORK ON FUNCTIONAL TEST GENERATION FOR AI HARDWARE ACCELERATORS AND COMPARISON

The DeepXplore [135] and DeepTest [136] algorithms generate error-inducing corner test cases, which are then used for re-training the ANN aiming at improving classification accuracy. The criterion is maximizing neuron coverage, such

that ideally for each neuron there is a test that makes it highly active. In the end, the generated synthetic samples represent real-world samples. For example, [136] starts with a subset of the testing set, called "seeds", then performs realistic transformations of seed images, such as changing brightness, changing contrast, rotation, blurring, fog effect, rain effect, etc. These test generation algorithms target improving classification performance and not hardware-level fault detection. However, it would be interesting to investigate whether synthetic samples generated in this fashion can also achieve high fault coverage.

In [125], a methodology to derive a diminutive set of functional test patterns for systolic array-based ANN accelerators is proposed. Two functional test pattern generation algorithms are proposed, namely an ANN model-agnostic algorithm and an ANN model-aided confidence-based algorithm. The ANN model-agnostic algorithm rates samples in the testing set based on their similarity to other samples belonging to different output classes. The similarity metric used is average pixel intensity. The ANN model-aided confidence-based algorithm searches for samples in the testing set that have been predicted correctly but with least confidence score. The proposed method in our work concerns SNNs and chooses samples that are prone to misclassification based on a different SNN-specific similarity metric defined based on the output spike trains.

In [126], it is proposed to rank and select a small subset of samples from the training set making the hypothesis that samples that require more neural network parameter tuning during training than others will be more sensitive to changes in neural network parameters due to faults. Tuning effort per sample is approximated with the change in the loss function in each training step. In the case where the model is pre-trained, a black-box approach is proposed to rank samples based on the difference in the loss function of a randomly initialized neural network instance and the pre-trained neural network. The methodology is demonstrated for memristive crossbar-based ANN hardware accelerators.

A functional test generation algorithm for SNNs is proposed in [93]. The algorithm produces a test set containing a mixture of available samples and adversarial examples. An adversarial example is generated by perturbing available samples by adding a minimum amount of noise such that the predictions of the nominal and faulty SNNs are differentiated. The algorithm starts by injecting a fault and examining if any of the available samples detects it. If not, up to D adversarial examples are generated, where D is a user-defined variable, aiming at finding one that detects the fault. If any available sample or adversarial example is found that detects the fault, then this successful test is tried out on all faults. It is placed in the kept list and the detected faults are dropped from the list. The algorithm reiterates targeting the next undetected fault. Since the number of synapse faults is too high, to solve the scalability issue only the last layer synapse faults are considered. In this work, we performed an experiment in Section 7.4 to justify fault space reduction. Furthermore, the algorithm in [93] follows a greedy approach where tests are repeatedly evaluated on the undetected faults, combining test generation with fault coverage estimation. Thus, the number of inferences required, i.e., the test generation time, is a summation over tests and

faults. On the contrary, the proposed algorithm in this paper dissociates test generation with fault simulation. Tests are first ranked by performing inference on the fault-free network, then fault coverage is computed only for the highly ranked tests. Thanks to the fault model agnostic ranking, the proposed method reduces dramatically the number of inferences. Finally, in the algorithm in [93], adversarial examples are useful for detecting benign faults since all critical faults are covered by original samples before entering the adversarial example generation loop. Adversarial example generation can be also seemingly added to the proposed algorithm in our work as a second step to boost benign fault detection.

8 NEURON FAULT TOLERANCE STRATEGY

By leveraging the observations made from the fault injection experiments in Chapter 4, in this chapter, we propose a neuron fault tolerance strategy for SNNs, optimized for low area and power overhead [80]. The fault tolerance strategy is composed of a preparatory passive part and a second-step active part. First, the network is prepared to passively eliminate some of the faults proactively. The rest of the faults, that cannot be confronted in the first step, are detected by on-die monitors, which are embedded within the network's components in a discreet way in terms of occupied area and power consumption and offer both an on-line and an off-line BIST mechanism. After a fault has been detected, the error mitigation mechanisms are activated and the performance of the network is recovered.

8.1 RELATED WORK ON FAULT TOLERANCE

The fault injection experiments conducted in Chapter 4 and the related work mentioned in sec 4.1 demonstrate that equipping AI hardware accelerators with a preventative fault tolerance strategy is a crucial requirement for mitigating risks in AI systems. The goal is to identify critical fault types and fault locations in the SNN architecture and, subsequently, take action to render the design fault-tolerant. Fault tolerance techniques can be proactive or reactive, and typically each can address a subset of fault types and locations.

Proactive techniques aim at making the SNN tolerate by design a number or certain types of faults. One approach is to perform fault-aware training where faults are injected during training epochs, for example as synapse weight perturbations, aiming at maximizing simultaneously accuracy and fault tolerance [92], [122], [149]. An advantage of this approach is that it allows margin for voltage reduction in the memory, thereby helping to reduce the energy consumption [39]. A second approach is to derive the memory fault map via testing, then prioritize placing of MSBs of network parameters on non-faulty memory cells [122]. A third approach is to adopt training algorithms that naturally offer fault tolerance [92].

Reactive fault tolerance techniques are implemented at hardware-level and, in general, are composed of two mechanisms, namely a self-test mechanism for fault detection and a fault-mitigation strategy following a fault occurrence. One approach is to focus on the most lethal faults, which include the saturated neuron fault and large synapse weight increases, shown in Chapter 4. This approach is the one used in Section 8.3 and in a similar, more recent work [94], where a neuron saturation detector is attached to each neuron and silences the neuron if it exhibits saturation behavior. For large synapse weight increases, it is proposed to replace the weight with a predefined value, for example a zero value [94]. These fault-mitigation approaches essentially translate a critical fault into a benign fault. Finally, another approach is to perform on-line re-learning by disabling components for which re-learning cannot make up for the damage [121].

In this context, standard fault-tolerance techniques for regular VLSI circuits can be employed, such as TMR and Error Correction Codes (ECCs) for memories. However, efficiency can be largely improved by exploiting the architectural particularities of AI hardware accelerators and targeting only those fault scenarios that have a measurable effect on performance [40]. One approach is to perform re-training to learn around faults, but this requires access to the training set and extra resources on-chip, thus it is impractical at chip-level.

The work in [92] studies the resilience of feed-forward SNNs to dead synapse faults when trained with different algorithms. Synapses are selected to be faulty at random with different fault rates. Three of scikit-learn's toy datasets are used: iris, wine, and breast cancer. Results show that resilience characteristics depend largely on the training algorithm and in all cases the accuracy drops rapidly with increasing fault rates. It is shown how to modify an evolutionary optimization-

based training algorithm so as to improve fault tolerance. In particular, the fitness function is re-designed to become a weighted sum of the baseline accuracy and the average accuracy obtained on versions of the SNN with dead synapse faults. The resilience is improved but the baseline accuracy is not recovered.

In [122], fault injection is performed in a Python-based SNN model. The fault model is bit-flips in the memories storing the weights of the network. A uniform random distribution with different rates is considered. Fault-tolerance schemes to mitigate memory failures are also proposed, namely Fault-Aware Mapping (FAM) and Fault-Aware Training and Mapping (FATM). First, the memory fault map, i.e., the location of the faulty memory cells, is derived using testing. FAM consists in identifying the memory segment with the highest number of subsequent non-faulty cells and prioritize placing the MSBs of the weight in this segment, which is done using a circular shift. FATM follows FAM and consists in performing re-training while considering bit-flips for different rates during training epochs. In this way, the network adapts its accuracy to different bit-flip probabilities.

In [121], re-learning is proposed as a reactive fault tolerance strategy for a high-level biologically-inspired model of the cortical structure of the brain, which is deployed on a GPU. The fault model considers neuron stuck-at faults, i.e., neurons that do not fire when they should (stuck-at-0) or they fire when they should not (stuck-at-1). Concerning the stuck-at-0 neurons, the re-learning of the network can easily accommodate their absence with the neighbor neurons covering up for them. Stuck-at-1 neurons, on the other hand, can severely degrade the performance, thus they are disabled, i.e., converted to stuck-at-0 neurons, and the network re-learns. In order to detect a stuck-at-1 neuron, a voting scheme is used after interrupting the operation and recomputing the response of the winning layer of neurons on two neighboring layers.

8.2 PASSIVE FAULT TOLERANCE USING DROPOUT

As a first step, we aimed at implementing a passive fault tolerance such that the SNN is by construction capable of withstanding some faults without any area and power overheads. To this end, we discovered that training the SNN with dropout [137] can nullify the effect of dead neuron faults and neuron timing variations in all hidden layers. In this way, active fault tolerance, which implies area and power overheads, gets simplified since it will need to focus solely on saturation neuron faults in the hidden layers and on all fault types only for the output layer.

The dropout training technique was originally proposed to prevent overfitting and reduce the generalization error on unseen data. The idea is to temporarily remove neurons during training with some probability p, along with their incoming and outgoing connections. At test time, the final outgoing synapse weights of a neuron are multiplied by p. For a network with n neurons, there are 2^n "thinned" scaled-down networks, and training with dropout combines exponentially many thinned network models. The motivation is that model

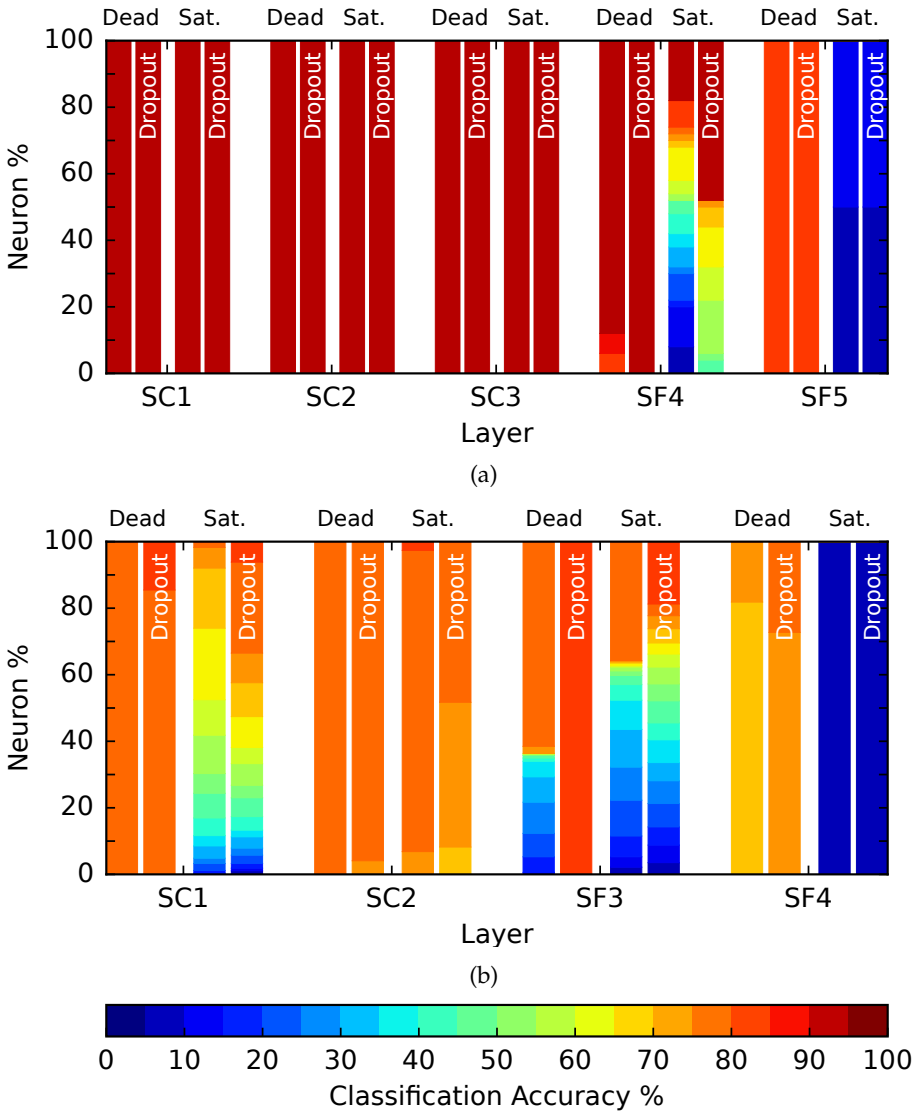


Figure 8.1: Effect of neuron faults on classification accuracy with and without dropout: (a) N-MNIST SNN; (b) gesture SNN.

combination nearly always improves performance, and dropout achieves this efficiently in one training session.

For the N-MNIST SNN we used p=10% in the input and SC1 layers, 20% in layers SC2 and SC3, and 50% in layer SF4. Training with dropout resulted in a slight improvement in the classification accuracy from 98.08% to 98.31%. For the gesture SNN we used p=50% only in layer SF3. In this case, dropout increased significantly the classification accuracy from 82.2% to 87.88%.

The beneficial effect of dropout on passively nullifying the effect of dead neuron faults is shown for each layer in Figs. 8.1a and 8.1b for the N-MNIST and gesture SNNs, respectively. For example, this is made largely evident by comparing the classification accuracy in the presence of dead faults for layer SF4 of the N-MNIST SNN and layer SF3 for the gesture SNN. The beneficial effect on nullifying the effect of neuron timing variations for the last hidden layer even for extreme variations of τ_s is shown in Figs. 8.2b and 8.3b for the N-MNIST

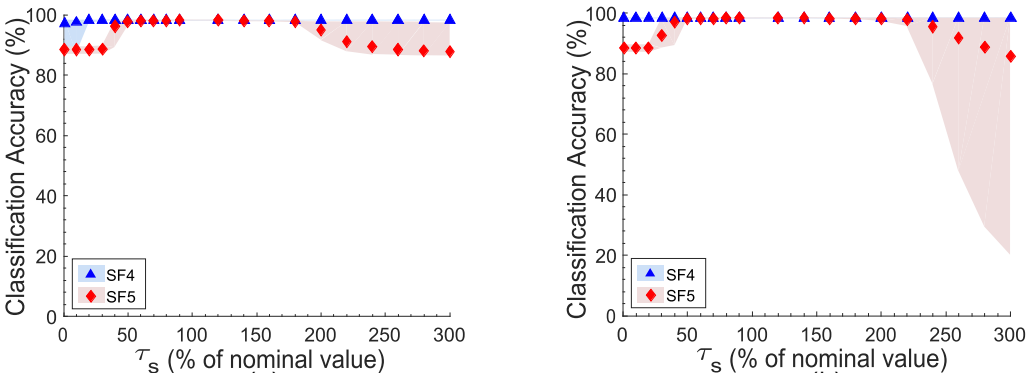


Figure 8.2: Effect of neuron timing variations for the N-MNIST SNN: (a) without dropout; (b) with dropout.

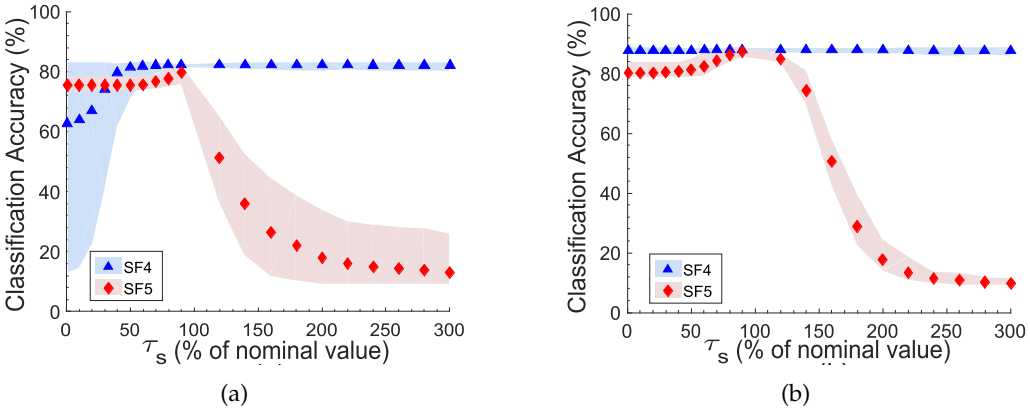


Figure 8.3: Effect of neuron timing variations for the gesture SNN: (a) without dropout; (b) with dropout.

and gesture SNNs, respectively. As can be seen, variations in τ_s from 1% to 300% have now no effect.

The reason behind this result is that dropout essentially equalizes the importance of neurons across the network, resulting in more uniform and sparse spiking activity across the network. Therefore, if a neuron in a hidden layer becomes dead or shows excessive timing variations, this turns out to have no effect on the overall classification accuracy. On the contrary, dropout may magnify the effect of saturation neuron faults, i.e. layer SF3 of the gesture SNN. Finally, we observe that dropout does not compensate for faults in the output layer since in this layer there is one neuron per class and any fault will either overshadow this class or cause it to dominate the other classes, while in the layer SC1 of the same network and in the SF4 of the N-MNIST network, the saturation faults affect the performance less after dropout.

Following a training that employs dropout, the network is left with the vulnerability of the output-layer neurons and the risk posed on the overall performance by a saturated neuron in a hidden layer. This means that a cost-effective strategy to implement fault-tolerance into a neural network would be to solely focus on these two cases.

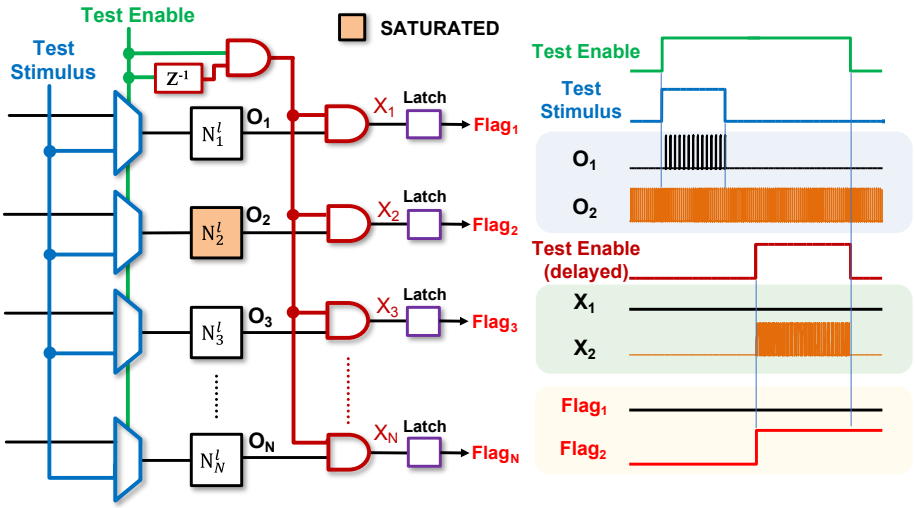


Figure 8.4: Offline self-test scheme.

8.3 ACTIVE FAULT TOLERANCE IN HIDDEN LAYERS

As explained in Section 8.2, active fault tolerance in hidden layers needs only to address neuron saturation. We propose two self-test schemes for neuron saturation detection, namely an offline scheme that can run during idle times of operation and an online scheme that can run concurrently with the operation. Regarding the fault recovery mechanism, we propose the "fault hopping" concept to simplify the hardware implementation, and we propose two recovery mechanisms at neuron-level and system-level.

8.3.1 *Offline Self-Test*

The offline self-test scheme is illustrated in Fig. 8.4. Neuron saturation is declared based on the neuron's activity in the absence of an input. A multiplexer is assigned to every neuron to switch between self-test and normal operation modes. During normal operation, neurons are receiving inputs from the previous layer through synapses, processing them and propagating them to the next layer. When the test enable signal is on, a short internally-generated current pulse is applied to all the neurons simultaneously as a test stimulus, thus test time for the complete network is very short. The neuron outputs are paired with a delayed version of the test enable signal through an AND gate. This is to ensure that any activity detected is uncorrelated with the input of the neuron and is indeed a result of saturation. The output of an AND gate going high indicates neuron saturation. This is captured by the latch which raises an error flag signal. A simulation is shown in Fig. 8.4 using the I&F neuron presented in Fig. 3.1. This self-test scheme adds a multiplexer, an AND gate, and a latch per neuron, thus the area overhead of the test circuitry is relatively small compared to a single neuron. It can detect aging-induced errors, possibly with some latency.

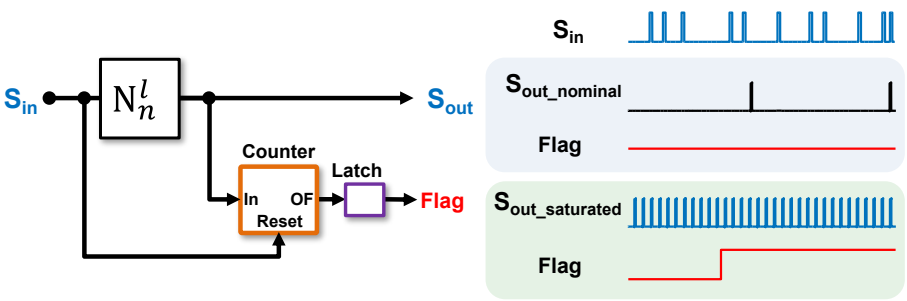


Figure 8.5: Online self-test scheme.

8.3.2 *Online Self-Test*

The online self-test scheme is illustrated in Fig. 8.5. It is applied on a per-neuron basis and takes advantage of the temporal dependency between the input and output of a spiking neuron. In particular, we count the number of spikes a neuron produces after every single input spike using a counter whose reset port is connected to the input of the neuron. In fault-free operation, the neuron needs to integrate multiple input spikes before it can produce a spike of its own, hence the counter is always reset, and the flag signal stays at zero. On the other hand, a saturated neuron will produce spikes with higher frequency than usual, causing the counter to overflow before an incoming spike resets it again. A latch is set when overflow happens and an error flag is raised and maintained. Based on our simulations, 2^3 uncorrelated spikes clearly indicate saturation, thus it suffices to use a 3-bit counter. Fig. 8.5 shows a simulation using the I&F neuron of Fig. 3.1. This online self-test scheme entails an area overhead comprised of a counter and a latch per neuron. All neurons are monitored individually and neuron saturation is detected in real-time.

8.3.3 *Recovery Mechanism*

The recovery mechanism is activated once neuron saturation is detected. We propose the concept of “fault hopping" where the critical saturation neuron fault is artificially translated to a dead neuron fault. The network is repaired since a dead neuron fault has no effect after dropout. This approach leads to an elegant hardware implementation and saves significant costs as opposed to the standard approach, which is to duplicate or triplicate neurons, or provision the SNN with spare neurons that are kept “fresh" and switch the connections of a detected saturated neuron to a spare neuron [40]. We propose two recovery mechanisms based on the concept of “fault hopping", at neuron-level and at system-level.

Neuron-level recovery is implemented by switching-off the saturated neuron. For example, for the I&F neuron in Fig. 3.1, this can be achieved by connecting a single extra transistor M_C in the tail part of the comparator inside the neuron as shown with red in Fig. 8.6. This transistor is controlled by the neuron error flag signal. When the neuron gets saturated, the biasing connection of the comparator is suddenly ceased, which deactivates the neuron tying its output to zero. The

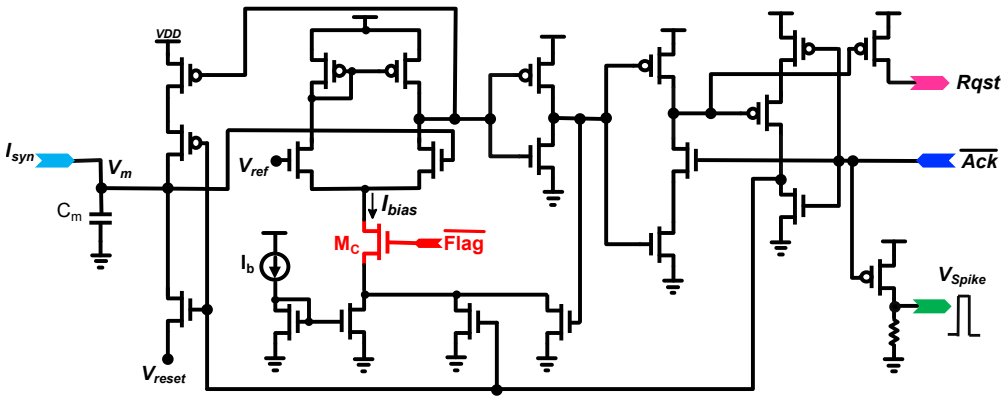


Figure 8.6: I&F neuron design showing the recovery operation at neuron-level.

area overhead is only one transistor per neuron and an auxiliary advantage is that faulty neurons get deactivated; thus, they stop consuming power.

System-level recovery is implemented by setting the outgoing synapse weights of the saturated neuron to zero. In this way, the saturated spike train gets trapped and does not propagate to neurons in the next layer. Typically, the communication between neurons in SNNs is performed by a controller that implements the AER protocol [24], like in the neuromorphic hardware experimentation platform presented in Chapter 5. AER controllers perform multiplexing/demultiplexing of spikes generated from or delivered to all neurons in a layer onto a single communication channel. Rather than delivering the actual spike, the controller encodes the address of the neuron that spiked and translates it into the addresses of the destination neurons, and then the weights corresponding to every synaptic connection are loaded accordingly. By leveraging this operation, the system-level recovery approach is based on equipping the controller with the ability to recognize the neuron error flag and update the outgoing synaptic weights to zero. This system-level recovery mechanism has a minimal area overhead since it is reused across all neurons. However, compared to the neuron-level recovery mechanism, saturated neurons stay on continuing consuming power.

8.4 ACTIVE FAULT TOLERANCE IN THE OUTPUT LAYER

As for the most critical output layer, we propose to directly use TMR for a seamless recovery solution from any single fault type. In particular, a group of three identical neurons vote for the decision of a certain class, as shown in Fig. 8.7. The voter is a simple 4-gate structure that propagates the output upon which the majority of neurons agree. This means that a faulty neuron in the group, be it dead, saturated or showing excessive timing variations, is outvoted and bypassed. Performing TMR only in the last layer will result in negligible increase in the area and power overhead and a reasonable overhead to pay to ensure strong fault tolerance. The reason is that the number of neurons in the output layer is typically small compared to the size of the whole network. For example, in the N-MNIST SNN, the output layer neurons account for about 0.57% of the neurons in the whole network. This percentage gets even less for the more

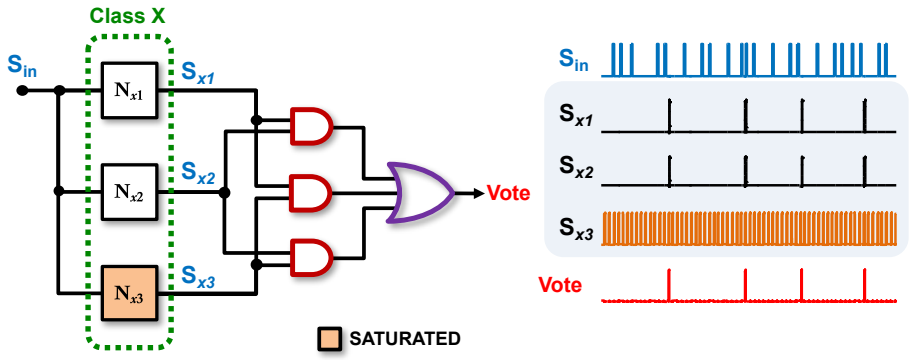


Figure 8.7: TMR at the output layer.

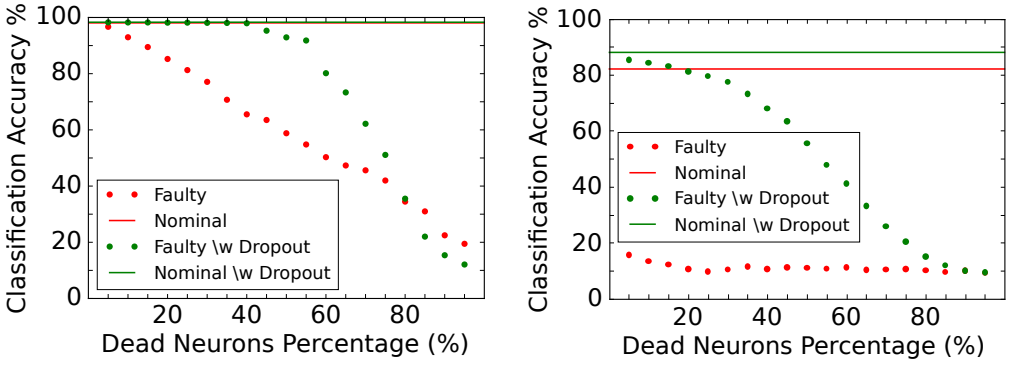


Figure 8.8: Fault tolerance for multiple fault scenarios: (a) N-MNIST SNN; (b) gesture SNN.

complicated gesture SNN, where the output layer represents around 0.04% of the total number of neurons.

8.5 MULTIPLE FAULT SCENARIO

So far, we have discussed fault tolerance considering a single fault assumption. Moreover, our experiments have shown that neuron timing variations start having an effect when the neuron approaches a dead or a saturated one, and our proposed fault tolerance strategy suggests turning a saturated neuron into a dead one. Hence, all faults eventually fold back to a dead neuron fault, arising the question of what percentage of dead neuron faults can the network withstand. Figs. 8.8a and 8.8b show the classification accuracy as a function of the percentage of dead neurons considering the last hidden layer, which is the most critical amongst all hidden layers, for the N-MNIST and gesture SNNs, respectively. Fig. 8.8 shows the baseline nominal classification accuracy with and without dropout and how the classification accuracy drops with the increase of dead neuron rate. As the results show, the SNNs employing dropout can withstand larger rates of dead neurons. More specifically, the N-MNIST SNN does not lose any classification accuracy with a dead neuron rate of up to 40%. As for the gesture SNN, the classification accuracy drops faster, but it is still able

to perform with over 80% classification accuracy at a dead neuron rate of 20%, which corresponds to 102 neurons.

9 ON-LINE TESTING OF NEUROMORPHIC HARDWARE

By exploiting the results of the reliability analysis presented in Chapter 6, we propose an on-line testing methodology for neuromorphic hardware supporting SNN functionality. Testing aims at detecting in real-time abnormal operation due to hardware-level faults, as well as screening of outlier or corner inputs that are prone to misprediction. Testing is enabled by two on-chip classifiers that prognosticate based on a low-dimensional set of features extracted with spike counting, whether the network will make a correct prediction. The system of classifiers is capable of evaluating the confidence of the decision, and when the confidence is judged low a reply operation helps to resolve the ambiguity.

This chapter demonstrates the above testing methodology by fully embedding it in the FPGA-based neuromorphic hardware platform described in Chapter 5. It operates in the background being totally non-intrusive to the network operation, while offering a zero-latency test decision for the vast majority of inferences [150].

9.1 PROPOSED TESTING APPROACH

Summarizing the state-of-the-art we make the following observations:

1. Proactive fault tolerance methods [80], [92], [122], [149] cannot compensate the effect of all faults, thus a number of faults remain critical and a dedicated test procedure is desired.

2. On-line concurrent error detection is implemented at the component-level, i.e., checking the status of synapses and neurons individually [80], [85], [94]. This approach may result in large overhead.

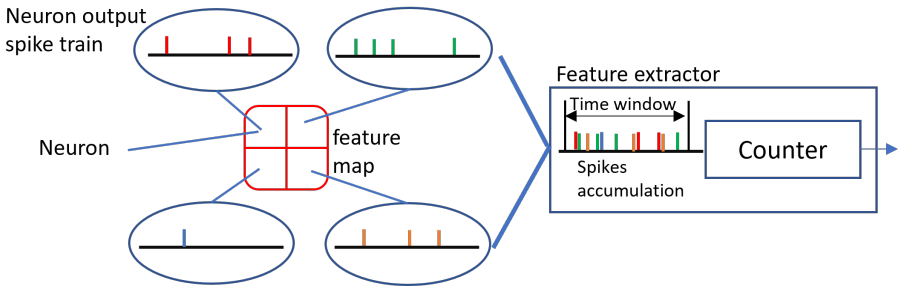
3. Functional testing based on selected fault-sensitizing inputs [93], [112] is primarily a post-manufacturing testing approach but can also be executed on-line periodically in idle times by storing the functional tests in an on chip memory. Besides the memory overhead, the assumption is that the memory remains fault-free. Moreover, this approach does not guarantee high safety standards as it misses transient errors and detects permanent errors with latency.

4. Testing approaches are demonstrated for hardware-level fault detection only, while their utilisation for outlier and corner input detection is not studied so far.

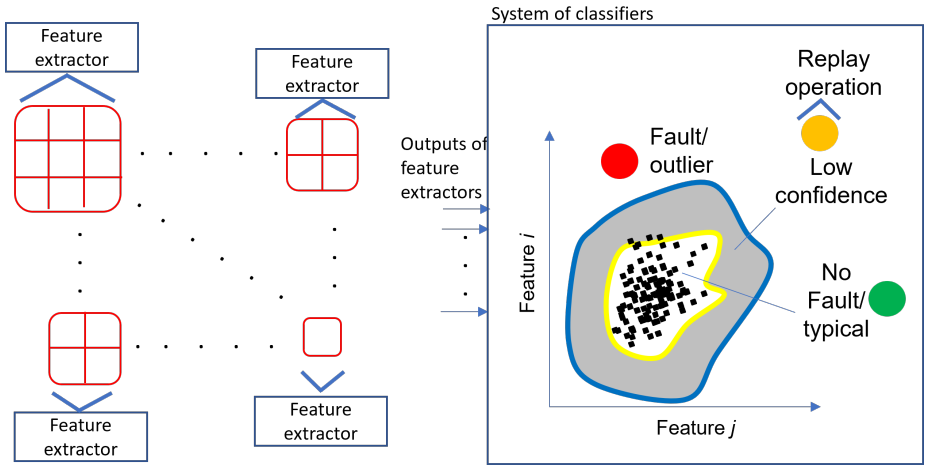
The primary objective of the proposed testing approach is to detect in real-time any abnormality in the SNN operation, either it is due to a fault occurring or due to an outlier or corner input.

Testing, in general, can be viewed as checking a set of symptoms that point to abnormal operation. We postulate that defining symptoms at the output of neurons is a good strategy since information flows in the form of spike trains and for the SNN prediction to be affected the output spike train of at least one neuron in the network should be appreciably altered.

However, checking the output of every single neuron results in test resources with large overhead. To this end, we propose defining symptoms at a higher-level of hierarchy in the network, specifically at the output of each feature map. By construction, according to the AER protocol, a feature map outputs a flow of spiking events $e(t, d)$, where t is the time of the event and d is the sender or the recipient neuron coordinates. Illustrated in Fig. 9.1a, we propose to project the spike events of neurons of the feature map in time, consider a pre-defined time window that is dependent on the duration of the input, and define a *test parameter* at the feature map-level equal to the count of accumulated spike events during the time window. The premise is that an abnormal operation will be manifested in the cumulative spiking activity at the output of at least one feature map both in terms of the number of produced spikes and the spike frequency, causing some test parameters to drift away from their expected values. This drift is a *symptom* of abnormal operation. In this way, we drastically reduce the test parameter dimensionality from the neuron size to the feature map size. In fact, it may not be necessary to consider all feature maps since an abnormal spiking



(a) Feature map extractor.



(b) Network-level test.

Figure 9.1: Principle of operation.

activity at the output of a feature map will propagate and spread through the network, thus it may be detectable at the output of feature maps in the next network hierarchy levels.

Next step is making a test decision based on the test parameters. We postulate that checking their combination across the network can be a better test criterion as opposed to checking them individually. This can be done by training a single one-class classifier to map the test parameters to an one-bit test decision addressing the complete network. In machine learning terminology, the test parameters serve as the input features of the classifier, not to be confused with the feature extraction performed by each feature map in the SNN. The classifier is trained on the fault-free network using the available training input samples. Each input is presented to the network and test parameters are collected at the outputs of the feature maps. This is an one-off effort that is already spent during training. The classifier will learn the area in the test parameter space that corresponds to normal operation, enclosing it with a classification boundary, as shown for example with the yellow classifier in Fig. 9.1b. In abnormal operation, the combination of test parameters will drift outside the classification boundary and the classifier will flag an error detection.

The performance of a classifier is assessed based on two metrics, namely false negatives or *test escapes*, i.e., abnormal operation goes undetected, and false positives or *overkill*, i.e., flagging an error when there is actually none. However, a single classifier is likely incapable of perfectly distinguishing normal from abnormal operation and is bounded to making errors. Using a single classifier, the classifier establishing the optimal trade-off between test escapes and overkill would have been decided based on test economics.

To this end, we adopt a two-tier test approach originally proposed for analog circuits [151]. We propose to avoid using a single classifier making a deterministic decision and instead use a system of two classifiers, as shown in Fig. 9.1b. The yellow classifier is designed to be *strict*, i.e., in its inner area it encloses only feature patterns corresponding to normal operation. The blue classifier is designed to be *lenient*, i.e., feature patterns falling in its outer area for sure correspond to abnormal operation. If the decision of the two classifiers agrees, i.e., the footprint of the feature pattern lies into the inner area of the yellow classifier or into the outer area of the blue classifier, in other words it lies outside the grey zone between the two boundaries, then this decision is deterministic and can be trusted. In contrast, if the footprint lies into the grey zone, then the test decision has low confidence. Essentially, the grey zone serves as a guard-band.

To deal with low confidence decisions, we need an extra fast test to make a final decision with incontestable accuracy. For this purpose, we investigate the different scenarios to understand how the system of the two classifiers responds in each case. The input of the SNN can be typical or can be an outlier or corner input that is foreign to the bulk of the training set and, thereby, is prone misprediction. On the other hand, the SNN can be fault-free or contain a fault. The case of fault occurrence or an outlier or corner input will be flagged by the yellow classifier by construction. The problem lies in the fact that the yellow classifier can also inadvertently flag an error for a typical input and a fault-free SNN.

We observed experimentally that this latter scenario happens due to the stochasticity of the SNN. More specifically, a neuromorphic design is by nature asynchronous, meaning that a neuron might receive a spike event at its input or fire one at its output anytime. On the other hand, the controlling processor operates synchronously based on a clock. The synchronization between the two could potentially create some micro-delays in their communication, which propagate during the inference of the SNN. Given that the decision-making in a SNN is based on the temporal characteristics of the spike trains, these delays result in a variance of the triggering time of neurons and, thereby, in a stochasticity at their output spike trains.

To understand the effect of stochasticity, using our case study described in Chapter 6, we repeated the SNN inference multiple times for each sample of the training set. We observed that the scenario where the yellow classifier flags an error for a typical input and fault-free SNN occurs for a handful of repetitions for

a few samples. The footprint of the feature pattern of these samples lies closer to the boundary of the yellow classifier, thus they can be marginally misclassified.

Based on this observation, to make a final decision when the system has low confidence, the strategy that we propose is to perform a replay operation presenting the input sample to SNN a number of times. If a repetition is found where the yellow classifier predicts normal operation, then the SNN prediction can be trusted to be correct. For normal operation, which is the prevalent scenario, typically one reply operation will suffice.

Based on this observation, to make a final decision when the system has low confidence, the strategy that we propose is to perform one replay operation presenting the same input sample to the SNN a second time. If now the yellow classifier predicts normal operation, then the SNN prediction can be trusted to be correct. Otherwise, the system flags an error.

9.2 CASE STUDY

As case study we use the convolutional SNN presented in Chapter 6 designed for recognizing the symbol on poker cards.

9.2.1 *Classifiers*

Each classifier is implemented with a Support Vector Machine (SVM) using a Radial Basis Function (RBF) kernel. The two hyper-parameters are ν, which controls the trade-off between overfitting and generalization of the SVM in one-class learning, and γ, which is the coefficient of the RBF kernel [152]. A small value of ν leads to fewer support vectors and, therefore, a smoother decision boundary, while a large value leads to more support vectors and, therefore, a curvy decision boundary [152]. We use the cross-language LIBSVM library [153]. The two SVM models are trained in MATLAB and then they are loaded by the C application running on the processor.

9.2.2 *Dataset Categorization*

To account for the SNN stochasticity, for a given input sample, the SNN inference is repeated multiple times and the samples are categorized as follows:

- *Group 1*: Samples whose class is consistently correctly predicted.
- *Group 2*: Samples whose class is consistently erroneously predicted.
- *Group 3*: Samples that are ambiguously predicted during the multiple repetitions due to the SNN stochasticity.

The two SVMs are trained using samples in Group 1 increasing also the training set size for the SVMs. As we care about the impact of faults when the SNN correctly predicts an input sample, the fault detection capability of the

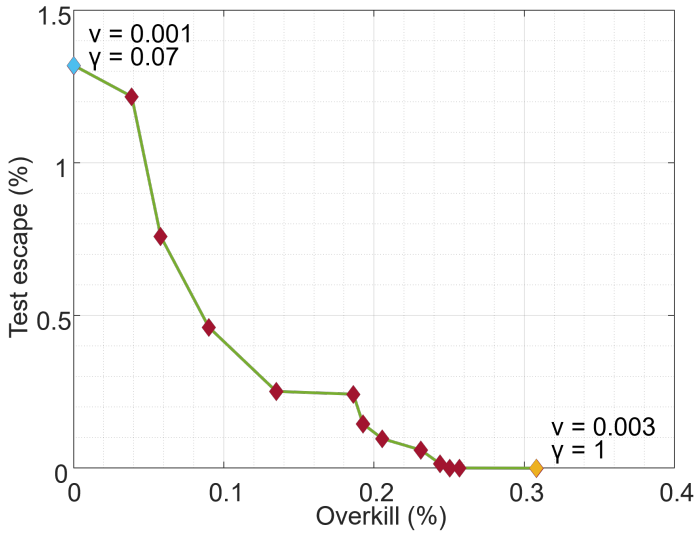


Figure 9.2: Pareto front curve test escape vs. overkill for SVM trained with different pairs of hyper-parameter values ν and γ.

SVMs is assessed on Group 1 only. Group 2 comprises the outlier input samples that the SNN did not learn to predict correctly after training. Group 3 comprises corner input samples for which the SNN prediction has high variance and can end up being incorrect. Groups 2 and 3 are not used for evaluating fault detection as the SNN already makes or is prone to making a wrong prediction anyways.

9.2.3 *Fault Model*

The fault model consists of permanent bit-flips in the memories that store the various SNN parameters of the neuromorphic design (i.e., synapse weights, neuron parameters, feature map parameters), similar to Section 6.2. We consider two scenarios, namely single bit-flips across different bit positions and multiple bit-flips uniformly distributed with a BER probability from 10^{-6} to 10^{-1}. In total, we consider 7404 SNN instances with single faults and 6278 SNN instances with multiple faults.

To assess the criticality of a fault, we consider its impact on the SNN classification result. Faults are categorized into:

- *Critical faults*: For a given input sample, a fault is critical if the predicted class of the faulty SNN is different than this of the fault-free SNN.
- *Benign faults*: For a given input sample, a fault is benign if the predicted class of the faulty SNN matches this of the fault-free SNN.

9.3 EXPERIMENTAL RESULTS

First we consider a single SVM and we assess the performance based on the resultant test escape and overkill. The trade-off is explored in Fig. 9.2 showing

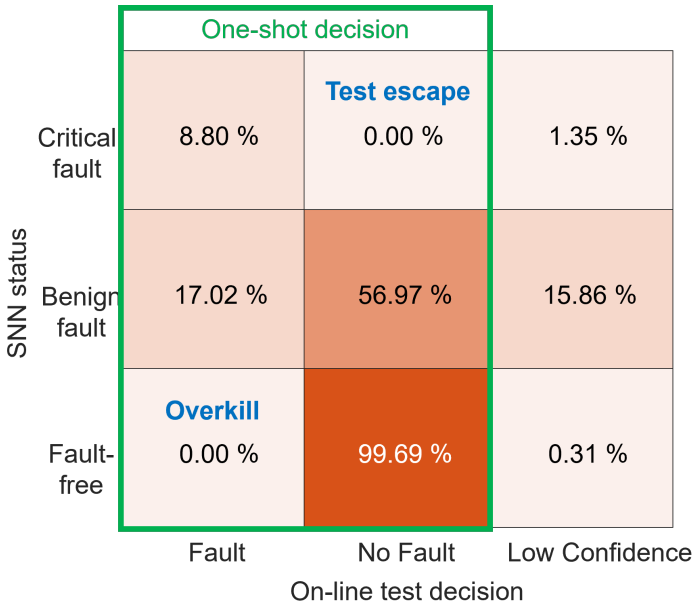


Figure 9.3: Performance of system with two SVM.

the Pareto front by training different SVMs while varying the combination of ν and γ values. The two end points marked in the Pareto front correspond to test escape for zero overkill and overkill for zero test escape, respectively. These two SVMs are the selected blue and yellow classifiers, respectively, shown in Fig. 9.1.

Fig. 9.3 shows the performance of the system of two SVMs for samples in Group 1. Rows correspond to the SNN status (i.e., critical fault, benign fault, fault-free) and the columns correspond to the on-line test decision. Out of all faults, $(8.8+0+1.35) = 10.15\%$ are critical and $(17.02+56.97+15.86) = 89.85\%$ are benign. As it can be seen, $(8.8/10.15)*100 = 86.7\%$ of critical faults are detected in real-time, while test escape and overkill are both zero for the one-shot decisions. However, the system has low confidence for $(1.35/10.15)*100 = 13.3\%$ of critical faults and 0.31% of fault-free inferences. Regarding benign faults, $(17.02/89.85)*100 = 18.94\%$ are proactively detected, $(56.97/89.85)*100 = 63.41\%$ are classified as no fault, and for $(15.86/89.85)*100 = 17.65\%$ the system has low confidence. With a replay operation all uncertainties are lifted. The 0.31% of fault-free inferences that were previously flagged as low-confidence are now flagged as “no-fault" since their footprint jumps inside the yellow boundary. Whereas, the inferences with the 13.3% of critical faults and 17.65% of benign faults that were previously flagged as low-confidence, are now flagged as “fault" since their footprint remains inside the grey zone.

Regarding the outlier and corner samples from Groups 2 and 3, the yellow SVM flags an error in all cases, thus the system successfully warns when SNN outputs incorrect predictions. Considering the system of two SVMs, all samples in Group 3 and 59.98% of samples in Group 2 lie in the low-confidence grey zone, while for the rest 40.02% of samples in Group 2 an error is flagged directly. The uncertainty is lifted with the replay operation as all these samples remain in the grey zone.

As a final note, deterministic one-shot decisions are completed without delaying the next SNN inference, while whenever there is a low-confidence decision there is a delay equal to the time of one inference due to the replay operation. The SVMs run in software on a second processor core, in parallel with the SNN operation. Thus, they are totally transparent to the SNN without interrupting or interfering with it. The only overhead in our hardware implementation is the utilization of the extra processor core dedicated to the SVMs that increases power consumption.

10 CONCLUSIONS

The last years have been characterized by a great improvement and a rapid evolution of AI, with numerous applications already embedding AI and many of them actually depending highly on it. The complexity of these applications requires very deep architectures of neural networks in order to satisfy the workload, which on their turn render necessary the employment of powerful hardware accelerators.

With this steep integration of AI in many fields, it is reasonable for questions to arise about the reliability of such applications. A starting point in an effort to answer them is to design more trustworthy neural networks based on the knowledge on what could fail in the network and how. Therefore, the network would be reinforced to withstand unwanted scenarios and avoid the worst consequences.

To achieve that, it is essential to remember that SNNs make up another human invention whose roots originate in biology. Thus, being inspired by the way a living brain works, a lot of remarkable characteristics emanate naturally and are inherited intact. However, since an imitation is very difficult to equal its original, the SNNs lack the great extend achieved by their biological counterparts in various of their features, with one of these being the fault tolerance capabilities. Consequently, when a structural component of a network, i.e. a neuron or a synapse, fails to operate properly, the cognition ability of the network is at risk and its performance may be degraded.

Trusting AI requires trusting the hardware accelerator on which the SNN is running, too. Taking into consideration that electronics and biology do not share any similarities in their fundamental principals, SNNs are inevitably susceptible to hardware-level faults.

As a conclusion, it is evident that neural networks and the AI hardware accelerators destined for them have a lot of vulnerabilities and addressing them is of utmost importance in order to ensure a reliable future for and with AI.

10.1 THESIS CONTRIBUTIONS

The purpose of this thesis was to explore the domain of reliability of AI hardware with the focus being on SNNs, so that more concrete answers can be provided

to the aforementioned questions. To this extend, the thesis contributions can be summarized as follows:

CHAPTER 3: We performed MC analysis and defect simulation for an analog I&F neuron at transistor level. Then we observed and categorized the resulting faulty behaviors to translate transistor-level faults into behavioral-level faults and errors. We then used our observations to form a comprehensive behavioral-level fault model that can be used to test spiking neurons regardless their implementation.

CHAPTER 4: We developed a GPU-accelerated fault injection framework to simulate faulty SNNs and observe the effects of the faulty behaviors at network level. The modeling of the neurons is quite modifiable, so that a range of faults and defects is covered, while the fault models derive from the ones described in Chapter 3. This way, a customization on the flow of computations is enough to accurately achieve the desired effects of the injected fault(s) on the network's performance. Using the fault injection framework, we inject a series of faults on two deep convolutional SNNs designed for the classification of the N-MNIST and the IBM's DVS128 Gesture datasets, accordingly. With the outcome of the experiments, we evaluate and demonstrate the criticality of faults and the severity of their effect on each network's performance, which concludes to a large-scale analysis of the resiliency of the two networks under study.

CHAPTER 5: We implemented a neuromorphic hardware experimentation platform specifically for SNNs. The base of the platform is the configurable event-driven SNN hardware architecture designed in VHDL and flushed on an FPGA board. Aside the network in hardware, we have built a support framework in MATLAB to set up and configure the network, perform the fault injection, map the network's faulty instance back into the hardware, and analyze the experimental results. All these come as an embedded system application operating under the aid of an on-board microprocessor that handles and monitors the underlying network.

CHAPTER 6: We used the experimentation platform from Chapter 5 to perform a fault injection experiment and fault resilience analysis for the SNN hardware accelerator. We then assessed the fault criticality on actual hardware to find that certain SNN parameters, i.e., splitter, router, and kernel parameters, are critical and must be protected, while for others, i.e., neuron threshold and kernel weights, the network shows some degree of resilience to faults occurring in LSBs. Therefore, these parameters can be the subject of selective fault tolerance reducing the cost of an all-around fault-tolerance.

CHAPTER 7: We presented a method to generate compact functional test sets for SNN hardware accelerators. A fault-agnostic metric is proposed to rank the available samples based on their fault coverage capability without performing any fault injection experiments. The functional test set is generated by performing inference only once for each available sample in

the training and/or testing sets and recording the output neuron spiking activity, i.e., an effort that is spent already during training. Thereafter, fault injection experiments are performed using only the highly ranked samples to compute fault coverage given a fault model. Results collected from our three SNN case studies as presented in Chapters 4 and 6, show that the proposed method generated highly compact functional test sets that can detect all faults resulting in even the smallest accuracy drop, i.e., one sample is misclassified.

CHAPTER 8: We leveraged the findings from the fault injection experiments from Chapter 4 to build a cost-effective neuron fault tolerance strategy for SNNs. The fault-tolerance strategy is a two-step procedure. In a first preparatory step, the SNN is trained using dropout which makes some neuron fault types for some layers passive. In a second step, we perform active fault tolerance to detect and recover from the remaining neuron faults in all layers. For hidden layers, we propose off-line and on-line fault detection schemes, a "fault hopping" concept to simplify the error recovery mechanism, and two different neuron-level and system-level recovery mechanisms. For the small output layer we simply use TMR.

CHAPTER 9: We presented a generic on-line testing methodology virtually applicable to any SNN hardware accelerator design and cognitive task. Two classifiers monitor the cumulative spike count at the output of feature maps of the SNN and are trained to detect in real-time outlier or corner inputs that are prone to misprediction, as well as hardware-level faults. This is achieved without generating any overkill thanks to the simultaneous assessment of the confidence in the decision. Whenever the confidence is low, a single replay operation suffices to resolve the ambiguity and make an accurate final decision. The methodology is fully demonstrated on our neuromorphic hardware experimentation platform presented in Chapter 5. It is shown that it enables trustworthy operation with zero-latency transparent decisions for over 99.6% of the SNN inferences, while for the rest the decision is made with a delay of one inference. The only overhead is the power consumption from the utilization of a second processor core dedicated to the integration of the two classifiers.

10.2 FUTURE PERSPECTIVES

Research is an ongoing process with many sub-parts popping up while going deeper in a domain. As extensive as a PhD thesis can be, there is always space for improvement and further exploration.

The fault injection framework presented in Chapter 4 was first developed to serve our experiments. The goal, which is already in the making, is to deliver it as a Python library ready to be easily used under the specification of a few commands by the user. It will be fast, meaning that the training or the inference times of a network will be affected in the slightest way possible, so that there is

no time overhead added to the application. The framework will be open-source and its code publicly available for everyone to use, contributing to the respective research field or serving as an industrial tool.

Another extension to the fault injection framework would be its integration with the experimentation platform presented in Chapter 5, so as to provide (i) a direct support for neuromorphic hardware; (ii) a unified working environment (no need for the current platform's assistive MATLAB framework); and (iii) an extra layer of acceleration by inserting the hardware in the loop for the training and inference of the studied network. This way the user experience is simplified and also the hardware plays an active role in all the experiments.

For the hardware oriented experiments, only the experimentation platform of Chapter 5 was considered. Although similar results are expected on the reliability analysis of the same and other networks on different platforms, e.g., if access to Loihi or SpiNNaker is granted, it would be interesting to verify this assumption. Also, applying the on-line testing method of Chapter 9 in more hardware platforms, helps in uncovering its portability and generality as a global testing mechanism.

Concerning the fault tolerance strategy proposed in Chapter 8, the next step would be to demonstrate it on actual neuromorphic hardware. For example, in architectures like our experimentation platform that groups the spiking neurons in nodes, it would be more convenient to check if nodes behave similarly to neurons, e.g., dead and saturated nodes, and if yes, modify the reactive fault tolerance strategy, so that it accommodates the needs of such designs.

BIBLIOGRAPHY

[1] A. Pinar Saygin, I. Cicekli, and V. Akman, "Turing test: 50 years later," *Minds and machines*, vol. 10, no. 4, pp. 463–518, 2000 (cit. on p. 3).

[2] A. M. Turing, "Computing Machinery and Intelligence," *Mind*, vol. LIX, no. 236, pp. 433–460, Oct. 1950 (cit. on p. 2).

[3] C. S. Strachey, "Logical or non-mathematical programmes," in *Proceedings of the 1952 ACM National Meeting (Toronto)*, ser. ACM '52, Toronto, Ontario, Canada: Association for Computing Machinery, 1952, pp. 46–49, ISBN: 9781450379250 (cit. on p. 2).

[4] M. Campbell, A. Hoane, and F.-h. Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, pp. 57–83, 2002 (cit. on p. 3).

[5] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943 (cit. on pp. 3, 34).

[6] W. A. Clark and B. G. Farley, "Generalization of pattern recognition in a self-organizing system," in *Proceedings of the March 1-3, 1955, western joint computer conference*, 1955, pp. 86–91 (cit. on p. 3).

[7] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958 (cit. on p. 3).

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998 (cit. on pp. 4, 56).

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989 (cit. on p. 4).

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017 (cit. on pp. 4, 9).

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015 (cit. on pp. 5, 6).

[12] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014. arXiv: 1409.1556 [cs.CV] (cit. on p. 4).

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016 (cit. on p. 4).

[14] H. Newquist, *The Brain Makers: Genius, Ego, And Greed In The Quest For Machines That Think*, 2nd. The Relayer Group, 2020 (cit. on p. 6).

[15] J. Lighthill, "Artificial intelligence: a paper symposium," *Science Research Council, London*, 1973 (cit. on p. 7).

[16] P. Jackson, "Introduction to expert systems," 1986 (cit. on p. 7).

[17] S. J. Russell and P. Norvig, "Artificial intelligence: a modern approach," *Prentice Hall Upper Saddle River, NJ, USA*, 2003 (cit. on p. 7).

[18] D. Steinkraus, I. Buck, and P. Simard, "Using gpus for machine learning algorithms," in *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, vol. 2, 2005, pp. 1115–1120 (cit. on pp. 9, 10).

[19] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *Tenth international workshop on frontiers in handwriting recognition*, 2006 (cit. on p. 9).

[20] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Twenty-second international joint conference on artificial intelligence*, 2011 (cit. on p. 9).

[21] N. Jouppi, *Google supercharges machine learning tasks with TPU custom chip*, https://cloud.google.com/blog/products/gcp/google-supercharges-machine-learning-tasks-with-custom-chip, Online, 2016 (cit. on p. 10).

[22] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017 (cit. on p. 10).

[23] C. Onay and E. Öztürk, "A review of credit scoring research in the age of big data," *Journal of Financial Regulation and Compliance*, 2018 (cit. on p. 12).

[24] S.-C. Liu, T. Delbruck, G. Indiveri, A. Whatley, and R. Douglas, *Event-based neuromorphic systems*. John Wiley & Sons, 2014 (cit. on pp. 12, 104).

[25] C. Mead, *Analog VLSI and Neural Systems*. Addison Wesley, 1989 (cit. on p. 13).

[26] G. Indiveri and T. K. Horiuchi, *Frontiers in neuromorphic engineering*, 2011 (cit. on p. 13).

[27] G. Volanis, A. Antonopoulos, A. A. Hatzopoulos, and Y. Makris, "Toward silicon-based cognitive neuromorphic ICs—a survey," *IEEE Design & Test*, vol. 33, no. 3, pp. 91–102, 2016 (cit. on pp. 13, 31, 34).

[28] R. Ananthanarayanan, S. K. Esser, H. D. Simon, and D. S. Modha, "The cat is out of the bag: cortical simulations with 10^9 neurons, 10^{13} synapses," in *Proceedings of the conference on high performance computing networking, storage and analysis*, 2009, pp. 1–12 (cit. on p. 13).

[29] C.-S. Poon and K. Zhou, "Neuromorphic silicon neurons and large-scale neural networks: challenges and opportunities," *Frontiers in Neuroscience*, vol. 5, 2011, Article 108 (cit. on p. 13).

[30] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014 (cit. on pp. 13, 34, 92).

[31] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, "Loihi: a neuromorphic manycore processor with on-chip learning," *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018 (cit. on pp. 14, 34, 35, 92).

[32] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Front. Neurosci.*, vol. 9, Apr. 2015, Article 141 (cit. on pp. 14, 34, 35, 93).

[33] L. Gomes, "Special report: can we copy the brain?-the neuromorphic chip's make-or-break moment," *IEEE Spectrum*, vol. 54, no. 6, pp. 52–57, 2017 (cit. on p. 14).

[34] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "SpiNNaker: a 1-W 18-core system-on-chip for massively-parallel neural network simulation," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, 2013 (cit. on pp. 14, 34, 35, 92).

[35] *ISO 26262: road vehicles-functional safety*, 2018 (cit. on p. 15).

[36] *IEC 61508: edition 2.0 functional safety*, 2010 (cit. on p. 15).

[37] S. Lawrence, C. L. Giles, and A. C. Tsoi, "What size neural network gives optimal generalization? convergence properties of backpropagation," 1998, Technical report (cit. on p. 16).

[38] P. Kerlirzin and F. Vallet, "Robustness in multilayer perceptrons," *Neural Computation*, vol. 5, no. 3, pp. 473–482, 1993 (cit. on p. 16).

[39] F. Su, C. Liu, and H.-G. Stratigopoulos, "Testability and dependability of AI hardware: survey, trends, challenges, and perspectives," *IEEE Design & Test*, 2023. DOI: 10.1109/MDAT.2023.3241116 (cit. on pp. 16, 22, 50, 98).

[40] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: a review," *IEEE Access*, vol. 5, pp. 17 322–17 341, Aug. 2017 (cit. on pp. 17, 82, 98, 103).

[41] European Commission, *Ethics guidelines for trustworthy artificial intelligence (AI)*, https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai, Online, Apr. 2019 (cit. on pp. 17, 18, 20).

[42] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Oct. 2004 (cit. on p. 22).

[43] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997 (cit. on p. 29).

[44] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952 (cit. on p. 30).

[45] R. FitzHugh, "Impulses and physiological states in models of nerve membrane," *Biophysical Journal*, vol. 1, pp. 445–466, 1961 (cit. on p. 30).

[46] J. Nagumo, S. Arimoto, and S. Yoshizawa, "An active pulse transmission line simulating nerve axon," *Proceedings of the IRE*, vol. 50, pp. 2061–2070, 1962 (cit. on p. 30).

[47] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003 (cit. on p. 30).

[48] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. v. Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers in neuroscience*, vol. 5, p. 73, 2011 (cit. on pp. 31, 34).

[49] W. Gerstner, "Time structure of the activity in neural network models," *Phys. Rev. E*, vol. 51, pp. 738–758, 1 1995 (cit. on p. 31).

[50] F. Ponulak and A. Kasiński, "Introduction to spiking neural networks: information processing, learning and applications," *Acta Neurobiol Exp*, vol. 71, no. 4, pp. 409–433, 2011 (cit. on p. 31).

[51] M. Abeles, *Local cortical circuits: an electrophysiological study*. Springer Science & Business Media, 1982, vol. 6 (cit. on p. 31).

[52] E. D. Adrian, "The impulses produced by sensory nerve endings: part i," *The Journal of physiology*, vol. 61, no. 1, p. 49, 1926 (cit. on p. 31).

[53] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *Nature*, vol. 381, pp. 520–522, 1996 (cit. on p. 31).

[54] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014 (cit. on pp. 32, 52).

[55] C. J. Shatz, "The developing brain," *Scientific American*, vol. 267, no. 3, pp. 60–67, 1992 (cit. on p. 32).

[56] J. Sjöström and W. Gerstner, "Spike-timing dependent plasticity," *Scholarpedia*, vol. 35, 2010 (cit. on p. 32).

[57] G. Bi and M. Poo, "Synaptic modification by correlated activity: hebb's postulate revisited," *Annual review of neuroscience*, vol. 24, no. 1, pp. 139–166, 2001 (cit. on p. 32).

[58] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International joint conference on neural networks (IJCNN)*, ieee, 2015, pp. 1–8 (cit. on p. 33).

[59] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience*, vol. 11, p. 682, 2017 (cit. on p. 33).

[60] S. K. Esser, P. A. Merolla, J. V. Arthur, *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, vol. 113, no. 41, pp. 11 441–11 446, Sep. 2016 (cit. on p. 33).

[61] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, 2002 (cit. on p. 33).

[62] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, vol. 10, p. 508, 2016 (cit. on p. 33).

[63] F. Zenke and S. Ganguli, "Superspike: supervised learning in multilayer spiking neural networks," *Neural computation*, vol. 30, no. 6, pp. 1514–1541, 2018 (cit. on p. 33).

[64] S. B. Shrestha and G. Orchard, "SLAYER: spike layer error reassignment in time," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, Dec. 2018, pp. 1412–1421 (cit. on pp. 33, 51, 52, 57).

[65] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Front. Neurosci.*, vol. 9, Nov. 2015, Article 437 (cit. on pp. 34, 56, 85).

[66] L. A. Camuñas-Mesa, Y. L. Domínguez-Cordero, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco, "A configurable event-driven convolutional node with rate saturation mechanism for modular convnet systems implementation," *Front. Neurosci.*, vol. 12, Feb. 2018, Article 63 (cit. on pp. 34, 68, 76, 85, 93).

[67] S. Panchapakesan, Z. Fang, and J. Li, "SyncNN: Evaluating and accelerating spiking neural networks on FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 4, pp. 1–27, 2022 (cit. on p. 34).

[68] J. Li, G. Shen, D. Zhao, Q. Zhang, and Z. Yi, "FireFly: A high-throughput and reconfigurable hardware accelerator for spiking neural networks," *arXiv preprint arXiv:2301.01905*, 2023 (cit. on p. 34).

[69] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, "Benchmarking keyword spotting efficiency on neuromorphic hardware," in *Proceedings of the 7th annual neuro-inspired computational elements workshop*, 2019, pp. 1–8 (cit. on p. 35).

[70] L. A. Camuñas-Mesa, T. Serrano-Gotarredona, and B. Linares-Barranco, "Event-driven sensing and processing for high-speed robotic vision," in *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*, IEEE, 2014, pp. 516–519 (cit. on p. 36).

[71] A. Joubert, B. Belhadj, O. Temam, and R. Héliot, "Hardware spiking neurons design: analog or digital?" In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2012, pp. 1–5 (cit. on p. 36).

[72] L. A. Camuñas-Mesa, B. Linares-Barranco, and T. Serrano-Gotarredona, "Neuromorphic spiking neural networks and their memristor-cmos hardware implementations," *Materials*, vol. 12, no. 17, p. 2745, 2019 (cit. on pp. 37, 38).

[73] L. Camunas-Mesa, A. Acosta-Jimenez, C. Zamarreno-Ramos, T. Serrano-Gotarredona, and B. Linares-Barranco, "A 32×32 pixel convolution processor chip for address event vision sensors with 155 ns event latency and 20 meps throughput," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 4, pp. 777–790, 2010 (cit. on p. 38).

[74] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: VGG and residual architectures," *Frontiers in neuroscience*, vol. 13, p. 95, 2019 (cit. on p. 38).

[75] S. A. El-Sayed, T. Spyrou, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Spiking neuron hardware-level fault modeling," in *Proc. 26th IEEE Int. Symp. On-Line Test. Robust Syst. Des. (IOLTS)*, Jul. 2020 (cit. on p. 39).

[76] A. N. Burkitt, "A review of the integrate-and-fire neuron model: i. homogeneous synaptic input," *Biological Cybernetics*, vol. 95, no. 1, pp. 1–19, 2006 (cit. on p. 40).

[77] R. Serrano-Gotarredona, T. Serrano-Gotarredona, A. Acosta-Jimenez, and B. Linares-Barranco, "A neuromorphic cortical-layer microchip for spike-based event processing vision systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 12, pp. 2548–2566, Dec. 2006 (cit. on pp. 40, 93).

[78] S. Sunter, K. Jurga, and A. Laidler, "Using mixed-signal defect simulation to close the loop between design and test," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2313–2322, 2016 (cit. on p. 42).

[79] B. Esen, A. Coyette, G. Gielen, W. Dobbelaere, and R. Vanhooren, "Effective DC fault models and testing approach for open defects in analog circuits," in *Proc. IEEE International Test Conference*, Paper 3.2, 2016 (cit. on p. 42).

[80] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Neuron fault tolerance in spiking neural networks," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Feb. 2021 (cit. on pp. 49, 82, 83, 86, 87, 97, 108).

[81] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, 2017, pp. 1–12 (cit. on pp. 50, 82, 83, 86).

[82] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei, "Ares: A framework for quantifying the resilience of deep neural networks," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, 2018, pp. 1–6 (cit. on pp. 50, 82, 86).

[83] J. J. Zhang, K. Basu, and S. Garg, "Fault-tolerant systolic array based accelerators for deep neural network execution," *IEEE Des. Test*, vol. 36, no. 5, pp. 44–53, Oct. 2019 (cit. on p. 50).

[84] E. Vatajelu, G. D. Natale, and L. Anghel, "Special session: reliability of hardware-implemented spiking neural networks (SNN)," in *Proc. IEEE VLSI Test Symp. (VTS)*, Apr. 2019 (cit. on pp. 50, 82, 86).

[85] S. A. El-Sayed, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Self-testing analog spiking neuron circuit," in *Proc. Int. Conf. Synth. Model. Anal. Simulat. Methods Appl. Circuit Design (SMACD)*, Jul. 2019 (cit. on pp. 50, 83, 108).

[86] F. F. dos Santos, P. F. Pimenta, C. Lunardi, L. Draghetti, L. Carro, D. Kaeli, and P. Rech, "Analyzing and increasing the reliability of convolutional neural networks on GPUs," *IEEE Trans. Reliab.*, vol. 68, no. 2, pp. 663–677, Jun. 2019 (cit. on pp. 50, 82, 83, 86).

[87] M. A. Neggaz, I. Alouani, S. Niar, and F. Kurdahi, "Are CNNs reliable enough for critical applications? An exploratory study," *IEEE Des. Test*, vol. 37, no. 2, pp. 76–83, Apr. 2020 (cit. on pp. 50, 82, 86).

[88] L.-H. Hoang, M. A. Hanif, and M. Shafique, "FT-ClipAct: resilience analysis of deep neural networks and improving their fault tolerance using clipped activation," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2020, pp. 1241–1246 (cit. on pp. 50, 83).

[89] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *Proc. Annual Int. Symp. Comput. Archit. (ISCA)*, Jun. 2012, pp. 356–367 (cit. on p. 50).

[90] G. Gambardella, J. Kappauf, M. Blott, C. Doehring, M. Kumm, P. Zipf, and K. Vissers, "Efficient error-tolerant quantized neural network accelerators," in *Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Nanotechnol. Syst. (DFT)*, Oct. 2019 (cit. on pp. 50, 83).

[91] L. Xia, W. Huangfu, T. Tang, X. Yin, K. Chakrabarty, Y. Xie, Y. Wang, and H. Yang, "Stuck-at fault tolerance in RRAM computing systems," *IEEE J. Emerg. Sel. Top. Circuits Syst.*, vol. 8, no. 1, pp. 102–115, 2018 (cit. on p. 50).

[92] C. D. Schuman, J. P. 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 *Proc. Int. Jt. Conf. Neural Netw. (IJCNN)*, 2020, pp. 1–10 (cit. on pp. 50, 82, 86, 98, 108).

[93] H.-Y. Tseng, I.-W. Chiu, M.-T. Wu, and J. C.-M. Li, "Machine learning-based test pattern generation for neuromorphic chips," in *IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2021 (cit. on pp. 50, 82, 83, 86, 87, 95, 96, 108).

[94] R. V. W. Putra, M. A. Hanif, and M. Shafique, "SoftSNN: low-cost fault tolerance for spiking neural network accelerators under soft errors," in *Proc. 59th Design Autom. Conf. (DAC)*, Jul. 2022, pp. 151–156 (cit. on pp. 50, 82, 83, 86, 98, 108).

[95] A. Mahmoud, N. Aggarwal, A. Nobbe, J. R. S. Vicarte, S. V. Adve, C. W. Fletcher, I. Frosio, and S. K. S. Hari, "PyTorchFI: a runtime perturbation tool for DNNs," in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, 2020, pp. 25–31 (cit. on p. 50).

[96] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman, and N. DeBardeleben, "TensorFI: a flexible fault injection framework for tensorflow applications," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. (ISSRE)*, 2020, pp. 426–435 (cit. on p. 50).

[97] Y. He, P. Balaprakash, and Y. Li, "FIdelity: efficient resilience analysis framework for deep learning accelerators," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Oct. 2020, pp. 270–281 (cit. on pp. 50, 82, 86).

[98] L. M. Luza, A. Ruospo, D. Söderström, C. Cazzaniga, M. Kastriotou, E. Sanchez, A. Bosio, and L. Dilillo, "Emulating the effects of radiation-induced soft-errors for the reliability assessment of neural networks," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 1867–1882, 2022 (cit. on p. 50).

[99] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben, "*BinFI*: an efficient fault injector for safety-critical machine learning systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Nov. 2019 (cit. on p. 50).

[100] A. Chaudhuri, J. Talukdar, F. Su, and K. Chakrabarty, "Functional criticality classification of structural faults in AI accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2022, early access (cit. on p. 50).

[101] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "SASSIFI: an architecture-level fault injection tool for GPU application resilience evaluation," in *IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2017, pp. 249–258 (cit. on p. 50).

[102] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler, "NVBitFI: dynamic fault injection for GPUs," in *Proc. 51st Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, 2021, pp. 284–291 (cit. on p. 50).

[103] M. Abadi, A. Agarwal, P. Barham, *et al.*, *TensorFlow: large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/ (cit. on p. 50).

[104] F. Chollet *et al.*, *Keras*, https://keras.io, 2015 (cit. on p. 50).

[105] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688 (cit. on p. 50).

[106] A. Paszke *et al.*, "PyTorch: an imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on pp. 50, 51).

[107] A. Amir *et al.*, "A low power, fully event-based gesture recognition system," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017 (cit. on pp. 56, 57, 85).

[108] Y. LeCun, C. Cortes, and C. J. Burges, *The MNIST database of handwritten digits*, http://yann.lecun.com/exdb/mnist/, Online (cit. on p. 56).

[109] C. Zamarreño-Ramos, A. Linares-Barranco, T. Serrano-Gotarredona, and B. Linares-Barranco, "Multicasting mesh AER: a scalable assembly approach for reconfigurable neuromorphic structured AER systems. application to ConvNets," *IEEE Trans. Biomed. Circuits Syst.*, vol. 7, no. 1, pp. 82–102, Feb. 2013 (cit. on p. 69).

[110] T. Spyrou, S. A. El-Sayed, E. Afacan, L. A. Camuñas-Mesa, B. Linares-Barranco, and H.-G. Stratigopoulos, "Reliability analysis of a spiking neural network hardware accelerator," in *Proc. Design Autom. Test Europe Conf. (DATE)*, Mar. 2022 (cit. on pp. 75, 82, 86).

[111] J. A. Pérez-Carrasco *et al.*, "Mapping from frame-driven to frame-free event-driven vision systems by low-rate rate coding and coincidence processing–application to feedforward ConvNets," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 11, pp. 2706–2719, Nov. 2013 (cit. on p. 76).

[112] S. A. El-Sayed, T. Spyrou, L. A. Camuñas-Mesa, and H.-G. Stratigopoulos, "Compact functional testing for neuromorphic computing circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2022 (cit. on pp. 81, 108).

[113] D. Maliuk, H.-G. Stratigopoulos, H. Huang, and Y. Makris, "Analog neural network design for RF built-in self-test," in *Proc. IEEE Int. Test Conf. (ITC)*, Paper 23.2, Nov. 2010 (cit. on pp. 82, 84).

[114] Y. Huang and R. Singhai, "Tutorial 1B: AI chip technologies and DFT methodologies," in *Proc. IEEE Int. Syst.-on-Chip Conf. (SOCC)*, Sep. 2019 (cit. on p. 82).

[115] S. Motaman, S. Ghosh, and J. Park, "A perspective on test methodologies for supervised machine learning accelerators," *IEEE Trans. Emerg. Sel. Topics Power Electron.*, vol. 9, no. 3, pp. 562–569, Aug. 2019 (cit. on p. 82).

[116] A. Gebregiorgis and M. B. Tahoori, "Testing of neuromorphic circuits: structural vs functional," in *Proc. IEEE Int. Test Conf. (ITC)*, Paper 3.2, Nov. 2019 (cit. on p. 82).

[117] A. Ankit, I. Chakraborty, A. Agrawal, M. Ali, and K. Roy, "Circuits and architectures for in-memory computing-based machine learning accelerators," *IEEE Micro*, vol. 40, no. 6, pp. 8–22, 2020 (cit. on p. 82).

[118] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: opportunities and challenges," *Front. Neurosci.*, vol. 12, Oct. 2018, Article 774 (cit. on p. 82).

[119] L. A. Camuñas-Mesa, B. Linares-Barranco, and T. Serrano-Gotarredona, "Spiking neural networks and their memristor-CMOS hardware implementations," *Materials*, vol. 12, no. 17, Aug. 2019, Article 2745 (cit. on p. 82).

[120] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: a survey," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 2, Apr. 2019 (cit. on p. 82).

[121] A. Hashmi, H. Berry, O. Temam, and M. Lipasti, "Automatic abstraction and fault tolerance in cortical microachitectures," in *Proc. ACM/IEEE Annual Int. Symp. Comput. Archit. (ISCA)*, Jun. 2011, pp. 1–10 (cit. on pp. 82, 98, 99).

[122] R. V. W. Putra, M. A. Hanif, and M. Shafique, "ReSpawn: energy-efficient fault-tolerance for spiking neural networks considering unreliable memories," in *Proc. ACM/IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2021 (cit. on pp. 82, 86, 98, 99, 108).

[123] A. Ruospo, A. Bosio, A. Ianne, and E. Sanchez, "Evaluating convolutional neural networks reliability depending on their data representation," in *Proc. 23rd Euromicro Conf. Digit. Syst. Des. (DSD)*, Aug. 2020, pp. 672–679 (cit. on pp. 82, 86).

[124] A. Chaudhuri, C. Liu, X. Fan, and K. Chakrabarty, "C-testing and efficient fault localization for AI accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2021, early access (cit. on p. 82).

[125] S. Kundu, S. Banerjee, A. Raha, S. Natarajan, and K. Basu, "Toward functional safety of systolic array-based deep learning hardware accelerators," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 485–498, Jan. 2021 (cit. on pp. 82, 83, 95).

[126] S. T. Ahmed and M. B. Tahoori, "Campact functional test generation for memristive deep learning implementations using approximate gradient ranking," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2022, pp. 239–248 (cit. on pp. 82, 83, 95).

[127] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech, "Selective hardening for neural networks in FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 66, no. 1, pp. 216–222, Jan. 2019 (cit. on p. 83).

[128] Z. Xu and J. Abraham, "Safety design of a convolutional neural network accelerator with error localization and correction," in *Proc. IEEE Int. Test Conf. (ITC)*, Paper 12.3, Nov. 2019 (cit. on p. 83).

[129] M. Liu, L. Xia, Y. Wang, and K. Chakrabarty, "Algorithmic fault detection for RRAM-based matrix operations," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 3, 29:1–29:31, May 2020 (cit. on p. 83).

[130] E. Ozen and A. Orailoglu, "Low-cost error detection in deep neural network accelerators with linear algorithmic checksums," *J. Electron. Test.: Theory Appl.*, vol. 36, no. 6, pp. 703–718, Dec. 2020 (cit. on p. 83).

[131] S. Hari, M. Sullivan, T. Tsai, and S. Keckler, "Making convolutions resilient via algorithm-based error detection techniques," *IEEE Trans. Dependable Secure Comput.*, Mar. 2021, early access (cit. on p. 83).

[132] A. Ruospo, D. Piumatti, A. Floridia, and E. Sanchez, "A suitability analysis of software based testing strategies for the on-line testing of artificial neural networks applications in embedded devices," in *Proc. IEEE Int. Symp. On-Line Test. Robust Syst. Des. (IOLTS)*, 2021 (cit. on p. 83).

[133] M. Liu and K. Chakrabarty, "Online fault detection in ReRAM-based computing systems by monitoring dynamic power consumption," in *Proc. IEEE Int. Test Conf. (ITC)*, Nov. 2020 (cit. on p. 83).

[134] N. I. Deligiannis, R. Cantoro, M. Sonza Reorda, M. Traiola, and E. Valea, "Towards the integration of reliability and security mechanisms to enhance the fault resilience of neural networks," *IEEE Access*, vol. 9, pp. 155 998–156 012, 2021 (cit. on p. 83).

[135] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: automated whitebox testing of deep learning systems," in *Proc. 26th ACM Symp. Oper. Syst. Princ. (SOSP)*, Oct. 2017 (cit. on pp. 83, 94).

[136] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: automated testing of deep-neural-network-driven autonomous cars," in *Proc. IEEE/ACM Int. Conf. Softw. Eng. (ICSE)*, 2018, pp. 303–314 (cit. on pp. 83, 84, 94, 95).

[137] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jun. 2014 (cit. on pp. 86, 99).

[138] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2010, pp. 1947–1950 (cit. on p. 93).

[139] B. V. Benjamin *et al.*, "Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, Apr. 2014 (cit. on p. 93).

[140] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 106–122, Nov. 2018 (cit. on p. 93).

[141] D. Ma *et al.*, "Darwin: a neuromorphic hardware co-processor based on spiking neural networks," *J. Syst. Archit.*, vol. 77, pp. 43–51, Jun. 2017 (cit. on p. 93).

[142] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, "A 4096-neuron 1M-synapse 3.8-pJ/SOP spiking neural network with on-chip STDP learning and sparse weights in 10-nm FinFET CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, Apr. 2019 (cit. on p. 93).

[143] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm^2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 1, pp. 145–158, Feb. 2019 (cit. on p. 93).

[144] J. D. Victor and K. P. Purpura, "Nature and precision of temporal coding in visual cortex: a metric-space analysis," *J. Neurophysiol.*, vol. 76, no. 2, pp. 1310–1326, Aug. 1996 (cit. on p. 94).

[145] M. C. van Rossum, "A novel spike distance," *Neural Comput.*, vol. 13, no. 4, pp. 751–763, Apr. 2001 (cit. on p. 94).

[146] T. Kreuz, J. S. Haas, A. Morelli, H. D. Abarbanel, and A. Politi, "Measuring spike train synchrony," *J. Neurosci. Methods*, vol. 165, no. 1, pp. 151–161, Sep. 2007 (cit. on p. 94).

[147] T. Kreuz, D. Chicharro, C. Houghton, R. G. Andrzejak, and F. Mormann, "Monitoring spike train synchrony," *J. Neurophysiol.*, vol. 109, no. 5, pp. 1457–1472, Mar. 2013 (cit. on p. 94).

[148] E. Satuvuori, M. Mulansky, N. Bozanic, I. Malvestio, F. Zeldenrust, K. Lenk, and T. Kreuz, "Measures of spike train synchrony for data with multiple time scales," *J. Neurosci. Methods*, vol. 287, pp. 25–38, Aug. 2017 (cit. on p. 94).

[149] R. V. W. Putra, M. A. Hanif, and M. Shafique, "SparkXD: a framework for resilient and energy-efficient spiking neural network inference using approximate DRAM," in *Proc. 58th Design Autom. Conf. (DAC)*, Dec. 2021, pp. 379–384 (cit. on pp. 98, 108).

[150] T. Spyrou and H.-G. Stratigopoulos, "On-line testing of neuromorphic hardware," in *Proc. IEEE Eur. Test Symp. (ETS)*, May 2023 (cit. on p. 107).

[151] H.-G. Stratigopoulos and Y. Makris, "Error moderation in low-cost machine-learning-based analog/RF testing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 2, pp. 339–351, Feb. 2008 (cit. on p. 110).

[152] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, Jul. 2001 (cit. on p. 111).

[153] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, 27:1–27:27, 3 Apr. 2011, Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm` (cit. on p. 111).