



**Politecnico
di Torino**

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Artificial Intelligence (37.th cycle)

Energy-Efficient Neuromorphic Hardware

Design and Optimization of Brain-Inspired Computing
Paradigms for Spiking Neural Networks

Alessio Carpegna

* * * * *

Supervisors

Prof. Stefano Di Carlo, Supervisor
Prof. Alessandro Savino Co-supervisor

Politecnico di Torino
April 9, 2025

This thesis is licensed under a Creative Commons License, Attribution - Noncommercial-NoDerivative Works 4.0 International: see www.creativecommons.org. The text may be reproduced for non-commercial purposes, provided that credit is given to the original author.

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

.....

Alessio Carpegna
Turin, April 9, 2025

Summary

This thesis investigates neuromorphic computing from multiple perspectives, including bio-inspired algorithms, specialized hardware accelerators, optimization techniques, and continual learning strategies. By drawing inspiration from biological processes, this research aims to enhance the efficiency of Artificial Intelligence (AI) in edge computing, facilitating the development of autonomous and real-time intelligent systems. AI has already revolutionized various industries, including healthcare, autonomous systems, manufacturing, space exploration, and sustainable energy, by leveraging large datasets, pattern recognition, and intelligent decision-making. However, the increasing complexity of AI models presents significant computational challenges, particularly in terms of energy efficiency and real-time processing. Traditional AI systems primarily rely on cloud computing, which, despite its computational power, introduces latency and privacy concerns, making it unsuitable for real-time and security-sensitive applications. Edge AI addresses these limitations by enabling local execution on resource-constrained devices such as IoT sensors, embedded systems, and autonomous machines. Nonetheless, deploying AI at the edge is constrained by limited power, memory, and computational capacity, necessitating innovative solutions to improve efficiency. To optimize AI for edge applications, several techniques have been developed. Among these, neuromorphic computing offers one of the most promising approaches by mimicking the brain's event-driven processing and distributed memory mechanisms. This biologically inspired paradigm enables highly efficient AI models that operate with minimal energy consumption. The contributions of this thesis to neuromorphic engineering and Spiking Neural Network (SNN) research span multiple areas. A primary focus is the development of specialized hardware accelerators to enhance SNN execution. This includes a high-level Python framework designed to simplify the design, training, and deployment of Field Programmable Gate Array (FPGA)-based SNNs, making these technologies more accessible to researchers without extensive hardware expertise. Additionally, this work explores automatic optimization techniques for SNN architectures, leveraging multi-objective optimization strategies to balance accuracy, power efficiency, and latency. Another key aspect is continual learning in SNNs, allowing models to dynamically adapt and refine their knowledge as they process new data, thereby replicating the flexibility of biological learning.

Furthermore, this research investigates emerging technologies and analog circuits to significantly reduce power consumption in healthcare applications. By integrating biological inspiration with computational efficiency, this work advances the field of edge AI through neuromorphic principles, hardware acceleration, and optimization techniques. The ultimate goal is to enable the development of more energy-efficient, adaptive, and autonomous AI systems, providing scalable solutions to real-world challenges in intelligent computing.

Acknowledgements

The deepest thank you goes to Stefano Di Carlo, without whom I would never have been where I am. During my Master's studies, pursuing a Ph.D. was not part of my plans. It was only through working with him that I decided to take this path, and it was absolutely worth it. Over these years, Stefano has been much more than a Ph.D. supervisor, he has been an example in many aspects of both academic and everyday life, a perfect travel companion at conferences, a constant source of guidance, and, above all, a friend. Thank you for giving me the opportunity to undertake this adventure. I've learned more than I could ever express within the pages of this thesis.

And immediately after, I want to thank Alessandro Savino, who taught me so much, offered me his unwavering support, and worked alongside Stefano to help me grow both professionally and personally over these years. I am profoundly grateful to you, and I deeply appreciate everything you've done for me.

I would like to acknowledge and thank Giacomo Indiveri, together with Elisa Donati, Chiara De Luca, and the entire team at Institute of Neuro-Informatics (INI) Zurich, for hosting me and providing me with the invaluable opportunity to learn and grow. I am grateful for the chance to observe exceptional researchers at work, which was an inspiring experience. Part of this thesis, and in particular chapter 6 is based on the work I did during my period at INI.

Another portion of my Ph.D. was dedicated to the collaboration with Davide Nadalini and Alberto Dequino from the PULP group, alongside professors Luca Benini and Francesco Conti. It was one of the most enjoyable and inspiring experiences of my Ph.D., and it led to a publication upon which chapter 5 is based. Thank you all, working with you was both a pleasure and an honor.

And finally, a heartfelt thank you to the SMILIES research group at Politecnico, from the long-time members to the newcomers. You are true friends to me, first and foremost, before being colleagues. If I have enjoyed these years as much as I have, it is in no small part thanks to all of you.

*Strange about learning; the farther I go
the more I see that I never knew even
existed. A short while ago I foolishly
thought I could learn everything - all the
knowledge in the world. Now I hope only
to be able to know of its existence, and to
understand one grain of it. Is there
time?*

*(Keyes D., 1958, Flowers For Algernon.
Weidenfeld & Nicolson.)*

List of Acronyms

- AdExIF** Adaptive Exponential Integrate-and-Fire. xxii, 13, 90, 92, 93
- ADSE** Automatic Design Space Exploration. 49, 50, 69
- AER** Address Event Representation. 25, 89, 90
- AI** Artificial Intelligence. iii, iv, 1–5, 8, 9, 17, 35, 50, 73, 99
- ANN** Artificial Neural Network. xix, 3, 4, 6, 9–11, 14, 17–19, 23, 25, 49, 50, 73, 74
- API** Application Programming Interface. 58
- ASIC** Application-Specific Integrated Circuit. 22, 24, 25
- AX** Adaptive eXperimentation. 58, 70
- BCM** Bienenstock–Cooper–Munro. 18
- BPM** Beats Per Minute. 92, 93
- BPTT** Back-Propagation Through Time. xix, 18, 19, 23, 36, 76, 77, 79, 85
- BRAM** Block RAM. 32, 34, 38, 39, 42, 45–47, 54
- CAM** Content Addressable Memory. 90
- CL** Continual Learning. 73–77, 79–81
- CMOS** Complementary Metal Oxide Semiconductor. 8, 89, 90
- CNN** Convolutional Neural Network. 2, 10, 25, 49, 73–76, 80
- CPU** Central Processing Unit. 5, 23, 24
- CSNN** Convolutional Spiking Neural Network. 70
- CU** Control Unit. xx, 26–28, 41, 42

DAC Digital to Analog Converter. 89

DL Deep Learning. 1, 2, 4, 22

DNN Deep Neural Network. 2

DRAM Dynamic Random Access Memory. 32

DSE Design Space Exploration. xxi, 25, 52, 53, 56, 58, 67

DVS Dynamic Vision Sensor. 32, 62, 65, 68, 69

E-I Excitatory-Inhibitory. xxii, 91–95

ECG Electrocardiography. xxii, 3, 88, 92, 96, 97

EDA Electronic Design Automation. 23

EIF Exponential Integrate and Fire. 13

ELUT Equivalent Look Up Table. 56

EMG Electromyography. 88

ETH Eidgenössische Technische Hochschule. 25

FAIR Findability, Accessibility, Interoperability, and Reusability. 1, 103

FC Fully-Connected. 35, 55, 70

FC-R Fully-Connected Recurrent. 12, 33, 35, 36, 39, 41, 47

FF Flip Flop. 32, 38

FF-FC Feed-Forward Fully-Connected. xx, 12, 26, 27, 33, 35, 36, 41, 46, 47, 56

FFNN Feed-Forward Neural Network. 10

FPGA Field Programmable Gate Array. iii, xvii, 5, 6, 22–25, 31, 32, 34–38, 46, 47, 50–52, 54, 56, 62, 63, 67–70

GLIF Generalized Leaky Integrate and Fire. 13

GPU Graphic Processing Unit. 2, 19, 22, 23

GRU Gated Recurrent Units. 18

H-H Hodgkin-Huxley. 12, 13

HFO High Frequency Oscillation. 88

HR Heart Rate. 87, 88, 92, 93, 96, 97

IBM International Business Machines. 8

IEEE Institute of Electrical and Electronics Engineers. xvii, xviii, xxi, xxii, 73, 76–78, 80–84, 87, 93–95, 97

IF Integrate and Fire. xx, 16, 29–31, 37, 47, 52, 54, 57

INI Institute of Neuro-Informatics. vi, 25

IoT Internet of Things. 2, 4, 25

LIF Leaky Integrate and Fire. xx, 8, 13–16, 29–31, 35, 37, 39, 46, 47, 54, 57, 63, 79, 90, 92

LR Latent Replay. 74–84

LSTM Long Short Term Memory. 18

LTD Long Term Depression. 18

LTP Long Term Potentiation. 18

LUT Look Up Table. 31, 32, 34, 38, 46, 54, 56

MAC Multiply and Accumulate. 29

MCP McCulloch-Pitts. 9

ML Machine Learning. 9, 16, 62, 70

NAS Network Architecture Search. 51

NE Network Evaluator. 52, 53

NG Network Generator. 53, 56

NIR Neuromorphic Intermediate Representation. 35

NN Neural Network. 11, 18

NSM Neural State Machine. xxii, 88, 91–95

PPG Photoplethysmograms. 92, 97

QAT Quantization Aware Training. 34

QIF Quadratic Integrate and Fire. 13

RAM Random Access Memory. 34

RC-R Randomly-Connected Recurrent. 12

ReLU Rectified Linear Unit. 10

RIF Resonator Integrate and Fire. 13

RL Reinforcement Learning. 50

ROM Read Only Memory. 34

RTL Register Transfer Level. 33, 51

SbS Spike-by-Spike. 37

SCNN Spiking Convolutional Neural Networks. 37

SG Surrogate Gradient. 19, 79

SHD Spiking Heidelberg Dataset. xvii, 23, 35, 39–42, 47, 62–65, 67–70, 74, 79

SIMD Single Instruction Multiple Data. 23, 24

SNN Spiking Neural Network. iii, xvii, xix, 5, 6, 8, 9, 11, 12, 16–19, 22–28, 31, 33–37, 40, 44, 47, 49–55, 58, 62, 64, 67–71, 73–77, 79, 80, 85

SRAM Static Random Access Memory. 32

STDP Spike-Timing-Dependent Plasticity. 18, 89, 90

TPU Tensor Processing Unit. 22, 23

UAT Universal Approximation Theorem. 10, 11

VHDL VHSIC Hardware Description Language. 34, 35

VLSI Very Large Scale Integration. 8

WTA Winner Takes All. xxii, 91–96

Contents

List of Acronyms	X
List of Tables	XVII
List of Figures	XIX
1 Thesis goals and scope	1
1.1 The Role of Artificial Intelligence	1
1.2 The need for local AI at the edge	2
1.3 Deploying AI at the edge: taking inspiration from the brain	4
1.4 Research contributions	5
2 Background	7
2.1 An historical perspective: drawing inspiration from the human brain	7
2.2 A computational perspective: spikes processing	8
2.3 Approximating neural computation: the birth of mathematical neuron models	9
2.4 From Neurons to Networks: The Universal Approximation Theorem	10
2.5 Networks of spiking neurons	11
2.6 Overview of mathematical models of spiking neurons	12
2.7 The Leaky Integrate and Fire model	14
2.8 Encoding information into sequences of spikes	16
2.9 Training	17
3 Spiker+: a framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge	21
3.1 Why accelerating Spiking Neural Networks	21
3.2 Spiker+ overview	22
3.3 Neuromorphic accelerators: related work	23
3.4 Spiker+ architecture	26
3.4.1 Network architecture	26
3.4.2 Network CU: global synchronization	27
3.4.3 Layer CU: deliver spikes to neurons	27

3.4.4	Neuron models	29
3.4.5	Synapses	31
3.4.6	I/O interface	32
3.5	Configuration framework	33
3.6	Experimental results	35
3.6.1	Benchmarking	37
3.6.2	Performance vs input activity	41
3.6.3	Performance vs quantization	44
3.6.4	Performance vs. size	45
3.7	Conclusions	47
4	SpikExplorer: hardware-oriented Design Space Exploration for Spiking Neural Networks on FPGA	49
4.1	Automatic Design Space Exploration	50
4.2	Related works	51
4.3	Materials and Methods	52
4.3.1	Network Generator and hardware neurons	53
4.3.2	Area	54
4.3.3	Accuracy and latency	56
4.3.4	Power	57
4.3.5	DSE engine	58
4.4	Experimental results	62
4.4.1	Experimental set-up	62
4.4.2	Global Exploration	63
4.4.3	Fixed neuron models and network size	66
4.4.4	Synthesis and comparison with State of Art	67
4.5	Conclusions and future work	69
5	Compressed Latent Replays for Lightweight Continual Learning on Spiking Neural Networks	73
5.1	Related Work	74
5.2	Latent Replay-based Continual Learning in SNNs	76
5.2.1	Latent Replays in SNNs	77
5.2.2	Optimization of Latent Replay memory	78
5.3	Experimental Results	79
5.3.1	Experimental Setup	79
5.3.2	Weights initialization	79
5.3.3	Sample-Incremental CL	79
5.3.4	Class-Incremental CL	80
5.3.5	Classification Accuracy vs Number of LRs	81
5.3.6	Compressed LRs	83
5.3.7	Comparison with other compressions	84

5.3.8	Multi-Class-Incremental Setup	84
5.4	Conclusions and future work	85
6	Neuromorphic Heart Rate Monitors: Neural State Machines for Monotonic Change Detection	87
6.1	Background	89
6.1.1	Neuromorphic Hardware	89
6.1.2	Neuromorphic primitives	91
6.2	Materials and Methods	92
6.2.1	Dataset and signal processing	92
6.2.2	Network on chip	92
6.3	Experimental Results	94
6.3.1	WTA network: non-monotonic state transitions	94
6.3.2	NSM network: monotonic state transitions	95
6.3.3	Network dynamics on real ECG signal	96
6.3.4	Power consumption	96
6.4	Conclusions	96
7	Conclusions	99
8	List of publications	101
9	Code availability	103
	Bibliography	105

List of Tables

3.1	Summary of the experimental set-up on the two datasets. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY. . . .	36
3.2	Comparison of Spiker+ to state-of-the-art FPGA accelerators for SNNs. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	37
3.3	Benchmarking on the Spiking Heidelberg Dataset (SHD) and AudioMNIST datasets. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	39
3.4	Synthesis of maximum size accelerator on different Xilinx™ FPGAs boards. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	46
4.1	Set of specifications that the user can provide. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.	61
4.2	Set of experimental parameters provided to SpikExplorer. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY. . . .	63
4.3	Best architectures with the four neuron models on the MNIST. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY. . .	66
4.4	Best architectures with the four neuron models on the SHD. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY. . . .	66
4.5	Best architectures with the four neuron models on the DVS. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY. . . .	66
4.6	Comparison of SpikExplorer to state-of-the-art FPGA accelerators for SNNs. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.	70
5.1	Memory-Accuracy tradeoff for Sample-Incremental CL, with 2560 LRs and variable C_r and LR index. Pre-training accuracy on the scenario to be learned was 65%. Reproduced from Dequino et al. 2024 [154] ©2024 Institute of Electrical and Electronics Engineers (IEEE).	82
5.2	Memory-Accuracy tradeoff for Class-Incremental CL, with 2432 LRs and variable C_r and LR index. Reproduced from Dequino et al. 2024 [154] ©2024 IEEE.	83

6.1 Connection types and average probabilities. Reproduced from Carpegna et al. 2024 [188] ©2024 IEEE. 94

List of Figures

1.1	Comparison between a cloud computing paradigm (1.1a) and an edge computing approach (1.1b). In the cloud-based model, all devices transmit data over the internet to a centralized and powerful main-frame that carries out tasks such as inference using the Artificial Neural Network (ANN) model, subsequently sending the results back to the devices. Conversely, the edge computing model integrates an onboard electronic unit within each device to perform inference locally. This decentralized approach mitigates issues such as unpredictable latency, excessive power consumption, bandwidth overload, and concerns related to data security and privacy.	3
1.2	Methods to reduce computing, area and energy requirements of a ANN	4
2.1	Artificial neuron model	10
2.2	Spiking neuron model	11
2.3	Typical architectures of an SNN. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	12
2.4	In the <i>syn</i> model, synapses exhibit a dynamic response characterized by a decay parameter α , which governs how incoming spikes are weighted and how the synaptic current decays over time in the absence of stimulation. The resulting synaptic currents are then integrated by the membrane potential, which itself follows a second dynamic governed by a time constant β . In the <i>lif</i> model, the temporal dynamics of the synapses are disregarded; each synapse simply scales the input spike by a fixed weight, and only the membrane exhibits leaky integration. Finally, in the simplest <i>if</i> model, even the membrane's decay is omitted: input spikes are accumulated over time without any leak or temporal dynamics.	14
2.5	Example of unrolling of the network in Back-Propagation Through Time (BPTT) (left), with surrogate gradient function for the spiking activation (right).	19
3.1	Landscape of neuromorphic hardware. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	24

3.2	Spiker+ example of Feed-Forward Fully-Connected (FF-FC) architecture. The example includes three layers with different numbers of neurons and depicts all the architecture’s main control blocks. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	27
3.3	Internal architecture of the Network and Layer Control Units (CUs). Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	28
3.4	Spiker+ neuron architectures in order of increasing complexity (Integrate and Fire (IF), I-order Leaky Integrate and Fire (LIF) and II-order LIF), with subtractive reset. In the multiplexers, channels labeled with I indicate the beginning of the Integrate path, R the beginning of the Reset path, and L the beginning of the Leakage path. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	30
3.5	Spiker+ neuron architectures in order of increasing complexity (IF, I-order LIF and II-order LIF), with fixed reset. In the multiplexers, channels labeled with I indicate the beginning of the Integrate path, R the beginning of the Reset path, and L the beginning of the Leakage path. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	31
3.6	Spiker+ configuration framework. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	33
3.7	Pareto optimal curves of the accelerators presented in Table 3.2. (a) Accuracy is plotted against the three key hardware metrics: latency, which reflects computation speed, power consumption, and area utilization, measured by the number of logic cells required by the accelerator. Memory usage is closely tied to area utilization, so it is not shown here. (b) Power consumption is plotted against latency and area utilization. (c) Area utilization is plotted versus latency. Spiker+ emerges as the optimal solution for all hardware-related metrics, though it still falls slightly short in terms of accuracy. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	40
3.8	Visual representation of the average number of active cycles at various stages of the network. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	42
3.9	Impact of input activity variations on energy, power, and latency of inference using Spiker+. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	43
3.10	Impact of quantization of neuron membrane potentials and synaptic weights on inference accuracy and power consumption for target datasets. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	44

3.11	Impact of quantization of neuron membrane potentials and synaptic weights on inference accuracy and power consumption for target datasets. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	45
3.12	Impact of quantization of neuron membrane potentials and synaptic weights on inference accuracy and power consumption for target datasets. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.	46
4.1	SpikExplorer general architectures, including (i) a library of hardware neurons, (ii) a network evaluator estimating the performance of selected implementations, and (iii) a Bayesian Design Space Exploration (DSE) engine. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.	53
4.2	Metrics estimation: the figure graphically showcases how the different metrics considered by SpikExplorer are estimated. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.	55
4.3	Pareto frontiers of the global exploration on the MNIST dataset targeting power, area, and accuracy optimization. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.	64
4.4	Pareto frontiers of the global exploration on the SHD dataset targeting power, area, and accuracy optimization. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.	65
4.5	Pareto frontiers of the global exploration on the DVS dataset targeting power, area, and accuracy optimization. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.	65
4.6	Pareto frontiers of the exploration with top accuracy neuron model for each benchmark. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.	68
4.7	Pareto frontiers of the exploration with the number of neurons constrained to 200 for each benchmark. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.	69
5.1	Visual depiction of a generic SNN with Recurrent Neurons and Latent Replays. In a), our Recurrent Neuron model; in b), the structure of a generic Fully-Connected model for Continual Learning; in c), an example of the data fed to the Latent Replay stage. Reproduced from Dequino et al. 2024 [154]©2024 IEEE.	76
5.2	Example of a lossy compression (1) to store LRs. Compression ratio is here set to 4:1. Shrinking LRs and activations reduces the memory by 4×. A de-compression step (2) is required to respect the SNN time scale. Reproduced from Dequino et al. 2024 [154]©2024 IEEE. . .	78

5.3	Measurement of (a) forgetting and (b) Top-1 accuracy on the new scenario (Sample-Incremental CL). Latent Replays were applied to the 2 nd to last layer of the considered model using 2,560 past samples for the replay. Reproduced from Dequino et al. 2024 [154]©2024 IEEE.	80
5.4	Measurement of (a) forgetting and (b) Top-1 accuracy on the new class (Class-Incremental CL). Latent Replays were applied to the 2 nd to the last layer of the model. Reproduced from Dequino et al. 2024 [154]©2024 IEEE.	81
5.5	Top-1 accuracy on SHD’s test set in Sample and Class-Incremental CL, in case of a variable number of LRs (a) with respect to the epochs, (b) with respect to the LR index. Reproduced from Dequino et al. 2024 [154]©2024 IEEE.	82
5.6	Multi-Class-Incremental CL on SHD. Here, a pre-trained model learned to classify 10 more classes, starting from a pre-training on 10 classes. Reproduced from Dequino et al. 2024 [154] ©2024 IEEE.	84
6.1	Excitatory-Inhibitory balanced computational primitive	91
6.2	Monotonic Neural State Machine (NSM). The input signal is filtered through four 4 th order Butterworth bandpass filters. Each filtered component is converted into spikes through a Adaptive Exponential Integrate-and-Fire (AdExIF) neuron (blue). The input spikes are fed into populations of AdExIF neurons (blue), encoding different states of the network, interconnected in a Winner Takes All (WTA) configuration via a common inhibitory population (red). States are connected to gating populations (yellow) to implement the monotonic computation. These are organized in a Excitatory-Inhibitory (E-I)-balanced configuration with an inhibitory population (red) limiting the overall activity. Reproduced from Carpegna et al. 2024 [188] 2024 IEEE.	93
6.3	Stable sustained activity of one of the E-I-balanced primitives included in the networks. Reproduced from Carpegna et al. 2024 [188] ©2024 IEEE.	95
6.4	Network behavior when stimulated with 50Hz Poissonian sequences testing all the possible transitions: (a) non-monotonic WTA dynamics; (b) monotonic NSM response. Reproduced from Carpegna et al. 2024 [188] ©2024 IEEE.	95
6.5	Response of the network when stimulated with a real Electrocardiography (ECG) signal: (a) intense 10 minutes bike session; (b) 6 minutes and 40 seconds of walk. Reproduced from Carpegna et al. 2024 [188] ©2024 IEEE.	97

Chapter 1

Thesis goals and scope

During my Ph.D., my research has focused on neuromorphic computing, analyzing the field from multiple perspectives. This thesis offers an in-depth exploration of various aspects of neuromorphic computing, covering a broad range of topics, including bio-inspired algorithms, specialized hardware accelerators, optimization techniques, and continual learning strategies.

Through this work, I aim to demonstrate how biologically inspired computation can enhance efficiency in edge Artificial Intelligence (AI) applications, addressing the critical need for intelligent systems capable of operating autonomously and in real-time. By providing insights into the design and implementation of bio-inspired technologies, I hope to equip future researchers with the knowledge and tools necessary to effectively build and utilize these innovations, fostering advancements in the fields of AI and Neuromorphic Computing.

This Ph.D. thesis adheres to the Findability, Accessibility, Interoperability, and Reusability (FAIR) principles by making the developed code publicly available, ensuring that the research is transparent, reproducible, and accessible to the scientific community. You can find the links to all the open source repos related to this research in chapter 9.

1.1 The Role of Artificial Intelligence

AI is revolutionizing numerous industries, unlocking capabilities that were once unimaginable [1]. Its ability to process vast amounts of data, recognize patterns, and make intelligent decisions is driving innovation across a wide range of emerging fields. In healthcare [2], AI is enabling personalized medicine [3], where algorithms analyze genetic data to tailor treatments to individual patients, as well as AI-assisted diagnostics [4], where Deep Learning (DL) models detect diseases from medical images with accuracy comparable to human experts [5]. In autonomous systems, AI is at the core of technologies such as self-driving cars [6], drones [7],

and industrial robots [8]. In the case of self-driving cars, AI algorithms process data from sensors to map and understand the environment [9], make driving decisions [10], and ensure passenger safety. Similarly, drones powered by AI are being used in industries like agriculture [11], logistics, and environmental monitoring, performing tasks like surveillance, delivery [12], and infrastructure inspection with high precision and minimal human oversight [13]. In manufacturing, AI is transforming industrial robots by enabling them to work autonomously on tasks like assembly, welding, and packaging [14]. In space exploration, AI-powered rovers and satellites can autonomously analyze planetary surfaces, detect anomalies, and make decisions without human intervention [15]. Furthermore, AI is transforming sustainable energy, where intelligent algorithms optimize smart grids, predict energy demand, and enhance battery storage efficiency [16]. As AI continues to evolve, its impact on these emerging fields is expected to grow, driving breakthroughs in technology and redefining the future of innovation.

1.2 The need for local AI at the edge

When comparing state-of-the-art AI Deep Neural Networks (DNNs), such as Convolutional Neural Network (CNN) [17] or transformers [18], with the brain in terms of power consumption and computational efficiency, AI still lags by orders of magnitude [1]. While originally inspired by neural processes, modern AI models have evolved into highly sophisticated mathematical frameworks, typically optimized for specific tasks. And as their complexity grows, so does their demand for immense computational power [19].

Similarly to what happened with more conventional digital services, early AI research was constrained by limited computational resources and often relied on centralized mainframes. The DL revolution of the 2010s [20], fueled by massive datasets and Graphic Processing Unit (GPU) acceleration, reinforced the dominance of cloud-based AI, where large-scale models could be trained and deployed efficiently in powerful data centers. However, just as in conventional computing, AI is now shifting toward the edge [21]. Driven by the need for real-time inference, privacy-focused applications, and energy-efficient processing, AI models are increasingly being deployed directly on embedded devices such as smartphones and Internet of Things (IoT) sensors.

In many applications, local execution at the edge is not only desirable but essential, primarily due to the non-deterministic latency associated with internet-based communication. For instance, an autonomous vehicle must continuously recognize obstacles, detect pedestrians, and navigate through a dynamic environment where other humans and vehicles behave unpredictably [22]. If there is a delay in transmitting information between the car and a remote server, the consequences could be severe, including potentially fatal accidents. A similar challenge arises in space

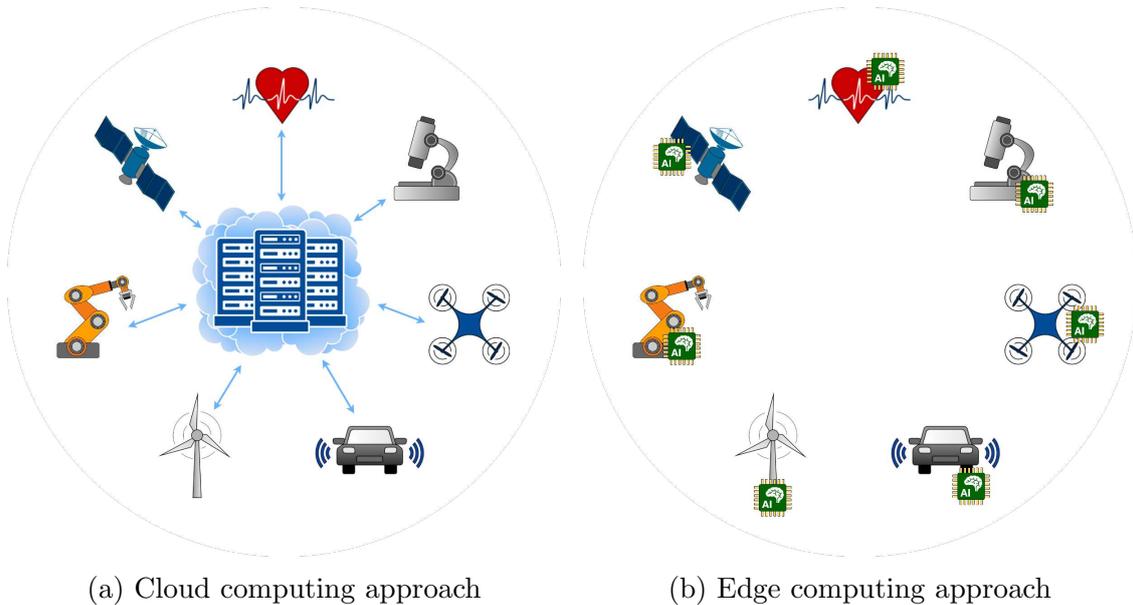


Figure 1.1: Comparison between a cloud computing paradigm (1.1a) and an edge computing approach (1.1b). In the cloud-based model, all devices transmit data over the internet to a centralized and powerful mainframe that carries out tasks such as inference using the Artificial Neural Network (ANN) model, subsequently sending the results back to the devices. Conversely, the edge computing model integrates an onboard electronic unit within each device to perform inference locally. This decentralized approach mitigates issues such as unpredictable latency, excessive power consumption, bandwidth overload, and concerns related to data security and privacy.

exploration: a planetary rover operating on a far distant planet cannot rely on real-time human guidance due to the significant communication delay. It must process sensor data locally and make autonomous decisions to avoid hazards. Wearable devices capable of real-time health monitoring—such as continuous glucose monitors for diabetes patients or Electrocardiography (ECG) sensors for detecting irregular heartbeats—must process data locally to immediately alert the user or medical personnel in case of an emergency [3]. Any delay caused by sending data to the cloud for processing could result in life-threatening consequences.

Security and privacy concerns also drive the need for edge AI [23]. Many applications involve processing sensitive data, such as facial recognition for authentication, biometric monitoring, or surveillance systems. If this data is transmitted to a remote cloud server, it could be intercepted, misused, or subjected to unauthorized access. By processing information locally, edge AI enhances privacy and reduces the risk of data breaches.

Despite its advantages, running AI models at the edge presents significant challenges. Unlike cloud servers, which benefit from virtually unlimited computational power, storage, and energy resources, edge devices are inherently constrained. Smartphones, IoT sensors, embedded systems, and autonomous robots must operate within strict limits on power consumption, memory, and processing capability. These constraints make it difficult to deploy large, compute-intensive DL models on edge devices. Furthermore, edge devices often have limited cooling and must balance AI processing with other system tasks. Finally, AI models are inherently data-driven, requiring extensive training before deployment.

As a result, AI generally follows a hybrid paradigm where training remains predominantly cloud-based, while inference is increasingly performed at the edge.

1.3 Deploying AI at the edge: taking inspiration from the brain

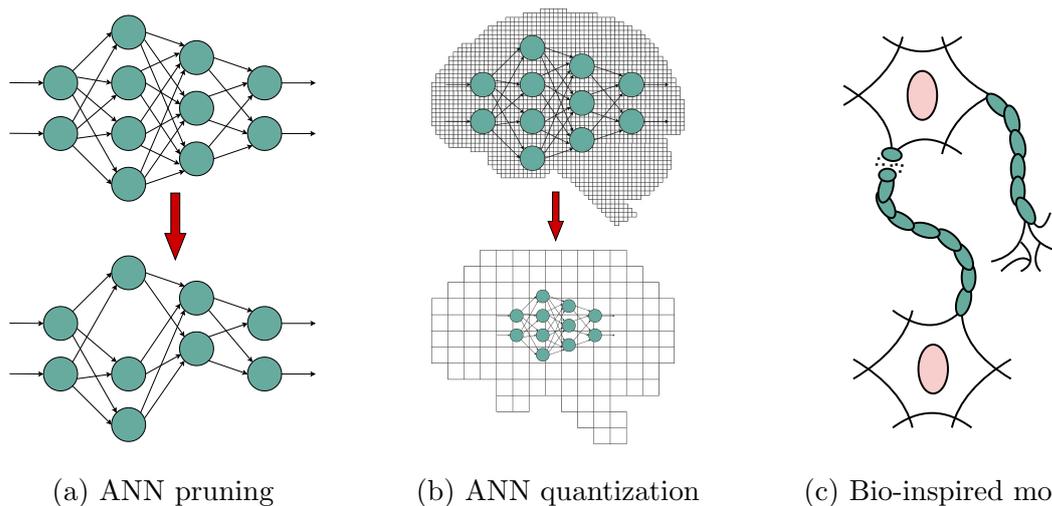


Figure 1.2: Methods to reduce computing, area and energy requirements of a ANN

To address these challenges, various techniques have been developed to optimize AI models, particularly ANNs, for edge deployment. Early research on optimal brain damage [24] revealed that many weights in a neural network contribute minimally to its final accuracy and can be removed without significant performance degradation. Model pruning (Figure 1.2a) builds on this concept by eliminating redundant or less significant parameters, thereby reducing both model size and computational complexity. Similarly, quantization (Figure 1.2b) replaces high-precision floating-point arithmetic (e.g., 32-bit) with lower-precision representations (e.g., 8-bit or even 4-bit integer operations), significantly decreasing memory usage and

computational cost while maintaining accuracy [25].

Fields like TinyML [26] and Frugal AI [27] leverages these techniques, along with other architectural optimizations, to develop ultra-low-power AI models capable of running on microcontrollers with minimal resources. Additionally, techniques such as knowledge distillation [28] enable a smaller and more efficient *student* model to learn from a larger *teacher* model, achieving comparable accuracy with significantly reduced complexity.

These and many other approaches are gaining increasing interest within the research community to try to bring all the advantage of AI on resource-constrained edge devices.

However, the most compelling inspiration for achieving efficient computation comes from nature itself: the brain. Over millions of years, evolution has fine-tuned biological computation to maximize survival while minimizing energy use. Neuromorphic computing [29] seeks to replicate this biological approach to computation (Figure 1.2c), striving to achieve similar levels of efficiency [30].

The main focus of this thesis is this last approach, with a particular emphasis on Spiking Neural Networks (SNNs).

1.4 Research contributions

This thesis explores the field of Neuromorphic Engineering and SNNs from different perspectives.

From a computational standpoint, our brain operates differently from the classical Von Neumann architecture [31] used in conventional computing systems. The brain consists of numerous small computational units, the neurons, interconnected in highly complex patterns and processing information concurrently. Moreover, memory in the brain is distributed throughout the system, primarily stored within synapses that connect neurons, according to current understanding. In contrast, a Von Neumann machine relies on one or a few highly powerful computing units, known as Central Processing Units (CPUs), with a centralized memory from which both instructions and data are retrieved. The first part of this thesis will focus on the design of specialized hardware processors, tailored to efficiently execute SNN models. chapter 3 presents the core of my Ph.D. research, which focused on creating a high-level Python framework to design, train, and deploy hardware SNNs. The core technology explored in this phase is the Field Programmable Gate Array (FPGA), a versatile platform that enables the creation of flexible, reconfigurable devices capable of targeting different applications. The goal is to provide a tool that allows users to create their own SNN implementations on an FPGA without requiring field-specific advanced skills. However, the vast number of configuration options available for such systems presents a challenge. As a second step, chapter 4 investigates the application of automatic optimization techniques to search

for the most efficient architecture. This multi-objective optimization takes into account both model performance metrics, such as accuracy, and hardware-specific parameters, such as area, latency, and power consumption. It enables users to specify target problems and acceptable parameter ranges for the model, allowing for the automatic tailoring of the SNN. Together with chapter 3, this approach aims to provide tools for automatically generating hardware-efficient architectures for SNNs.

As mentioned earlier, the brain has the remarkable ability to evolve and improve itself through experience. Similarly, training ANNs and SNNs typically occurs on high-performance centralized computers, and once deployed, the network remains fixed, primarily used for inference. However, akin to the brain’s capability, many applications require the network to continue learning from new incoming data, allowing it to specialize or adapt to recognize previously unseen information. This concept is the focus of chapter 5, which delves into the field of continual learning [32] applied to SNNs. Supervised learning techniques are adapted and applied to SNNs, exploring methods to enable networks to learn from new data without forgetting previously learned information. A particular emphasis is placed on minimizing the computational and memory requirements necessary for continual learning, ensuring that SNNs can evolve in real-time while maintaining previously acquired knowledge. This chapter aims to provide solutions for enhancing the learning capabilities of SNNs, making them more adaptable to dynamic environments, just as the biological brain does.

Finally, while the approach outlined so far leverages mature technology with well-established advantages, such as FPGAs, it remains distant from the real behavior of biological brains. The next step involves moving beyond simply optimizing the execution of mathematical models of biological primitives and instead transitioning to a computational substrate closer to the biological target. This is the fundamental concept of neuromorphic engineering [29]. The aim is to shift computations to the analog domain [33], despite the inherent challenges of analog computing—such as noise, production mismatches causing variability, and other constraints akin to those observed in the brain, which is inherently an analog computer. chapter 6 explores the development of computational primitives using transistors operating in the sub-threshold regime as the target technology. This approach significantly reduces power consumption, bringing it closer to biological levels, albeit at the cost of lower robustness. Designing such systems requires addressing the same limitations that biological evolution has confronted. The ultimate objective is to understand how nature has successfully adapted to these constraints and to translate these insights into hardware design, even though we are still far from achieving the complexity and efficiency of biological computation.

Chapter 2

Background

2.1 An historical perspective: drawing inspiration from the human brain

The evolution of our understanding of biological neurons and their mathematical modeling is marked by transformative breakthroughs over the centuries [34]. In the 18th century, Luigi Galvani and Lucia Galeazzi Galvani discovered that muscle contractions are triggered by electrical impulses rather than fluid movements [35], challenging the then-prevailing *balloonist theory* [36]. Their work introduced the idea of animal electricity. During the 19th century, Carlo Matteucci, starting from the Galvani's work, demonstrated that nervous cells produce direct current [37], setting the stage for Emil Du Bois-Reymond's discovery of action potentials [38] and Hermann von Helmholtz's measurements of their propagation speed [39]. Initially, proponents like Joseph von Gerlach and Camillo Golgi supported the reticular theory, which viewed the nervous system as a continuous network [40]. However, Santiago Ramón y Cajal, using Golgi's staining method, showed that neurons are discrete cells [41], a finding later cemented by Heinrich Wilhelm von Waldeyer-Hartz, who introduced the term *neuron* in 1891 [42]. Julius Bernstein further contributed by developing the membrane theory of electrical potentials, bridging neuroscience with biophysics [43].

The modern understanding of neural excitability took a giant leap with Alan Hodgkin and Andrew Huxley's work on the giant squid axon [44]. Their mathematical model, based on differential equations, described how voltage-gated sodium (Na^+) and potassium (K^+) channels govern the generation and propagation of action potentials. This model not only provided critical insights into neuronal behavior but also laid the groundwork for computational neuroscience. Building on these discoveries, the mid-20th century witnessed significant strides in understanding neural networks. Early models by McCulloch and Pitts (1943) demonstrated that neurons could perform logical operations [45], while Donald Hebb's (1949)

principles of synaptic plasticity became foundational [46]. Research by Hubel and Wiesel in 1962 revealed the visual cortex’s role in edge detection [47], and David Marr’s work between 1969 and 1982 offered theories on memory, vision, and motor control [48]. These studies led to the development of spiking neuron models, such as the Leaky Integrate and Fire (LIF) model [49], and later, other simplified yet effective models like FitzHugh-Nagumo [50] and Izhikevich [51], which balanced computational efficiency with biological realism. Together, these models enabled large-scale simulations of neural networks, deepening the interplay between neuroscience and AI. In the 1980s, Carver Mead pioneered Neuromorphic Engineering, aiming to build electronic circuits that mimic the brain’s processing [29]. His innovations in sub-threshold Complementary Metal Oxide Semiconductor (CMOS) circuits and analog Very Large Scale Integration (VLSI) for sensory processing introduced event-driven computation, echoing the spike-based communication of neurons. Today, neuromorphic computing has reached new heights with processors designed to implement SNNs directly in hardware. Recently, major technology companies have shown increasing interest in this field. Examples include International Business Machines (IBM) Corporation’s TrueNorth [52], Intel Loihi [53] and platforms like Heidelberg University’s BrainScaleS [54] and Manchester University’s SpiNNaker [55], which enable ultra-fast, massively parallel brain simulations.

2.2 A computational perspective: spikes processing

Throughout evolution, biological systems have been shaped by the need for efficient, robust, and low-energy information processing. One of the most striking outcomes of this optimization is the use of spikes, discrete, all-or-nothing electrical impulses, as the fundamental unit of communication in neural networks [56]. Unlike continuous signals, which suffer from attenuation over distance and are susceptible to noise, spikes are propagated actively and reliably across long axons without loss of information. This biological form of digitization ensures that neural signals remain distinguishable, even in complex and dynamic environments, much like how digital communication systems maintain signal integrity over noisy channels [57]. However, the true computational power of spiking neurons does not lie in isolated spikes but in the precise timing and patterns of spike sequences. The brain does not simply encode information in the rate of firing but also in the relative timing of spikes between neurons. This temporal coding mechanism allows neural networks to process information with high efficiency, enabling rapid decision-making, sensory processing, and complex pattern recognition. Sequences of spikes act as dynamic signals that can represent temporal correlations in data, store information in recurrent loops, and even perform computations similar to digital logic circuits. For example, in sensory systems, time-to-first-spike encoding allows neurons to rapidly

detect changes in stimuli, optimizing reaction speed while minimizing energy use.

By leveraging these spike-based representations, biological neurons achieve an optimal balance between computational power and metabolic cost. Inspired by these principles, SNNs [58] seek to replicate the efficiency of biological computation. Unlike traditional ANN, which rely on continuous activations and dense matrix operations, SNNs process information asynchronously through spike timing and event-driven updates [59]. This allows neuromorphic systems to achieve significant power savings while maintaining computational expressiveness, making them ideal for edge computing and low-power applications. In both natural and artificial intelligence, spike-based computation represents a fundamental shift towards energy-efficient, noise-resilient, and highly dynamic information processing.

2.3 Approximating neural computation: the birth of mathematical neuron models

In the early stages of AI, the focus was on developing mathematical models that could approximate the fundamental functions of biological neurons, without the complexity of spike-based communication. Numerous studies conducted over the decades on these simplified models laid the foundation for much of today's knowledge in AI. Many of these early concepts were later adapted to SNNs, making it worthwhile to briefly revisit them. Pioneers such as Warren McCulloch and Walter Pitts proposed a simple mathematical model of neurons, conceptualizing them as binary units capable of processing information [45]. The McCulloch-Pitts (MCP) neuron represents an idealized binary threshold unit, and while it can classify linearly separable problems, it cannot handle non-linear decision boundaries or more complex issues without a learning mechanism. This limitation restricts its practical application in real-world problems. In 1958, Frank Rosenblatt introduced the perceptron, designed for binary classification tasks [60]. It was celebrated as a breakthrough in Machine Learning (ML): the perceptron can solve linearly separable problems where data can be divided by a single hyperplane. It uses the perceptron learning rule to iteratively adjust its weights until convergence, forming the foundation for modern ANNs.

From a mathematical perspective, the perceptron, and most neuron models derived from it, comprises two fundamental steps, as shown in Figure 2.1: integration and a non-linear function, called the activation function. The integration step is where the artificial neuron aggregates its inputs, calculating a weighted sum to represent the total signal received. This linear operation acts as a feature extraction mechanism, combining multiple input signals into a single value. Following integration, the neuron applies a nonlinear activation function, denoted as $f(S)$, to the summed input. This step is crucial for two main reasons:

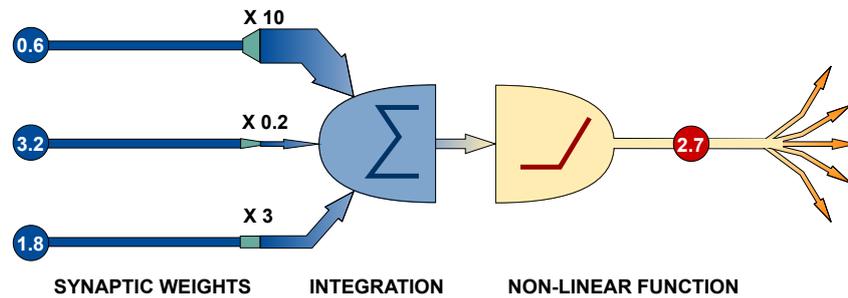


Figure 2.1: Artificial neuron model

1. It allows the network to approximate non-linear functions, which is necessary for solving complex problems, such as image recognition and language modeling.
2. It prevents the entire network from collapsing into a single linear transformation, thereby maintaining the effectiveness of deeper layers.

Common non-linear activation functions include:

- Rectified Linear Unit (ReLU) [61]: used in CNNs and deep networks, enabling sparsity and mitigating the vanishing gradient problem.
- Sigmoid: used in older networks and for generating probability-like outputs, such as in binary classification.
- Tanh: similar to the sigmoid but centered around zero, promoting balanced gradients.

2.4 From Neurons to Networks: The Universal Approximation Theorem

Despite an artificial neuron's ability to solve non-linear problems, its computational power is limited. A single neuron can only handle a small amount of information due to its simple architecture, which prevents it from processing complex, high-dimensional data and capturing intricate patterns across a broad range of tasks. By combining neurons into networks, they can operate in parallel, processing large datasets, performing distributed computations, and capturing hierarchical patterns. A pivotal concept in ANNs theory is the Universal Approximation Theorem (UAT) [62], which asserts that a Feed-Forward Neural Network (FFNN) with a single hidden layer, sufficient neurons, and a non-linear activation function can approximate any continuous function on a closed, bounded subset of real numbers

to an arbitrary degree of accuracy, provided enough neurons are used. This means that even a relatively simple network can theoretically model a wide variety of behaviors and patterns. As previously noted, the activation functions in the hidden layers must be non-linear to enable the network to model complex functions. Notably, the UAT holds true even with just one hidden layer, though a large number of neurons may be needed. However, while the UAT ensures that a Neural Network (NN) can approximate any continuous function, it does not address the efficiency of such an approximation—such as the optimal number of neurons or the training time required. In practice, deeper networks with multiple layers and specialized architectures, like convolutional layers for image processing, are often employed [63]. These architectures can learn hierarchical features and generalize more effectively than shallow networks.

2.5 Networks of spiking neurons

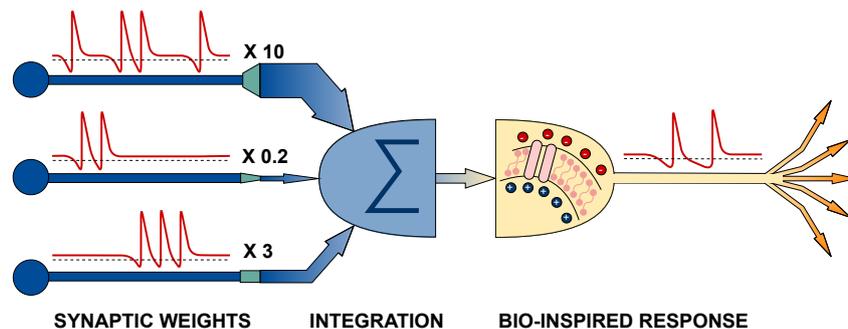


Figure 2.2: Spiking neuron model

Having established the foundational theory of ANNs computation, attention can now be directed back to SNNs [58]. From a mathematical standpoint, spikes are often treated as all-or-nothing events. This approach allows them to be represented as binary values, ignoring the shape of the spikes themselves. While this remains a debated topic in the research community, it is widely accepted as the standard theory and provides a useful starting point for bridging the computational frameworks of ANNs with the brain’s computing paradigm.

Similar to classical neuron models, spiking neurons rely on two fundamental operations: input integration and a non-linear response. However, unlike the models discussed in section 2.3, the non-linearity in spiking neurons is replaced by a biologically inspired membrane model, as shown in Figure 2.2

Representing information in binary form brings about two primary effects: first, the integration step is simplified; in this model, weighting inputs becomes a binary multiplication, where the synaptic weight is considered only when a spike is present

(multiplied by 1), and disregarded when no spike occurs (multiplied by 0). The second advantage is a shift in complexity from space to time: rather than using high-precision multi-bit numbers to encode information, the sequence of spikes itself carries the information. The timing and order of the spikes become the crucial element. As a result, SNNs are particularly well-suited for processing data with inherent temporal information. To further emphasize temporal correlations, recurrent architectures can be employed.

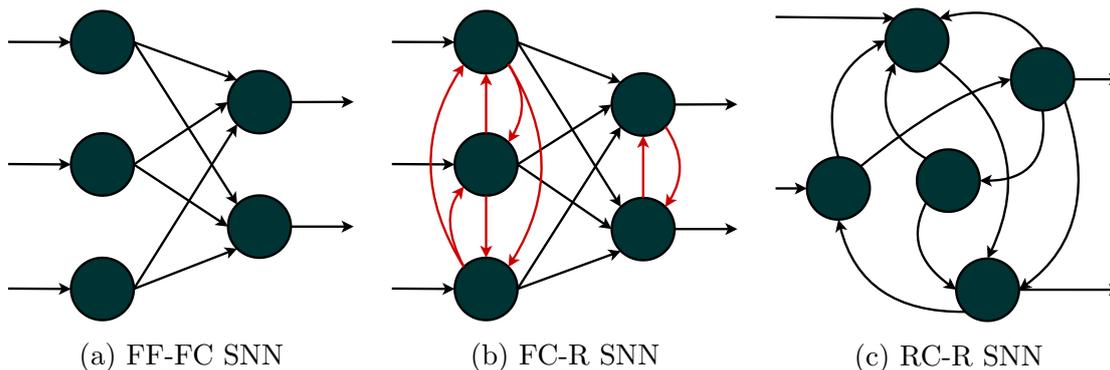


Figure 2.3: Typical architectures of an SNN. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

Figures 2.3a, 2.3b, and 2.3c illustrate three potential SNN architectures. Figure 2.3a displays a Feed-Forward Fully-Connected (FF-FC) architecture, capable of modeling intricate relationships between inputs and outputs. Figure 2.3b introduces recurrent connections between neurons within the same layer. These recurrent links enhance the temporal aspect of SNNs, enabling neurons to modify their outputs based on previous states, which improves the network’s ability to comprehend context and predict future events based on prior sequences. Figure 2.3c shows a randomly connected structure, commonly used in reservoir computing and similar applications. This approach relies on random connections to capture dynamic temporal patterns, which can be particularly powerful in certain types of learning tasks.

2.6 Overview of mathematical models of spiking neurons

Over the past decades, numerous computational neuron models have emerged, derived from electrical conductance measurements and mathematical formulations. The Hodgkin-Huxley (H-H) model [44], based on squid giant axon experiments, offers a detailed description of neuronal dynamics using four coupled differential equations. It models the membrane potential in terms of ion channel conductances,

which are regulated by voltage-dependent gating variables. Despite its biological realism, the H-H model's computational complexity has driven the development of simpler alternatives. The FitzHugh-Nagumo model [50] simplifies this further, reducing the system to two equations: one for membrane potential and another for a slow recovery variable, capturing excitability dynamics with cubic nonlinearities. The LIF model, first proposed by Louis Lapicque in 1907 [65], models neuronal activity with a single first-order differential equation. The LIF model treats the membrane as an electrical circuit with a capacitor and resistor, producing an exponential decay response to an input current. When the potential reaches a threshold, it resets, mimicking the spiking behavior of real neurons in a computationally efficient way. The Adaptive Exponential Integrate-and-Fire (AdExIF) model [66], introduced by Brette and Gerstner in 2005, enhances the LIF model by incorporating an exponential term for spike generation, allowing for a smoother and more biologically realistic spike onset. Additionally, it includes a second equation for adaptation currents, which accumulate with each spike and reduce firing probability, reproducing behaviors such as spike-frequency adaptation and bursting. Several other LIF-based models offer further refinements while maintaining computational efficiency [67]. The Quadratic Integrate and Fire (QIF) model smooths spike initiation with a quadratic membrane potential function, bridging the LIF model and more complex models. The Exponential Integrate and Fire (EIF) model, a precursor to AdExIF, replaces the LIF's hard threshold with an exponential voltage-dependent activation term, improving spike onset realism. The Conductance-Based LIF model introduces synaptic conductance dynamics, making it more suitable for network simulations. The Resonator Integrate and Fire (RIF) model extends the LIF framework by adding subthreshold oscillations, capturing resonant behaviors seen in certain cortical and thalamic neurons. The Generalized Leaky Integrate and Fire (GLIF) framework [49] systematically extends the LIF model by incorporating the above mentioned biological features, such as spike-triggered adaptation, voltage-dependent thresholds, and nonlinear dynamics. The GLIF hierarchy ranges from GLIF1, which behaves like the basic LIF model, to GLIF5, which introduces non-linear membrane properties for more accurate subthreshold dynamics and spike generation. These models have become essential for neuromorphic computing and large-scale brain simulations, such as those conducted by the Allen Institute for Brain Science. Finally, to balance computational efficiency with biological realism, Izhikevich's model [51] combines a QIF system with an adaptive recovery variable. With just two equations, it replicates a wide range of spiking behaviors, including regular spiking, fast spiking, bursting, and resonant dynamics.

These models form the foundation of large-scale neural simulations and neuromorphic computing, each offering varying trade-offs between biological plausibility and computational efficiency.

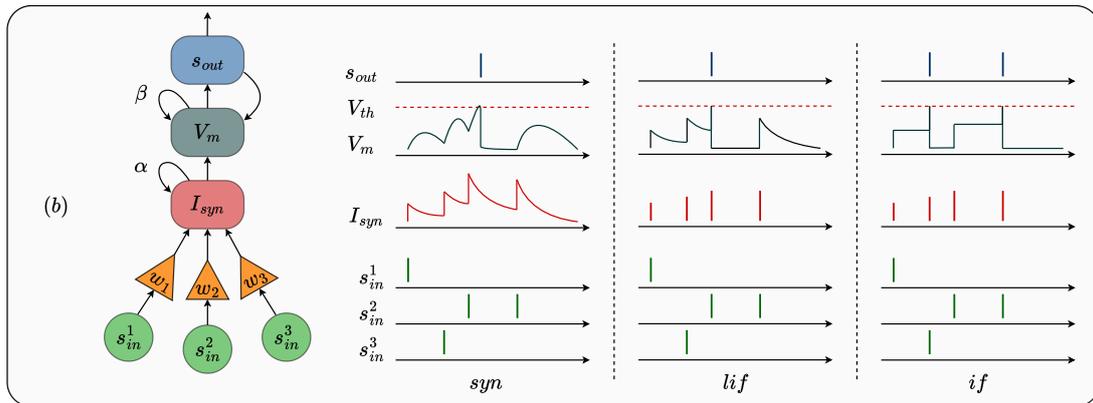


Figure 2.4: In the *syn* model, synapses exhibit a dynamic response characterized by a decay parameter α , which governs how incoming spikes are weighted and how the synaptic current decays over time in the absence of stimulation. The resulting synaptic currents are then integrated by the membrane potential, which itself follows a second dynamic governed by a time constant β . In the *lif* model, the temporal dynamics of the synapses are disregarded; each synapse simply scales the input spike by a fixed weight, and only the membrane exhibits leaky integration. Finally, in the simplest *if* model, even the membrane’s decay is omitted: input spikes are accumulated over time without any leak or temporal dynamics.

2.7 The Leaky Integrate and Fire model

Among the many available neuron models, the LIF model stands out as an optimal balance between complexity and efficiency, particularly when the goal is not to strictly mimic biological neurons. In practical applications such as pattern recognition, data classification, and function approximation, common tasks for ANNs, LIF models are well-suited to capture temporal dependencies and integrate input information effectively. Their simplicity also makes them computationally efficient, allowing them to be executed with limited processing power or on specialized hardware.

A LIF neuron is a recurrent neuron, meaning its output depends not only on the current inputs but also on its previous state, which is represented by the membrane potential. In biological neurons, the membrane selectively allows certain ions to pass, creating a charge difference between the input and output of the cell, which gives rise to the membrane potential. Physically, the membrane can be modeled as a capacitor-resistor pair: the capacitor accumulates charge in response to an input current, and if no input is present, it gradually discharges through the resistor, returning to the resting potential. For biological cells, this resting potential is typically around

$$V_{rest} = -70mV \quad (2.1)$$

However, for simplicity, this thesis sets the resting potential at 0V for all computations, which streamlines the calculations without affecting the model’s mathematical properties.

The LIF model discussed in this thesis represents a family of neuron models with varying levels of simplification.

Equation 2.2 introduces the discrete-time formulation of a Synaptic Conductance-based II-order LIF model [66], which is the most comprehensive model explored in this work, and will be called *syn* model along the thesis.

Operating in discrete time enables the iterative solution of the differential equations governing the temporal evolution of the membrane potential (V_m). In this model, input spikes (s_{in}) are integrated by synapses with conductance weights (W), influencing the membrane potential based on the input’s significance ($W \cdot s_{in}$). The integrated spikes form the synaptic current (I_{syn}), which undergoes capacitive discharge ($\alpha \cdot I_{syn}$, with $0 < \alpha < 1$). The membrane integrates this current, resulting in increased or decreased potential based on the current sign. The current sign is influenced by the excitatory or inhibitory nature of the input spikes, impacting the neuron’s firing probability. This effect can be positive or negative. The capacitive component (β) discharges the membrane toward a resting state in the absence of stimuli ($\beta \cdot V_m$, with $0 < \beta < 1$). Lastly, the reset parameter (r) models the reset process, as explained later in this section.

$$\begin{cases} I_{syn}[n] = \alpha \cdot I_{syn}[n - 1] + W \cdot s_{in}[n] \\ V_m[n] = (\beta \cdot V_m[n - 1] + I_{syn}[n - 1]) \cdot r \end{cases} \quad (2.2)$$

An action potential, represented by a separate variable s_{out} , occurs when the membrane voltage (V_m) surpasses a threshold value (V_{th}). This dual-variable approach (membrane and action potential), simplifies the description by treating the spike as a binary variable. Equation 2.3 illustrates the relationship between the output spike and membrane potential V_m .

$$s_{out}[n] = \begin{cases} 1, & \text{if } V_m > V_{th} \\ 0, & \text{if } V_m \leq V_{th} \end{cases} \quad (2.3)$$

Finally, to model the complete discharge of the membrane when the neuron fires, a reset term is used (r in Equation 2.2). There are different alternatives to applying the reset: a *hard-reset*, shown in Equation 2.4, in which the membrane is instantly brought to zero when a spike is generated, and a *subtractive-reset*, detailed in Equation 2.5. In the latter, the threshold value is subtracted by the membrane potential.

$$r = 1 - s_{out}[n - 1] \quad (2.4)$$

$$r = 1 - \frac{V_{th}}{\beta \cdot V_m[n - 1] + I_{syn}[n - 1]} \quad (2.5)$$

The LIF model, being a family of models, allows for various simplifications to derive different LIF descriptions. For instance, setting $\alpha = 0$ transforms the II-order model in Equation 2.2 into a I-order LIF, where input spikes directly influence the membrane potential. This will be called *lif* along the thesis. Additionally, with $\beta = 1$, a basic Integrate and Fire (IF) model is obtained, maintaining a constant membrane value without input spikes. This will be referenced to as *if* model in the next chapters.

These different models serve diverse tasks. The II-order LIF excels in handling input sequences with high temporal information content, capturing longer correlations in precise spike sequences. On the other hand, I-Order LIF and IF models are simpler and preferred for processing static data converted into spikes, e.g., images. Working solely with LIF offers six distinct models: II- and I-order LIF, and IF, each with a hard or subtractive reset. These models, combined with the architectures in section 2.5, address various classification and regression problems, making them a robust basis for solving ML tasks with SNNs. In the next chapters the presence of recurrent connections between neurons will be identified with a prepended r (*rsyn*, *rlif*, *rif*).

2.8 Encoding information into sequences of spikes

A critical consideration when working with SNNs is that they communicate in a distinct manner compared to most other digital systems, sensors, storage, and computing platforms. Specifically, SNNs process and interpret sequences of binary spikes, providing their outputs in the same spike-based format [68]. In contrast, data are typically sampled, recorded, and stored in numerical forms. Consequently, for SNNs to interact with systems that do not natively generate spikes, they require an encoding step to convert numerical data into spike sequences. Various encoding techniques have been developed to achieve this transformation [69].

One of the most widely adopted methods is rate coding, which translates the intensity of an input signal into the instantaneous firing rate of a spike train. In biology, this approach is commonly seen in tactile sensing, where increased pressure corresponds to higher spike rates [70]. These spike sequences generally follow a Poisson distribution [71], meaning there is a stochastic component to the spike timing, but the average firing rate correlates with the input intensity. Rate encoding is particularly useful for training SNNs on static data, such as images, by approximating the continuous input values using spike trains. Although SNNs are

inherently recurrent and well-suited for temporal data, rate coding extends their usability to static applications, making it a versatile method. Rate coding will be employed throughout this thesis for benchmarking purposes.

A more efficient but less robust method involves encoding the input data in the precise timing of individual spikes [72]. This is exemplified by time-to-first-spike encoding [73], where higher input values correspond to earlier spike timings. This approach significantly reduces the number of spikes compared to rate encoding, as it encodes information with a single spike. However, it is more sensitive to noise; if the spike is delayed, lost, or shifted in time, the encoded information can be entirely corrupted. As such, while this method is efficient, it is prone to errors in noisy environments, where temporal precision is difficult to maintain.

Several additional techniques exist for encoding temporal data into spike sequences, exploiting the intrinsic properties of SNNs [74]. These methods include sigma-delta converters, adaptive delta modulation, simple thresholding of temporal sequences, and trainable encoding. In trainable encoding, input data are used as an input current to a set of neurons, whose spiking activity then encodes the data. The encoding process itself can be optimized during training, allowing the network to learn the best way to represent the data with spikes.

Finally, encoding can also occur at the population level [75]. In population coding, multiple neurons are activated concurrently to represent a specific pattern or situation. For example, in a navigation task, if a stimulus originates from the right, a specific population of neurons may fire, triggering a corresponding turn in that direction. Conversely, if the stimulus comes from the left, a different population may respond, causing the system to steer accordingly. This method allows SNNs to efficiently represent complex patterns by leveraging the collective activity of multiple neurons.

While encoding is not the primary focus of this thesis, as it warrants its own dedicated line of research, several of the aforementioned encoding methods will be utilized to benchmark and evaluate the solutions developed within this work.

2.9 Training

The real power of ANNs and SNNs lies in their ability to be automatically optimized to perform a specific task, an ability that mirrors the learning process in biological brains. This process, known in the realm of AI as training, is a crucial step in making SNNs work. Although a full treatment of training methods would require a dedicated discussion, the following text provides an overview of the key concepts.

Historically [34], the idea that learning occurs through changes in neural connections has its roots in the work of early researchers such as Alexander Bain [76] and William James [77]. Their ideas were later formalized by Donald Hebb [46],

who proposed that repeated neural activity strengthens the synaptic connections between neurons, a concept famously summarized by Carla Shatz as “cells that fire together wire together.” [78] These early insights laid the groundwork for understanding how synaptic modifications can encode memory. Subsequent contributions by researchers like Jerzy Konorski introduced the concept of inhibitory connections [79], and the development of the Bienenstock–Cooper–Munro (BCM) learning rule [80] in 1970 added the important notion of frequency-dependent synaptic modifications, including Long Term Potentiation (LTP) and Long Term Depression (LTD). More recently, experiments by Henry Markram in 1995 provided evidence for the role of precise spike timing in synaptic plasticity [81], leading to the formulation of Spike-Timing-Dependent Plasticity (STDP). Despite these groundbreaking findings, biologically inspired learning methods have not achieved the same level of performance as traditional supervised techniques in engineered systems.

In contrast, supervised learning has become the dominant paradigm for training ANNs, largely due to the effectiveness of the backpropagation algorithm [82]. Backpropagation, popularized by the work of Rumelhart, Hinton, and Williams in 1986, uses the chain rule of differentiation to propagate errors from the output layer back through the network, allowing for efficient weight adjustments. This method has proven remarkably successful and remains a cornerstone of modern NN training. For sequential data, the technique of Back-Propagation Through Time (BPTT) was developed [83]. BPTT works by unrolling a recurrent network over multiple time steps and applying backpropagation over this extended network structure, thereby enabling the training of models that capture temporal dependencies. Despite challenges such as vanishing and exploding gradients, advances like Long Short Term Memory (LSTM) [83] networks and Gated Recurrent Units (GRU) [84] have enhanced the ability of recurrent networks to learn long-term dependencies.

When it comes to training SNNs, the task becomes more challenging due to the inherent non-differentiability of the spike function. One common strategy is the so-called ANN-to-SNN method. In this approach, an ANN with an architecture identical to that of the target SNN is first trained using standard supervised methods, and the learned weights are then transferred to the SNN. Although this conversion enables the use of highly effective supervised techniques, it is often accompanied by a reduction in accuracy due to the differences between continuous activations in ANNs and the discrete spiking behavior in SNNs.

A more direct approach to training SNNs involves replacing the non-differentiable spike function with a differentiable surrogate function during the backward pass [85]. This substitution allows the use of BPTT directly on the SNN, effectively bridging the gap between the spike-based operation of the network and the gradient-based optimization used in supervised learning. Common surrogate functions include smoothed versions of the step function, such as sigmoid variants and hyperbolic tangent functions, which approximate the gradient well enough to allow effective training. As illustrated in Figure 2.5, these surrogate gradients are used

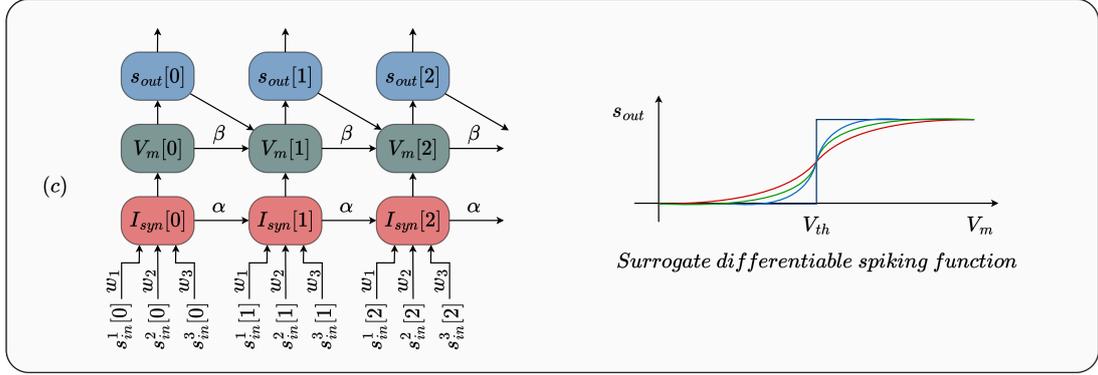


Figure 2.5: Example of unrolling of the network in BPTT (left), with surrogate gradient function for the spiking activation (right).

to update the network’s weights during the backward pass, thereby enabling the supervised training of SNNs.

Since the focus of this thesis is on the practical applications of SNNs, supervised methods, and in particular BPTT with Surrogate Gradient (SG), are the primary emphasis. This approach leverages the mature and powerful optimization techniques developed for ANNs while adapting them to the unique requirements of spike-based computation. As a result, despite the challenges presented by the non-differentiability of spikes, supervised training remains the most effective method for optimizing the performance of SNNs in practical applications.

Moreover, several frameworks support supervised training of SNNs. This thesis primarily employs `snnTorch` [86], an open-source library built on PyTorch, which facilitates efficient training and accelerates computations using GPU.

Chapter 3

Spiker+: a framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge

Adapted, with permission, from Alessio Carpegna; Alessandro Savino; Stefano Di Carlo, "Spiker+: a framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge," in IEEE Transactions on Emerging Topics in Computing, 2024, doi: 10.1109/TETC.2024.3511676. Licensed under CC-BY.

3.1 Why accelerating Spiking Neural Networks

One of the main challenges in neuromorphic computing is the limited understanding of how biological substrates perform the complex computations underlying neural network function. While ongoing research continues to uncover these mechanisms, the past 70 years have seen remarkable progress in computational models using non-biological substrates, particularly silicon. As a practical starting point, leveraging conventional silicon-based technologies provides an effective approach to implementing brain-inspired computation. The term neuromorphic engineering originally referred to analog silicon circuits designed to mimic biological processes. Over time, its definition has expanded to include digital implementations [86]. In this thesis, neuromorphic encompasses brain-inspired algorithms and architectures that enable efficient computation within constrained power budgets.

Neuromorphic principles can be explored at various levels, with digital electronics being the most mature and widely adopted. However, the brain’s computational model differs significantly from the Von Neumann architecture that underpins traditional microprocessor-based systems. While conventional architectures rely on a few powerful processing cores with separate memory units, the brain consists of numerous small, parallel computing units, the neurons, with localized memory [87]. To replicate this efficiency, specialized hardware accelerators are essential [88]. General-purpose accelerators such as GPUs and Tensor Processing Units (TPUs) have been widely used for DL tasks. However, accurately emulating biological computation requires considering the brain’s dynamic nature, where neurons communicate through asynchronous, all-or-nothing spike events, as discussed in previous chapters. Conventional GPUs are not optimized for accelerating SNNs, making dedicated neuromorphic hardware a necessity. The choice of hardware generally falls into two categories: fixed or reprogrammable. Fixed hardware, such as Application-Specific Integrated Circuits (ASICs), provides optimal performance but comes with high development costs and long design cycles. In contrast, reprogrammable solutions, such as FPGAs, offer greater flexibility at the expense of some performance trade-offs. Many embedded systems already incorporate FPGAs, making them a practical choice for embedded and edge neuromorphic applications.

3.2 Spiker+ overview

The flexibility of specialized hardware design poses a challenge, with applications often requiring diverse network architectures, encoding methods, and neuron models. While awaiting the maturity of SNN accelerators based on emerging technologies, existing literature proposes various digital hardware solutions (refer to section 3.3). Unfortunately, these solutions often constrain network topology to circuit architecture, limiting exploration of the broader design space, whether considering ASICs or FPGAs. We propose an alternative strategy, optimizing the network architecture for specific needs and leveraging FPGAs for deploying custom hardware blocks. This approach enables efficient and low-power SNN inference engines at the edge, supporting real-time data processing. FPGAs provide high parallelism and reconfigurability, making them ideal for accelerating neural network computations with minimal latency.

To support this, this chapter presents Spiker+, a complete framework for generating efficient low-power and low-area customized SNN accelerators on FPGAs for inference at the edge. Spiker+ introduces several pivotal contributions. At its core, it provides a fully configurable multi-layer hardware architecture implementing both fully connected and recurrent SNNs. This architecture is a significant step from the preliminary Spiker model, initially presented in [89]. It introduces a library of highly efficient architectures delving into a range of approximation

techniques to implement remarkably low-area and low-power neurons, thus optimizing resource utilization while maintaining high performance. Notably, Spiker+ brings a complete design framework to the forefront, a comprehensive toolkit for developing complex SNNs accelerators. This framework empowers researchers and developers to describe target network architectures with great flexibility, enabling the specification of layers, neuron types, and input encoding techniques using a few lines of Python code. Integrating sophisticated off-line server-based training algorithms like BPTT [85] ensures the network has cutting-edge learning capabilities. Additionally, Spiker+ emphasizes the significance of optimizing networks through quantization techniques, reducing complexity while judiciously balancing approximation and accuracy. Finally, the framework seamlessly generates a VHDL model of the accelerator, primed for deployment on Xilinx™ FPGA boards. These contributions make Spiker+ a robust solution in the hardware-accelerated SNN landscape. Spiker+ has been tested on the well-known MNIST dataset [61] and compared to state-of-the-art SNN accelerators for FPGAs, demonstrating superior performance. Moreover, two accelerators for the Spiking Heidelberg Dataset (SHD) [90] and AudioMNIST [91] have been generated and evaluated to illustrate the framework’s flexibility when handling different problems. The primary aim of Spiker+ is to offer an Electronic Design Automation (EDA) framework that simplifies the design of SNN accelerators for FPGA, addressing a gap that is still underrepresented in the literature. The tool does not include features for searching optimal architectures or optimizing hyper-parameters. However, it seamlessly integrates with existing tools such as SpikExplorer [92], able to find an optimal network configuration, which can then be used by Spiker+ to generate the corresponding hardware accelerator.

The rest of the chapter is organized as follows: section 3.3 reviews relevant literature on accelerating SNNs. section 3.4 describes the Spiker+ architecture, with all the design choices that it involves, and section 3.5 introduces the framework able to configure, design, and generate custom hardware accelerators. Finally, section 3.6 presents the results obtained by applying the designed accelerators on the MNIST, SHD, and AudioMNIST, and section 3.7 concludes the chapter.

3.3 Neuromorphic accelerators: related work

In the past, SNNs were primarily implemented using software frameworks like Brian/Brian2 [93]. However, their unique features, including high parallelism, temporal evolution, and event-driven computation, are ill-suited for dominant Von-Neumann CPU architectures with one or a few powerful computational units. Unfortunately, Single Instruction Multiple Data (SIMD) architectures, such as GPUs and TPUs, optimized for standard ANNs workloads, are also not well-equipped for efficiently processing event-driven information across multiple timesteps [94]. Furthermore, the binary spike encoding of SNNs does not align with the typical 64,

32, or 16-bit numeric representations of these SIMD architectures. Therefore, dedicated neuromorphic hardware is crucial for ensuring the widespread adoption of SNNs. Figure 3.1 provides an overview of state-of-the-art accelerators for SNNs, offering a general perspective rather than an exhaustive list of solutions. For detailed information, refer to [95][96].

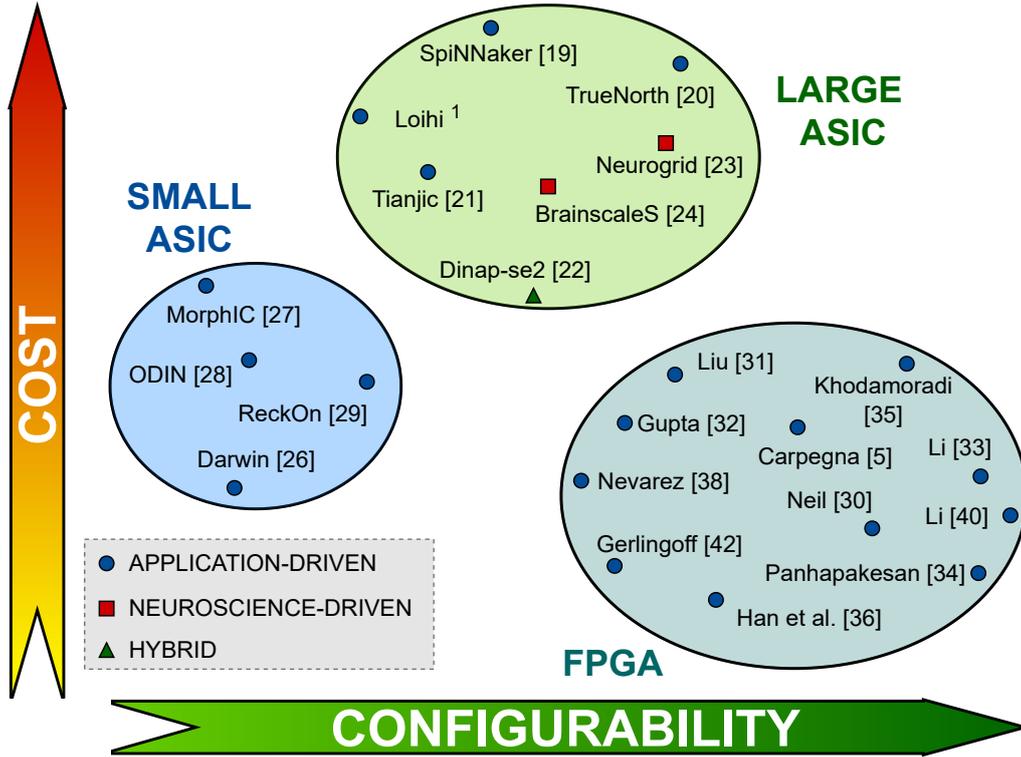


Figure 3.1: Landscape of neuromorphic hardware. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

Contributions in this domain span various design dimensions, including application-driven solutions focused on specific applications and those aimed at modeling biological neuron dynamics. However, this chapter primarily emphasizes the hardware technology dimension. The research effort is divided between analog solutions based on emerging technologies and efficient digital implementations [96]. In the digital realm, presented solutions differ on the target platform (ASIC or FPGA) and accelerator size, tailored for either large-scale systems or small applications.

Examining large network models, the SpiNNaker system developed at Manchester University is implemented using standard 32-bit ARM M4F CPUs simulating neuron activity. It optimizes spike routing between units [97]. Promisingly, major computer companies invest in developing their neuromorphic accelerators, such as

Intel Loihi [53] and IBM True-North [52]. These accelerators provide additional optimizations using specialized hardware to execute the neuron model. Another solution, Tianjic [98] from the University of Beijing, aims to implement a hybrid SNN/ANN model, benefiting from both domains.

An alternative design approach is recognizing that real biological neural networks function as analog physical systems. Emulating their efficiency involves using hardware components that approximate biological elements. Chips like Dynap-se2 [99] by SynSense, a spinoff of the Institute of Neuro-Informatics (INI) in Switzerland, exemplify this concept. Neurogrid [100] from Stanford University and the European project BrainscaleS [54] focus on faithfully simulating portions of a biological brain. Due to their complexity, these systems have programming tools for automatic SNN configuration. Tools like Rockpool [101] by SynSense and Nengo [102] facilitate automatic configuration and acceleration of SNNs on neuromorphic hardware based on Python model descriptions.

While these accelerators suit large-scale neuromorphic systems with good programmability, applications like the IoT, wearable devices, and biomedical sensors demand small sizes, computation robustness, and low power consumption. In such cases, designing specialized digital hardware accelerators to execute specific tasks, such as classification or regression, efficiently becomes a viable solution.

To pursue this direction, the first option is designing a compact, programmable ASIC supporting various architectures and models. The focus is on digital solutions for networks ranging from hundreds to a few thousand neurons. A preliminary comparison with non-spiking hardware accelerators, dedicated to efficient convolution execution in CNNs, is discussed in [103], highlighting the energy efficiency of SNNs. These accelerators often use the Address Event Representation (AER) protocol for compatibility with neuromorphic sensors. An example is found in [104], developed at Zhejiang University. Charlotte Frenkel’s work at the University of Delft and Eidgenössische Technische Hochschule (ETH) Zurich introduces three chips — MorphIC [105], ODIN [106], and ReckOn [107] — exploring online learning on small, low-power, and efficient accelerators.

The final digital accelerator option, central to this chapter, involves developing a specialized FPGA-based accelerator, offering advantages like cost reduction and increased flexibility by bypassing tape-out design needs. Many IoT edge systems now integrate FPGAs for task acceleration, potentially expanding the application of neuromorphic processors. An early example is the event-driven Minitaur [108], and subsequent alternatives feature diverse update policies, neuron models, architectural choices, and network sizes [109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 89]. Another key advantage of using an FPGA is its intrinsic reconfigurability. It allows hardware reprogramming to modify the network architecture or neuron model, tailoring the accelerator to specific application requirements. However, existing accelerators still need to exploit this characteristic efficiently. The first example of FPGA-oriented Design Space Exploration (DSE) is found in [119],

focusing on the best encoding technique for input data translation into spike sequences. Conversely, *E³NE* [120] provides a block library to configure networks for specific applications, optimizing data movement and hardware utilization, with a dedicated section for input encoding.

In this scenario, there is still a lack of a comprehensive framework that conceals the internal details of the architecture, enabling the user to operate at higher abstraction levels. For instance, a Python description of the model could be automatically translated into custom blocks. Spiker+ moves in this direction, trying to address this challenge.

3.4 Spiker+ architecture

This section presents the Spiker+ hardware architecture, which serves as the central component of the Spiker+ SNN hardware acceleration framework. The architecture is introduced top-down, beginning with the high-level network model and then delving into the neurons and input/output interfaces.

3.4.1 Network architecture

The SNN architecture presented here builds upon the initial Spiker architecture introduced in [89]. While our earlier work provided a proof of concept tailored for inference on the MNIST dataset [61], derived from the SNN model by Diehl and Cook [121], Spiker+ focuses on a generic and fully configurable architecture adaptable to various problems.

Figure 3.2 depicts the high-level architecture of a toy example of a three-layer FF-FC architecture used to introduce the three hierarchical levels of Control Units (CUs) that characterize Spiker+: (i) the *network CU*, responsible for synchronizing the various components within the network; (ii) the *layer CUs*, orchestrating the update of the neurons of a layer based on a set of input spikes; (iii) the *neuron CU*: the accelerator core controlling the update of the membrane potential in each neuron. This organization represents a highly optimized architecture in terms of performance and space utilization.

Block communication is based on a simple two-signal (**start/ready**) handshake protocol to ensure high modularity while minimizing design complexity. When a block (i.e., a neuron or a layer) is ready to work, it notifies the corresponding CU through the **ready** signal and awaits a new **start** signal to begin the computation. Consequently, if two blocks need synchronization, combining the two **ready** signals with an **AND** gate ensures that the CU waits for both before initiating a new computation. This protocol is also employed at the interface with the external world. Such an approach maintains modularity in the design and paves the way for various architectural solutions.

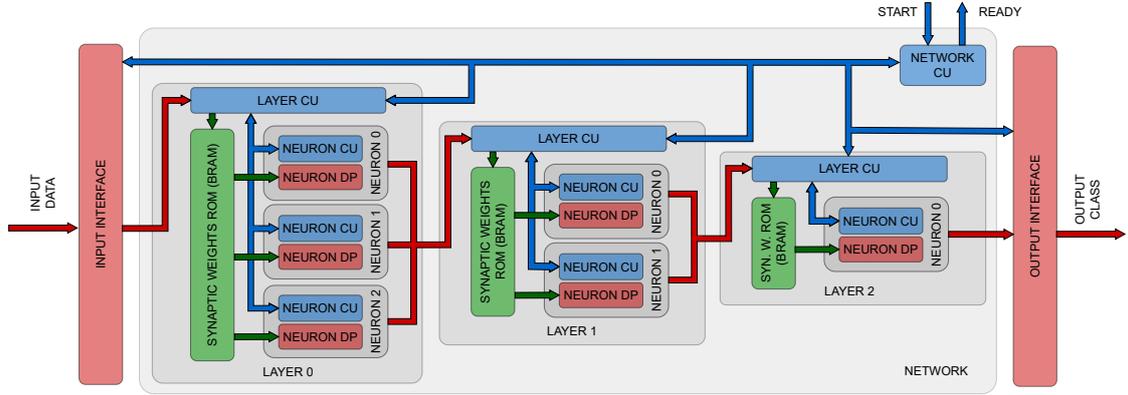


Figure 3.2: Spiker+ example of FF-FC architecture. The example includes three layers with different numbers of neurons and depicts all the architecture’s main control blocks. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

3.4.2 Network CU: global synchronization

The primary function of the Network CU is to coordinate the temporal evolution of the neurons of the different layers during an inference. As previously mentioned, in an SNN, information is encoded as trains of spikes (i.e., sequences of bits) received on every input. Each train is characterized by a given duration (i.e., the number of transmitted bits) denoted as `N_cycles`. Thus, the network performs inference by evolving over `N_cycles` temporal steps to analyze the temporal patterns.

The Network CU reported in Figure 3.3a receives a `start` signal when a new inference must be initiated. It then manages an iterative process. At every iteration, it awaits the `ready` signal from all Layer CUs composing the network to ensure all layers are prepared to work (i.e., `L1 ... Ln ready`). The Network CU also synchronizes with the input/output interface to ensure that data are available and results are correctly delivered (i.e., `IN/OUT ready`). Subsequently, it asserts a set of `start` signals, enabling all connected blocks to perform a computation step. At the same time, the internal counter (`CNT`) increases to track the computation length. Upon reaching the desired cycle count (`N_cycles`), the loop concludes, and the `ready` signal is asserted to indicate the end of the inference process.

3.4.3 Layer CU: deliver spikes to neurons

In a fully connected multi-layer SNN as the one proposed in Figure 3.2, a parallel update of each layer involves three dimensions: (i) the number of neurons, (ii) the number of inputs processed by each neuron, and (iii) the temporal dimension of each input, representing the number of cycles.

The last dimension is inherently sequential and cannot be parallelized, as it depends on the temporal evolution of the inputs. This dimension is managed by

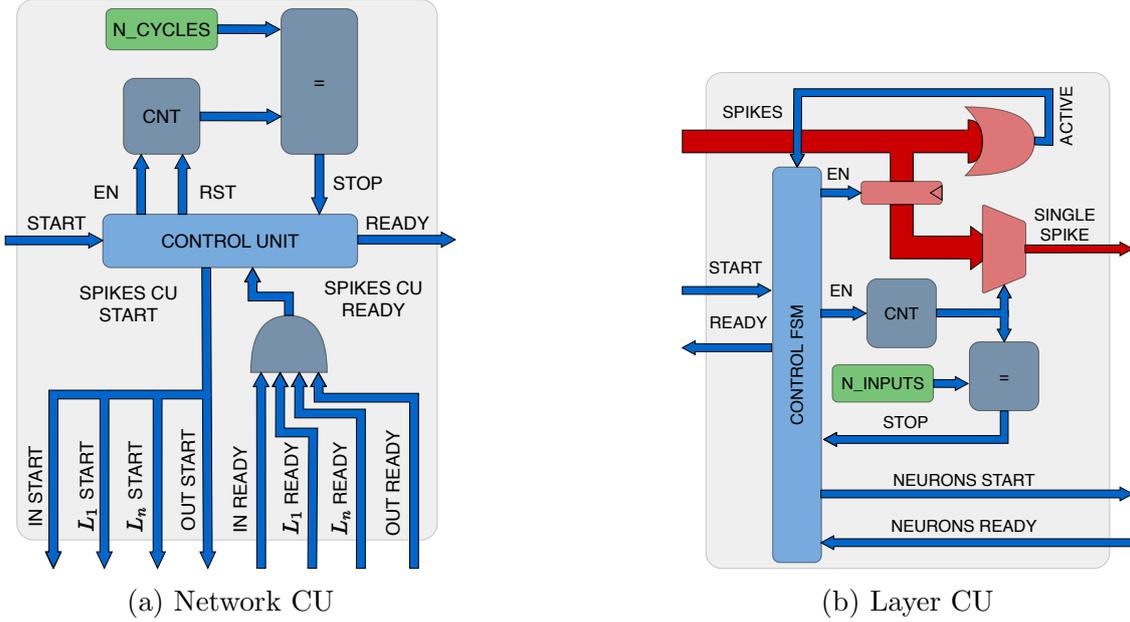


Figure 3.3: Internal architecture of the Network and Layer CUs. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

the Network CU discussed in subsection 3.4.2, which manages the update during each cycle.

If the network is sufficiently small, updating all neurons with their inputs in parallel within a single cycle could be feasible. However, the network and input data sizes are typically too large to achieve such a degree of parallelism. Consequently, Spiker+ exploits only one dimension to obtain parallelism, concurrently updating all neurons within a layer while sequentially providing inputs to each neuron. The Network CU, depicted in Figure 3.3b, oversees this process.

Once again, this circuit operates based on a **start/ready** protocol. The control unit receives a **start** signal from the Network CU and enters a loop: it awaits the readiness of the neurons composing the layer to process a new spike (**neurons ready** signal), initiates the computation (**neurons start** signal), and increments the internal counter (CNT). The counter directly selects the spike to be processed from the sampled inputs. When all input spikes have been provided to the neurons (N_INPUTS), the loop concludes, and the control unit asserts the **ready** signal.

An additional component visible in the upper section of Figure 3.3b is an **OR** gate utilized to verify if there is at least one active spike among the inputs. Currently, no encoding or compression has been applied to the spikes. However, considering that SNNs typically exhibits sparse activity, avoiding unnecessary computations when there are no spikes can significantly save time and power. The role of the **OR** gate is to compress along the time dimension: if there is no active spike in a

particular cycle, looping over all inputs to provide no spike to the neuron becomes unnecessary.

3.4.4 Neuron models

All the different LIF neuron models presented in section 2.7 are translated into dedicated hardware implementations in Spiker+, trying to minimize the required components. Building upon the groundwork laid in [89], the proposed neuron functions as a Multiply and Accumulate (MAC) unit, augmented with additional components and controls to manage its various states. Figure 3.4 shows the obtained architectures with subtractive reset, in order of increasing complexity (IF, I-order LIF and II-order LIF). Figure 3.5 shows the same alternatives, but with fixed reset.

From a hardware perspective, the most critical factors of the characteristic equation of the neurons are the multiplications. Four of them can be found in Equation 2.2:

1. The synapses weighting: $W \cdot s_{in}[n]$
2. The reset: $V[n - 1] \cdot r$
3. The exponential decays: $\alpha \cdot I_{syn}[n - 1]$ and $\beta \cdot V_m[n - 1]$

For the first one, exploiting the binary nature of the spikes reduces the operation to a simple selection: zero if there is no spike, W if a spike is present. This can be implemented as a bitwise AND between the weight and the spike.

The reset operation can be applied in two ways, as shown in equations 2.4 and 2.5. The first case exploits the binary nature of the spike: either the membrane is kept at its value or reset to zero, so this is again a selection process more than a multiplication. The hardware implementation is a bit more general since it allows to explicitly choose the value of V_{reset} , which in this case can also be different from 0, as shown in Figures 3.5a, 3.5b and 3.5c. The second reset method can be obtained by simply subtracting the threshold voltage V_{th} from the computed value of V_m , as shown in Figures 3.4a, 3.4b and 3.4c.

At this point, the last critical multiplication is the one required to compute the step-by-step exponential decay of the membrane. The problem exists only for the two LIF models (in the IF model, the membrane is kept fixed without stimuli), with one multiplication needed in the I-order version and two multiplications in the II-order one. The criticality is solved once again, exploiting the characteristics of binary operations. If one of the operators is representable as a power of two, the multiplication can be reduced to a simple bit-shift. Since there is no control on the values of I_{syn} and V_m , which evolve dynamically during the update of the network,

the only parameters on which it is possible to act are the constant hyper-parameters α and β . The values can vary between 0 and 1, with larger values corresponding to slower exponential decay. Generally, a value near to 1 is observed. In this case α and β can be approximated as $\alpha = 1 - \alpha'$ and $\beta = 1 - \beta'$, where α' and β' are negative powers of 2. As shown in [89], the overall accuracy has no notable impact if such an approximation is applied during the training phase.

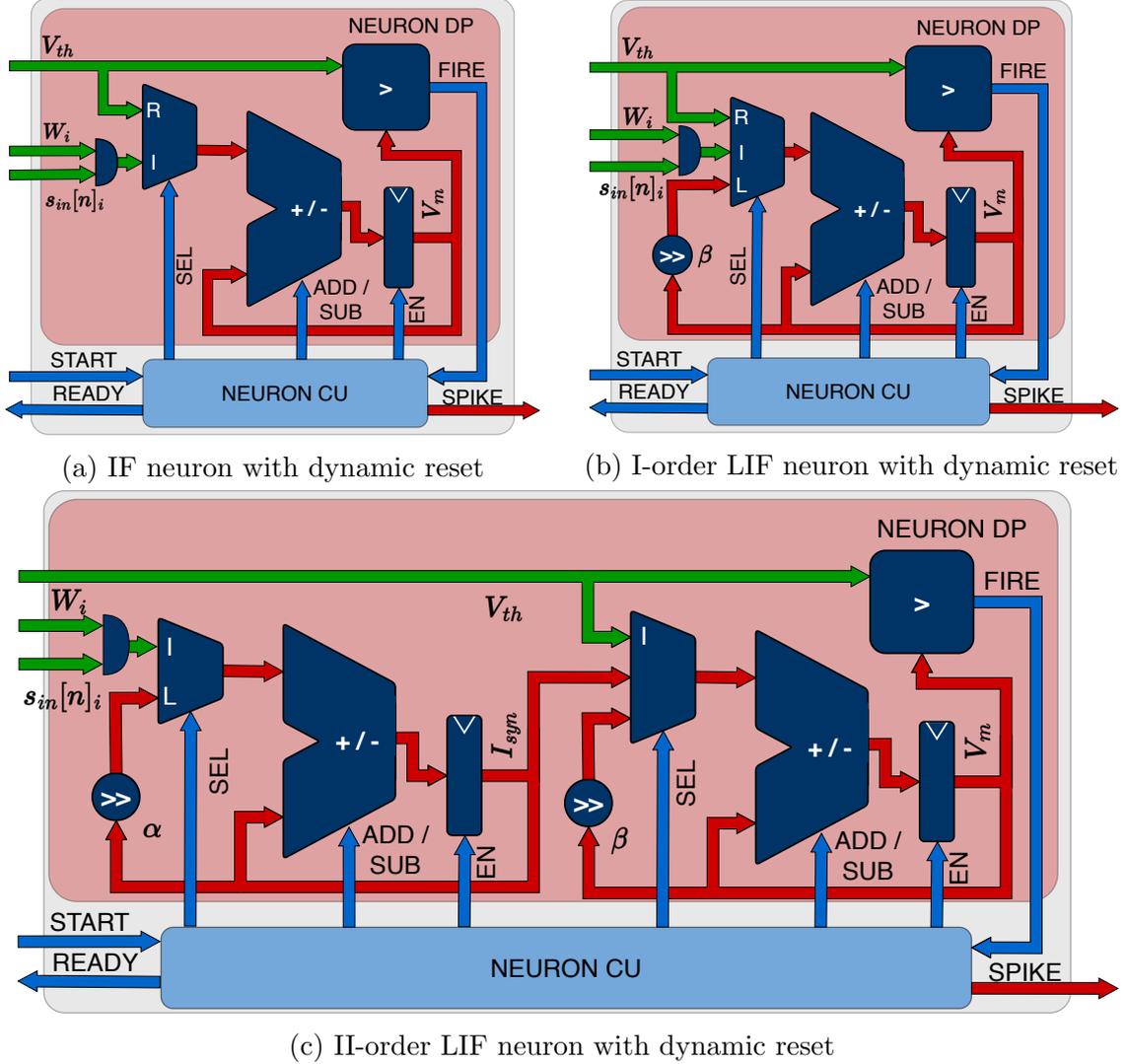


Figure 3.4: Spiker+ neuron architectures in order of increasing complexity (IF, I-order LIF and II-order LIF), with subtractive reset. In the multiplexers, channels labeled with I indicate the beginning of the Integrate path, R the beginning of the Reset path, and L the beginning of the Leakage path. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

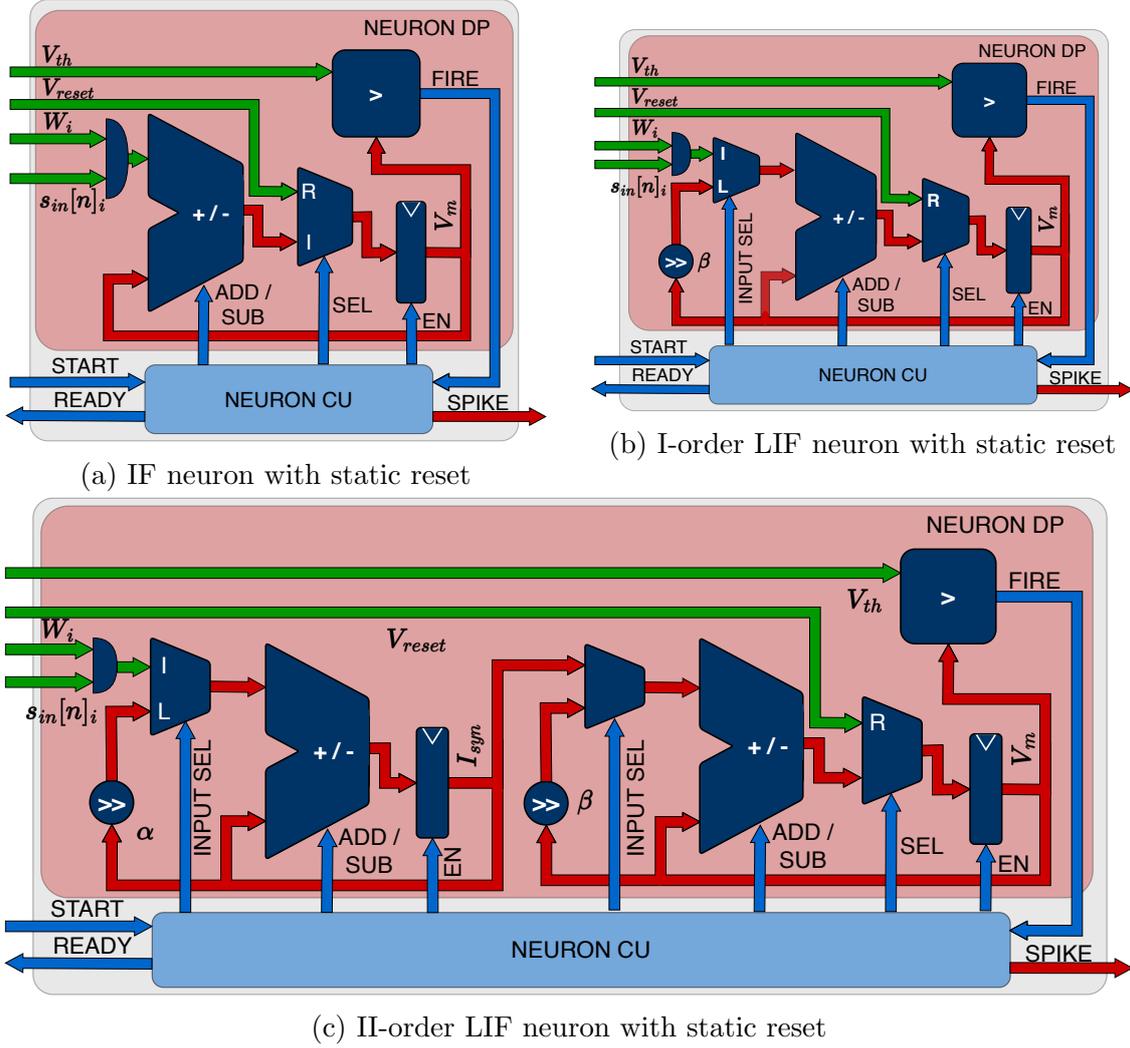


Figure 3.5: Spiker+ neuron architectures in order of increasing complexity (IF, I-order LIF and II-order LIF), with fixed reset. In the multiplexers, channels labeled with I indicate the beginning of the Integrate path, R the beginning of the Reset path, and L the beginning of the Leakage path. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

3.4.5 Synapses

The primary advantage of implementing SNNs on dedicated hardware, alongside the execution parallelism, lies in the opportunity to integrate memory and computation. On an FPGAs, this integration can be achieved through two distinct methods.

For relatively small memory requirements, such as the internal parameters of the neurons, the internal Look Up Tables (LUTs) can serve as a viable memory

solution. This approach offers superior speed, leveraging Flip Flops (FFs) and registers. However, the available space is limited, primarily due to the necessity of accommodating the logical functions of the network within the LUTs.

In scenarios where a larger memory capacity is necessary, particularly for synaptic weights, many FPGAs grant access to discrete units of Static Random Access Memory (SRAM) strategically positioned close to the computing elements, commonly referred to as Block RAM (BRAM).

Spiker+ provides a synapse interface, implementing the **start/ready** handshake protocol, and relies on an initialization file containing quantized weights. Weights are stored into BRAMs. Spiker+ expects all neurons to access their respective weights in parallel upon activating the ready signal by the synapse; therefore, it strongly relies on the high parallelism provided by on-board BRAMs. Spiker+ also permits storing weights in an external Dynamic Random Access Memory (DRAMs) when on-board space is insufficient. In such situations, the synaptic interface loads the weights for the current cycle before asserting the ready signal, impacting the accelerator’s speed.

A secondary configurable attribute concerning synapses involves incorporating feedback connections, as mentioned in section 2.5. Spiker+ can be configured to include or exclude these connections, depending on the application requirements.

3.4.6 I/O interface

Spiker+ requires an input/output interface to receive data and transmit results. Spiker+ supports two scenarios. In the simple scenario, inputs have already been encoded as spikes. For instance, these data may originate from neuromorphic sensors, such as a Dynamic Vision Sensor (DVS) cameras or a silicon cochlea. Alternatively, they could be pre-encoded by an external block before being stored.

In a more complex scenario, data are stored in a raw numeric format and converted on-board into spike streams. There are different methods available for this conversion, as mentioned in section 2.8. An efficient rate encoding structure such as the one proposed in [89] can be directly connected to Spiker+. Furthermore, several possibilities exist concerning data transmission: data may arrive as a continuous stream directly from a sensor or be transmitted through an external link. Alternatively, data may be stored in memory, necessitating memory access by the accelerator. To accommodate these diverse scenarios, Spiker+ uses the **start/ready** handshake protocol to manage the communication with the input interface.

At the accelerator output, decisions are usually taken based on the firing activity of the last layer. Spiker+ implements the output interface using simple counters, one for each output neuron, that can be post-processed outside the accelerator (e.g., the most active neuron wins). This simple implementation is only one possible option and can be easily customized to specific needs. The only requirement of the output interface is to implement the **start/ready** protocol to synchronize with the

network.

It has to be noted that the latency measures reported in this piece of work have been taken considering input and output interfaces that can keep the pace of the accelerator. Overall, the network performance strongly depends on the interfaces. Results in section 3.6 aim to show the maximum performance Spiker+ can reach.

3.5 Configuration framework

Spiker+ goes beyond being a mere hardware accelerator; it is a comprehensive design framework that facilitates easy customization of the SNN accelerator for specific applications. As detailed in section 3.4, the platform encompasses six distinct neuron models, a modular layer interface allowing instantiating any desired number of layers, and customizable inter-layer feedback connections. However, manually defining the architecture at the Register Transfer Level (RTL) requires substantial effort. To tackle this challenge, Spiker+ incorporates a Python-based configuration framework, streamlining the customization process to just a few lines of code. The customization and tuning flow for a specific target application within Spiker+ is depicted in Figure 3.6.

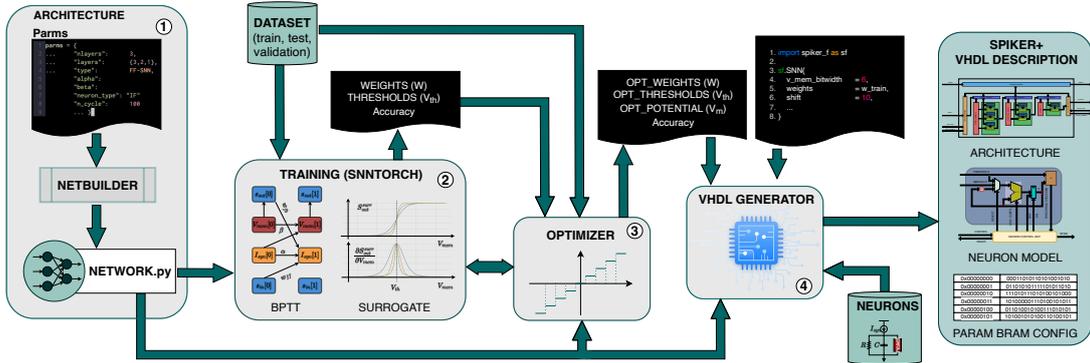


Figure 3.6: Spiker+ configuration framework. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

The configuration of the network, its training, and the optimization towards its hardware realization are performed at the software level, using Python functionalities. These first steps are built on top of the open-source framework `snn_torch` [86]. The description of the network is based on a Python dictionary that contains various parameters, such as the SNN model and architecture (e.g., FF-FC or Fully-Connected Recurrent (FC-R) SNN), specifying relevant parameters like the number of layers, neurons per layer, neuron type, and timing constants.

First (step 1), the dictionary is processed by the `netbuilder` module, which converts it into a Python object that serves as a key component for subsequent

steps. Once the network is constructed, it undergoes training (step 2), utilizing `snntorch` and `pytorch`. During training, as per [89], the time constants α and β are rounded to the nearest power of two to adjust for this approximation. The outcome of the training phase includes a set of learned parameters such as weights and thresholds along with an accuracy evaluation of the trained model.

As training typically occurs in floating-point precision, unsuitable for edge hardware accelerators, Spiker+ implements functions to quantize the network, automating the exploration of the quantization impact on SNN accuracy (step 3). Spiker+ employs a two's complement N-bit fixed-point representation, and if any values exceed the representable range during quantization, they are saturated to either the maximum or minimum value. Users can define the search interval (default: 16 to 0), and the tool iterates through these values, performing full inferences and returning the resulting accuracy values. Currently, the `optimizer` performs a complete grid search over the quantization range, but future improvements may include more advanced search techniques, such as Bayesian optimization, to accelerate the process. Additionally, a Quantization Aware Training (QAT) could help mitigate the accuracy loss. Plans also include merging the `trainer` and `optimizer` using tools like Brevitas [122] to handle quantization.

Thanks to the modularity of the Spiker+ blocks and their interfaces, the tool can integrate seamlessly with other open-source frameworks like `spikingjelly` [123]. This allows users to configure custom `netbuilder`, `trainer`, and `optimizer` modules based on other tools. However, the built-in functionality of Spiker+ relies on `snntorch`, benefiting from its active community and extensive documentation. The goal is to offer an open-source, user-friendly tool that aligns with ongoing efforts to streamline the development and deployment of neuromorphic systems.

In the final step (step 4), the chosen model, architecture, and trained parameters are delivered to the `VHDL Generator`, which automatically translates them into a VHSIC Hardware Description Language (VHDL) description of the SNN. This stage capitalizes on the modularity of the proposed architecture and utilizes an available library of neuron models. This is the core and the main contribution of Spiker+ tool. It relieves the user from the need to have specific skills in hardware design: starting from a high-level description of the network, it automatically generates the desired architecture using the VHDL language.

Furthermore, the tool generates configuration files for storing the trained parameters in the FPGA memory. This provides flexibility to the user in selecting the type of memory. Modern hardware design tools, such as Xilinx Vivado[™], support the automatic generation of Read Only Memory (ROM) memories, allowing users to choose the desired hardware platform, such as LUTs, BRAMs, or distributed Random Access Memory (RAM). These tools expect a configuration file as input, precisely what Spiker+ provides.

To the best of our knowledge, Spiker+ is the first complete high-level synthesis tool for an SNN, generating VHDL code automatically from a high-level Python

description of the network. Additionally, the VHDL description of the accelerator includes a testbench for enhanced development and simulation convenience. This testbench allows inputs to be read from a file, enabling users to provide the desired input vector stimuli. It provides a straightforward interface to ensure the device synthesizes on the selected FPGA. Only recently, a new intermediate representation, Neuromorphic Intermediate Representation (NIR), was introduced to decouple the network description from its specific implementation [124]. NIR enables the description of any SNN as a graph, making explicit the connection patterns, computational primitives, and other key aspects. Given that Spiker+ specifically targets Fully-Connected (FC) and FC-R architectures, incorporating a translation into NIR at this stage would have added unnecessary complexity. However, this approach shows great potential for future iterations of Spiker+. A `vhdl generator` capable of interpreting NIR descriptions would further decouple the network configuration from the hardware architecture, enhancing the modularity and flexibility of the framework, in line with its current design principles.

3.6 Experimental results

Spiker+ has been evaluated using two widely recognized benchmark datasets, (i) MNIST [61] and (ii) SHD [90], plus an additional audio dataset, the AudioMNIST [91].

MNIST comprises grayscale images of handwritten digits from 0 to 9, commonly used to benchmark AI algorithms. This dataset is ideal for comparing Spiker+ with other SNN accelerators. Images are converted into spikes using Poisson-distributed rate encoding. Due to the dataset simplicity, a basic I-order LIF model with a FF-FC structure suffices for accurate classification.

SHD is explicitly designed as an SNN benchmark, containing recordings of people pronouncing numbers in English and German. It requires a more complex neuron model, specifically a II-order LIF, and a network architecture with inter-layer recurrent connections to account for the importance of the time dimension in achieving acceptable classification accuracy.

AudioMNIST is again a dataset of spoken numbers from 0 to 9, pronounced in English. To convert audio samples into spikes, spectrograms were first generated using 40 Mel-spaced band-pass filters. These spectrograms were then binarized by setting values to 1 only where the spectrogram exceeded a defined threshold. This process reduces the dataset’s size compared to the other two, making it suitable for real-time applications where an electronic system could leverage Spiker+ support. The AudioMNIST dataset has not yet been widely explored with SNNs in the literature, making it an intriguing case.

These datasets differ significantly, demanding SNNs with distinct models and complexities, providing an opportunity to test Spiker+ reconfigurability for three

different tasks. All the models are trained offline employing BPTT with two different surrogate gradient functions. Table 3.1 summarizes the considered setup.

Table 3.1: Summary of the experimental set-up on the two datastes. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

	MNIST	SHD	Audio MNIST
Data type	Gray-scale images	Audio recordings	Audio recordings
# inputs	784	700	40
Spikes time-steps	100	100	73
Encoding	Rate code	Custom [90]	Binarized spectrogram
Training samples	60,000	8,156	24,000
Test samples	10,000	2,264	6,000
Net type	FF-FC SNN	FC-R SNN	FF-FC SNN
Net arch	784-128-10	700-200-20	40-150-10
Neuron model	I-order LIF	II-order LIF	I-order LIF
Reset	Subtractive	Subtractive	Subtractive
Training method	BPTT with SG	BPTT with SG	BPTT with SG
Surrogate function	Arctan	Fast Sigmoid	Arctan
Model accuracy	96.83%	75.44%	95.23%

The remainder of this section is structured as follows: subsection 3.6.1 presents results from tuning Spiker+ on the three target datasets and compares it with state-of-the-art SNN accelerators on FPGA. Subsections 3.6.2, 3.6.3, and 3.6.4 analyze various Spiker+ configurations, examining the influence of architectural choices and data characteristics on area, latency, and power consumption. The goal of this analysis is to show the importance of a tool able to automatically generate different designs when exploring different alternatives to solve a specific task.

3.6.1 Benchmarking

Table 3.2 provides a comprehensive comparison of Spiker+ with recent state-of-the-art FPGA accelerators designed for SNNs on the MNIST dataset. The table is split into two sections. The upper section covers Spiking Convolutional Neural Networks (SCNN) accelerators, where spiking layers are strategically placed after or interleaved with standard convolutional layers, gradually identifying key features in input images. The lower section considers pure spiking accelerators with fully connected layers of either IF or LIF neurons. Notably, the comparison focuses on works published from 2020 onward, while references such as [89] and [118] provide information on the performance of older accelerators.

Table 3.2: Comparison of Spiker+ to state-of-the-art FPGA accelerators for SNNs. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

Design	Lin et al. [109]	Nevarez et al. [116]	Li et al. [111]	Gerlinghoff et al. [120]	Panchapakesan et al. [112]	Khodamoradi et al. [113]
Year	2023	2021	2023	2022	2021	2021
f_{clk} [MHz]	100	200	300	200	200	N/R
Neuron bw	8	8	12	N/R	4	N/R
Weights bw	8	8	8	3	4	N/R
Update	Clock	Clock	Clock	Clock	Event	Event
Model	IF	Spike-by-Spike (SbS)	LIF	LIF	IF	LIF
FPGA	XA7Z020	XC7Z020	XCZU3EG	XCVU13P	XCZU9EG	XA7Z020
Avail. BRAM	140	140	216	2688	912	140
Used BRAM	N/R	16	50	N/R	N/R	40.5
Avail. DSP	220	220	360	12288	2520	220
Used DSP	0	46	288	0	N/R	11
Avail. logic cells	159,600	159,600	211,680	3,088,800	822,240	159,600
Used logic cells	27,551	23,704	15,000	51,000	N/R	39,368
Arch	28x28-32c3-p2-32c3-p2-256-10	28x28x2-32c5-p2-64c5-p2-1024-10	28x28-16c3-64c3-p2-128c3-p2-256c3-256c3-10	32x32x1-6c5-p2-16c5-p2-120c5-120-84-10	28x28-32c3-p2-32c3-p2-256-10	28x28-16c7-24c7-32c7-10
#syn	8,960	75,776	2,560	25,320	10,752	320
T_{int}/img [ms]	0.27	1.67	0.49	0.29	0.08	N/R
Power [W]	0.28	0.22	2.55	3.40	N/R	N/R
E/img [mJ]	0.076	0.37	1.250	0.986	N/R	N/R
E/syn [nJ]	8.48	4.88	488	38.9	N/R	N/R
Accuracy	99.00%	98.84%	98.12%	99.10%	99.30%	98.50%
Design	Han et al. [114]	Gupta et al. [110]	Li et al. [118]	Carpegna et al. [89]	SPIKER+ (this work)	
Year	2020	2020	2021	2022	2024	
f_{clk} [MHz]	200	100	100	100	100	
Neuron bw	16	24	16	16	6	
Weights bw	16	24	16	16	4	
Update	Event	Event	Hybrid	Clock	Clock	
Model	LIF	LIF	LIF	LIF	LIF	
FPGA	XC7Z045	XC6VLX240T	XC7VX485	XC7Z020	XC7Z020	
Avail. BRAM	545	416	2,060	140	140	
Used BRAM	40.5	162	N/R	45	18	
Avail. DSP	900	768	2,800	220	220	
Used DSP	0	64	N/R	0	0	
Avail. logic cells	655,800	452,160	485,760	159,600	159,600	
Used logic cells	12,690	79,468	N/R	55,998	7,612	
Arch	784-1024-1024-10	784-16	784-200-100-10	784-400	784-128-10	
#syn	1,861,632	12,544	177,800	313,600	101,632	
T_{int}/img [ms]	6.21	0.50	3.15	0.22	0.78	
Power [W]	0.477	N/R	1.6	59.09	0.18	
E/img [mJ]	2.96	N/R	5.04	13	0.14	
E/syn [nJ]	1.59	N/R	28	41	1.37	
Accuracy	97.06%	N/R	92.93%	73.96%	93.85%	

In image classification tasks, SCNNs accelerators achieve superior accuracy, peaking at 99.30% with larger and more complex networks, often utilizing DSP cores on the FPGA (e.g., [116, 113, 111]). However, despite Spiker+ being explicitly designed to trade off accuracy with other design dimensions (i.e., power, area, latency), it still achieves a commendable accuracy of 93.85%. Among purely spiking

accelerators, it is only surpassed by [114], which employs an accelerator with higher latency, power, and size. Remarkably, Spiker+ excels in compactness and low power consumption. On a low-end Xilinx™ XC7Z020 FPGA board, it uses only 7,612 logic cells (4.8% of available cells) and 18 BRAMs (13% of available BRAMs), with a total power requirement of 180mW. This makes it an optimal solution for limited space or power-constrained applications. It is important to note that Table 3.2 measures the area with a general "logic cells" value obtained by combining LUTs and FFs for a concise overview. For Spiker+, these values are 4,314 and 3,298, respectively. Detailed values for other accelerators can be found in their respective papers.

A noteworthy observation from the comparison in Table 3.2 is that the top power-efficient accelerators employ a clock-driven update policy. This counter-intuitive finding contradicts the general literature assertion favoring event-driven approaches for power efficiency. A clock-driven approach seems the most effective solution for relatively small-sized accelerators lacking sufficient sparsity in spiking activities. Li et al. attempted to address this with a hybrid architecture, adapting its update strategy to input sparsity. However, this strategy did not yield significant power savings, even with added complexity [118]. Importantly, the power consumption of the accelerator reported in [89] is unrealistically high. This is due to an error in mapping I/O ports during the accelerator implementation, leading to an overestimated value.

Regarding latency, Spiker+ requires $780\mu s$ to classify an input image. Although not the fastest result in Table 3.2, this achievement is noteworthy considering the limited hardware resources and power consumption. Factors influencing this latency include: (i) the clock frequency capped at 100MHz due to BRAM access time; (ii) the image encoding using 100 time-steps (the window size impacts inference time directly); (iii) the speed of the input and output interfaces; (iv) the input spiking activity affecting the classification time, as explained in subsection 3.6.2. For comparison, Carpegna et al. [89] achieved $220\mu s$ image classification time with a 3500 time-step encoding window. This work employed a different encoding that privileged spike sparsity, impacting accuracy (i.e., 73.96%) but demonstrating how sparsity can reduce inference time.

To better compare Spiker+ with other accelerators Figure 3.7 shows the Pareto optimal curves comparing the considered architectures and highlights the optimal trade-offs between different metrics. Looking at Figure 3.7(a) it can be observed that Spiker+ lies a little behind the majority of the others in terms of accuracy, with an average ranking in terms of latency. From the point of view of power consumption and area utilization, expressed in terms of the required logic cells, Spiker+ is dominant. This becomes even more evident in Figures 3.7(b) and (c), where the hardware-related features are compared against one another. Across all three plots, Spiker+ consistently emerges as the optimal choice, highlighting the effectiveness of the design strategies discussed in section 3.4. It is important

to note that, because the focus was primarily on hardware design, there is still potential for accuracy improvements using more advanced training and parameter tuning techniques. Notably, a multi-objective Bayesian optimization conducted in [92] enabled us to optimize the encoding process, reducing the required number of time steps, directly linked to latency, and improving accuracy by approximately 2%. Thus, there is still room for further enhancements at the algorithmic level. Ultimately, what stands out is that Spiker+ offers the best balance in terms of the consumption-area-latency trade-offs.

Apart from the MNIST use case, Spiker+ is the first accelerator tested on SHD and AudioMNIST. Consequently, results reported in Table 3.3 are presented independently, without comparison to other architectures.

Table 3.3: Benchmarking on the SHD and AudioMNIST datasets. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

	SHD	AudioMNIST
f_{clk} [MHz]	100	100
Avail. logic cells	159,600	159,600
Used logic cells	18,268	10,124
Neuron bw	8	8
FF weights bw	6	5
Arch	700-200-20	40-150-10
RR weights bw	5	0
#syn	184,000	7,500
Update	Clock	Clock
$T_{lat}/\text{recording}$ [ms]	0.54	0.08
$T_{lat}/\text{timestep}$ [us]	5.4	1.0
Model	II order LIF	I order LIF
Power [W]	0.43	0.29
FPGA	XA7Z020	XA7Z020
Energy/recording [mJ]	0.23	0.02
Energy/synapse [nJ]	1.25	2.67
Avail. BRAM	140	140
Used BRAM	51	16
Accuracy	72.99%	89.82%

The hardware requirements for processing the SHD exceed those used for MNIST and AudioMNIST due to several factors: the network architecture required to process this complex dataset is larger, the neuron model is a more complex II-order LIF (i.e., double size compared to a I-order LIF), and the accelerator uses a FC-R model featuring inter-layer feedback connections with weights stored in BRAM. The bit widths of the neuron membrane potential and weights are also higher.

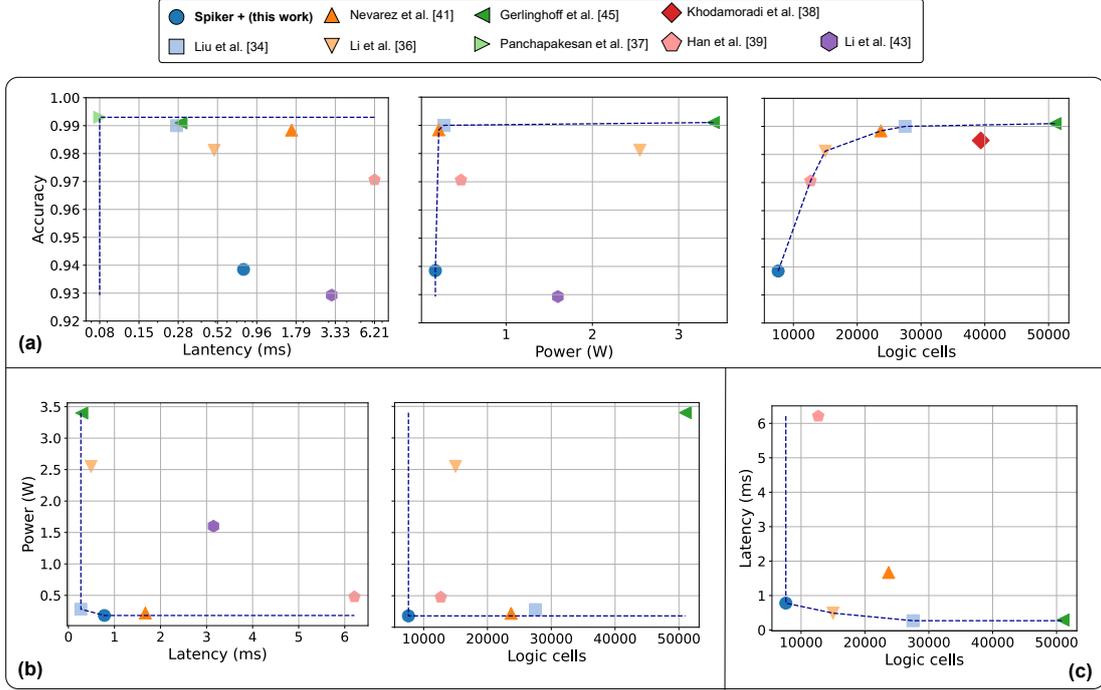


Figure 3.7: Pareto optimal curves of the accelerators presented in Table 3.2. (a) Accuracy is plotted against the three key hardware metrics: latency, which reflects computation speed, power consumption, and area utilization, measured by the number of logic cells required by the accelerator. Memory usage is closely tied to area utilization, so it is not shown here. (b) Power consumption is plotted against latency and area utilization. (c) Area utilization is plotted versus latency. Spiker+ emerges as the optimal solution for all hardware-related metrics, though it still falls slightly short in terms of accuracy. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

However, Spiker+ remains smaller and more power efficient than most accelerators in Table 3.2. Latency is reduced from $780\mu s$ for MNIST to $540\mu s$ for SHD thanks to lower input activity in the biologically inspired encoding used for this dataset. Similarly, latency is reduced to $80\mu s$ for the AudioMNIST, thanks to the lower dimensionality of the inputs (40 spike channels are sufficient to capture the most relevant features of the spoken digits), and a smaller amount of timesteps. Differently from the MNIST task, both the SHD and AudioMNIST are based on time-varying signals, which are intrinsically more suitable for SNN processing. Table 3.3 reports the latency required for a single time step, which is $5.4\mu s$ and $1\mu s$ respectively. This makes Spiker+ on its own able to process in real-time signals sampled up to 185kHz for the SHD and 1MHz for the AudioMNIST. While this does not account for input pre-processing or communication between Spiker+ and

the processor or main memory, these tasks are not computationally intensive, particularly for AudioMNIST, due to the conversion protocol we developed and the reduced size of the input data.

3.6.2 Performance vs input activity

As previously mentioned, the input spiking activity, influenced by the encoding method, plays a critical role in the accelerator’s performance. Before delving into a detailed analysis, Figure 3.8 illustrates the average number of active cycles across different network layers. A significant variation is observed among the datasets. In MNIST, activity consistently decreases as it progresses through the network. For SHD, there is a peak in activity in the first hidden layer, while in AudioMNIST, the activity remains steady in the first two layers and increases in the output layer. This discrepancy could be attributed to the inter-layer feedback in SHD, which causes more synchronized activity compared to the FF-FC architecture used for the other two datasets. Other contributing factors include differences in input data statistics and the impact of the training process

Since all layers update in parallel and process inputs sequentially, latency is determined by the slowest layer (i.e., the layer handling the largest set of inputs). For the MNIST and SHD architectures, detailed in Table 3.1, the slowest layer is the input layer, processing 784 and 700 inputs respectively. In this layer, 100% of time-steps contain at least one spike for MNIST, while for SHD, the percentage is around 48%. Consequently, SHD, with the combination of a lower number of inputs and lower activity in the input layer, enables increased inference speed. Since both models use the same number of time steps and clock frequency, and the difference in the number of inputs is not significant, one might expect about 48% inference time reduction for SHD compared to MNIST due to the reduced activity (i.e., about 0.37 ms). However, the observed value in Table 3.3 is 0.54 ms. The higher latency is explained by the FC-R model used by SHD incorporating inter-layer feedback connections, processed sequentially. Therefore, an additional set of 200 feedback inputs must be processed for the first layer, with an average of 93% active time steps. For AudioMNIST, the latency is primarily determined by the hidden layer, which consists of 150 neurons with an average activity of around 71%. Given the reduced number of time steps compared to the other datasets, the resulting latency is significantly lower.

The input activity not only impacts inference latency, as explained earlier. When the activity decreases, there is a higher probability of having spare cycles with no spikes, allowing the Layer CU to skip the sequential processing of all inputs. This reduction in calculations also affects power consumption. Figure 3.9 analyzes how power, latency, and energy (i.e., power times latency) change with different input activities on the two datasets, highlighting counterintuitive behaviors.

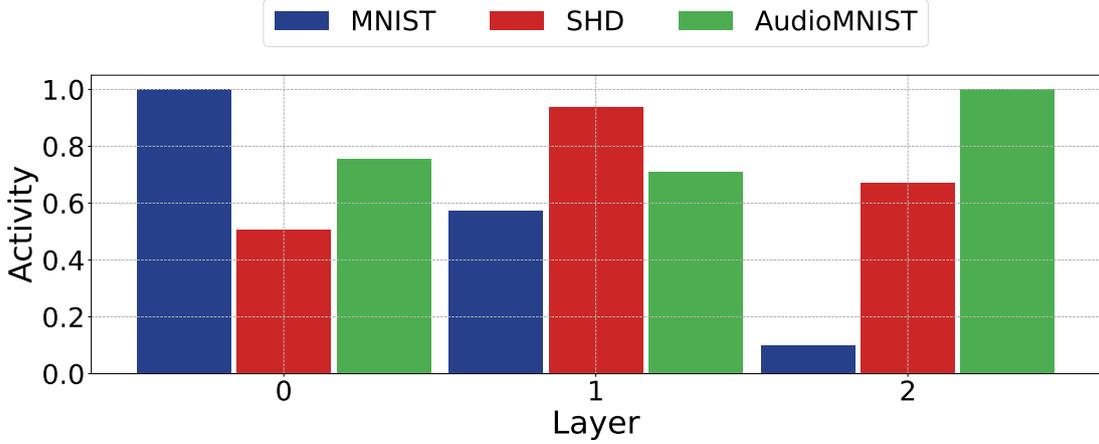


Figure 3.8: Visual representation of the average number of active cycles at various stages of the network. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

Examining Figure 3.9a that reports the average power consumption of an inference task, we observe two distinct behaviors. In the case of MNIST and AudioMNIST, reducing input activity increases power consumption, reaching a limit of around 200mW and 300mW respectively as activity approaches zero and stabilizing at 180mW and 295mW under full activity. This behavior is explained by the clock-driven nature of the accelerator, where every clock cycle triggers a network update regardless of active spikes in the input. As mentioned earlier, Spiker+ skips time steps without active inputs. However, the exponential decay of the membrane is computed step by step. Therefore, without input stimuli, Spiker+ dedicates one clock cycle to decay all membranes before returning to an idle state, awaiting the next input set. The two situations are similar from the perspective of neuron switching power, as the membrane is updated in both cases. However, the layer CU continually switches between two states, resulting in a high total switching activity and higher power consumption. Conversely, when executing a sequential update on the inputs, the CU enters the update state and then awaits the completion of the loop. This behavior is not observed in SHD. Given the larger architecture and higher weight bit-width, power consumption in SHD is likely dominated by BRAMs access during input processing.

In conclusion, latency heavily depends on input activity. Without active spikes, the execution time tends to $N_{\text{cycles}} \times T_{\text{clk}}$, as a single clock cycle is sufficient to decay the membranes. Consequently, overall energy consumption is reduced with decreasing input activity.

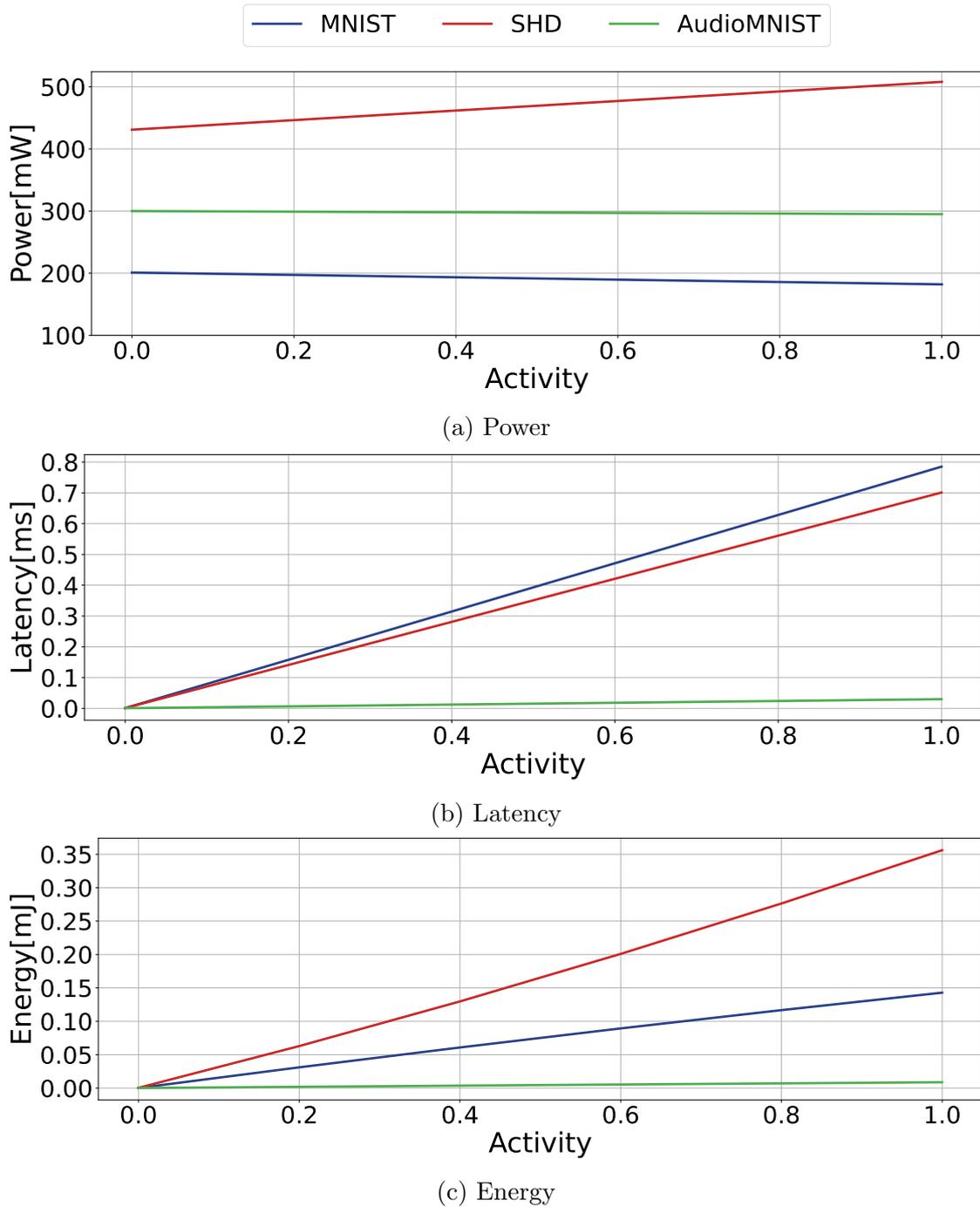
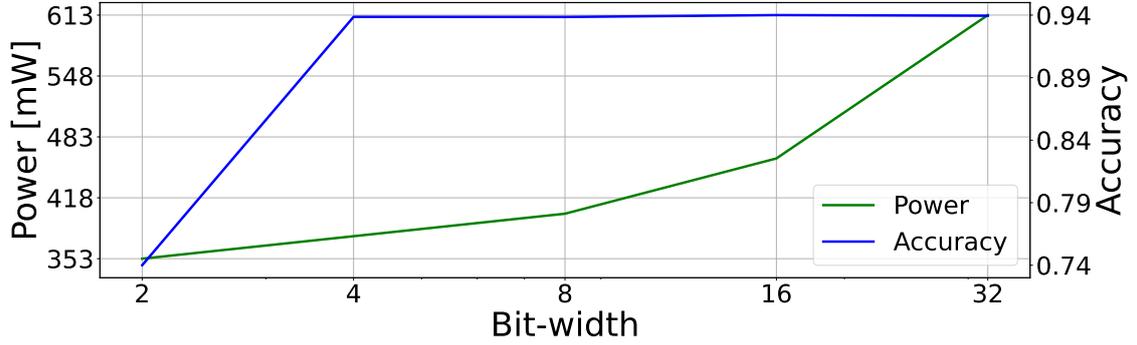
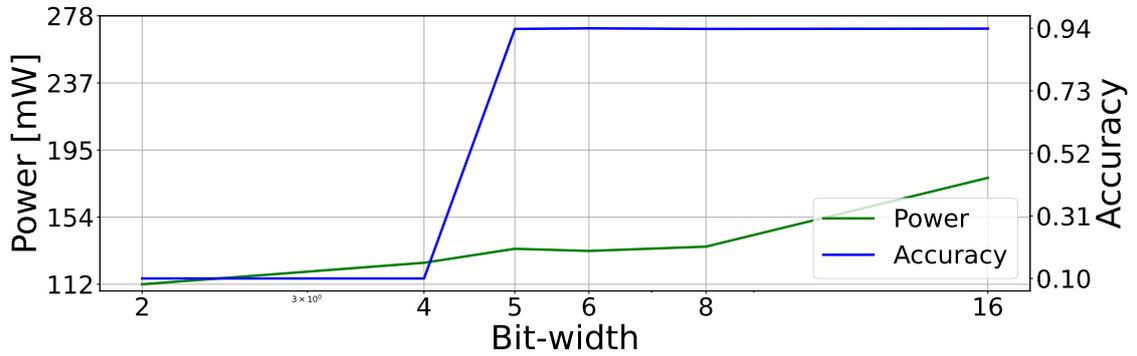


Figure 3.9: Impact of input activity variations on energy, power, and latency of inference using Spiker+. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.



(a) Membrane potential quantization on the MNIST dataset



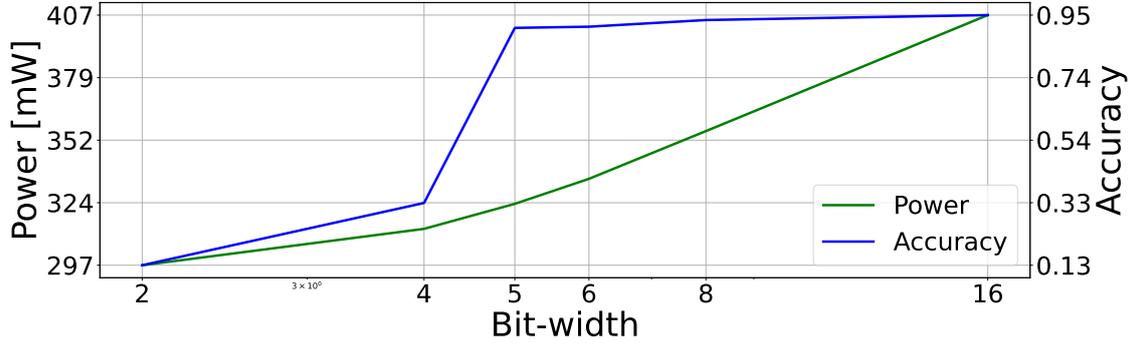
(b) Weights quantization on the MNIST dataset

Figure 3.10: Impact of quantization of neuron membrane potentials and synaptic weights on inference accuracy and power consumption for target datasets. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

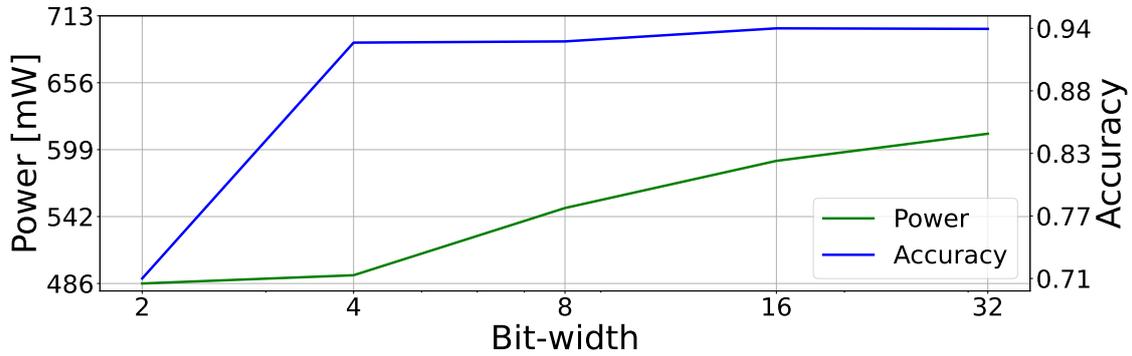
3.6.3 Performance vs quantization

As one of the key features of Spiker+ is the optimization of the accelerator through quantization of weights and membrane potentials, it is crucial to examine how these design choices influence performance. Latency remains unaffected by the selected bit-widths, as it is determined by the required amount of processing, which depends on: i) input size, ii) number of time steps, iii) network size, where each layer processes the outputs from the previous one, and iv) spike sparsity. Since all neurons operate in parallel, latency is dictated by the layer that needs to process the largest set of spikes. The primary presumed impacts of quantization are on power consumption and network accuracy. Figure 3.10, Figure 3.11 and Figure 3.12 illustrate the results of quantization on the considered datasets.

The resilience of SNNs to quantization is notable. In all the models, despite their differences and the unique information encoding methods employed, the decrease in accuracy with different quantization values is minimal. However, reducing precision by even a single bit has an evident effect on power consumption, owing to the large



(a) Membrane potential quantization on the Audio MNIST dataset



(b) Weights quantization on the Audio MNIST dataset

Figure 3.11: Impact of quantization of neuron membrane potentials and synaptic weights on inference accuracy and power consumption for target datasets. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

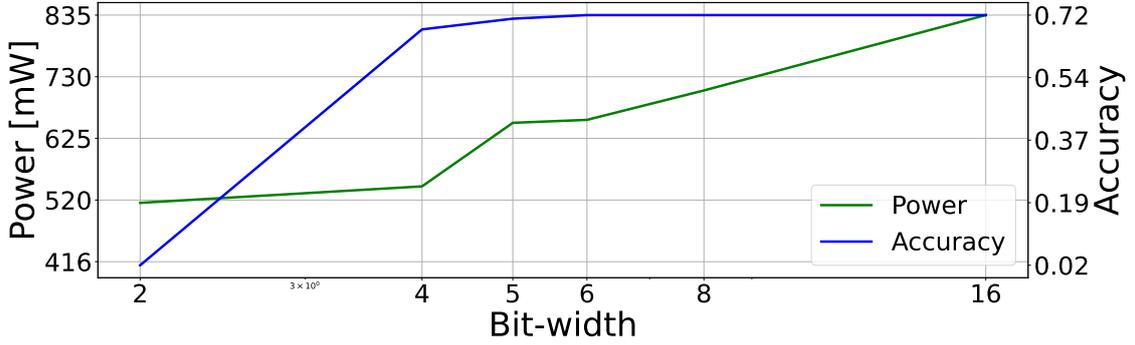
number of units working in parallel.

Finally, as explained in subsection 3.6.4, it is crucial to note that the primary constraint on Spiker+ size is the number of weights that can be accessed in parallel. A smaller weight bit-width reduces the required interconnections and the amount of memory used, including the total number of BRAMs. This not only conserves power but also facilitates the implementation of larger architectures.

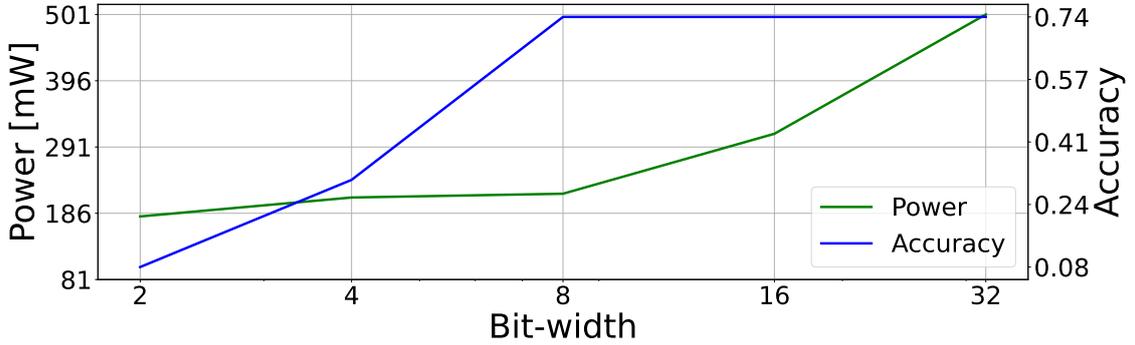
3.6.4 Performance vs. size

Finally, let us explore the model complexity achievable with Spiker+ on selected Xilinx™ FPGA boards. Synthesis results for three Xilinx™ boards, particularly low-end ones suitable for resource-constrained edge applications, are presented in Table 3.4.

On the Xilinx™ XC7Z020/XA7Z020 boards discussed in subsection 3.6.1, the



(a) Membrane potential quantization on the SHD dataset



(b) Weights quantization on the SHD dataset

Figure 3.12: Impact of quantization of neuron membrane potentials and synaptic weights on inference accuracy and power consumption for target datasets. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

Table 3.4: Synthesis of maximum size accelerator on different Xilinx™ FPGAs boards. Reproduced from Carpegna et al. 2024 [64], licensed under CC-BY.

FPGA	Net	Neur.	BRAM	LUT	Pow(mW)
XC7Z020 / XA7Z020	FFFC	1224	138	44330	1070
	RSNN	550	135	24,660	720
XCZU3EG	FFFC	1900	215	62989	1200
	RSNN	690	213	29,809	780

largest FF-FC network possible using a I-order LIF consists of 1,220 neurons, utilizing 138 BRAMs (98.5%) and 42,430 LUTs (26.7%). The BRAM size of the FPGA emerges as the limiting factor. On the slightly more advanced Xilinx™ XCZU3EG board, a larger 1,900-neuron architecture, utilizing 215 BRAMs (99.5%) and 62,989 LUTs (29.8%), can be implemented. Notably, the place-and-route algorithm encounters no obstacles, reinforcing that BRAM limitations govern the network size.

For FC-R architectures using a II-order LIF, where both feed-forward and feedback weights need storage in BRAM, the maximum network size is influenced. Specifically, it is capped at 550 neurons for Xilinx™ XC7Z020/XA7Z020 boards and 690 neurons for the Xilinx™ XCZU3EG board.

Potential issues such as place-and-route complexities or excessive power consumption due to the fully parallel nature of the accelerator are foreseeable. A prospective solution involves implementing a certain degree of time multiplexing to address these challenges while expanding the architecture size. This approach entails sharing hardware components between neurons, introducing a trade-off with performance, and is currently under study as an enhancement.

3.7 Conclusions

This chapter introduced Spiker+, a versatile framework to design low-power and resource-efficient hardware accelerators for SNNs targeting edge inference on FPGA platforms. It features a Python configuration framework that facilitates easy reconfiguration of the accelerator, allowing users to choose from six neuron models (IF, I-order LIF, and II-order LIF, each with the option of a *hard* or *subtractive* reset) and two network architectures (FF-FC and FC-R). The tool enables the automatic selection of training and quantization parameters directly through Python. The results are significant, boasting a 93.85% accuracy on MNIST, with a classification latency of $780\mu s$ per image and power consumption of $180mW$. Additionally, it achieves a 72.99% and 89.82% accuracy on SHD and AudioMNIST respectively, corresponding to a $540\mu s$ latency and power consumption of $430mW$ for the SHD, and $80\mu s$ latency and $290mW$ power for the AudioMNIST. These metrics are highly competitive compared to state-of-the-art FPGA accelerators for SNNs, demonstrating high performance in both power efficiency and area. This work lays a solid foundation for deploying specialized, low-power, and efficient SNN accelerators in resource and power-constrained edge applications. Spiker+ is a live project, and ongoing work focuses on enlarging the library of available neurons, input encoding blocks, and network architectures.

Chapter 4

SpikExplorer: hardware-oriented Design Space Exploration for Spiking Neural Networks on FPGA

Adapted, with permission, from Padovano, D.; Carpegna, A.; Savino, A.; Di Carlo, S. SpikeExplorer: Hardware-Oriented Design Space Exploration for Spiking Neural Networks on FPGA. Electronics 2024, 13, 1744., doi: 10.3390/electronics13091744. Licensed under CC-BY.

In the context of hardware accelerators for neuromorphic systems, one of the main challenges is how to construct the SNN to fit the target application best: there are many different neuron models with varying degrees of biological plausibility and computing efficiency; a single model has a lot of internal parameters to tune; the network architecture itself can be modified depending on the task to perform. A manual selection of all these hyper-parameters can be very complex and could bring a non-optimal solution. At the same time, an exhaustive search for the best configuration would require too much time, given the search space size. Automatic Design Space Exploration (ADSE) can represent a solution. However, while the literature is rich in works about ADSE in the field of CNNs [125, 126] and other ANN models [127], this is not true for SNNs. The few existing works on the topic focus on a single-objective optimization directed towards the improvement of the accuracy [128] or concentrate the search on a particular aspect of the network, like the input data encoding, using fixed neuron models and parameters and performing only a tiny grid search between a limited set of network architectures [119].

This chapter presents SpikExplorer, a flexible Hardware-Oriented ADSE framework to automatically optimize SNN models for their deployment on digital hardware accelerators targeting FPGA implementations. The tool supports multi-objective ADSE driven by power consumption, latency, area, and accuracy, leveraging Bayesian optimization. It empowers users to fine-tune network architecture, neuron models, and internal settings, explicitly tailoring them for FPGA deployment. SpikExplorer can be specialized for whatever neuron model and hardware implementation, allowing to easily customize SNN co-processors depending on the user requirements. This can help leverage the benefits of SNNs in power and resource-constrained edge applications [30], simplifying the configuration and tuning of these new networks in various problems.

The chapter is organized as follows: section 4.1 provides a background on ADSE and section 4.2 overviews related work on ADSE for SNNs. section 4.3 overviews the proposed method and section 4.4 shows its capabilities on a set of case studies. Finally, section 4.5 concludes the chapter and highlights future extensions.

4.1 Automatic Design Space Exploration

When working with complex systems such as SNNs, the numerous degrees of freedom make a comprehensive exploration of the design space impractical. This challenge is compounded when crafting specific hardware implementations, where synthesizing and simulating architectures can consume significant time. Over the past few decades, researchers have sought ways to optimize and speed up the search for optimal architectures in electronic systems, a pursuit intensified by the proliferation of AI and ANNs. One approach to reducing the search space involves randomly selecting a subset of points and focusing exploration solely on them. Despite its simplicity, this can prove effective in many cases [129]. Nevertheless, structured and systematic alternatives abound in the literature, many drawing inspiration from biological evolution, like evolutionary and genetic algorithms [130], Extremal Optimization [131], and Reinforcement Learning (RL) [132].

Among optimization techniques, Bayesian optimization [133] stands out as a robust solution, particularly for its ability to converge rapidly even with complex models. It effectively addresses the exploration-exploitation dilemma [134], balancing exploring new solutions and exploiting known ones. Instead of directly interacting with the objective function (e.g., accuracy of the network), which might be computationally expensive to evaluate, Bayesian optimization builds a simpler, approximate model. It is typically based on Gaussian processes or other probabilistic models. Initially, this surrogate model makes some assumptions about the objective function based on the limited information available. As more data points are collected through evaluations of the actual objective, the surrogate model becomes refined and better approximates the actual function. A Bayesian optimizer relies

on an acquisition function, considering both exploration and exploitation aspects to decide which point in the search space to evaluate next. Exploration involves trying out points in the search space that are uncertain or have yet to be explored to gain more information about the objective function and potentially discover better solutions. Moreover, exploitation involves focusing on areas of the search space likely to yield good results based on the current knowledge provided by the surrogate model. The acquisition function balances these two aspects to guide the search effectively. Thanks to this methodical approach, Bayesian optimization demonstrates efficacy in converging to solutions, even in scenarios involving numerous parameters in the search, rendering it a valuable tool for optimizing SNNs. Interested readers may refer to [135] for a broader topic overview.

4.2 Related works

Automatic optimization for SNN architectures is critical when considering dedicated hardware implementations of SNNs, particularly for resource-constrained edge applications. In such scenarios, the optimization of the network targets multiple objectives: together with the fine-tuning of the model on a specific problem, minimizing power consumption, area occupancy, and latency become integral parts of the optimization goals.

Within this framework, tools are available to support FPGA hardware designs. For instance, *E³NE* [120] provides a library of elementary blocks to build RTL descriptions of SNN architectures. On the other hand, Spiker+ [64] provides a framework to automatize the generation of the SNN RTL models starting from a high-level network description, providing a library of possible models and network architectures.

However, a crucial gap remains: given the availability of various neuron blocks and architectures, how can the network be optimized to achieve the highest possible accuracy while constraining other metrics such as latency, power consumption, or area? Some works on Network Architecture Search (NAS) for SNNs exist. For example, authors in [136] propose an ADSE methodology to perform a single-objective search, targeting the optimization of SNN accuracy only. They mainly focus on convolutional architectures targeting image datasets, such as CIFAR-10, CIFAR-100, and TinyImageNet, applying a NAS strategy to select between different convolutional kernel and pooling sizes. So, the target application is specific, and the work considers the software model only without considering the actual hardware implementation. On the other hand, [128] proposes NeuroXplorer, a hardware-oriented ADSE tool to optimize SNN deployment on existing neuromorphic hardware. There is no search for the network structure, neuron model, and parameters. Conversely, starting from a trained SNN model, the tool tries to organize computations to fit the target platform at best, for example, clustering groups of neurons to minimize the

transport of spikes over long distances. It focuses on the computational paradigms used within existing neuromorphic processors, such as the Dynap-se1 [33]. Eventually, the first attempt at creating an FPGA-oriented optimizer was performed in [119]. However, the work is focused on finding the optimal encoding for the input data and on the fast evaluation of the optimization metrics (such as power, area, and latency) performed with a novel system C simulator of the hardware accelerator called NAXT. The optimal SNN search is a grid search conducted within a small set of pre-defined architectures with a fixed IF neuron model without using any specific optimization algorithm.

4.3 Materials and Methods

SpikExplorer has been designed as a modular Python tool with different components connected in a closed loop. Figure 4.1 shows a high-level view of the complete framework.

The DSE engine is the core of the optimization framework. It aims at finding the optimal SNN architecture and its related parameters for a given problem within a user-defined design space. The user imposes constraints by specifying which parameters must remain fixed and which require optimization. An infinite search space risks prolonged search duration and potential converging failure. Hence, users are prompted to define search limits for each optimized parameter. This ensures that the search remains bounded and manageable. For instance, limits can be set on the maximum network size, considering the available hardware resources on the target platform.

The optimization process follows a multi-objective Bayesian approach. The user can select a set of optimization targets: accuracy, area, latency, and power. While Figure 4.1 illustrates an exploration encompassing all four potential metrics, the optimization can focus solely on a subset or even just one in the extreme case. Once the optimization objectives are defined, the DSE engine constructs a surrogate model for each of them and starts an iterative optimization process. The surrogate models determine the next point to explore at each iteration, aiming to optimize all required metrics. A point within the search space is defined by a set of values associated with the parameters used for the optimization.

For each explored design option, the specific SNN architecture and configuration is forwarded to the Network Evaluator (NE), responsible for the network construction, training, and performance evaluation. This, in turn, requires providing a training dataset. This block closes the loop by giving the DSE engine the characterization of the selected observation points in terms of accuracy, area, latency, and power required to update the internal surrogate models. This task requires comprehensively characterizing the various neuron models and computing their individual area occupancy, power consumption, and latency. SpikExplorer has been

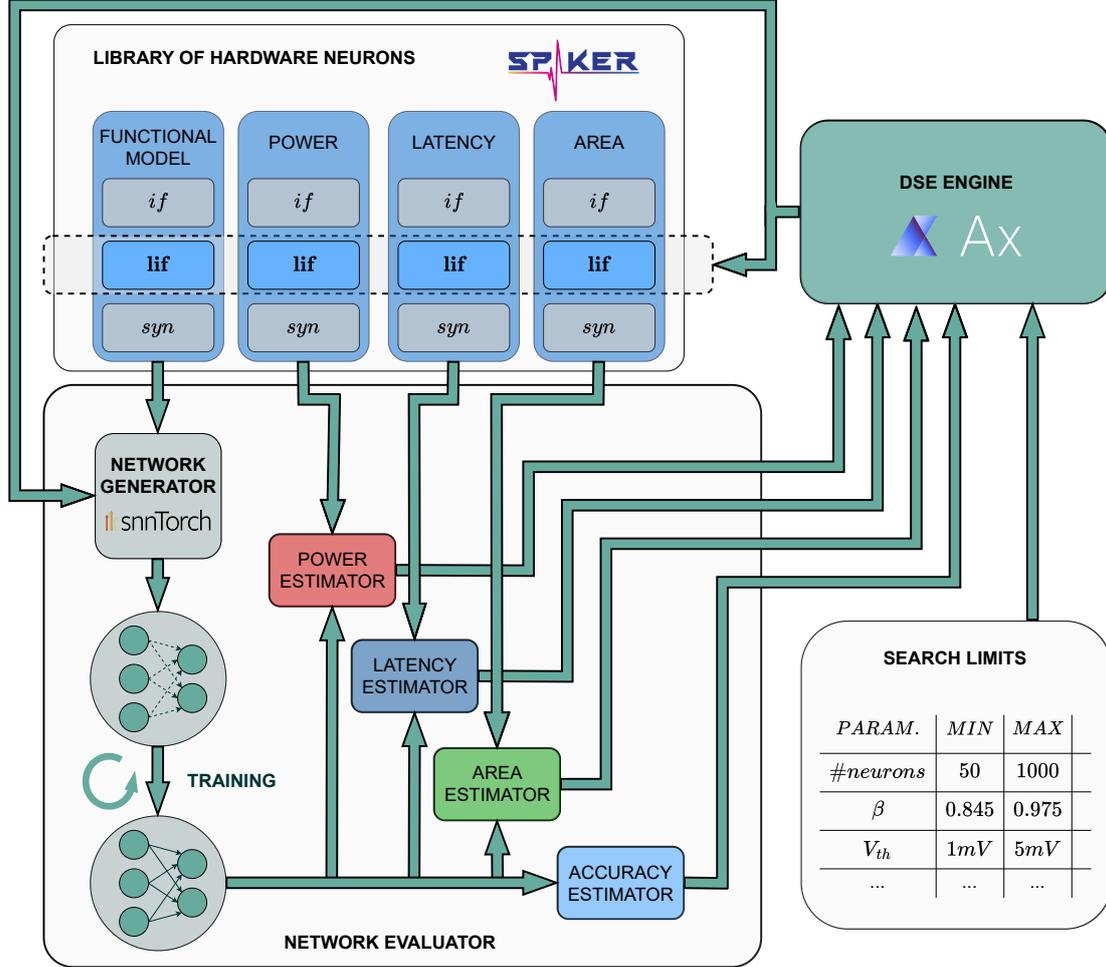


Figure 4.1: SpikExplorer general architectures, including (i) a library of hardware neurons, (ii) a network evaluator estimating the performance of selected implementations, and (iii) a Bayesian DSE engine. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

intentionally designed to be versatile and compatible with any user-defined neuron characterization model. However, this chapter utilizes a comprehensive characterization library derived from open-source experiments, leveraging the Spiker+ framework [64]. Given the framework’s complexity, the following sections overview each component separately.

4.3.1 Network Generator and hardware neurons

The Network Generator (NG) is the submodule of the NE block in charge of building the SNN network models required for performance evaluations. It involves

a set of functional models of the available neurons that can be incorporated into the network architecture. Each functional model must be characterizable for the considered optimization targets. For instance, if the optimization target is area minimization, the user must provide a characterization detailing the area occupation of each considered neuron model. This facilitates fine-tuning the search process with specific neuron implementations, which will be integrated into the customized SNN co-processor on FPGA.

In its current implementation, SpikExplorer supports a set of default neuron functional models based on the IF variants described in section 2.7. From a functional point of view, the models are defined using the `snnTorch` framework [86]. This facilitates the creation of a range of networks suitable for various problems where IF models are applicable. `snnTorch` enables the modeling and approximation of the hardware neuron behavior without a precise knowledge or description of all internal details.

Each available neuron model is associated with hardware-related information obtained using the open-source hardware models provided by the Spiker+ framework [64]. These models are synthesized on a Xilinx XC7Z020 reference FPGA board, and the corresponding performance metrics are extracted, such as area, power, and latency. The following sections outline the techniques to characterize the default SpikExplorer neuron models. This presentation aims to provide insight into the available estimates and to explain how neurons can be described to fine-tune the search on a specific implementation.

4.3.2 Area

The neuron area estimation consists of two primary components: (i) the area occupied by the computational elements and (ii) the memory utilized by the synaptic connections. Both are estimated through hardware synthesis of available implementations. Figure 4.2a shows an example of the synthesis of a simple LIF model, reporting the corresponding LUT count and the amount of memory required by synaptic weights, in this case, stored in FPGA BRAM.

Quantization is a well-known technique exploited to reduce the memory footprint of SNNs models, and several quantization frameworks exist [137, 138, 139]. To avoid overlap with existing solutions, the default neuron library provided by SpikExplorer does not aim at optimizing quantization, focusing on higher-level architectural optimizations. Therefore, the default neuron characterization uses 32-bit data representations. Experimental results later show that this data representation preserves full-precision accuracy without introducing bias from precision reductions across different models. Although the resulting architecture may appear oversized, what truly influences the optimization process is the relative dimensions of the neurons. Nevertheless, the user can enlarge the library of available neurons, including architectures with different quantization levels, to drive the search toward

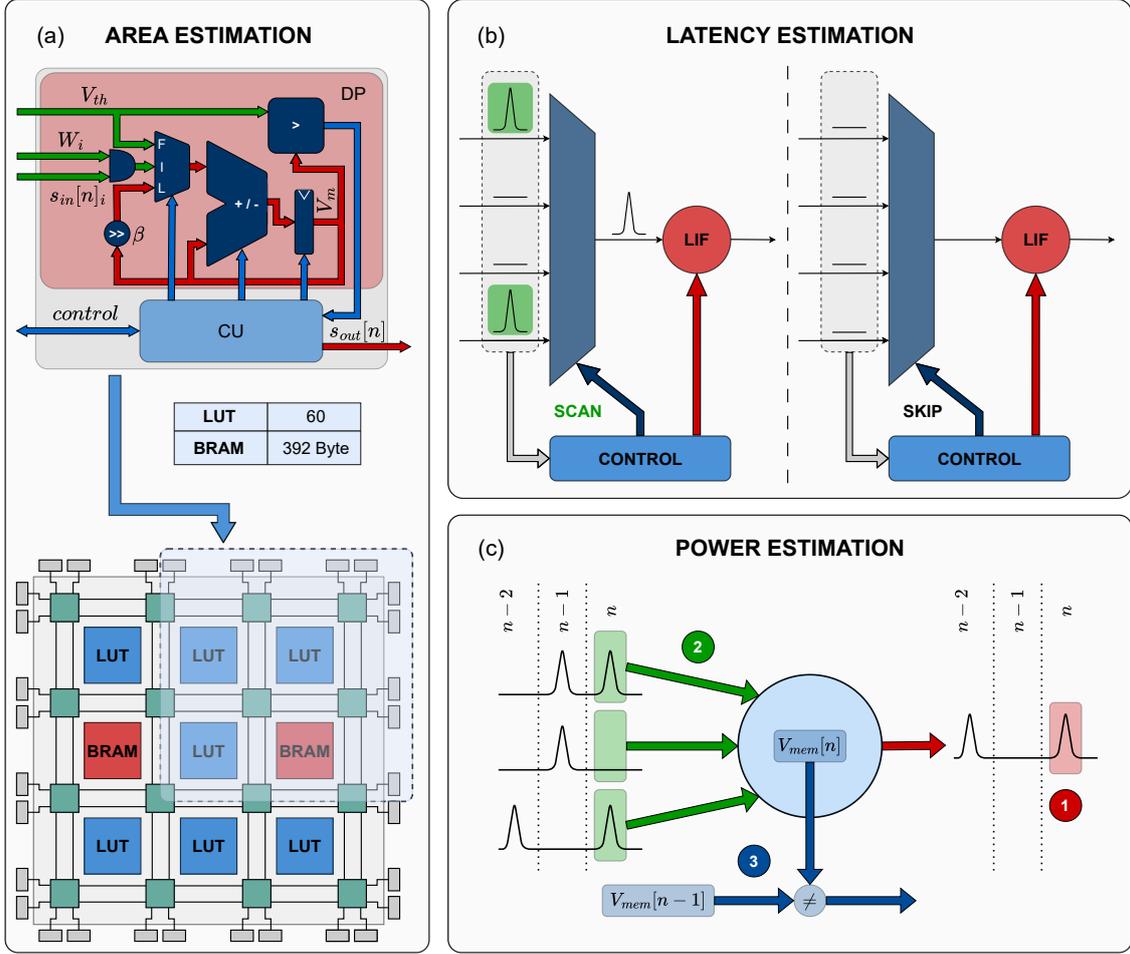


Figure 4.2: Metrics estimation: the figure graphically showcases how the different metrics considered by SpikExplorer are estimated. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

smaller neurons with more aggressive quantization.

The total number of weights is computed at run-time after defining the SNN architecture and integrated into the area estimation to account for different architectural structures and their impact on the area. This allows us to consider the diverse memory footprints of different architectural choices. For instance, in fully connected architectures with identical neuron count on each layer, a deeper network featuring smaller layers will incorporate fewer synaptic weights, thus necessitating less memory. Moreover, recurrent architectures introduce an area overhead due to FC recurrent connections that can be computed according to Equation 4.1.

$$R = \frac{\text{Recurrent layer area}}{\text{FF - FC layer area}} = \frac{N_{in} \cdot N_{neurons} + N_{neurons} \cdot N_{neurons}}{N_{in} \cdot N_{neurons}} = \frac{N_{in} + N_{neurons}}{N_{in}} \quad (4.1)$$

Here, N_{in} denotes the number of inputs, and $N_{neurons}$ signifies the number of neurons within a specific layer.

SpikExplorer measures the overall area occupancy in terms of Equivalent Look Up Table (ELUT) count:

$$N_{ELUT} = \sum_{l=0}^{N_{layers}} N_{neurons}^l \cdot (N_{LUT32} + N_{in}^l \cdot r) \quad (4.2)$$

Where l denotes the layer index, N_{layers} the total number of layers, N_{ELUT} represents the total number of ELUTs occupied by the network, N_{LUT32} is the number of LUTs required by a single neuron with a 32-bit precision, and

$$r = \begin{cases} R, & \text{if } l \text{ is recurrent} \\ 1, & \text{if } l \text{ is FF-FC} \end{cases} \quad (4.3)$$

For clarity in visualization, the distinction between FF-FC and recurrent architectures is expressed using the neuron model nomenclature. The default supported models encompass *if*, *rif*, *lif*, *rlif*, *syn*, and *rsyn*, where the prefix *r* signifies a recurrent architecture.

4.3.3 Accuracy and latency

Since quantization is not the primary focus of SpikExplorer, the accuracy estimation of various network configurations used to drive the DSE process is based on full-precision 64-bit floating-point software models constructed by the NG using the *snnTorch* framework. These estimations are crucial for guiding the optimization process but should not be regarded as precise accuracy measurements for the target hardware co-processor. They represent an upper bound on the final accuracy that depends on the quantization applied when deploying the model on a real FPGA.

In terms of latency, a clock-driven reference model is considered. In particular, SpikExplorer implements two different latency estimation models: a fixed latency model, in which each neuron is characterized by a single latency value, independent of the spiking activity, which accounts for the time required to integrate spikes and to decay or reset the neuron; and an optimized latency model, following a computational methodology like that described in [64]. In this case, two latency values are considered: a high latency occurs when at least one input spike is present, prompting neurons to scan all inputs to identify the active ones, and a low value occurs without spikes, where the scanning process is omitted. Figure 4.2b shows

the two considered cases. Since all the inputs are processed sequentially, the largest the number of inputs to a neuron, the highest will be the latency of that neuron in case it receives input spikes. The computational process is considered entirely parallel, making the overall latency independent of the overall number of neurons. The approach is highly tailored to fully parallel clock-driven implementations. Alternatively, an activity-based methodology resembling the one utilized for power consumption (refer to subsection 4.3.4) could be adopted to accommodate event-driven approaches.

4.3.4 Power

The neurons’ power consumption generally depends on their activity levels. This holds for clock-driven architectures, as evidenced in [64], and is even more pronounced in event-driven alternatives. To understand how SpikExplorer estimates the overall power consumption, it is convenient to analyze the operations involved in updating a LIF neuron. Equation 4.4 shows the mathematical operations involved, obtained by merging Equation 2.2, Equation 2.3 and Equation 2.4, setting $\alpha = 0$ and re-ordering the terms according to [92].

$$V_m[n] = \underbrace{\beta \cdot V_m[n-1]}_{(3) \text{ Leak}} + \underbrace{W \cdot s_{in}[n]}_{(2) \text{ Integrate}} - \underbrace{\beta \cdot s_{out}[n-1] \cdot R[n]}_{(1) \text{ Fire}} \quad (4.4)$$

As the name of the model suggests, the neuron executes three primary operations: *Leakage* (3), *Integration* (2), and *Firing* (1). The equation defines the evolution of the membrane potential in its discrete-time form. SpikExplorer examines the state of each neuron at every time step to evaluate the instantaneous power consumption. It expects a characterization of the power consumed by the neuron when executing each of the reported operations. The overall power consumption is then computed by averaging the instantaneous values over the entire sequence of time steps. To accomplish this task, SpikExplorer must monitor (i) the presence of an output spike, (ii) the presence of input spikes, and (iii) the value of the state variables that change dynamically during the network operations. With LIF and IF models, the only state variable involved is V_m , while with a synaptic model, I_{syn} is monitored as well. Using these, SpikExplorer understands the current state of the neuron and infers the relative consumed power, as illustrated in Figure 4.2c.

Observing the neuron’s output reveals whether the neuron has ”fired” a spike. If a spike is generated due to the threshold potential being exceeded, the membrane is reset; this is associated with a first power contribution. Inspecting the inputs, if spikes are present, they are weighted and integrated into the membrane potential, implying an additional power contribution. Eventually, without spikes, the membrane decays toward its resting value, consuming extra power. This condition happens when the membrane potential at time step n differs from that at time step

$n - 1$. This approach facilitates a highly adaptable evaluation. For instance, in clock-driven update policies, decay consumes power at every time step, which can be factored into the Leak contribution. Conversely, in an event-driven approach, computations occur solely in the presence of input spikes, potentially resulting in zero power consumption for the leak term. In this case, the decay power can be merged into the "integrate" term. Alternatively, a custom functional model can be used for the neuron, in which the membrane is updated only when input spikes are received. Here, by checking whether the membrane has changed value, the decay contribution can be considered only in the presence of input stimuli. Lastly, depending on factors like recent resetting or reaching asymptotic decay values due to finite precision platforms, the neuron may remain in a constant state without necessitating significant power-consuming updates.

4.3.5 DSE engine

As detailed in section 4.1, Bayesian optimization emerges as the preferred method for DSE in SNNs. This preference stems from several factors, including the abundance of tunable hyperparameters, inherent noise in the objective function due to the spiking information encoding, and the long training times associated with large SNNs. Bayesian optimization is advantageous for its rapid convergence, facilitated by a simplified surrogate model, and its inherently parallelizable nature, accelerating the exploration process.

The DSE engine of SpikExplorer is built resorting to the Adaptive eXperimentation (AX) optimization package, an open-source solution developed at MetaTM [140]. It provides high-level Application Programming Interfaces (APIs) that SpikExplorer uses to iterate through the optimization efficiently. Listing 4.1 shows a summarized version of the code used to perform the optimization. The DSE engine receives in input a set of configuration parameters, indicating the number of iterations involved in the optimization (line 7), the objectives of the search (line 8), the metrics to optimize, each associated with the range in which to perform the search (line 9), and the set of candidate neuron architectures, including the functional models and their characterization (line 10). The optimization process (lines 12-40) starts initializing the Bayesian surrogate model using the AX APIs (lines 14-20) and then performs an iterative procedure (lines 24-38). A set of parameters is selected at each iteration, following the predictions performed with the surrogate model (line 27). The network is configured with the chosen parameters (line 29), and its performance is evaluated (line 30). The results are then provided to the optimizer (line 33), which uses them to update the surrogate model (line 36). The process continues until the required number of iterations is completed (line 24). The full set of explored points (line 38) is returned (line 40). This can be used to find the best configurations on the Pareto frontier and select the configuration that best fits the desired requirements.

```
1 from ax.service.ax_client import AxClient
2
3 class SpikExplorer:
4
5     def __init__(self, config: dict):
6
7         self.num_trials      = config.get("num_trials")
8         self.objectives      = config.get("objectives")
9         self.search_param    = config.get("search_param")
10        self.neurons          = config.get("neurons")
11
12    def optimize(self):
13
14        dse_engine = AxClient()
15
16        # Initialize Bayesian optimization
17        dse_engine.create_experiment(
18            parameters=self.search_params,
19            objectives=self.objectives,
20        )
21
22        search_points = []
23
24        for _ in range(self.num_trials):
25
26            # Select initial point in the search space
27            net_config = dse_engine.get_next_trial()
28
29            snn = self.net_generator(net_config)
30            results = self.train_evaluate(snn)
31
32            # Give the results to the optimizer
33            dse_engine.complete_trial(results)
34
35            # Update the Bayesian surrogate model
36            dse_engine.update()
37
38            search_points.append((net_config, results))
39
40    return search_points
```

Listing 4.1: Summarized code of spike explorer

Table 4.1 displays the available optimization parameters, organized into three groups: network architecture (net), neuron model (neuron), and training process (training). Numeric parameters are "discrete" or "continuous" ranges. In the former case, only discrete integer values within the specified range are considered, while in the latter case, a continuous interval of real values is analyzed. Additionally, numeric values can be defined as sets of predefined values to try. For non-numerical parameters, enumerative lists of options are used.

Regarding network architecture, SpikExplorer offers constraints for optimizing the model. These constraints include the number of layers to use (discrete range), the number of neurons in each layer (set of options), and the network architecture (feed-forward or recurrent). As exploring the dimensionality of the network is computationally intensive, selecting the number of neurons per layer from a set allows for reducing the search space by performing a coarser search among a predefined range of layer sizes. Conversely, for a finer search, SpikExplorer can be left to select any layer size, and a set containing all integer numbers between the desired minimum and maximum can be provided. The final parameter related to the network allows for including recurrent connections within layers. This specification occurs at the network level, configuring the entire network with the specified layer type. Hybrid solutions are not currently considered in the search process. As discussed in subsection 4.3.2, rather than directly specifying whether layers must be recurrent, users can select models that inherently incorporate recurrence.

Nearly all internal parameters can be adjusted at the neuron level after selecting a specific model among the six options listed in Table 4.1 and elaborated upon in subsection 4.3.2. The reset mechanism can be configured as hard or subtractive (refer to Equation 2.3). Optimization of the exponential decay for both the synaptic current and membrane potential can be achieved through the α and β parameters ($0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$). In this case, the search can involve continuous values, with users specifying the limits of the search range or selecting from a predefined set of powers of two. The last option aligns with hardware optimization principles, where using powers of two allows for replacing the multiplication involved in exponential decay with a simple bit shift, as demonstrated in [64]. Given that exponential decay generally does not require rapid attenuation, α and β typically approach values close to one. Consequently, the search primarily focuses on the upper portion of the interval $[0, 1]$, utilizing the expression outlined in Table 4.1. Additionally, the firing threshold can be adjusted within a continuous range of values to regulate neuron activity. Finally, users can select the number of time steps involved in computation by specifying a set of values. Similar to the approach for choosing the number of neurons, users can limit the set of sequence lengths, tailoring the set's granularity based on the desired search precision. Alternatively, to grant SpikExplorer flexibility in selecting from all possible sequence lengths, users can provide a set containing all integer numbers between the desired minimum and maximum.

Table 4.1: Set of specifications that the user can provide. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

	Parameter	Values	
Net	# layers	Discrete	
	# neurons/layer	Set	
	Architecture	Feed Forward	Recurrent
Neuron	Model	if lif syn	rif rlif rsyn
	Reset	Hard	
		Subtractive	
	α, β	Continuous	
	V_{th}	$1 - 2^{-n}$	
	Time-steps	Set	
Training	Learning rate	Continuous	
	Optimizer		
	Regularizer		
	Surrogate slope		
	Surrogate	Sigmoid, Fast Sigmoid, ATan, Straight Through Estimator, Triangular, SpikeRateEscape, Custom [86]	

In addition to tuning the network architecture, SpikExplorer offers optimization options for the training process. This includes fine-tuning parameters such as the learning rate, optimizer settings —such as the Adam [141] parameters β_1 and β_2 , controlling the decay rates of moving averages of gradients and squared gradients respectively, and influencing the retention of historical information when updating model parameters —regularization parameters like λ , affecting the strength of L1 and L2 regularization [142], and modifying the penalty for large weights, and the surrogate function employed in the backward pass, as elaborated in section 2.9. In this scenario, SpikExplorer can select the function itself and adjust its steepness.

Given the many parameters involved, optimization efforts can focus on specific subsets. An illustration of such a targeted search is presented in section 4.4.

4.4 Experimental results

This section demonstrates the capabilities of SpikExplorer through selected case studies designed to test its internal optimization engine.

4.4.1 Experimental set-up

The exploration capabilities of SpikExplorer were evaluated using three distinct datasets, each with varying complexity and characteristics commonly employed for benchmarking SNNs:

1. MNIST [61]: grey-scale images of handwritten digits, converted into sequences of spikes using rate encoding. The corresponding number of inputs is $28 \times 28 = 784$.
2. SHD [90]: audio recordings of numbers pronounced in English and German, converted to spikes through a faithful emulation of the human cochlea. Recordings were performed with 700 channels, corresponding to the number of inputs of the network.
3. DVS128 [143]: video recordings of 11 gestures through a DVS converting images into spikes. The sensor’s resolution is 128×128 pixels, accounting for 16,384 inputs.

Two optimization experiments were conducted using the three datasets. In the first experiment, a broad exploration was undertaken, allowing SpikExplorer the freedom to optimize the training process while seeking optimal neuron models and network architectures. The objective was to minimize area and power consumption while maximizing accuracy. The search parameters provided to SpikExplorer are detailed in Table 4.2. The exponential decay rates were set to $\alpha = 0.9$ and $\beta = 0.82$ and maintained constant throughout the search process. The number of optimization iterations was selected to constrain the optimization time. It was set to 25 for MNIST and DVS128. Conversely, achieving acceptable accuracy with SHD requires more training epochs, so the number of search iterations was capped at 15 to control search duration. The second experiment focused on a more specific optimization goal. Here, the neuron model was fixed initially, and attention shifted to optimizing the total number of neurons within the network.

Lastly, hardware synthesis of the optimized architecture identified by SpikExplorer for the MNIST dataset was conducted to allow for a comparison with state-of-the-art FPGA accelerators for SNNs. The dataset is typically used as the reference benchmark to evaluate ML model in general and SNN accelerators specifically. The target hardware platform is a *PYNQTM – Z2* board, hosting a *Xilinx[®] Zynq – 7000 XC7Z020 – 1CLG400C* System on Chip (SoC). This

Table 4.2: Set of experimental parameters provided to SpikExplorer. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

	MNIST		SHD		DVS128	
	min	max	min	max	min	max
Learning rate	10^{-4}	$1.2 \cdot 10^{-4}$	10^{-4}	$1.2 \cdot 10^{-4}$	10^{-4}	$1.2 \cdot 10^{-4}$
Adam β_{optim}	0.9	0.999	0.9	0.999	0.9	0.999
# layers	1	3	1	3	1	3
Model	lif, syn, rlif, rsyn		lif, syn, rlif, rsyn		lif, syn, rlif, rsyn	
Reset	subtractive		subtractive		subtractive	
Time steps	10, 25, 50		10, 25, 50		10, 25, 50	
# neurons/layers	200, 100, 50		200, 100, 50		200, 100, 50	
Search iterations	25		15		25	
Training epochs	50		100		50	

features the XC7Z020 FPGA, and a Dual ARM[®] CortexTM – A9 MPCoreTM. The FPGA can be programmed with the free version of the Xilinx[®] Vivado suite, making the results strongly reproducible.

4.4.2 Global Exploration

Figure 4.3, Figure 4.3 and Figure 4.3 summarize the performance of SpikExplorer when optimizing the network architecture and parameters for the three selected datasets. The figure demonstrates a strong correlation between power consumption (figure on the left in Figure 4.3, Figure 4.3 and Figure 4.3) and area (figure on the right in Figure 4.3, Figure 4.3 and Figure 4.3). While this behavior is expected, it is noteworthy because previous publications predominantly emphasized the correlation between power consumption and spiking activity [64]. For the MNIST dataset (refer to Figures 4.3a and 4.3b), non-recurrent models emerge as the preferred choice. This preference is evident from the Pareto frontier, where virtually all top-performing models are *lif* and *syn* without recurrence. Notably, the highest accuracy is achieved with a first-order LIF model, devoid of any feedback connection (refer to Table 4.3). This is consistent with expectations since simple architectures without explicit recurrent connections should be enough, given the static nature of MNIST data transformed into spike sequences via rate coding. In this scenario, crucial information is not embedded in the temporal dimension but encoded in the average spike sequence rate. Consistently with what is expected, SpikExplorer converges towards these more straightforward solutions.

Conversely, in the case of SHD and DVS128, acquired through biologically inspired sensors and containing substantial information in spike timing, SpikExplorer

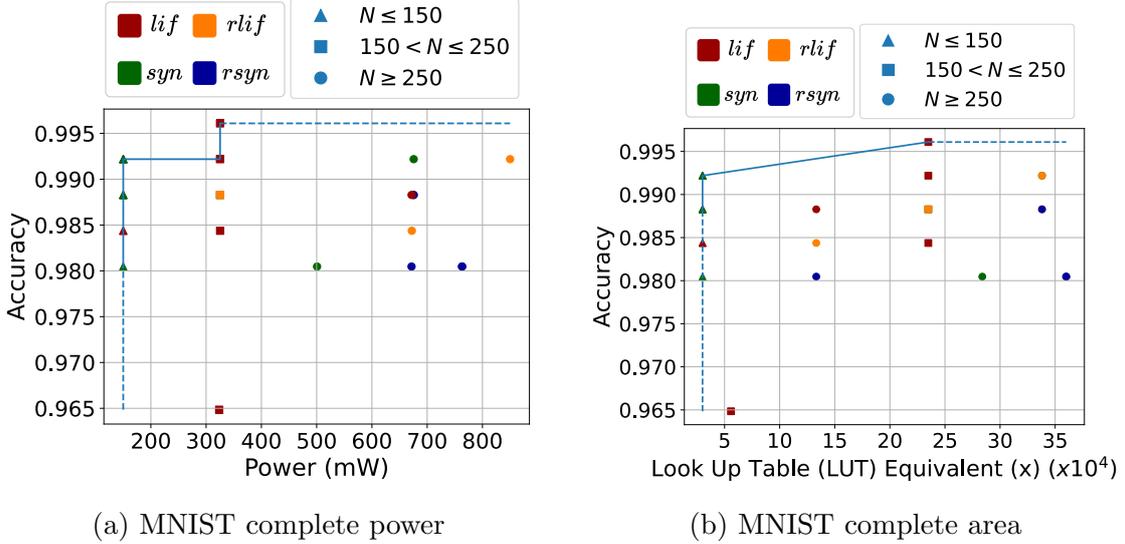


Figure 4.3: Pareto frontiers of the global exploration on the MNIST dataset targeting power, area, and accuracy optimization. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

generally leans towards recurrent structures such as *rlif* and *rsyn*, along with higher-order models (*syn*). Specifically, for SHD, a recurrent structure comprising *rlif* neurons emerges as the favored solution. At the same time, the search tends to diversify more toward both *rsyn* and *rlif*, occasionally incorporating *syn* instances for the DVS128.

It is noteworthy to observe how SpikExplorer can discover superior architectures in terms of accuracy by utilizing the same neuron model and comparable numbers of neurons while playing on other parameters, allowing us to keep the power consumption unchanged while better tuning them on the target task. This is visible when looking at the left section of the Pareto frontier across all three datasets.

In summary, Tables 4.3, 4.4, and 4.5 showcase the top-1 accuracy optimized SNN architecture, parameters, and performance identified by SpikExplorer for the three benchmarks, categorized by neuron model. The optimization is performed according to the setup summarized in Table 4.2.

To showcase the capability of SpikExplorer across different use cases, this study limited the maximum number of time steps to 50 to mitigate training time. However, upon reviewing the accuracy achieved by the optimized models, it seems reasonable to assume that these datasets may benefit from longer sequences. For instance, [64] reports a 75% accuracy for SHD with a 200-20 network using *rsyn* neurons and 100-time steps. Conversely, models tailored for MNIST can achieve nearly state-of-the-art accuracies with minimal time steps. Regarding architectures, the search for DVS128 tends toward larger structures, which correspondingly

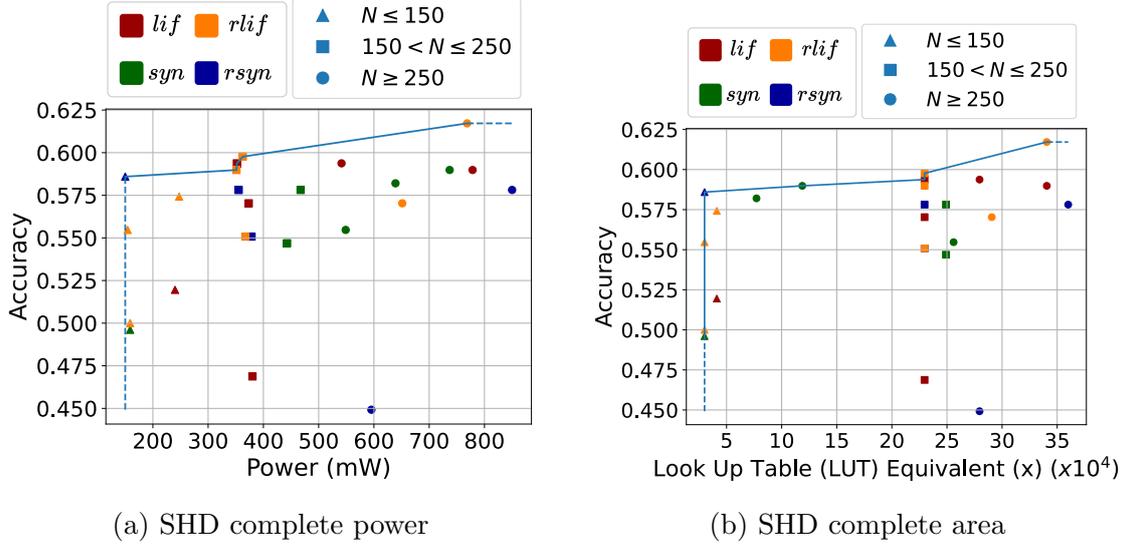


Figure 4.4: Pareto frontiers of the global exploration on the SHD dataset targeting power, area, and accuracy optimization. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

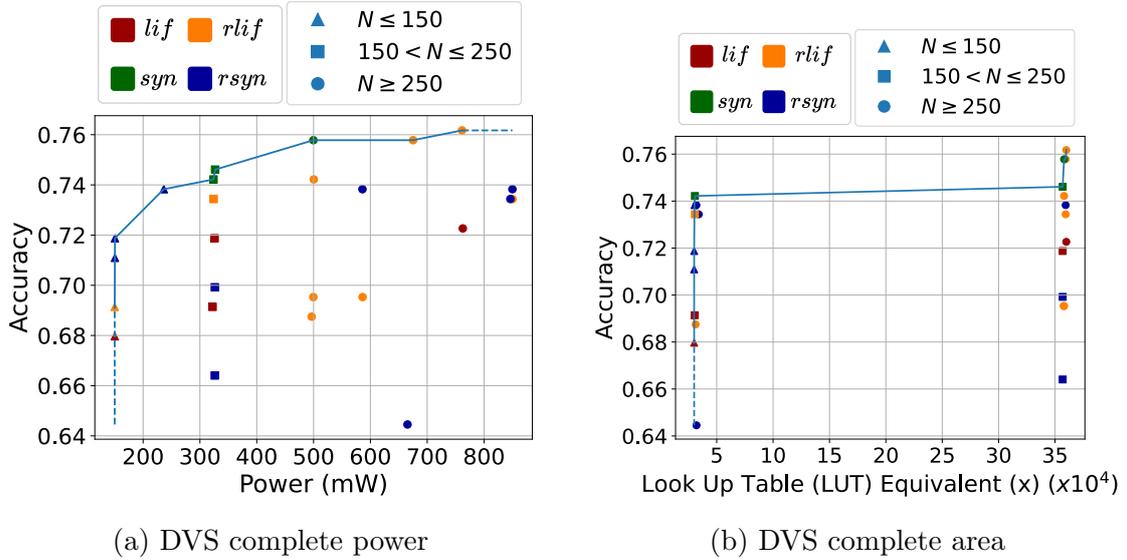


Figure 4.5: Pareto frontiers of the global exploration on the DVS dataset targeting power, area, and accuracy optimization. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

increases power consumption.

In terms of computing time, the exploration took approximately 5 hours for both MNIST and DVS, and approximately 16 hours for SHD, conducted on an

AMD Ryzen 9 7950X 16-Core Processor and an Nvidia RTX400 GPU. It is worth noting that the primary time consumption arises from training the network, which is more time-intensive for recurrent models than non-recurrent models due to the inability to accelerate the explicit time dependence through GPU.

Table 4.3: Best architectures with the four neuron models on the MNIST. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

Model	Arch.	TS	Acc.	Power (mW)
LIF	200-10	10	99.61%	310
RLIF	200-100-200-10	10	99.22%	860
SYN	200-200-10	25	99.22%	680
RSYN	100-10	25	99.22%	140

Table 4.4: Best architectures with the four neuron models on the SHD. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

Model	Arch.	TS	Acc.	Power (mW)
LIF	200-20	50	59.41%	360
RLIF	200-200-20	50	61.70%	760
SYN	100-100-200-20	10	58.98%	720
RSYN	100-20	50	58.59%	140

Table 4.5: Best architectures with the four neuron models on the DVS. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

Model	Arch.	TS	Acc.	Power (mW)
LIF	200-200-50-11	50	72.27%	500
RLIF	200-200-50-11	25	76.17%	760
SYN	200-100-11	50	75.78%	500
RSYN	100-200-50-10	50	73.83%	590

4.4.3 Fixed neuron models and network size

After showcasing the overall optimization capabilities of SpikExplorer, additional experiments were performed to highlight its behavior in constrained optimization problems.

Figure 4.6 showcases the capability of SpikExplorer to optimize the network with

a predefined neuron model, solely using the network architecture and parameters. In this case, the Pareto frontiers are dominated by small architectures ($N \leq 250$). It is interesting to observe that the optimizer is very effective when selecting the network architecture: for example, for the MNIST, an architecture with 200-10 neurons (square on the top left of Pareto frontier) can obtain the same accuracy of bigger solutions (circles on the top right of the Pareto frontier), reducing the power requirements by factors of 1.5 and more than 2, respectively. Results reported in Figure 4.6 also highlight the capability of SpikExplorer in supporting designers in finding the suitable trade-off between different metrics. For example, the Pareto frontier in Figure 4.6b shows that the power can be reduced almost three times by accepting an accuracy loss of 3%. Interestingly, the accuracy on the DVS128 is pushed up to the best value of 81.6%, improving by around 5% concerning a more agnostic search, indicating that a more specialized search can reach even better results.

Finally, Figure 4.7 shows the behavior of SpikExplorer when constraining the total number of neurons to 200 to study how different models behave. Interestingly, this produces different observations compared to results reported in subsection 4.4.2. The optimization for SHD privileges now the *syn* model, either with or without recurrent connections, while the *lif* model dominates the Pareto frontier in the DVS128 case. Again, the top-1 accuracy is increased, even if less than in the search with a fixed neuron model, reaching around 80%. This again supports the utility of a tool like SpikExplorer when exploring different design opportunities.

4.4.4 Synthesis and comparison with State of Art

As discussed in section 4.3, the power and area values provided by SpikExplorer are estimations to guide the DSE process and do not represent the actual values of the final FPGA implementation of the respective model. To obtain actual values and compare the performance of the models optimized by SpikExplorer with state-of-the-art SNN accelerators designed for FPGAs, a synthesis of an optimized architecture was conducted using the Spiker+ framework to generate the VHDL description [64]. This process generated the hardware implementation of a 128-10 architecture optimized with SpikExplorer for the MNIST dataset. This architecture is compared with other accelerators in Table 4.6.

An important observation is that the same architecture, with the same neuron model used in [64], is considered for a direct comparison. All other parameters are optimized following the approach outlined in Subsections 4.4.2 and 4.4.3. In this scenario, SpikExplorer optimizes the model by reducing the time steps from 100 to 16, decreasing the overall latency by more than six times, from $780\mu s$ to $120\mu s$. Simultaneously, the optimized training increases the accuracy by almost 3%, reaching 95.8%, thereby establishing the new optimized model as the best one among those considered, both in terms of power consumption and latency, while also positioning

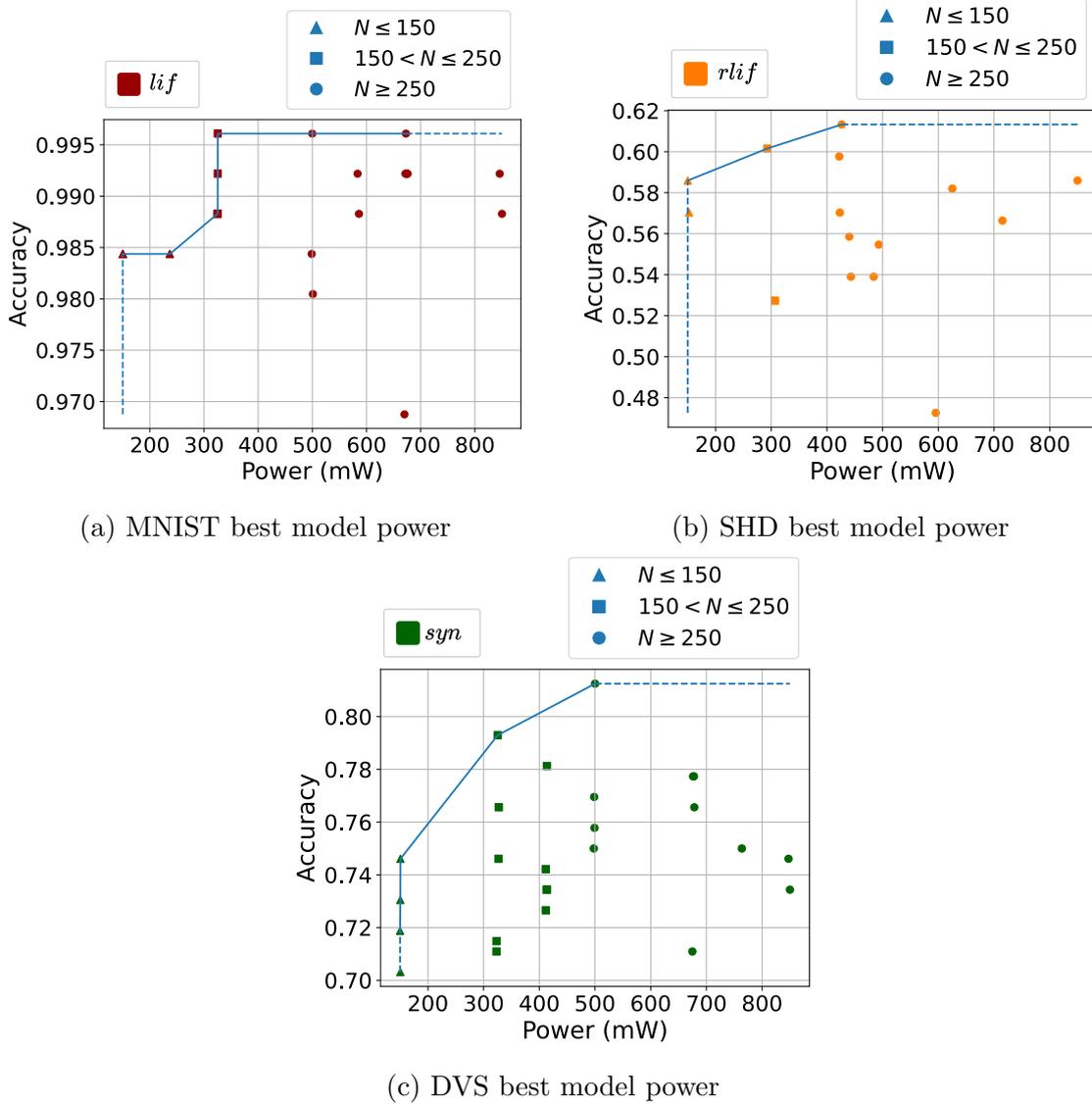


Figure 4.6: Pareto frontiers of the exploration with top accuracy neuron model for each benchmark. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

it close to the best-performing model in terms of accuracy [114]. Thus, SpikExplorer demonstrates its capability to enhance the design of FPGA accelerators for SNNs, simplifying the selection of the optimal architecture and effectively tailoring it to the desired application. It must be noted that SpikExplorer can optimize a target accelerator, starting from an existing set of hardware blocks. If the goal is to optimize latency and power further, the tool requires a more efficient neuron implementation tailored explicitly for low-power or high-performance applications.

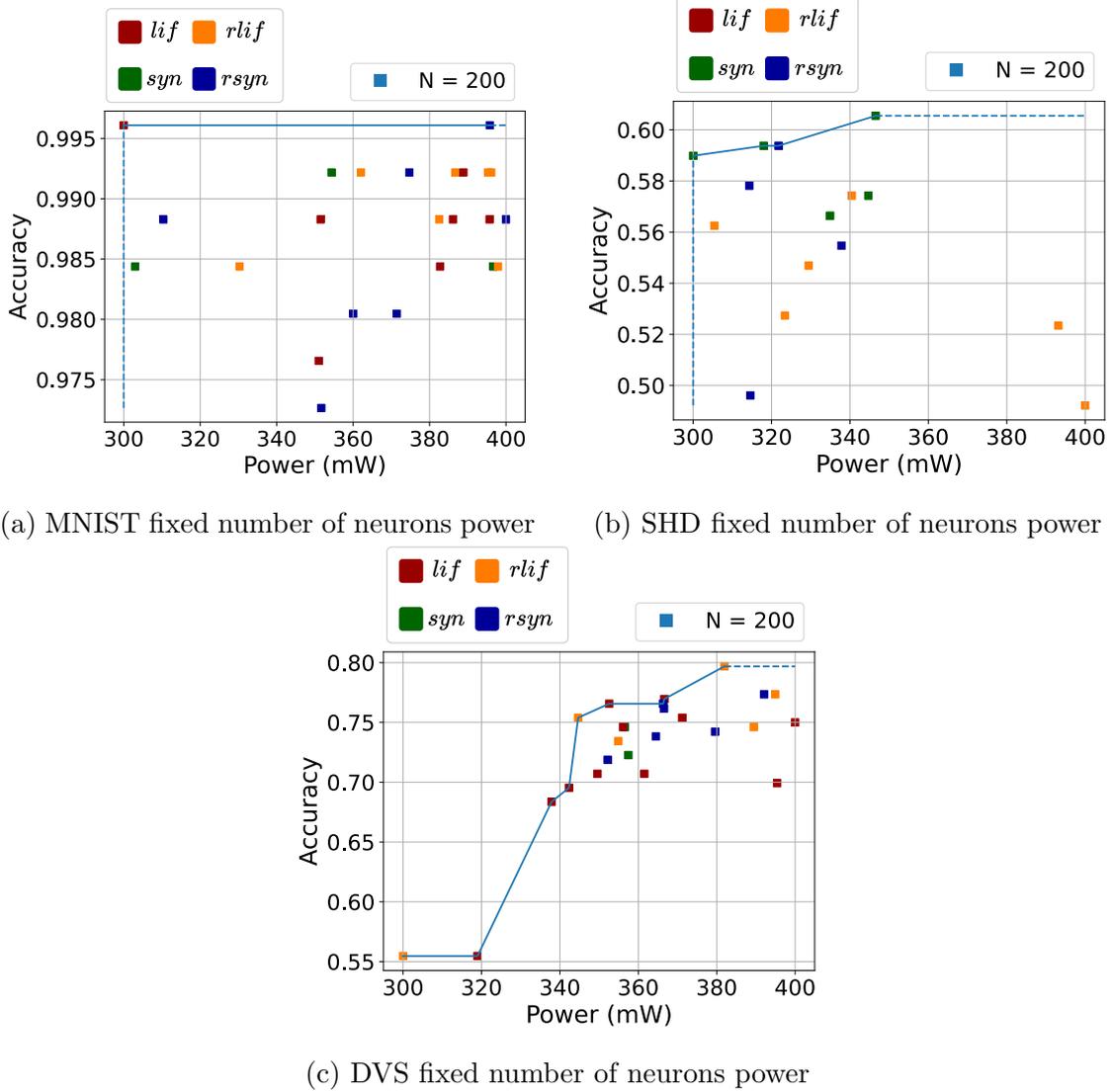


Figure 4.7: Pareto frontiers of the exploration with the number of neurons constrained to 200 for each benchmark. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

4.5 Conclusions and future work

This chapter introduced SpikExplorer, a tool tailored for hardware-centric ADSE in SNNs. Specifically designed for crafting and fine-tuning specialized hardware accelerators intended for deployment on FPGA, this tool showcases the effectiveness of Bayesian optimization within the context of SNNs. It enables an easy and flexible multi-objective search, considering model accuracy and critical hardware-specific metrics such as power consumption, area utilization, and latency. The design of

Table 4.6: Comparison of SpikExplorer to state-of-the-art FPGA accelerators for SNNs. Reproduced from Padovano et al. 2024 [92], licensed under CC-BY.

Design	Han et al. [114]	Gupta et al. [110]	Li et al. [118]	Spiker [89]	Spiker+ [64]	This work
Year	2020	2020	2021	2022	2024	
f_{clk} [MHz]	200	100	100	100		
Neuron bw	16	24	16	16	6	
Weights bw	16	24	16	16	4	
Update	Event	Event	Hybrid	Clock		
Model	LIF	LIF [144]	LIF	LIF		
FPGA	XC7Z045	XC6VLX240T	XC7VX485	XC7Z020		
Avail. BRAM	545	416	2,060	140		
Used BRAM	40.5	162	N/R	45	18	
Avail. DSP	900	768	2,800	220		
Used DSP	0	64	N/R	0		
Avail. logic cells	655,800	452,160	485,760	159,600		
Used logic cells	12,690	79,468	N/R	55,998	7,612	
Arch	1024-1024-10	784-16	200-100-10	400	128-10	
#syn	1,861,632	12,544	177,800	313,600	101,632	
T_{lat}/img [ms]	6.21	0.50	3.15	0.22	0.78	0.12
Power [W]	0.477	N/R	1.6	59.09	0.18	
E/img [mJ]	2.96	N/R	5.04	13	0.14	0.02
E/syn [nJ]	1.59	N/R	28	41	1.37	0.22
Accuracy	97.06%	N/R	92.93%	73.96%	93.85%	95.8%

SpikExplorer builds upon three open-source projects: `snnTorch`, `AX`, and `Spiker+`. Being open-source, SpikExplorer offers a robust solution for optimizing SNNs.

The capabilities of SpikExplorer were evaluated across three distinct tasks: static image recognition using the MNIST dataset, a prevalent benchmark in ML; speech recognition on the SHD dataset; and gesture recognition on the DVS128 dataset. In the MNIST scenario, the tool achieved outstanding performance, surpassing existing solutions in terms of latency by classifying images in approximately $120\mu s$ while consuming minimal power (180mW) and achieving high accuracy (95.8%). On the SHD task, it encountered challenges, achieving a top-1 accuracy of approximately 62%, possibly due to the limited number of time steps used for spike sequences during optimization. Regarding the DVS128 dataset, SpikExplorer delivered promising results, achieving 81.6% top-1 accuracy. Notably, the high dimensionality of the inputs of this dataset, with 128×128 event-based channels, made the use of FC networks suboptimal and fully parallel processing infeasible. Nevertheless, this dataset served as a valuable case study for evaluating the optimization tool with a complex dataset.

Despite the promising experimental results, additional testing with more complex case studies will be conducted to identify and solve cold boot and scalability issues that may affect Bayesian Optimization. Future work also involves expanding the framework’s scope to encompass different architectures such as Convolutional Spiking Neural Network (CSNN) and generalizing the tool to accommodate diverse computing paradigms like event-driven processors. Despite not explicitly

tailored for such hardware accelerators, SpikExplorer exhibits considerable flexibility, supporting custom neuron models and configurable metric assessments during optimization. This lays a solid foundation for automating the optimization of SNN co-processors, thereby facilitating the adoption of neuromorphic solutions in resource-constrained edge applications.

Chapter 5

Compressed Latent Replays for Lightweight Continual Learning on Spiking Neural Networks

©2024 Institute of Electrical and Electronics Engineers (IEEE). Adapted, with permission, from Alberto Dequino, Alessio Carpegna, Davide Nadalini, Alessandro Savino, Luca Benini, Stefano Di Carlo, Francesco Conti, "Compressed Latent Replays for Lightweight Continual Learning on Spiking Neural Networks," 2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Knoxville, TN, USA, 2024, doi: 10.1109/ISVLSI61997.2024.00052

Edge AI models are susceptible to errors, once deployed, due to shifts in data distribution between training and operational environments [145]. Also, an increasing number of applications require adapting AI algorithms to individual users while maintaining privacy and minimizing internet connectivity.

Continual Learning (CL) — i.e., the ability to continually learn from evolving environments, without forgetting previously acquired knowledge — emerges as a novel paradigm to address these challenges. *Rehearsal-based* methods [146, 147], the most accurate CL solutions, mitigate forgetting by continually training the learner on a mix of new data and a stored set of samples from previously learned tasks, albeit at the expense of additional on-device storage. *Rehearsal-free* methods [148, 149] rely on tailored modifications to network architecture or learning strategy to ensure model resilience to forgetting, without saving samples on-device, but with a potential trade-off in accuracy. CL at the edge, especially Rehearsal-based methods, can be resource-intensive for various ANN models, as CNNs, demanding substantial on-device storage for complex learning data.

Also in this context, SNNs emerge as a promising energy-efficient paradigm. While online learning has been explored in both hardware and software SNNs implementations [107], CL strategies in SNNs are only partially investigated in

Rehearsal-free methods [150, 151, 152]. At the moment of writing this thesis, only Proietti, et al. [153] addressed Rehearsal-based CL in SNNs. However, their work shows limited accuracy and does not optimize memory storage, which is vital for edge devices. This chapter introduces a new rehearsal-based CL solution for SNNs with the following contributions:

- A cutting-edge, memory efficient implementation of Rehearsal-based CL for SNNs, designed to seamlessly integrate with resource-constrained devices. We enable CL on SNNs by exploiting a Rehearsal-based algorithm — i.e., *Latent Replay (LR)* — proving to achieve State-of-the-Art classification accuracy on CNNs [147].
- A novel approach to reduce the rehearsal memory, based on the robustness of information encoding in SNNs to precision reduction, which allows us to apply a lossy compression on the time axis.
- The method is validated targeting a keyword spotting application, in which a recurrent SNN must recognize a predefined set of words, on two common CL setups: *Sample-Incremental* and *Class-Incremental* CL.
- Finally, to highlight the proposed approach’s robustness, it is tested in an extensive Multi-Class-Incremental CL routine, learning 10 new classes from an initial set of 10 pre-learned ones.

In the *Sample-Incremental* setup, a Top-1 accuracy of 92.46% was achieved on the SHD test set, requiring 6.4 MB for the LRs. This occurs while incorporating a new scenario, for which the accuracy is improved by 23.64%, without forgetting the previously learned ones. In the *Class-Incremental* setup, a Top-1 accuracy of 92% was attained, requiring 3.2 MB, when learning a new class with an accuracy of 92.50%, accompanied by a loss of up to 3.5% on the old classes. When jointly applying compression and selecting the most performing LR index, the required memory was reduced by 140× for the rehearsal data, while losing only up to 4% accuracy, with respect to the naïve approach. Moreover, in the *Multi-Class-Incremental* setup, a 78.4% accuracy was achieved with compressed rehearsal data, while learning the set of 10 new keywords. These results pave the way to a new low-power and high-accuracy approach for CL on edge.

5.1 Related Work

CL on ANNs involves two main approaches: Rehearsal-based and Rehearsal-free methods. In Rehearsal-based methods, mitigating catastrophic forgetting involves training the learner on a mix of newly acquired data and samples from previously

learned tasks. To increase learned classes, iCaRL [146] utilizes a set of representative old class examples as rehearsal data, chosen to maintain a balanced set of classes. To improve the efficiency of Rehearsal-based methods, Pellegrini et al. [147] propose storing rehearsal data as Latent Replays, i.e., activations produced as the output of one of the hidden layers of the learner, specifically a feed-forward CNN. Only the last layers are retrained, while the earliest backbone’s weights are frozen. Results, compared to iCaRL, demonstrate three orders of magnitude faster execution, with an almost 70% improvement in Top-1 accuracy on a Class-Incremental setup using the CORe50 dataset. In Rehearsal-free methods, addressing forgetting involves modifying the learner’s architecture or customizing the training procedure [148, 149]. However, their accuracy lags behind Rehearsal-based approaches. The Rehearsal-based approach using LRs was adopted, given its demonstrated effectiveness on CNNs.

Continual Learning in SNNs has primarily focused on Rehearsal-free methods. Skatchkovsky et al. [150] propose a Bayesian continual learning framework, providing well-calibrated uncertainty quantification estimates for new data. In contrast, the SpikeDyn framework [151] introduces unsupervised continual learning on a model search algorithm, supporting adaptive hyperparameters. However, this approach performs poorly, achieving 90% accuracy on average for a new class in the MNIST dataset while only maintaining 55% accuracy on the old classes. Drawing inspiration from human brain neurons, Han et al. [152] propose Self-Organized Regulation networks, capable of learning new tasks by self-organizing the neural connections of a fixed architecture. Additionally, they simulate irreversible damage to SNNs structures by pruning the models. Results on the CIFAR100 and Mini-ImageNet datasets, using DNN-inspired Convolutional and Recurrent networks, demonstrate an average accuracy of around 80% and 60%, respectively, for all learned classes. To the best of our knowledge, only the work of Proietti, et al., [153] targets Rehearsal-based CL on SNNs. In their approach, a Convolutional SNN model is trained to learn in a Class-Incremental and Task-free manner on the MNIST dataset, learning multiple binary classification tasks in sequence. However, their exploration doesn’t involve any memory optimization technique, storing raw data for the rehearsal phase. Additionally, they report a top-1 accuracy of 51% after learning sequentially the 10 target classes, while requiring 16MB of rehearsal memory.

To improve accuracy and optimize memory efficiency, the rehearsal-based domain was explored, applying efficient time-compressed LRs to SNNs.

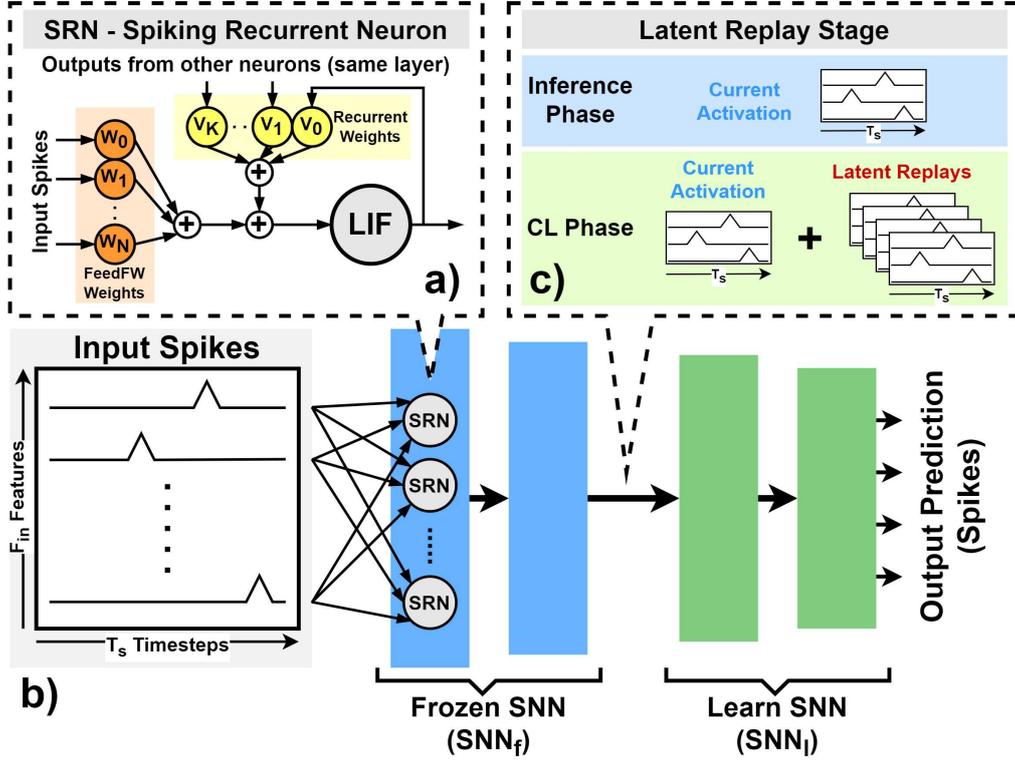


Figure 5.1: Visual depiction of a generic SNN with Recurrent Neurons and Latent Replays. In a), our Recurrent Neuron model; in b), the structure of a generic Fully-Connected model for Continual Learning; in c), an example of the data fed to the Latent Replay stage. Reproduced from Dequino et al. 2024 [154]©2024 IEEE.

5.2 Latent Replay-based Continual Learning in SNNs

In this chapter, inspiration is drawn from the work of Pellegrini et al. [147] on LR, initially designed for CNNs. Several crucial steps must be addressed to adapt this paradigm for SNNs. First, rehearsal data need to encapsulate the temporal evolution of a layer of spiking neurons, in contrast to the static input images in the original LR framework. Second, this temporal evolution must fit into the learning algorithm, i.e., BPTT, while mixing with newly acquired data. Furthermore, our attention is directed towards two CL scenarios: a Sample-Incremental setup, where a learner is trained to classify unseen scenarios, and a Class-Incremental setup, where the learner is tasked with identifying an increasing number of classes.

Algorithm 1 SNN Latent Replay training. Reproduced from Dequino et al. 2024 [154]©2024 IEEE.

```

1: Input:  $TR_{pre}, TR_{cl}$ .
2: Parameters:  $E_{pre}, E_{cl}, K, \eta$ .
3:
4: Initial Training phase:
5: Initialize SNN weights  $W, V$  randomly
6: for  $e \leftarrow 1$  to  $E_{pre}$  do
7:    $(W, V) \leftarrow \text{BPTT.train}(\text{SNN}, TR_{pre}, (W, V), \eta)$ 
8: end for
9:
10: Prepare network for CL:
11:  $(\text{SNN}_f, \text{SNN}_l) = \text{split}(\text{SNN}, K)$ 
12:  $LR = \text{Inference}(TR_{replay} \subseteq TR_{pre}, \text{SNN}_f)$ 
13:
14: Train network on new data:
15: for  $e \leftarrow 1$  to  $E_{cl}$  do
16:    $A = \text{Inference}(TR_{cl}, \text{SNN}_f)$ 
17:    $(W_l, V_l) = \text{BPTT.train}(\text{SNN}_l, A \cup LR, (W_l, V_l), \eta)$ 
18: end for

```

5.2.1 Latent Replays in SNNs

Algorithm 1 outlines the proposed Latent Replay-based training for an SNN with L layers. The neural network undergoes pre-training over E_{pre} epochs using BPTT on an initial training set TR_{pre} with an η learning rate (lines 4-8). As illustrated in Figure 5.1-b, the network is then split into two sections: (SNN_f) , comprising the first K layers denoted as *frozen layers*, and (SNN_l) , encompassing layers from the $L - K + 1$ -th to the L -th, marked as *learning layers* (line 11). The latent replays (Figure 5.1-c), denoted as LR , constitute a collection of the output activations of the K^{th} layer when exposed to a subset TR_{replay} of the pre-training set. This collection must be stored for later use during CL training (line 12), necessitating onboard storage. Since input data are trains of spikes (i.e., binary values) distributed over T_s time-steps, the stored LRs are in turn sequences of T_s single-bit activations.

When training the network on new data, belonging to a new scenario or class, only the *learning layers*, are trained: the *frozen layers* simply propagate the input spikes sequences in the forward direction, processing them through the function learned in the pre-training phase (line 16). The *learning layers* are then trained for E_{cl} epochs, using the output activations of the K^{th} layer, blended with the stored LRs to keep memory of the learned data (line 17).

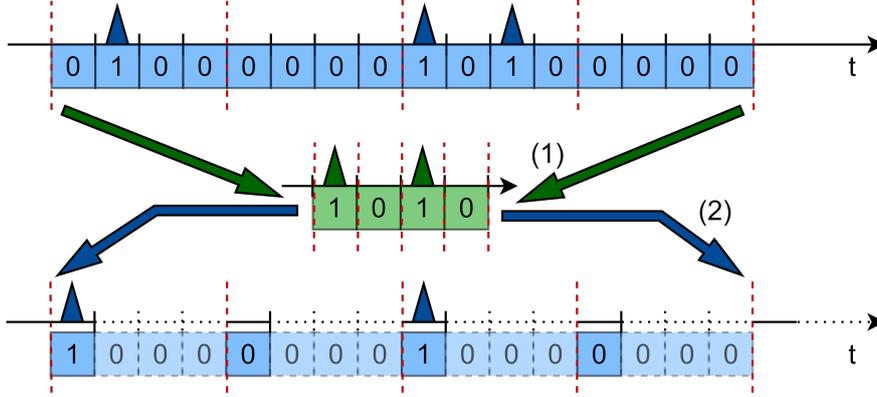


Figure 5.2: Example of a lossy compression (1) to store LRs. Compression ratio is here set to 4:1. Shrinking LRs and activations reduces the memory by $4\times$. A de-compression step (2) is required to respect the SNN time scale. Reproduced from Dequino et al. 2024 [154]©2024 IEEE.

5.2.2 Optimization of Latent Replay memory

To keep memory of previously learned information, an additional storage is required for the LRs. The problem becomes increasingly serious when more classes or scenarios are learned sequentially, making the required memory grow over time. To limit the phenomenon, a time-domain lossy compression on LRs is used.

Figure 5.2 illustrates the compression technique applied to a generic spiking sequence. Given a compression ratio (C_r), each component of a LRs sample is compressed into a sequence of T_s/C_r binary activations. The algorithm involves two steps: first, the uncompressed sequence is divided into chunks composed of C_r activations. Subsequently, each chunk is compressed in a single time step using a lossy compression based on thresholding: if the number of spikes within the chunk reaches a given *threshold* (e.g., 1), a spike is generated in the compressed sequence. The impact on computational resources manifests as a $C_r\times$ memory reduction for the LRs.

When processing a LR with the compressed data, decompression is needed to respect the time scale of the model. To avoid this step, different model’s time constants were tried, adapting them to the compressed duration of the spike sequences. However, the accuracy loss was dramatic, especially with compressed LR performed in the first layers. The inter-layer recurrent connections in these layers were likely tuned to a specific time scale and did not adapt well to its variation. In support of this idea, observations prove that a compressed LR on the last layer, which doesn’t contain any feedback connection, almost doesn’t affect the final accuracy. The solution, which also works for recurrent layers, is to uncompress the sequence by interleaving the compressed samples with zeroes to match the time scale of the spikes’ sequence at run-time.

5.3 Experimental Results

5.3.1 Experimental Setup

The proposed methodology was evaluated using the SHDs dataset [90], which comprises 10,420 spiking trains of 100 timesteps each. These trains consist of audio samples obtained through a conversion system inspired by the human cochlea. The dataset is categorized into 20 classes, corresponding to numbers 0 to 9, pronounced by 12 speakers in both English and German. A recurrent SNN was employed, with 700 input neurons and 20 outputs (one for each class), including 4 layers with decreasing output size (200-100-50-20). As described in chapter 2, the target neuron model was a Synaptic Conductance-based Second Order LIF neuron model, trained using BPTT and a fast-sigmoid SG. The proposed training setup incorporated a learning rate of $\eta = 10^{-3}$ for the Sample Incremental task and $\eta = 2 \cdot 10^{-4}$ for the Class Incremental task.

5.3.2 Weights initialization

When pre-training an SNNs for a Class-Incremental setup, neurons associated with unlearned classes were trained to be inactive. Therefore, when adding a new class to the pre-trained classifier, the yielded accuracy was poor, reaching a maximum of 57%. This issue was addressed by re-initializing the neuron weights devoted to detect the new class. When performing weight re-initialization, the obtained accuracy was strictly linked to the re-initialization strategy. Notably, random, constant, and Xavier-Glorot initializations [155] proved inefficient for proficient learning of the new class. Instead, a normal random distribution was adopted, with mean and variance aligned with those of the classifier’s weights trained on the old classes. This approach enabled the target model to achieve a remarkable 92.9% accuracy on the new class. Further details are discussed in subsection 5.3.4. Unlike the Class-Incremental scenario, the Sample-Incremental scenario did not require any initialization, as the number of classes did not change between samples.

5.3.3 Sample-Incremental CL

The target model underwent pre-training on 11 out of 12 scenarios (speakers) to simulate user personalization in keyword spotting tasks. The 12th scenario was introduced using a Sample-Incremental CL approach. To benchmark the proposed technique against methodologies like [153], experiments were conducted in three setups: (i) a *naïve incremental* setup, where the 12th speaker was learned without rehearsal; (ii) a *naïve rehearsal* setup, mixing the 12th speaker’s input activations with 2,560 samples from the training step; (iii) the proposed LR setup, integrating 2,560 LRs stored at the output of the 2nd-to-last layer. Figure 5.3-a) displays the

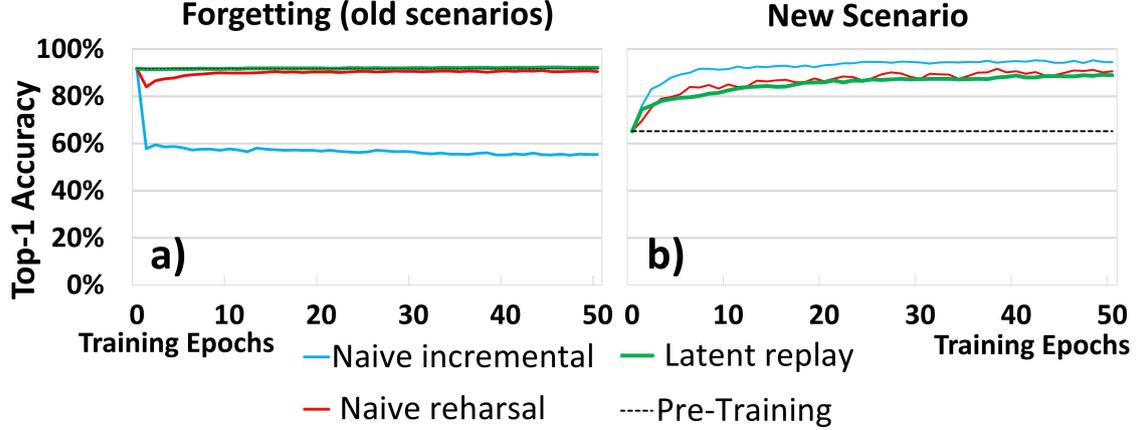


Figure 5.3: Measurement of (a) forgetting and (b) Top-1 accuracy on the new scenario (Sample-Incremental CL). Latent Replays were applied to the 2nd to last layer of the considered model using 2,560 past samples for the replay. Reproduced from Dequino et al. 2024 [154] ©2024 IEEE.

forgetting measurement, represented by the Top-1 accuracy of the SNN after 50 CL epochs, focusing on the old scenarios. Figure 5.3-b) showcases the accuracy of the new scenario.

As observed in CNNs, SNNs face Catastrophic Forgetting when learning new samples without rehearsal. In naïve incremental, few epochs specialized the model toward the new speaker ($> 90\%$ accuracy), resulting in almost 40% accuracy loss on the old speakers. In naïve rehearsal, rehearsal data enabled the model to retain nearly the same accuracy before the CL routine, limiting the accuracy drop on the old speakers to a 2%. Conversely, the new speaker was learned with almost 90% accuracy. Retraining only the last 2 layers with LRs maximized accuracy retention on old scenarios. The pre-training accuracy of 11 speakers was enhanced by 1%, as the new speaker’s samples strengthen classification in classes seen during pre-training. The Top-1 accuracy on the new speaker was increased by 24%, reaching an overall 88%. This was 2% lower than naïve rehearsal, but it was associated with a 7 \times reduction in the memory required to store rehearsal data.

5.3.4 Class-Incremental CL

In this scenario, the introduction of a new keyword was simulated by pre-training the model on 19 out of 20 classes and adding the 20th class subsequently. Similar to the Sample-Incremental setup, three scenarios were compared: (i) a *naïve incremental*, (ii) a *naïve rehearsal*, and (iii) *the proposed LR setup*. Figure 5.4-a) measures forgetting, while Figure 5.4-b) shows Top-1 accuracy on the new class. In terms of forgetting, LRs showed no accuracy drop, outperforming both naïve

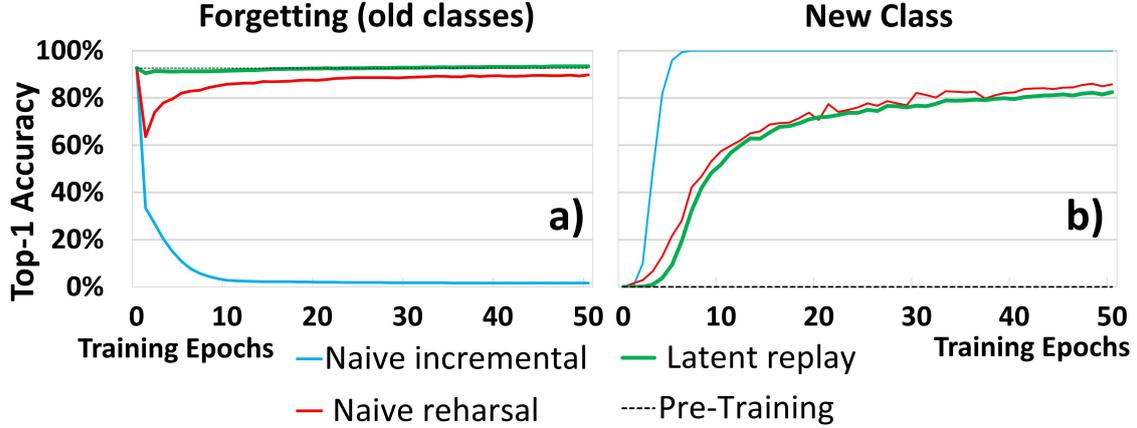


Figure 5.4: Measurement of (a) forgetting and (b) Top-1 accuracy on the new class (Class-Incremental CL). Latent Replays were applied to the 2^{nd} to the last layer of the model. Reproduced from Dequino et al. 2024 [154] ©2024 IEEE.

rehearsal ($\simeq 3\%$ drop) and naïve incremental (complete forgetting). The new class was learned within 50 epochs for all three methods. While naïve incremental overfitted the new class, completely forgetting the past ones, LRs achieved the highest accuracy of 83% without notable forgetting. Again, the accuracy reached by naïve rehearsal was a bit higher on the new data, and exceeded LRs by 3%. However this came at the cost of a 3% forgetting of the past classes, with a $7\times$ memory occupation.

5.3.5 Classification Accuracy vs Number of LRs

Figure 5.5 comprehensively analyzes the impact of various hyperparameters on accuracy, examining (a) the number of LRs and (b) LR indices across different epochs. In the Sample-Incremental setup, increasing LRs positively influenced Top-1 accuracy. The primary effect was on the maximum accuracy, rather than the convergence slope, which remained relatively constant across all cases. An initial forgetting was observed with 640 and 1280 past samples, becoming more and more accentuated with decreasing numbers of LRs. However, in all the cases the network was able to retrieve the past information, bringing the accuracy back to its initial value, or exceeding it. The choice of LRs index impacted maximum accuracy as well: Figure 5.5 shows the superiority of LRs (indexes 1, 2 and 3) with respect to naïve rehearsal (index 0). The highest accuracy of 92.5% was achieved with 5,120 LRs at layer index 1, as layer 0 retains knowledge of old data while extending towards old and new samples.

Increasing LRs has similar effects for the Class-Incremental CL setup. While varying the layer index, the peak accuracy was obtained at LRs index 2, making

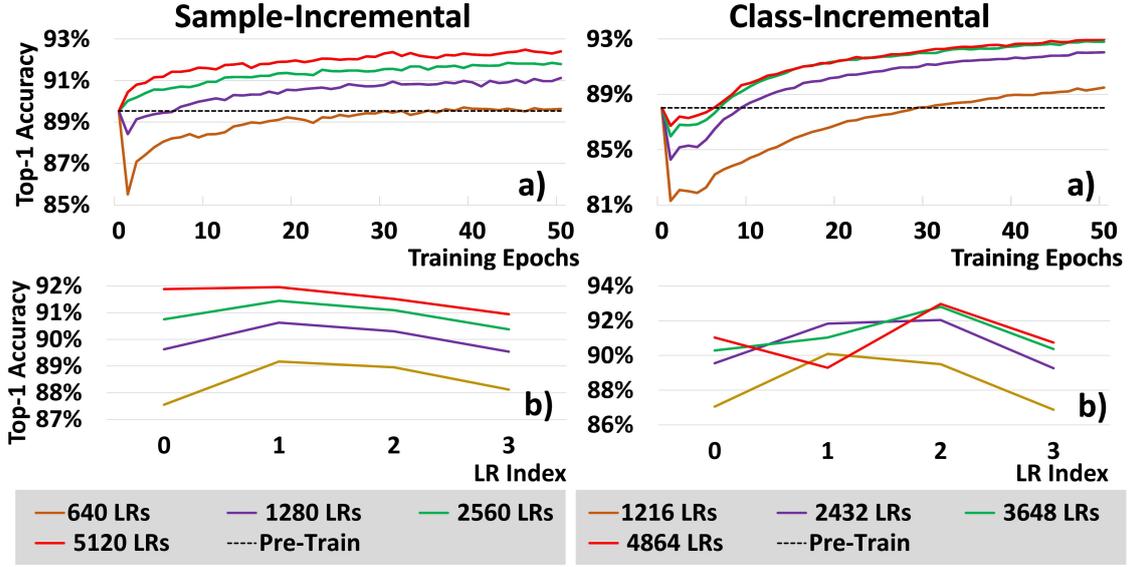


Figure 5.5: Top-1 accuracy on SHD’s test set in Sample and Class-Incremental CL, in case of a variable number of LR’s (a) with respect to the epochs, (b) with respect to the LR index. Reproduced from Dequino et al. 2024 [154] ©2024 IEEE.

Table 5.1: Memory-Accuracy tradeoff for Sample-Incremental CL, with 2560 LR’s and variable C_r and LR index. Pre-training accuracy on the scenario to be learned was 65%. Reproduced from Dequino et al. 2024 [154] ©2024 IEEE.

C_r \ LR’s		LR’s			
		Naïve rehearsal	Layer 1	Layer 2	Layer 3
1:1	Acc	90.50%	92.46%	91.79%	90.37%
	Mem	22.4 MB	6.4 MB	3.2 MB	1.6 MB
1:5	Acc	67.81%	79.71%	89.91%	90.27%
	Mem	4.48 MB	1.28 MB	640 KB	320 KB
1:10	Acc	67.12%	68.12%	84.73%	88.79%
	Mem	2.24 MB	640 KB	320 kB	160 kB

Top-1 accuracy reach 93% with 4,864 LR’s. This indicates the capability of the final layers to learn the new class, exploiting the higher-level features learned by previous layers. An interesting effect can be noted at index 1, where the curve is non-monotonic: in this case, a larger number of LR’s delayed the convergence of the model, preventing it from reaching an acceptable accuracy within the 50 epochs.

Table 5.2: Memory-Accuracy tradeoff for Class-Incremental CL, with 2432 LRs and variable C_r and LR index. Reproduced from Dequino et al. 2024 [154] ©2024 IEEE.

C_r \ LRs		Naïve rehearsal	Layer 1	Layer 2	Layer 3
1:1	Acc	89.55%	91.83%	92.05%	89.26%
	Mem	22.4 MB	6.4 MB	3.2 MB	1.6 MB
1:5	Acc	84.93%	57.43%	82.03%	86.78%
	Mem	4.48 MB	1.28 MB	640 KB	320 KB
1:10	Acc	77.43%	30.75%	69.42%	85.53%
	Mem	2.24 MB	640 KB	320 kB	160 kB

5.3.6 Compressed LRs

At this point, the memory-accuracy trade-off of the proposed LR compression algorithm was studied. The choice of the LRs’ layer index and the time compression factor C_r was based on the trade-off between accuracy and memory requirements. Table 5.1 and Table 5.2 present the results obtained on the full SHD test set, for Sample and Class-Incremental setups respectively. In the case of *naïve rehearsal*, the rehearsal data were stored as past input activations, consisting of 100 timesteps and 700 inputs. The spatial size of LRs aligned with the size of the layer on which they were stored as input, such as 100 inputs for layer 2. After selecting a C_r , the rehearsal data were additionally compressed on the time axis; for example, with $C_r = 5$, the data were reduced by $5\times$, featuring 20 timesteps. Therefore, all the data were scaled proportionally. The experiments showed a minimal size of the rehearsal data, with LRs collected on layer 3 using $C_r = 10$, being $140\times$ smaller than *naïve rehearsal* without compression. For the Sample-Incremental setup (Table 5.1), the results showed that a Top-1 accuracy of 92.46% for uncompressed LRs with index 1, surpassing the naïve rehearsal by 2%. Compressed LRs reduced the maximum accuracy: in case of a 1:5 ratio, 90.27% was achieved with index 3, while the accuracy drop on earlier LR indices became prohibitive before layer 2. This drop was accentuated for 1:10 compression, achieving a Top-1 accuracy of 88.79% on index 3. However, a memory save of $10\times$ was attested.

Also in case of a Class-Incremental setup (Table 5.2), the Top-1 accuracy of Latent-Replay, obtained at LR index 2, surpassed the Naive-Incremental by a 2.5%. Performing a LR at index 3 brought back the accuracy to 89.26%, comparable with Naïve rehearsal, but with a memory requirement $14\times$ lower. Finally 1:5 and 1:10 time compressions corresponded to a further drop of 2.5% and 3.7%, but with a memory saving of $5\times$ and $10\times$. The accuracy loss for previous indexes was still too high, indicating that a lighter compression is required.

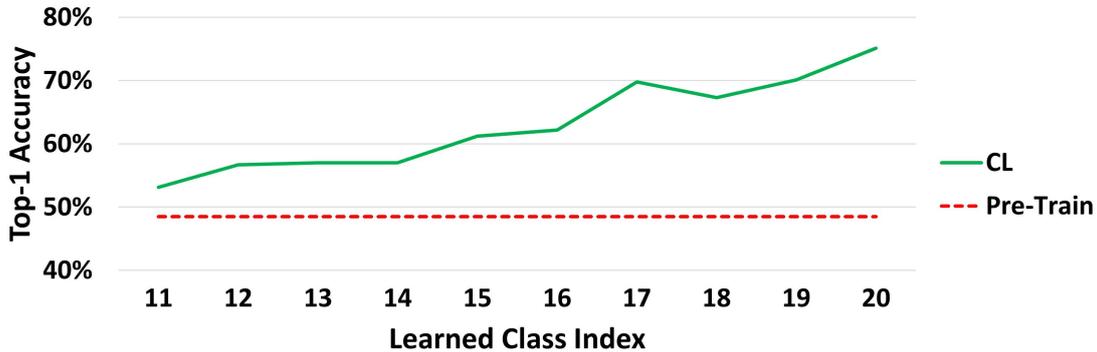


Figure 5.6: Multi-Class-Incremental CL on SHD. Here, a pre-trained model learned to classify 10 more classes, starting from a pre-training on 10 classes. Reproduced from Dequino et al. 2024 [154] ©2024 IEEE.

5.3.7 Comparison with other compressions

The technique focuses on preserving most of the temporal content of spike sequences, while allowing for an easy decompression at run-time. Alternatively, latent spikes can be aggregated by storing them as active spike counts.

Two additional compression methods were compared with the proposed subsampling approach (Figure 5.2): first, spikes were aggregated over the full sequence and expanded during rehearsal, placing all spikes in the first time-steps. While this drastically reduces the required memory to $\log_2(N)$ bits/sequence, the time information was lost, causing a 12% to 14% accuracy drop compared to the proposed technique, with $C_r = 10$ and 20. The second was a hybrid approach: sequences were split into chunks and spikes accumulated within each chunk. This led to a 2% accuracy improvement with respect to the proposed method, at the cost of a 3× and 4× memory occupation with $C_r = 20$ and $C_r = 10$, respectively. Therefore, for the considered keyword spotting setup, the proposed method provides an optimal trade-off between accuracy, memory occupation and run-time decompression complexity. Also, the temporal properties of the spike trains proved to be more relevant than the absolute number of spikes.

5.3.8 Multi-Class-Incremental Setup

Finally, a more complex testing scenario was considered, i.e., a keyword spotting language shift in which the model learned to classify 10 classes of digits in German, starting from 10 pre-learned English-spoken digits. The pre-trained knowledge was stored as 2560 LRs, 256 per-class. After each new class was learned, 256 LRs were added to the memory. The model was trained for 50 epochs per class, with $C_r = 2$.

Figure 5.6 shows the Top-1 accuracy over the 20-class SHD test set. The pre-training accuracy was of 48.5%, corresponding to an accuracy of 97% on the first 10 classes only. While learning with CL, a monotonic growth was assessed, learning each successive class with 88.2% average accuracy. For each new class learned, an average forgetting of 2.2% was observed on all the previous classes. In the end, a final accuracy of 78.4% on the full test set was obtained.

5.4 Conclusions and future work

This work represents one of the early attempts to apply rehearsal-based methods to SNNs. Rehearsal techniques provide several practical advantages over other continual learning strategies, such as regularization-based approaches [156, 157] and architectural methods [158, 159]. Unlike regularization methods, which constrain weight updates to preserve prior knowledge, rehearsal strategies explicitly revisit prior data, resulting in more robust retention—especially for complex, high-dimensional tasks. Compared to architectural solutions, which expand the model or allocate task-specific subnetworks (e.g., Progressive Neural Networks), rehearsal methods are more memory-efficient and scalable, particularly when enhanced with sample selection or compression.

Biological inspiration underpins this strategy, drawing from hippocampal replay phenomena observed in the brain during sleep or rest. Such replay, which involves the reactivation of neuronal sequences linked to past experiences, is thought to support memory consolidation and long-term learning.

Our experiments with compression techniques demonstrated that the network could retain knowledge of previous tasks even when presented with an approximated version of past activations. This is encouraging both computationally—indicating that the model is not tied to exact activation patterns—and in terms of memory efficiency, as compression significantly reduces storage demands, as shown in Section 5.3. However, this study remains preliminary, as it relied on supervised training with BPTT, which requires unrolling the network over time and storing all intermediate activations—an approach that dramatically increases memory usage. In future work, we aim to extend this framework by exploring more efficient learning algorithms such as E-prop [160] and STDP [81], with the goal of enabling on-device training and continual learning directly on edge platforms in real-time scenarios.

Chapter 6

Neuromorphic Heart Rate Monitors: Neural State Machines for Monotonic Change Detection

©2024 IEEE. Adapted, with permission, from Alessio Carpegna, Chiara De Luca, Federico Emanuele Pozzi, Alessandro Savino, Stefano Di Carlo, Giacomo Indiveri, Elisa Donati, "Neuromorphic Heart Rate Monitors: Neural State Machines for Monotonic Change Detection," 2024 IEEE Biomedical Circuits and Systems Conference (BioCAS), Xi'an, China, 2024, doi: 10.1109/BioCAS61083.2024.10798178

As last contribution to this thesis, transitioning from conventional digital architectures to neuromorphic systems enables energy-efficient computation by leveraging biologically inspired principles. In this final chapter, the application of neuromorphic technology to real-world biomedical signal processing is explored, focusing on Heart Rate (HR) monitoring. Detecting and quantifying HR has emerged as a critical tool for identifying potential pathologies and providing valuable insights into cardiovascular health [161]. Among variable behaviors, monotonic HR changes indicate unidirectional trends, either increasing or decreasing, in average HR over time. In non-clinical settings, such as general well-being and athletic training, tracking monotonic changes in HR is essential for evaluating physical fitness and recovery rates [162]. In clinical settings, monitoring monotonic changes in HR is crucial for medical diagnosis and patient monitoring. A consistent monotonic increase or decrease in heart rate can be an early indicator of cardiac conditions such as arrhythmia, bradycardia, or tachycardia. Early detection of these trends enables timely medical intervention, preventing more severe complications and improving patient outcomes [163].

Understanding monotonic increases in HR is also particularly important in patients with dementia, as it can help detect physiological stress or discomfort that

could precede or accompany agitation states [109, 164]. By monitoring trends in dementia care and detecting early signs of agitation, healthcare providers and caregivers can intervene more promptly and effectively. This may potentially reduce the severity and frequency of agitation episodes, thereby improving the quality of care and the quality of life for dementia patients and reducing the burden on caregivers [165].

Due to the importance and, at the same time, the difficulty of detecting these changes for human interpreters, an always-on system capable of continuously monitoring and detecting changes in average HRs can be extremely impactful. Always-on devices for health monitoring must meet stringent requirements, including low power consumption and real-time processing. For patients with dementia, these devices must operate for extended periods without frequent recharging, as subjects might not remember to charge the device regularly. Therefore, a low-power device that can be used as a “wear and forget” system is essential to ensure effective health management.

To this end, neuromorphic technology offers a compelling solution, providing energy-efficient and reliable processing capabilities [166]. Sensory-processing systems built using mixed-signal neuromorphic circuits are well-suited to the demands of continuous health monitoring [167]. Examples have already been successfully applied in a wide range of wearable applications, such as ECG anomaly detection [168, 169], High Frequency Oscillation (HFO) detection [170], and Electromyography (EMG) decoding [171, 172].

This chapter presents for the first time a neuromorphic implementation of an on-line signal processing system to specifically detect monotonic changes in average HRs over a long period of time. Computational primitives of analog neuron circuits are deployed, such as recurrent neural network models of finite state machines (i.e., Neural State Machines (NSMs)) [173, 174] that can switch between states, each encoding different average HR conditions, independent of the time elapsed between state changes. Here, the focus is on a monotonic state switch, tested on ECG, to detect a progressive HR increase. The model’s ability to track HR changes during activities (i.e., walking, cycling) is evaluated by testing it on a dataset with varying patterns. This chapter demonstrates how the model accurately follows monotonic changes (steady increase or decrease) while remaining inactive for non-monotonic signals. It further shows how the robust computational properties of NSMs allow the mixed-signal neuromorphic processor to produce accurate and reliable results. The low power consumption, $90\mu W$, and real-time processing features of these neuromorphic circuits make them ideal candidates for building continuous, long-term health monitoring devices in both clinical and non-clinical settings.

6.1 Background

6.1.1 Neuromorphic Hardware

The approaches presented up to now exploited traditional technologies to emulate biological behaviors. However the original idea of Neuromorphic Computing [29] was to physically emulate the behavior of neurons using silicon technology.

In this context, analog CMOS-based neuromorphic architectures represent a class of circuits specifically designed to mimic the computational properties of biological neural systems using continuous-time, continuous-valued analog signals [175]. Unlike digital or voltage-based implementations, these architectures encode information through currents, leveraging subthreshold CMOS circuits to enable ultra-low power consumption, real-time operation, and biologically realistic dynamics [176]. This makes them particularly well-suited for edge always-on sensing applications [177]. In current-mode operation, signals are represented by electrical currents rather than voltages. This paradigm aligns closely with the behavior of biological neurons and offers several practical advantages:

1. Neurons and synapses in biological systems naturally operate using currents, such as ion channel currents.
2. Current-mode circuits can achieve low impedance, high-speed, and low-voltage swings—leading to extreme energy efficiency.
3. Summation and integration of inputs are naturally performed by Kirchhoff’s current law.

Analog neurons in this architecture are typically implemented using transistors operating in the subthreshold regime, where the drain current depends exponentially on the gate voltage. This enables smooth, biologically plausible dynamics. The membrane potential is modeled as a capacitor that integrates incoming synaptic currents. When the voltage crosses a threshold, the neuron emits a spike and resets its potential. Analog synapses modulate the amplitude of the current flowing into the neuron [178]. These synapses can incorporate programmable weights through technologies such as floating-gate transistors, memristors, or Digital to Analog Converter (DAC)-controlled current mirrors. Furthermore, they can support plasticity mechanisms like STDP, which are implemented in continuous time by exploiting the temporal overlap of pre- and post-synaptic spikes.

While the internal processing is analog, communication between neurons is typically event-driven using the AER protocol. In this scheme, spike events are encoded as asynchronous digital packets containing the source address and the spike timestamp. This enables sparse, scalable, and energy-efficient communication between neurons and synapses. Additionally, local learning circuits can be embedded

to support on-chip implementation of learning rules such as Hebbian learning or STDP.

Several pioneering systems have demonstrated the viability of analog CMOS-based neuromorphic computing:

- Stanford’s Neurogrid [100] is a large-scale CMOS-based analog neuromorphic system which uses subthreshold circuits to emulate cortical columns.
- BrainScaleS [54] is a hybrid system with accelerated analog neuron circuits and digital communication for high-throughput brain simulation.
- DYNAP-SE [33] is a mixed-signal neuromorphic processor based on analog LIF neurons with configurable synapses.

These architectures benefit from extremely low power consumption, often in the microwatt or nanowatt range per neuron, real-time dynamics, and high integration density, enabling the implementation of large-scale networks in compact silicon footprints.

However, analog architectures also come with notable limitations. Analog components are inherently sensitive to process variations, temperature fluctuations, and device aging, and analog signals tend to be more susceptible to noise compared to their digital counterparts. Rather than viewing these imperfections solely as drawbacks, neuromorphic designers often embrace them, using these constraints as an opportunity to develop computational primitives that reflect the same challenges faced by biological brains. For instance, instead of relying on the precision of individual neurons, these systems employ populations of neurons to achieve robust, fault-tolerant computation. This collective behavior allows neuromorphic circuits to operate reliably in noisy, variable environments, much like their biological inspiration.

The neuromorphic processor used in this study is the DYNAP-SE chip [33]. It is a custom-designed asynchronous mixed-signal processor that features analog spiking neurons and synapses that mimic the biophysical properties of their biological counterparts in real-time. The chip comprises four cores, each containing 256 AdExIF neurons. Each synapse can be configured as one of four types: slow/fast and inhibitory/excitatory. Each neuron includes a Content Addressable Memory (CAM) block with 64 addresses, representing the pre-synaptic neurons to which it is connected. Digital peripheral asynchronous input/output logic circuits receive and transmit spikes via the AER communication protocol [179]. In this system, each neuron is assigned a unique address encoded as a digital word, which is transmitted using asynchronous digital circuits as soon as an event is generated. The chip features a fully asynchronous inter-core and inter-chip routing architecture, allowing flexible connectivity with microsecond precision even under heavy system loads.

6.1.2 Neuromorphic primitives

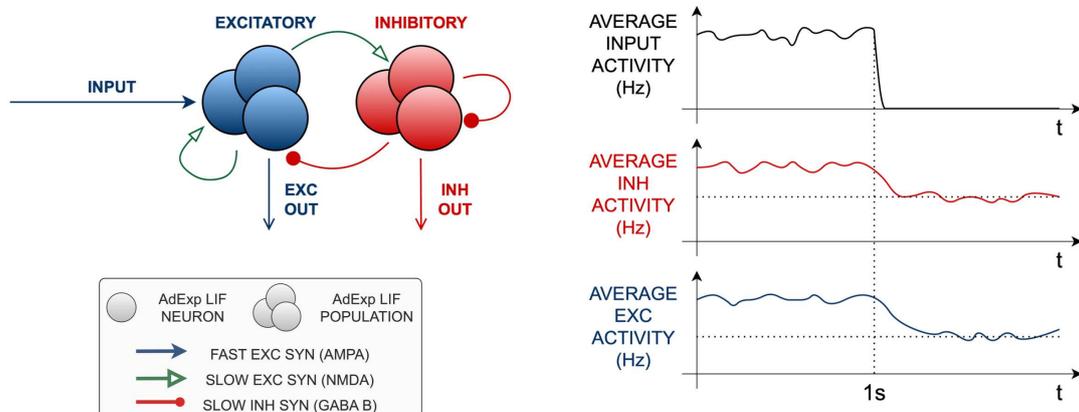


Figure 6.1: Excitatory-Inhibitory balanced computational primitive

Subsection 6.1.1 introduced the constraints of analog neuromorphic systems. To overcome these challenges, design approaches and methodologies are essential for building effective systems from noisy and unstable platforms. Nature, however, has had millions of years of evolution to develop solutions for performing complex tasks on biological systems that face similar constraints. One computational primitive observed in nature is the concept of excitatory-inhibitory (E-I) balanced networks [180]. These are recurrent neural populations that, once activated, can sustain their activity even when input is suppressed, maintaining network stability and preventing runaway excitation or inhibition. This balance between excitatory and inhibitory neurons is thought to be crucial for a wide range of brain functions, including sensory processing, motor control, and cognition. The E-I balance ensures stable network dynamics, supporting complex computations without leading to chaotic or oscillatory behavior [174]. By regulating network activity within a functional range, this balance prevents both hyperactivity, which can cause seizures, and hypoactivity, which can result in cognitive deficits. This principle of E-I balance is fundamental to the normal functioning of various brain regions, including the cortex, basal ganglia, hippocampus, cerebellum, and thalamocortical networks.

Figure 6.1 shows an example of an Excitatory-Inhibitory (E-I)-balanced network, along with the average firing activity of the excitatory and inhibitory populations. It can be observed that, even after the external input is removed, the two populations are able to maintain sustained activity. This enables the network to operate with time constants significantly longer than those achievable by individual neurons alone.

Such E-I-balanced primitives can be composed to construct more complex architectures, including Winner Takes All (WTA) [181] networks and NSMs [182].

These higher-order structures implement forms of computation inspired by biological circuits and are particularly suited for modeling sequential and state-dependent behaviors.

6.2 Materials and Methods

6.2.1 Dataset and signal processing

To test the model, an available dataset that includes left wrist Photoplethysmogramss (PPGs) and chest ECG recordings taken while participants used an indoor treadmill and exercise bike is used, accompanied by simultaneous motion data from accelerometers and gyroscopes [183]. Participants performed various exercises, such as walking, light jogging/running on a treadmill, and pedaling at low and high resistance, each for up to 10 minutes. The dataset includes records from 8 participants, with most participants spending 4 to 6 minutes per activity. The signals were sampled at 256 Hz, and the ECG records were processed with a 50 Hz notch filter to remove mains interference.

An energy-based approach was devised for signal-to-spike conversion [184]. The proposed method comprises two stages: bandpass filtering and LIF neurons [185, 186, 187]. Each input channel was processed through a series of bandpass filters. Four bands were used: 60-82 (#0), 82-105 (#1), 105-128 (#2), and 128-150 (#3) Beats Per Minute (BPM), employing fourth-order Butterworth filters. This covers a reasonable range of HR variation, spanning from a relaxed state to a tachycardiac one. Afterward, the signal was full-wave rectified and injected as a time-varying current into a simple LIF neuron.

6.2.2 Network on chip

Figure 6.2 shows the designed NSM. The architecture is based on E-I-balanced populations of AdExIF neurons [66], introduced in section 2.6. As mentioned in subsection 6.1.2, these populations can maintain sustained activity for extended periods, much larger than the time constants of the neurons themselves. This allows the network to implement a working memory capable of processing signals that change slowly compared to the time scales used within the chip.

The core of the network is a set of four E-I-balanced populations, organized in a WTA architecture. The shared inhibition population (red) encodes the state of the network by sustaining the activity of a specific population while silencing all the others. This guarantees that when receiving the input signal, such as an ECG recording, only the population corresponding to the most active frequency range is activated.

A second set of E-I-balanced populations, named Gating E-I-balanced populations, is used to control the transition between different states. The goal is to

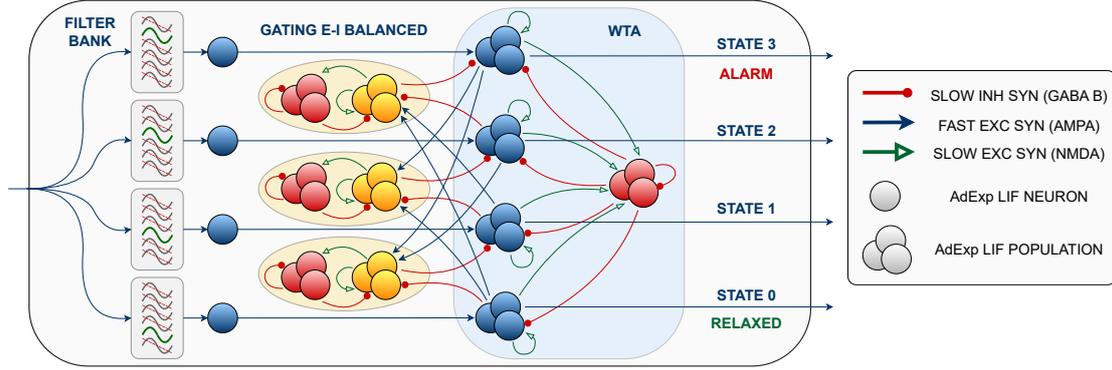


Figure 6.2: Monotonic NSM. The input signal is filtered through four 4th order Butterworth bandpass filters. Each filtered component is converted into spikes through a AdExIF neuron (blue). The input spikes are fed into populations of AdExIF neurons (blue), encoding different states of the network, interconnected in a WTA configuration via a common inhibitory population (red). States are connected to gating populations (yellow) to implement the monotonic computation. These are organized in a E-I-balanced configuration with an inhibitory population (red) limiting the overall activity. Reproduced from Carpegna et al. 2024 [188] 2024 IEEE.

target a progressive increase in HR, making the system robust to short fluctuations or temporary BPMs decreases, following only the average trend of the input. The mechanism of dis-inhibition [174] is exploited to ensure the network switches only to increasing states. The gating populations continuously inhibit the inactive elements of the WTA network, making them insensitive to any input stimulus. When a state is activated, it turns off (inhibits) the gating population of the subsequent state. This leads to the dis-inhibition of the next state, which makes it sensitive to the input. At the same time, it activates all the gating populations of previous states and the states following the next. This guarantees that the network follows only monotonic transitions between states and does not require precise tuning of the populations and connection synapses to make the system work. What matters is (i) that populations are sufficiently stable to maintain a sustained activity and (ii) that the inhibition between the gating populations and the WTA states is strong enough to silence any spiking activity completely. Finally, state 0 has no gating populations connected to it. This allows it to be used as a reset state: if the monotonic increase is only partial, and the HR returns to the relaxation range before reaching the alarming threshold, the network restarts, waiting for a new ramp-up in the input.

6.3 Experimental Results

To obtain reliable computation in the NSM while minimizing the overall network size, 16 neurons per population were used, with a total requirement of 176 neurons. Table 6.1 shows the average connection probabilities for different populations. The resulting network is compact, fitting well within a single core of the target chip [33]. Each population exhibits an average firing rate of approximately 50Hz. The total power consumption can be estimated by integrating the various contributions required for spike generation and communication [189, 190]. The obtained average power consumption is around $90\mu W$, indicating that our network successfully balances stable, sustained activity with low power consumption.

Table 6.1: Connection types and average probabilities. Reproduced from Carpegna et al. 2024 [188] ©2024 IEEE.

	Connection	Synapse type	Probability
WTA	state - i inh	NMDA	60%
	inh - i state	GABA B	60%
	state - i state	NMDA	83%
	inh - i inh	GABA B	20%
GATING	gate - i inh	NMDA	30%
	inh - i gate	GABA B	30%
	gate - i gate	NMDA	50%
	inh - i inh	GABA B	50%
MONOTONIC	gate - i state	GABA B	100%
	state - i gate	GABA B	100%
	state - i gate	AMPA	100%
INPUT	lif - i state	AMPA	100%

The behavior of the hardware system was first tested using control stimuli produced as 50Hz Poisson input spike trains. Figure 6.3 shows the sustained activity of one E-I-balanced population stimulated for one second. As shown, the network can keep a stable activity even when the input is removed.

6.3.1 WTA network: non-monotonic state transitions

As a second test, the dynamics of the WTA network were tested. Figure 6.4(a) shows the response of the network when stimulated with Poisson spike trains through a protocol covering all the possible transitions. In this case, each state is activated when receiving the corresponding input, regardless of the order of arrival. Once stimulated, the population becomes active, it suppresses the activity of all other populations, and stays active until a new input is provided to a different population.

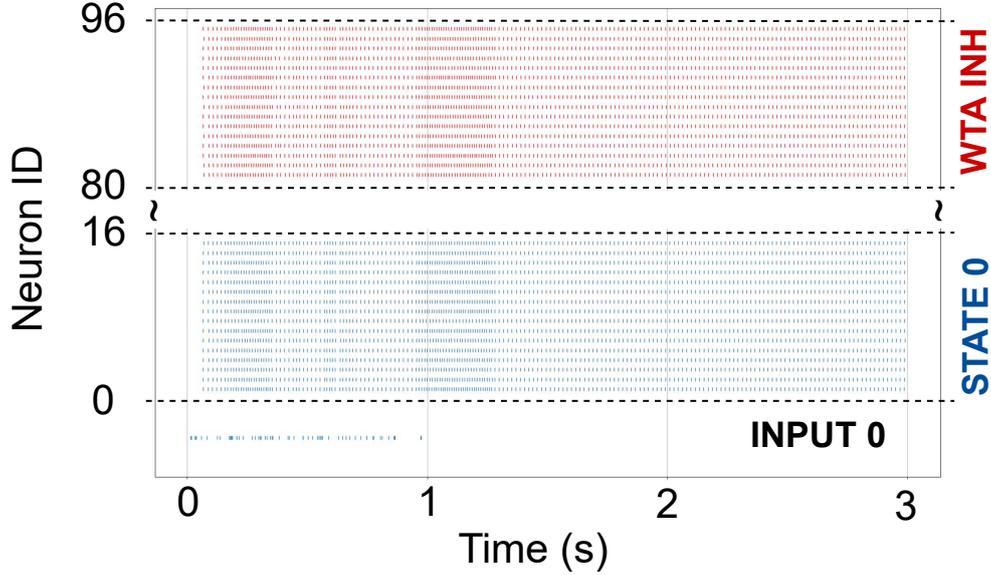


Figure 6.3: Stable sustained activity of one of the E-I-balanced primitives included in the networks. Reproduced from Carpegna et al. 2024 [188] ©2024 IEEE.

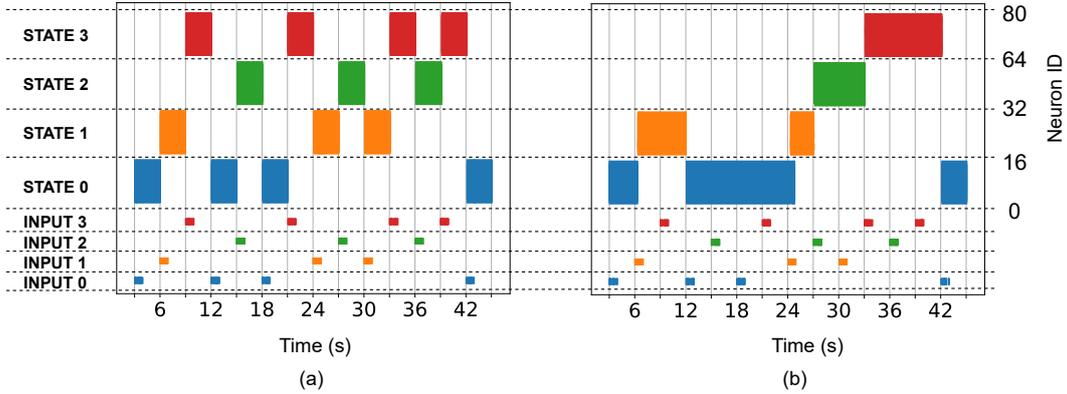


Figure 6.4: Network behavior when stimulated with 50Hz Poissonian sequences testing all the possible transitions: (a) non-monotonic WTA dynamics; (b) monotonic NSM response. Reproduced from Carpegna et al. 2024 [188] ©2024 IEEE.

6.3.2 NSM network: monotonic state transitions

When adding the gating connections, the response of the network becomes the one shown in Figure 6.4(b). The stimulation protocol is the same as in subsection 6.3.1. In this case, however, the network switches state only (i) towards a higher state, following the expected monotonic behavior, or (ii) towards state 0, which behaves as a reset state for the network, as mentioned in subsection 6.2.2

6.3.3 Network dynamics on real ECG signal

Finally, the network performance is evaluated using real ECG signals. Here, its robust and coherent ability to monitor and detect monotonic increases in HRs is demonstrated. To achieve this, the network was fed with pre-processed ECG recordings from the dataset described in subsection 6.2.1. Specifically, the network was tested under two different conditions: during an intense physical activity, namely cycling, lasting approximately 10 minutes (Figure 6.5(a)), and during a walking session of around 6 minutes (Figure 6.5(b)). The first scenario correctly detects a complete ramp-up of the HR, leading to the activation of the alarming state. These results show how the network is robust to spurious transitions: at time zero, the stimulation on input 3, caused by noise in the recording upper band, is ignored since state 3 is completely inhibited by its gating population. The same effect can be observed between 300 and 400 seconds with the network ignoring noisy stimuli from input 1. Note that also the weak stimuli coming from input 0 are ignored both in state 1 and 2, despite the absence of a gating population in this case. This shows that the WTA dynamics by themselves are robust to spurious transitions and a complete relaxation is required to reset the network, thus restarting the monotonic increase. This is even more evident in the walking session: in this case, the ramp-up is only partial, given the lower effort required. The network remains stable in state 1, waiting for a further increase in the HR and ignoring the noise on input 0.

6.3.4 Power consumption

To estimate the average power consumed by the system the different contributes necessary for spike generation and communication were integrated, following [189] and [190]. Since the network fits on a single core of the DYNAP-SE chip [33], the total power can be estimated as:

$$P_n = r_{inp}(E_{spike} + E_{pulse}) + r_{out}(E_{en} + E_{br}) \quad (6.1)$$

Where r_{inp} , r_{out} = mean input and output firing rate; $E_{spike} = 883$ pJ, energy to generate one spike; $E_{pulse} = 324$ pJ, energy of the pulse extender circuit; $E_{en} = 883$ pJ, energy to encode one spike and append destination; and $E_{br} = 6.84$ nJ, energy to broadcast event to same core. Given the average spiking activity $r_{inp} \simeq r_{out} \simeq 50Hz$, each neuron consumes around $500nW$. With the considered network, composed of 176 neurons the overall average power consumption is around $90\mu W$.

6.4 Conclusions

Our results show that a small and simple network, implemented with mixed-signal analog/digital neuromorphic circuits, can reliably monitor a monotonic HR

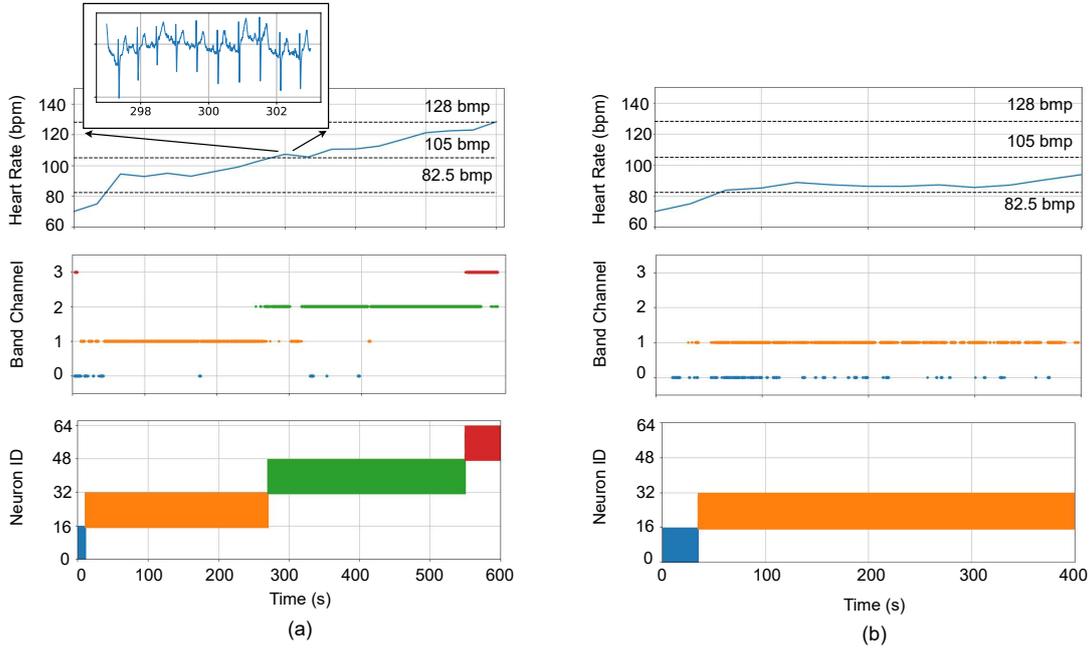


Figure 6.5: Response of the network when stimulated with a real ECG signal: (a) intense 10 minutes bike session; (b) 6 minutes and 40 seconds of walk. Reproduced from Carpegna et al. 2024 [188] ©2024 IEEE.

trends over extended periods, paving the way for always-on health monitoring systems that are both efficient and long-lasting. The low power consumption of the neuromorphic circuits enables continuous operation for extended amounts of time, making it ideal for wearable devices for health monitoring. Future research will focus on replacing ECG with PPG signals measured at the wrist and addressing the challenges of noise and movement artifacts. Additionally, the study will transition from healthy subjects to patients affected by neuropathologies, such as dementia, which impact HR over long periods, despite the absence of cardiac pathology.

This technology promises significant improvements in patient care and health management, especially in scenarios requiring constant monitoring and rapid response, thus enhancing overall quality of life.

Chapter 7

Conclusions

During this Ph.D. journey, I had the opportunity to explore the field of neuromorphic engineering from multiple perspectives. This interdisciplinary experience allowed me to engage with diverse domains, including high-level software modeling, the mathematical foundations of AI methodologies, biological mechanisms, and hardware design. Neuromorphic engineering is a rapidly growing and highly promising research field, with the potential to address the urgent issue of energy consumption in large-scale AI models. Its relevance is particularly pronounced in edge computing, where it can enable pervasive low-power AI systems.

The human brain stands as one of the greatest unsolved challenges in science. Its staggering complexity—arising from billions of neurons and trillions of dynamic synaptic connections—gives rise to intelligence, perception, memory, and consciousness, yet remains only partially understood despite decades of research. This complexity is not only structural but also functional: the brain operates with remarkable energy efficiency, adaptability, and robustness in the face of noise and incomplete information. Neuromorphic engineering emerges from the recognition that traditional computing paradigms are fundamentally ill-suited to replicate these capabilities. By taking inspiration directly from the brain’s architecture and operating principles, neuromorphic systems aim to create hardware and algorithms that are not only more efficient but also capable of real-time, context-aware, and adaptive processing. In doing so, neuromorphic engineering does more than mimic biology—it becomes a scientific tool to probe and model the brain itself. As such, the field occupies a unique position at the intersection of neuroscience, engineering, and artificial intelligence: striving to unravel the brain’s secrets by building systems in its image, while simultaneously using those same systems to better understand the very organ they emulate.

Despite its promise, neuromorphic engineering is still in an early stage of development and faces several important challenges. One of the main hurdles is the lack of mature, standardized tools and methodologies across hardware, software, and algorithmic layers. Unlike conventional AI systems, which benefit from

well-established architectures, frameworks, and training techniques, neuromorphic systems are still fragmented, with limited interoperability and few widely adopted platforms. On the hardware side, existing neuromorphic chips are often specialized and difficult to scale, with constraints on memory, programmability, and integration with mainstream computing infrastructure.

Over the past few years, I have tried to contribute to addressing these limitations by designing configurable systems and making neuromorphic research more accessible and practical. If, through my work, I have helped advance the research frontier—even if only by a small step—I consider my goal as a researcher fulfilled.

Chapter 8

List of publications

1. A. Carpegna, A. Savino and S. D. Carlo, "Spiker+: a framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge," in IEEE Transactions on Emerging Topics in Computing, doi: 10.1109/TETC.2024.3511676.
2. A. Carpegna, A. Savino and S. Di Carlo, "Spiker: an FPGA-optimized Hardware accelerator for Spiking Neural Networks," 2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Nicosia, Cyprus, 2022, pp. 14-19, doi: 10.1109/ISVLSI54635.2022.00016.
3. D. Padovano, A. Carpegna, A. Savino, and S. Di Carlo, "SpikeExplorer: Hardware-Oriented Design Space Exploration for Spiking Neural Networks on FPGA," Electronics, vol. 13, no. 9, Art. no. 9, Jan. 2024, doi: 10.3390/electronics13091744.
4. A. Dequino et al., "Compressed Latent Replays for Lightweight Continual Learning on Spiking Neural Networks," 2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Knoxville, TN, USA, 2024, pp. 240-245, doi: 10.1109/ISVLSI61997.2024.00052.
5. A. Carpegna et al., "Neuromorphic Heart Rate Monitors: Neural State Machines for Monotonic Change Detection," 2024 IEEE Biomedical Circuits and Systems Conference (BioCAS), Xi'an, China, 2024, pp. 1-5, doi: 10.1109/BioCAS61083.2024.10798178.
6. F. Pavanello et al., "Special Session: Neuromorphic hardware design and reliability from traditional CMOS to emerging technologies," 2023 IEEE 41st VLSI Test Symposium (VTS), San Diego, CA, USA, 2023, pp. 1-10, doi: 10.1109/VTS56346.2023.10139932.
7. F. Pavanello et al., "NEUROPULS: NEUROMorphic energy-efficient secure accelerators based on Phase change materials aUgmented siLicon photonicS,"

- 2023 IEEE European Test Symposium (ETS), Venezia, Italy, 2023, pp. 1-6, doi: 10.1109/ETS56758.2023.10173974.
8. S. Saeedi, A. Carpegna, A. Savino and S. Di Carlo, "Prediction of the Impact of Approximate Computing on Spiking Neural Networks via Interval Arithmetic," 2022 IEEE 23rd Latin American Test Symposium (LATS), Montevideo, Uruguay, 2022, pp. 1-6, doi: 10.1109/LATS57337.2022.9936999.
 9. Alessio Carpegna, Stefano Di Carlo, and Alessandro Savino. 2022. Artificial Resilience in neuromorphic systems. In Proceedings of the 12th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART '22). Association for Computing Machinery, New York, NY, USA, 112–114. <https://doi.org/10.1145/3535044.3535062>
 10. D. Kasap, A. Carpegna, A. Savino and S. Di Carlo, "Micro-Architectural features as soft-error markers in embedded safety-critical systems: preliminary study," 2023 IEEE European Test Symposium (ETS), Venezia, Italy, 2023, pp. 1-5, doi: 10.1109/ETS56758.2023.10174219.
 11. A. B. Göğebakan, E. Magliano, A. Carpegna, A. Ruospo, A. Savino and S. D. Carlo, "SpikingJET: Enhancing Fault Injection for Fully and Convolutional Spiking Neural Networks," 2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS), Rennes, France, 2024, pp. 1-7, doi: 10.1109/IOLTS60994.2024.10616060.
 12. S. Saeedi, A. Carpegna, A. Savino and S. Di Carlo, "Fast Exploration of the Impact of Precision Reduction on Spiking Neural Networks," 2024 IEEE International Conference on Design, Test and Technology of Integrated Systems (DTTIS), Aix-EN-PROVENCE, France, 2024, pp. 1-6, doi: 10.1109/DTTIS62212.2024.10780090.
 13. M. Traiola et al., "Approximate Fault-Tolerant Neural Network Systems," 2024 IEEE European Test Symposium (ETS), The Hague, Netherlands, 2024, pp. 1-10, doi: 10.1109/ETS61313.2024.10567290.
 14. E. Magliano, A. Carpegna, A. Savino and S. D. Carlo, "A Micro Architectural Events Aware Real-Time Embedded System Fault Injector," 2024 IEEE 25th Latin American Test Symposium (LATS), Maceio, Brazil, 2024, pp. 1-6, doi: 10.1109/LATS62223.2024.10534595.

Chapter 9

Code availability

In line with the open-source philosophy, this Ph.D. thesis prioritizes transparency and collaboration by making the developed code publicly available. The decision to release the code under an open-source license reflects a commitment to reproducibility, ensuring that the methods and results can be independently verified by other researchers. By embracing the FAIR, this work not only fosters innovation by allowing others to build upon and extend the research but also contributes to the broader scientific community's collective knowledge base.

1. **Spiker+**: <https://github.com/smilies-polito/Spiker>.
2. **SpikExplorer**: <https://github.com/smilies-polito/SpikExplorer.git>
3. **Continual Learning**: <https://github.com/Dequino/Spiking-Compressed-Continual-Learning.git>

Bibliography

- [1] Bo Xu and Mu-ming Poo. «Large language models and brain-inspired general intelligence». In: *National Science Review* 10.10 (Oct. 2023), nwad267. ISSN: 2095-5138. DOI: 10.1093/nsr/nwad267. URL: <https://doi.org/10.1093/nsr/nwad267> (visited on 02/06/2025).
- [2] Shuroug A. Alowais et al. «Revolutionizing healthcare: the role of artificial intelligence in clinical practice». In: *BMC Medical Education* 23.1 (Sept. 2023), p. 689. ISSN: 1472-6920. DOI: 10.1186/s12909-023-04698-z. URL: <https://doi.org/10.1186/s12909-023-04698-z> (visited on 03/30/2025).
- [3] Najma Taimoor and Semeen Rehman. «Reliable and Resilient AI and IoT-Based Personalised Healthcare Services: A Survey». In: *IEEE Access* 10 (2022). Conference Name: IEEE Access, pp. 535–563. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3137364. URL: <https://ieeexplore.ieee.org/document/9658494> (visited on 03/30/2025).
- [4] Feng Shi et al. «Review of Artificial Intelligence Techniques in Imaging Data Acquisition, Segmentation, and Diagnosis for COVID-19». In: *IEEE Reviews in Biomedical Engineering* 14 (2021). Conference Name: IEEE Reviews in Biomedical Engineering, pp. 4–15. ISSN: 1941-1189. DOI: 10.1109/RBME.2020.2987975. URL: <https://ieeexplore.ieee.org/document/9069255> (visited on 03/30/2025).
- [5] S. Kevin Zhou et al. «A Review of Deep Learning in Medical Imaging: Imaging Traits, Technology Trends, Case Studies With Progress Highlights, and Future Promises». In: *Proceedings of the IEEE* 109.5 (May 2021). Conference Name: Proceedings of the IEEE, pp. 820–838. ISSN: 1558-2256. DOI: 10.1109/JPROC.2021.3054390. URL: <https://ieeexplore.ieee.org/document/9363915> (visited on 03/30/2025).
- [6] Pranav Singh Chib and Pravendra Singh. *Recent Advancements in End-to-End Autonomous Driving using Deep Learning: A Survey*. arXiv:2307.04370 [cs]. Sept. 2023. DOI: 10.48550/arXiv.2307.04370. URL: <http://arxiv.org/abs/2307.04370> (visited on 02/06/2025).

- [7] Daniele Palossi et al. «A 64-mW DNN-Based Visual Navigation Engine for Autonomous Nano-Drones». In: *IEEE Internet of Things Journal* 6.5 (Oct. 2019). Conference Name: IEEE Internet of Things Journal, pp. 8357–8371. ISSN: 2327-4662. DOI: 10.1109/JIOT.2019.2917066. URL: <https://ieeexplore.ieee.org/document/8715489> (visited on 03/30/2025).
- [8] Prashant Johri et al. «Sustainability of Coexistence of Humans and Machines: An Evolution of Industry 5.0 from Industry 4.0». In: *2021 10th International Conference on System Modeling & Advancement in Research Trends (SMART)*. ISSN: 2767-7362. Dec. 2021, pp. 410–414. DOI: 10.1109/SMART52563.2021.9676275. URL: <https://ieeexplore.ieee.org/document/9676275> (visited on 03/30/2025).
- [9] Guillaume Bresson et al. «Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving». In: *IEEE Transactions on Intelligent Vehicles* 2.3 (Sept. 2017). Conference Name: IEEE Transactions on Intelligent Vehicles, pp. 194–220. ISSN: 2379-8904. DOI: 10.1109/TIV.2017.2749181. URL: <https://ieeexplore.ieee.org/document/8025618> (visited on 03/30/2025).
- [10] Jienan Chen et al. «Driving Maneuvers Prediction Based Autonomous Driving Control by Deep Monte Carlo Tree Search». In: *IEEE Transactions on Vehicular Technology* 69.7 (July 2020). Conference Name: IEEE Transactions on Vehicular Technology, pp. 7146–7158. ISSN: 1939-9359. DOI: 10.1109/TVT.2020.2991584. URL: <https://ieeexplore.ieee.org/document/9082903> (visited on 03/30/2025).
- [11] N. N. Misra et al. «IoT, Big Data, and Artificial Intelligence in Agriculture and Food Industry». In: *IEEE Internet of Things Journal* 9.9 (May 2022). Conference Name: IEEE Internet of Things Journal, pp. 6305–6324. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2998584. URL: <https://ieeexplore.ieee.org/document/9103523> (visited on 03/30/2025).
- [12] Suttinee Sawadsitang et al. «Shipper Cooperation in Stochastic Drone Delivery: A Dynamic Bayesian Game Approach». In: *IEEE Transactions on Vehicular Technology* 70.8 (Aug. 2021). Conference Name: IEEE Transactions on Vehicular Technology, pp. 7437–7452. ISSN: 1939-9359. DOI: 10.1109/TVT.2021.3090992. URL: <https://ieeexplore.ieee.org/document/9462455> (visited on 03/30/2025).
- [13] Omar Banimelhem and Baghdad Al-khateeb. «Explainable Artificial Intelligence in Drones: A Brief Review». In: *2023 14th International Conference on Information and Communication Systems (ICICS)*. ISSN: 2573-3346. Nov. 2023, pp. 1–5. DOI: 10.1109/ICICS60529.2023.10330543. URL: <https://ieeexplore.ieee.org/document/10330543> (visited on 03/30/2025).

- [14] Senthil Kumar Jagatheesaperumal et al. «The Duo of Artificial Intelligence and Big Data for Industry 4.0: Applications, Techniques, Challenges, and Future Research Directions». In: *IEEE Internet of Things Journal* 9.15 (Aug. 2022). Conference Name: IEEE Internet of Things Journal, pp. 12861–12885. ISSN: 2327-4662. DOI: 10.1109/JIOT.2021.3139827. URL: <https://ieeexplore.ieee.org/document/9667102> (visited on 03/30/2025).
- [15] S. Chien et al. «The Future of AI in Space». In: *IEEE Intelligent Systems* 21.4 (July 2006). Conference Name: IEEE Intelligent Systems, pp. 64–69. ISSN: 1941-1294. DOI: 10.1109/MIS.2006.79. URL: <https://ieeexplore.ieee.org/document/1667956> (visited on 03/30/2025).
- [16] Bimal K. Bose. «Artificial Intelligence Techniques in Smart Grid and Renewable Energy Systems—Some Example Applications». In: *Proceedings of the IEEE* 105.11 (Nov. 2017). Conference Name: Proceedings of the IEEE, pp. 2262–2273. ISSN: 1558-2256. DOI: 10.1109/JPROC.2017.2756596. URL: <https://ieeexplore.ieee.org/document/8074546> (visited on 03/30/2025).
- [17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. «ImageNet classification with deep convolutional neural networks». In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: <https://dl.acm.org/doi/10.1145/3065386> (visited on 03/30/2025).
- [18] Ashish Vaswani et al. «Attention is all you need». In: *Proceedings of the 31st International Conference on Neural Information Processing Systems. NIPS’17*. Red Hook, NY, USA: Curran Associates Inc., Dec. 2017, pp. 6000–6010. ISBN: 978-1-5108-6096-4. (Visited on 03/30/2025).
- [19] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. «Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning». In: *Sustainable Computing: Informatics and Systems* 38 (Apr. 2023), p. 100857. ISSN: 2210-5379. DOI: 10.1016/j.suscom.2023.100857. URL: <https://www.sciencedirect.com/science/article/pii/S2210537923000124> (visited on 03/30/2025).
- [20] Jeffrey Dean. «A Golden Decade of Deep Learning: Computing Systems & Applications». In: *Daedalus* 151.2 (May 2022), pp. 58–74. ISSN: 0011-5266. DOI: 10.1162/daed_a_01900. URL: https://doi.org/10.1162/daed_a_01900 (visited on 03/30/2025).
- [21] Zhi Zhou et al. «Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing». In: *Proceedings of the IEEE* 107.8 (Aug. 2019). Conference Name: Proceedings of the IEEE, pp. 1738–1762. ISSN: 1558-2256. DOI: 10.1109/JPROC.2019.2918951. URL: <https://ieeexplore.ieee.org/document/8736011> (visited on 03/30/2025).

- [22] Yuchuan Fu et al. «A Survey of Driving Safety With Sensing, Vehicular Communications, and Artificial Intelligence-Based Collision Avoidance». In: *IEEE Transactions on Intelligent Transportation Systems* 23.7 (July 2022). Conference Name: IEEE Transactions on Intelligent Transportation Systems, pp. 6142–6163. ISSN: 1558-0016. DOI: 10.1109/TITS.2021.3083927. URL: <https://ieeexplore.ieee.org/document/9447828> (visited on 03/30/2025).
- [23] Liang Xiao et al. «IoT Security Techniques Based on Machine Learning: How Do IoT Devices Use AI to Enhance Security?». In: *IEEE Signal Processing Magazine* 35.5 (Sept. 2018). Conference Name: IEEE Signal Processing Magazine, pp. 41–49. ISSN: 1558-0792. DOI: 10.1109/MSP.2018.2825478. URL: <https://ieeexplore.ieee.org/document/8454402> (visited on 03/30/2025).
- [24] Yann Le Cun, John S. Denker, and Sara A. Solla. «Optimal brain damage». In: *Proceedings of the 3rd International Conference on Neural Information Processing Systems*. NIPS'89. Cambridge, MA, USA: MIT Press, Jan. 1989, pp. 598–605. (Visited on 02/06/2025).
- [25] Raghuraman Krishnamoorthi. *Quantizing deep convolutional networks for efficient inference: A whitepaper*. arXiv:1806.08342 [cs]. June 2018. DOI: 10.48550/arXiv.1806.08342. URL: <http://arxiv.org/abs/1806.08342> (visited on 02/06/2025).
- [26] Luigi Capogrosso et al. «A Machine Learning-Oriented Survey on Tiny Machine Learning». In: *IEEE Access* 12 (2024). Conference Name: IEEE Access, pp. 23406–23426. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3365349. URL: <https://ieeexplore.ieee.org/document/10433185> (visited on 03/30/2025).
- [27] Mikhail Evchenko et al. *Frugal Machine Learning*. arXiv:2111.03731 [cs]. Nov. 2021. DOI: 10.48550/arXiv.2111.03731. URL: <http://arxiv.org/abs/2111.03731> (visited on 03/30/2025).
- [28] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. arXiv:1503.02531 [stat]. Mar. 2015. DOI: 10.48550/arXiv.1503.02531. URL: <http://arxiv.org/abs/1503.02531> (visited on 03/30/2025).
- [29] C. Mead. «Neuromorphic electronic systems». In: *Proceedings of the IEEE* 78.10 (Oct. 1990). Conference Name: Proceedings of the IEEE, pp. 1629–1636. ISSN: 1558-2256. DOI: 10.1109/5.58356. URL: <https://ieeexplore.ieee.org/document/58356> (visited on 02/06/2025).

- [30] Jianwei Xue et al. «EdgeMap: An Optimized Mapping Toolchain for Spiking Neural Network in Edge Computing». en. In: *Sensors* 23.14 (Jan. 2023). Number: 14 Publisher: Multidisciplinary Digital Publishing Institute, p. 6548. ISSN: 1424-8220. DOI: 10.3390/s23146548. URL: <https://www.mdpi.com/1424-8220/23/14/6548> (visited on 04/24/2024).
- [31] J. von Neumann. «First draft of a report on the EDVAC». In: *IEEE Annals of the History of Computing* 15.4 (1993). Conference Name: IEEE Annals of the History of Computing, pp. 27–75. ISSN: 1934-1547. DOI: 10.1109/85.238389. URL: <https://ieeexplore.ieee.org/document/238389> (visited on 03/30/2025).
- [32] Robert M. French. «Catastrophic forgetting in connectionist networks». English. In: *Trends in Cognitive Sciences* 3.4 (Apr. 1999). Publisher: Elsevier, pp. 128–135. ISSN: 1364-6613, 1879-307X. DOI: 10.1016/S1364-6613(99)01294-2. URL: [https://www.cell.com/trends/cognitive-sciences/abstract/S1364-6613\(99\)01294-2](https://www.cell.com/trends/cognitive-sciences/abstract/S1364-6613(99)01294-2) (visited on 03/30/2025).
- [33] Saber Moradi et al. «A Scalable Multicore Architecture With Heterogeneous Memory Structures for Dynamic Neuromorphic Asynchronous Processors (DYNAPs)». In: *IEEE Transactions on Biomedical Circuits and Systems* 12.1 (Feb. 2018). Conference Name: IEEE Transactions on Biomedical Circuits and Systems, pp. 106–122. ISSN: 1940-9990. DOI: 10.1109/TBCAS.2017.2759700. URL: <https://ieeexplore.ieee.org/document/8094868> (visited on 04/20/2024).
- [34] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. «A History of Spike-Timing-Dependent Plasticity». English. In: *Frontiers in Synaptic Neuroscience* 3 (Aug. 2011). Publisher: Frontiers. ISSN: 1663-3563. DOI: 10.3389/fnsyn.2011.00004. URL: <https://www.frontiersin.org/journals/synaptic-neuroscience/articles/10.3389/fnsyn.2011.00004/full> (visited on 04/03/2025).
- [35] Luigi Galvani et al. *Aloysii Galvani De viribus electricitatis in motu musculari commentarius*. lat. Bononiae : Ex Typographia Institutii Scientiarum, 1791. URL: <http://archive.org/details/AloysiiGalvaniD00Galv> (visited on 04/03/2025).
- [36] «Description of the Human Body». In: *Descartes: The World and Other Writings*. Ed. by René Descartes and Stephen Gaukroger. Cambridge Texts in the History of Philosophy. Cambridge: Cambridge University Press, 1998, pp. 170–205. ISBN: 978-0-521-63158-7. DOI: 10.1017/CB09780511605727.009. URL: <https://www.cambridge.org/core/books/descartes-the-world-and-other-writings/description-of-the-human-body/522AD668A94539C5841A8738E9B0DF6C> (visited on 04/03/2025).

- [37] Carlo Matteucci, Carlo Matteucci, and Paul Savi. *Traité des phénomènes électro-physiologiques des animaux*. Paris: Fortin, Masson, 1844. DOI: 10.5962/bhl.title.51425. URL: <https://www.biodiversitylibrary.org/bibliography/51425>.
- [38] Emil Du Bois-Reymond. *Untersuchungen über thierische Elektrizität*. ger. Berlin : G. Reimer, 1848. URL: http://archive.org/details/bub_gb_AtkPAAAAQAAJ (visited on 04/03/2025).
- [39] Hermann von Helmholtz. «Über die Geschwindigkeit einiger Vorgänge in Muskeln und Nerven». de. In: *Hermann von Helmholtz: Versuche zur Fortpflanzungsgeschwindigkeit der Reizung in den Nerven*. Ed. by Henning Schmidgen. Berlin, Heidelberg: Springer, 2021, pp. 165–168. ISBN: 978-3-662-63833-0. DOI: 10.1007/978-3-662-63833-0_9. URL: https://doi.org/10.1007/978-3-662-63833-0_9 (visited on 04/03/2025).
- [40] Ennio Pannese. «Il contributo di Camillo Golgi alla conoscenza della struttura del sistema nervoso». it. In: *Rendiconti Lincei* 18.2 (June 2007), pp. 123–127. ISSN: 1720-0776. DOI: 10.1007/BF02967219. URL: <https://doi.org/10.1007/BF02967219> (visited on 04/03/2025).
- [41] Santiago Ramón y Cajal. *Estructura de los centros nerviosos de las aves*. Madrid: [s. n.], 1924.
- [42] Arpan R Mehta et al. «Etymology and the neuron(e)». In: *Brain* 143.1 (Jan. 2020), pp. 374–379. ISSN: 0006-8950. DOI: 10.1093/brain/awz367. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6935745/> (visited on 04/03/2025).
- [43] Germana Pareti and A. Palma. «A conservative revolution: Julius Bernstein and his trip toward a membrane theory. A focus on bioelectrical processes in Mid-Nineteenth Century Germany». In: *Archives Italiennes de Biologie* 149 (Dec. 2011), pp. 47–56.
- [44] A. L. Hodgkin and A. F. Huxley. «A quantitative description of membrane current and its application to conduction and excitation in nerve». eng. In: *The Journal of Physiology* 117.4 (Aug. 1952), pp. 500–544. ISSN: 0022-3751. DOI: 10.1113/jphysiol.1952.sp004764.
- [45] Warren S. McCulloch and Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». en. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: <https://doi.org/10.1007/BF02478259> (visited on 04/03/2025).
- [46] D. O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. New York: Psychology Press, Apr. 2005. ISBN: 978-1-4106-1240-3. DOI: 10.4324/9781410612403.

- [47] D. H. Hubel and T. N. Wiesel. «Receptive fields of single neurones in the cat's striate cortex». In: *The Journal of Physiology* 148.3 (Oct. 1959), pp. 574–591. ISSN: 0022-3751. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1363130/> (visited on 04/03/2025).
- [48] David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. The MIT Press, July 2010. ISBN: 9780262514620. DOI: 10.7551/mitpress/9780262514620.001.0001. URL: <https://doi.org/10.7551/mitpress/9780262514620.001.0001>.
- [49] Corinne Teeter et al. «Generalized leaky integrate-and-fire models classify multiple neuron types». en. In: *Nature Communications* 9.1 (Feb. 2018). Publisher: Nature Publishing Group, p. 709. ISSN: 2041-1723. DOI: 10.1038/s41467-017-02717-4. URL: <https://www.nature.com/articles/s41467-017-02717-4> (visited on 03/21/2024).
- [50] Richard FitzHugh. «Impulses and Physiological States in Theoretical Models of Nerve Membrane». In: *Biophysical Journal* 1.6 (July 1961), pp. 445–466. ISSN: 0006-3495. DOI: 10.1016/S0006-3495(61)86902-6. URL: <https://www.sciencedirect.com/science/article/pii/S0006349561869026> (visited on 04/03/2025).
- [51] E.M. Izhikevich. «Simple model of spiking neurons». In: *IEEE Transactions on Neural Networks* 14.6 (Nov. 2003). Conference Name: IEEE Transactions on Neural Networks, pp. 1569–1572. ISSN: 1941-0093. DOI: 10.1109/TNN.2003.820440. URL: <https://ieeexplore.ieee.org/document/1257420> (visited on 01/24/2024).
- [52] Filipp Akopyan et al. «TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip». In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (Oct. 2015). Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 1537–1557. ISSN: 1937-4151. DOI: 10.1109/TCAD.2015.2474396. URL: <https://ieeexplore.ieee.org/document/7229264> (visited on 12/07/2023).
- [53] Mike Davies et al. «Loihi: A Neuromorphic Manycore Processor with On-Chip Learning». In: *IEEE Micro* 38.1 (Jan. 2018). Conference Name: IEEE Micro, pp. 82–99. ISSN: 1937-4143. DOI: 10.1109/MM.2018.112130359. URL: <https://ieeexplore.ieee.org/document/8259423> (visited on 12/07/2023).
- [54] Christian Pehle et al. «The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity». In: *Frontiers in Neuroscience* 16 (2022). ISSN: 1662-453X. URL: <https://www.frontiersin.org/articles/10.3389/fnins.2022.795876> (visited on 12/07/2023).

- [55] Eustace Painkras et al. «SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation». In: *IEEE Journal of Solid-State Circuits* 48.8 (Aug. 2013). Conference Name: IEEE Journal of Solid-State Circuits, pp. 1943–1953. ISSN: 1558-173X. DOI: 10.1109/JSSC.2013.2259038. URL: <https://ieeexplore.ieee.org/abstract/document/6515159> (visited on 12/07/2023).
- [56] James Stone. *Principles of Neural Information Theory: Computational Neuroscience and Metabolic Efficiency*. June 2018. ISBN: 978-0-9933679-2-2.
- [57] W. Gerstner. «SPIKING NEURON MODELS Single Neurons , Populations , Plasticity». In: 2002. URL: <https://www.semanticscholar.org/paper/SPIKING-NEURON-MODELS-Single-Neurons-%2C-Populations-Gerstner/849f849f647bc8d49c025dc408f4e927e12bceec> (visited on 04/03/2025).
- [58] Wolfgang Maass. «Networks of spiking neurons: The third generation of neural network models». In: *Neural Networks* 10.9 (Dec. 1997), pp. 1659–1671. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(97)00011-7. URL: <https://www.sciencedirect.com/science/article/pii/S0893608097000117> (visited on 01/31/2024).
- [59] Shih-Chii Liu et al. *Event-Based Neuromorphic Systems*. en. Google-Books-ID: MuvsBQAAQBAJ. John Wiley & Sons, Feb. 2015. ISBN: 978-0-470-01849-1.
- [60] F. Rosenblatt. «The perceptron: A probabilistic model for information storage and organization in the brain». In: *Psychological Review* 65.6 (1958). Place: US Publisher: American Psychological Association, pp. 386–408. ISSN: 1939-1471. DOI: 10.1037/h0042519.
- [61] Y. Lecun et al. «Gradient-based learning applied to document recognition». In: *Proceedings of the IEEE* 86.11 (Nov. 1998). Conference Name: Proceedings of the IEEE, pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791. URL: <https://ieeexplore.ieee.org/document/726791> (visited on 10/04/2023).
- [62] G. Cybenko. «Approximation by superpositions of a sigmoidal function». en. In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/BF02551274. URL: <https://doi.org/10.1007/BF02551274> (visited on 04/03/2025).
- [63] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. en. Google-Books-ID: Np9SDQAAQBAJ. MIT Press, Nov. 2016. ISBN: 978-0-262-03561-3.

- [64] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. «Spiker+: a framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge». In: *IEEE Transactions on Emerging Topics in Computing* (2024). Conference Name: IEEE Transactions on Emerging Topics in Computing, pp. 1–15. ISSN: 2168-6750. DOI: 10.1109/TETC.2024.3511676. URL: <https://ieeexplore.ieee.org/document/10794606> (visited on 02/05/2025).
- [65] Nicolas Brunel and Mark C. W. van Rossum. «Quantitative investigations of electrical nerve excitation treated as polarization». In: *Biological Cybernetics* 97.5 (Dec. 2007), pp. 341–349. ISSN: 1432-0770. DOI: 10.1007/s00422-007-0189-6. URL: <https://doi.org/10.1007/s00422-007-0189-6>.
- [66] Romain Brette and Wulfram Gerstner. «Adaptive exponential integrate-and-fire model as an effective description of neuronal activity». eng. In: *Journal of Neurophysiology* 94.5 (Nov. 2005), pp. 3637–3642. ISSN: 0022-3077. DOI: 10.1152/jn.00686.2005.
- [67] G. Bard Ermentrout and David H. Terman. *Mathematical Foundations of Neuroscience*. en. Google-Books-ID: ZkWiVu_OXgIC. Springer Science & Business Media, July 2010. ISBN: 978-0-387-87707-5.
- [68] Wenzhe Guo et al. «Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems». In: *Frontiers in Neuroscience* 15 (2021). ISSN: 1662-453X. URL: <https://www.frontiersin.org/articles/10.3389/fnins.2021.638474> (visited on 12/08/2023).
- [69] Daniel Auge et al. «A Survey of Encoding Techniques for Signal Processing in Spiking Neural Networks». en. In: *Neural Processing Letters* 53.6 (Dec. 2021), pp. 4693–4710. ISSN: 1573-773X. DOI: 10.1007/s11063-021-10562-2. URL: <https://doi.org/10.1007/s11063-021-10562-2> (visited on 12/08/2023).
- [70] Roger M. Enoka and Jacques Duchateau. «Rate Coding and the Control of Muscle Force». eng. In: *Cold Spring Harbor Perspectives in Medicine* 7.10 (Oct. 2017), a029702. ISSN: 2157-1422. DOI: 10.1101/cshperspect.a029702.
- [71] Professor David Heeger. «Poisson Model of Spike Generation». en. In: ().
- [72] Anne-Marie M. Oswald, Brent Doiron, and Leonard Maler. «Interval coding. I. Burst interspike intervals as indicators of stimulus intensity». eng. In: *Journal of Neurophysiology* 97.4 (Apr. 2007), pp. 2731–2743. ISSN: 0022-3077. DOI: 10.1152/jn.00987.2006.

- [73] Stefano Panzeri et al. «The Role of Spike Timing in the Coding of Stimulus Location in Rat Somatosensory Cortex». In: *Neuron* 29.3 (Mar. 2001), pp. 769–777. ISSN: 0896-6273. DOI: 10.1016/S0896-6273(01)00251-3. URL: <https://www.sciencedirect.com/science/article/pii/S0896627301002513> (visited on 04/03/2025).
- [74] Balint Petro, Nikola Kasabov, and Rita M. Kiss. «Selection and Optimization of Temporal Spike Encoding Methods for Spiking Neural Networks». In: *IEEE Transactions on Neural Networks and Learning Systems* 31.2 (Feb. 2020). Conference Name: IEEE Transactions on Neural Networks and Learning Systems, pp. 358–370. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2019.2906158. URL: <https://ieeexplore.ieee.org/document/8689349> (visited on 04/03/2025).
- [75] Zihan Pan et al. «Neural Population Coding for Effective Temporal Classification». In: *2019 International Joint Conference on Neural Networks (IJCNN)*. ISSN: 2161-4407. July 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8851858. URL: <https://ieeexplore.ieee.org/document/8851858> (visited on 04/03/2025).
- [76] Alexander Bain. *The Senses and the Intellect*. en. Google-Books-ID: iJgu5v1CJ8gC. J. W. Parker, 1855.
- [77] William James. *The principles of psychology, Vol I*. The principles of psychology, Vol I. Pages: xii, 697. New York, NY, US: Henry Holt and Co, 1890. DOI: 10.1037/10538-000.
- [78] Carla J. Shatz. «The Developing Brain». In: *Scientific American* 267.3 (1992). Publisher: Scientific American, a division of Nature America, Inc., pp. 60–67. ISSN: 0036-8733. URL: <https://www.jstor.org/stable/24939213> (visited on 04/03/2025).
- [79] Jerzy Konorski. *Conditioned reflexes and neuron organization*. Conditioned reflexes and neuron organization. Pages: xiv, 267. New York, NY, US: Cambridge University Press, 1948.
- [80] Elie L. Bienenstock, Leon N Cooper, and Paul W. Munro. «Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex». In: *How We Learn; How We Remember: Toward an Understanding of Brain and Neural Systems*. Vol. Volume 10. World Scientific Series in 20th Century Physics Volume 10. WORLD SCIENTIFIC, Sept. 1995, pp. 79–95. ISBN: 978-981-02-1814-0. DOI: 10.1142/9789812795885_0006. URL: https://www.worldscientific.com/doi/abs/10.1142/9789812795885_0006 (visited on 04/03/2025).

-
- [81] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. «Spike-Timing-Dependent Plasticity: A Comprehensive Overview». English. In: *Frontiers in Synaptic Neuroscience* 4 (July 2012). Publisher: Frontiers. ISSN: 1663-3563. DOI: 10.3389/fnsyn.2012.00002. URL: <https://www.frontiersin.org/articles/10.3389/fnsyn.2012.00002> (visited on 03/13/2024).
- [82] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. «Learning representations by back-propagating errors». en. In: *Nature* 323.6088 (Oct. 1986). Number: 6088 Publisher: Nature Publishing Group, pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: <https://www.nature.com/articles/323533a0> (visited on 01/25/2024).
- [83] Richard S. Sutton. «Learning to predict by the methods of temporal differences». en. In: *Machine Learning* 3.1 (Aug. 1988), pp. 9–44. ISSN: 1573-0565. DOI: 10.1007/BF00115009. URL: <https://doi.org/10.1007/BF00115009> (visited on 04/03/2025).
- [84] Kyunghyun Cho et al. «Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation». In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: <https://aclanthology.org/D14-1179/> (visited on 04/03/2025).
- [85] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. «Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks». In: *IEEE Signal Processing Magazine* 36.6 (Nov. 2019). Conference Name: IEEE Signal Processing Magazine, pp. 51–63. ISSN: 1558-0792. DOI: 10.1109/MSP.2019.2931595. URL: <https://ieeexplore.ieee.org/document/8891809> (visited on 10/04/2023).
- [86] Jason K. Eshraghian et al. *Training Spiking Neural Networks Using Lessons From Deep Learning*. en. Sept. 2021. URL: <https://arxiv.org/abs/2109.12894v5> (visited on 06/22/2023).
- [87] Hyunho Seok et al. «Beyond von Neumann Architecture: Brain-Inspired Artificial Neuromorphic Devices and Integrated Computing». en. In: *Advanced Electronic Materials* 10.8 (2024). eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/aelm.2300839>. ISSN: 2199-160X. DOI: 10.1002/aelm.202300839. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/aelm.202300839> (visited on 02/06/2025).

- [88] Pudi Dhilleswararao et al. «Efficient Hardware Architectures for Accelerating Deep Neural Networks: Survey». In: *IEEE Access* 10 (2022). Conference Name: IEEE Access, pp. 131788–131828. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3229767. URL: <https://ieeexplore.ieee.org/document/9988986> (visited on 12/13/2024).
- [89] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. «Spiker: an FPGA-optimized Hardware accelerator for Spiking Neural Networks». In: *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. ISSN: 2159-3477. July 2022, pp. 14–19. DOI: 10.1109/ISVLSI54635.2022.00016.
- [90] Benjamin Cramer et al. «The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks». In: *IEEE Transactions on Neural Networks and Learning Systems* 33.7 (July 2022). Conference Name: IEEE Transactions on Neural Networks and Learning Systems, pp. 2744–2757. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2020.3044364. URL: <https://ieeexplore.ieee.org/document/9311226> (visited on 10/04/2023).
- [91] Sören Becker et al. «AudioMNIST: Exploring Explainable Artificial Intelligence for audio analysis on a simple benchmark». In: *Journal of the Franklin Institute* 361.1 (Jan. 2024), pp. 418–428. ISSN: 0016-0032. DOI: 10.1016/j.jfranklin.2023.11.038. URL: <https://www.sciencedirect.com/science/article/pii/S0016003223007536> (visited on 09/05/2024).
- [92] Dario Padovano et al. «SpikeExplorer: Hardware-Oriented Design Space Exploration for Spiking Neural Networks on FPGA». en. In: *Electronics* 13.9 (Jan. 2024). Number: 9 Publisher: Multidisciplinary Digital Publishing Institute, p. 1744. ISSN: 2079-9292. DOI: 10.3390/electronics13091744. URL: <https://www.mdpi.com/2079-9292/13/9/1744> (visited on 09/06/2024).
- [93] Marcel Stimberg, Romain Brette, and Dan FM Goodman. «Brian 2, an intuitive and efficient neural simulator». In: *eLife* 8 (Aug. 2019). Ed. by Frances K Skinner et al. Publisher: eLife Sciences Publications, Ltd, e47314. ISSN: 2050-084X. DOI: 10.7554/eLife.47314. URL: <https://doi.org/10.7554/eLife.47314> (visited on 01/29/2024).
- [94] Nitin Rathi et al. «Exploring Neuromorphic Computing Based on Spiking Neural Networks: Algorithms to Hardware». In: *ACM Comput. Surv.* 55.12 (Mar. 2023), 243:1–243:49. ISSN: 0360-0300. DOI: 10.1145/3571155. URL: <https://dl.acm.org/doi/10.1145/3571155> (visited on 12/13/2024).
- [95] Arindam Basu et al. *Spiking Neural Network Integrated Circuits: A Review of Trends and Future Directions*. arXiv:2203.07006 [cs]. Mar. 2022. DOI: 10.48550/arXiv.2203.07006. URL: <http://arxiv.org/abs/2203.07006> (visited on 06/22/2023).

-
- [96] Fabio Pavanello et al. «Special Session: Neuromorphic hardware design and reliability from traditional CMOS to emerging technologies». In: *2023 IEEE 41st VLSI Test Symposium (VTS)*. ISSN: 2375-1053. Apr. 2023, pp. 1–10. DOI: 10.1109/VTS56346.2023.10139932.
- [97] Christian Mayr, Sebastian Hoepfner, and Steve Furber. *SpiNNaker 2: A 10 Million Core Processor System for Brain Simulation and Machine Learning*. arXiv:1911.02385 [cs]. Nov. 2019. DOI: 10.48550/arXiv.1911.02385. URL: <http://arxiv.org/abs/1911.02385> (visited on 12/07/2023).
- [98] Lei Deng et al. «Tianjic: A Unified and Scalable Chip Bridging Spike-Based and Continuous Neural Computation». In: *IEEE Journal of Solid-State Circuits* 55.8 (Aug. 2020). Conference Name: IEEE Journal of Solid-State Circuits, pp. 2228–2246. ISSN: 1558-173X. DOI: 10.1109/JSSC.2020.2970709. URL: <https://ieeexplore.ieee.org/document/8998338> (visited on 12/07/2023).
- [99] Ole Richter et al. *DYNAP-SE2: a scalable multi-core dynamic neuromorphic asynchronous spiking neural network processor*. arXiv:2310.00564 [cs]. Nov. 2023. URL: <http://arxiv.org/abs/2310.00564> (visited on 12/07/2023).
- [100] Ben Varkey Benjamin et al. «Neurogrid: A Mixed-Analog-Digital Multichip System for Large-Scale Neural Simulations». In: *Proceedings of the IEEE* 102.5 (May 2014). Conference Name: Proceedings of the IEEE, pp. 699–716. ISSN: 1558-2256. DOI: 10.1109/JPROC.2014.2313565. URL: <https://ieeexplore.ieee.org/document/6805187> (visited on 12/07/2023).
- [101] Dylan Muir, Felix Bauer, and Philipp Weidel. «Rockpool Documentation». eng. In: (Oct. 2024). Publisher: Zenodo. DOI: 10.5281/zenodo.14032139. URL: <https://zenodo.org/records/14032139> (visited on 04/06/2025).
- [102] Trevor Bekolay et al. «Nengo: a Python tool for building large-scale functional brain models». English. In: *Frontiers in Neuroinformatics* 7 (Jan. 2014). Publisher: Frontiers. ISSN: 1662-5196. DOI: 10.3389/fninf.2013.00048. URL: <https://www.frontiersin.org/journals/neuroinformatics/articles/10.3389/fninf.2013.00048/full> (visited on 04/06/2025).
- [103] Surya Narayanan et al. «SpinalFlow: An Architecture and Dataflow Tailored for Spiking Neural Networks». In: *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. May 2020, pp. 349–362. DOI: 10.1109/ISCA45697.2020.00038. URL: <https://ieeexplore.ieee.org/document/9138926> (visited on 12/07/2023).
- [104] De Ma et al. «Darwin: A neuromorphic hardware co-processor based on spiking neural networks». In: *Journal of Systems Architecture* 77 (June 2017), pp. 43–51. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2017.01.003. URL: <https://www.sciencedirect.com/science/article/pii/S1383762117300231> (visited on 12/06/2023).

- [105] Charlotte Frenkel, Jean-Didier Legat, and David Bol. «A 65-nm 738k-Synapse/mm² Quad-Core Binary-Weight Digital Neuromorphic Processor with Stochastic Spike-Driven Online Learning». In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. ISSN: 2158-1525. May 2019, pp. 1–5. DOI: 10.1109/ISCAS.2019.8702793. URL: <https://ieeexplore.ieee.org/document/8702793> (visited on 12/07/2023).
- [106] Charlotte Frenkel et al. «A 0.086-mm² 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28-nm CMOS». In: *IEEE Transactions on Biomedical Circuits and Systems* 13.1 (Feb. 2019). Conference Name: IEEE Transactions on Biomedical Circuits and Systems, pp. 145–158. ISSN: 1940-9990. DOI: 10.1109/TBCAS.2018.2880425.
- [107] Charlotte Frenkel and Giacomo Indiveri. «ReckOn: A 28nm Sub-mm² Task-Agnostic Spiking Recurrent Neural Network Processor Enabling On-Chip Learning over Second-Long Timescales». In: *2022 IEEE International Solid-State Circuits Conference (ISSCC)*. Vol. 65. ISSN: 2376-8606. Feb. 2022, pp. 1–3. DOI: 10.1109/ISSCC42614.2022.9731734.
- [108] Daniel Neil and Shih-Chii Liu. «Minitaur, an Event-Driven FPGA-Based Spiking Network Accelerator». In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.12 (Dec. 2014). Conference Name: IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 2621–2628. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2013.2294916. URL: <https://ieeexplore.ieee.org/document/6701396> (visited on 12/06/2023).
- [109] Hanwen Liu et al. «A Low Power and Low Latency FPGA-Based Spiking Neural Network Accelerator». In: *2023 International Joint Conference on Neural Networks (IJCNN)*. ISSN: 2161-4407. June 2023, pp. 1–8. DOI: 10.1109/IJCNN54540.2023.10191153. URL: <https://ieeexplore.ieee.org/document/10191153> (visited on 12/06/2023).
- [110] Shikhar Gupta, Arpan Vyas, and Gaurav Trivedi. «FPGA Implementation of Simplified Spiking Neural Network». In: *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. Nov. 2020, pp. 1–4. DOI: 10.1109/ICECS49266.2020.9294790. URL: <https://ieeexplore.ieee.org/abstract/document/9294790> (visited on 12/06/2023).
- [111] Jindong Li et al. «FireFly: A High-Throughput Hardware Accelerator for Spiking Neural Networks With Efficient DSP and Memory Optimization». In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 31.8 (Aug. 2023). Conference Name: IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 1178–1191. ISSN: 1557-9999. DOI: 10.1109/TVLSI.2023.3279349. URL: <https://ieeexplore.ieee.org/document/10143752> (visited on 12/06/2023).

- [112] Sathish Panchapakesan, Zhenman Fang, and Jian Li. «SyncNN: Evaluating and Accelerating Spiking Neural Networks on FPGAs». In: *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. ISSN: 1946-1488. Aug. 2021, pp. 286–293. DOI: 10.1109/FPL53798.2021.00058. URL: <https://ieeexplore.ieee.org/document/9556367> (visited on 12/06/2023).
- [113] Alireza Khodamoradi, Kristof Denolf, and Ryan Kastner. «S2N2: A FPGA Accelerator for Streaming Spiking Neural Networks». In: *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '21. New York, NY, USA: Association for Computing Machinery, Feb. 2021, pp. 194–205. ISBN: 978-1-4503-8218-2. DOI: 10.1145/3431920.3439283. URL: <https://dl.acm.org/doi/10.1145/3431920.3439283> (visited on 12/06/2023).
- [114] Jianhui Han et al. «Hardware implementation of spiking neural networks on FPGA». In: *Tsinghua Science and Technology* 25.4 (Aug. 2020). Conference Name: Tsinghua Science and Technology, pp. 479–486. ISSN: 1007-0214. DOI: 10.26599/TST.2019.9010019. URL: <https://ieeexplore.ieee.org/document/8954866> (visited on 12/06/2023).
- [115] Guohe Zhang et al. «A low-cost and high-speed hardware implementation of spiking neural network». In: *Neurocomputing* 382 (Mar. 2020), pp. 106–115. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2019.11.045. URL: <https://www.sciencedirect.com/science/article/pii/S0925231219316479> (visited on 12/06/2023).
- [116] Yarib Nevarez et al. «Accelerating Spike-by-Spike Neural Networks on FPGA With Hybrid Custom Floating-Point and Logarithmic Dot-Product Approximation». In: *IEEE Access* 9 (2021). Conference Name: IEEE Access, pp. 80603–80620. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3085216. URL: <https://ieeexplore.ieee.org/document/9444398> (visited on 12/06/2023).
- [117] Hajar Asgari et al. «Low-Energy and Fast Spiking Neural Network For Context-Dependent Learning on FPGA». In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 67.11 (Nov. 2020). Conference Name: IEEE Transactions on Circuits and Systems II: Express Briefs, pp. 2697–2701. ISSN: 1558-3791. DOI: 10.1109/TCSII.2020.2968588. URL: <https://ieeexplore.ieee.org/document/8966255> (visited on 12/06/2023).
- [118] Sixu Li et al. «A Fast and Energy-Efficient SNN Processor With Adaptive Clock/Event-Driven Computation Scheme and Online Learning». In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 68.4 (Apr. 2021). Conference Name: IEEE Transactions on Circuits and Systems I: Regular Papers, pp. 1543–1552. ISSN: 1558-0806. DOI: 10.1109/TCSI.2021.3052885.

- [119] Nassim Abderrahmane, Edgar Lemaire, and Benoît Miramond. «Design Space Exploration of Hardware Spiking Neurons for Embedded Artificial Intelligence». In: *Neural Networks* 121 (Jan. 2020), pp. 366–386. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2019.09.024. URL: <https://www.sciencedirect.com/science/article/pii/S0893608019303041> (visited on 12/06/2023).
- [120] Daniel Gerlinghoff et al. «E3NE: An End-to-End Framework for Accelerating Spiking Neural Networks With Emerging Neural Encoding on FPGAs». English. In: *IEEE Transactions on Parallel and Distributed Systems* 33.11 (Nov. 2022). Publisher: IEEE Computer Society, pp. 3207–3219. ISSN: 1045-9219. DOI: 10.1109/TPDS.2021.3128945. URL: <https://www.computer.org/csdl/journal/td/2022/11/09619972/1yDfD2jSkBW> (visited on 12/06/2023).
- [121] Peter Diehl and Matthew Cook. «Unsupervised learning of digit recognition using spike-timing-dependent plasticity». In: *Frontiers in Computational Neuroscience* 9 (2015). ISSN: 1662-5188. URL: <https://www.frontiersin.org/articles/10.3389/fncom.2015.00099> (visited on 09/13/2023).
- [122] Devansh Chawda and Benaoumeur Senouci. «Fast prototyping of Quantized neural networks on an FPGA edge computing device with Brevitas and FINN». In: *2024 Fifteenth International Conference on Ubiquitous and Future Networks (ICUFN)*. ISSN: 2165-8536. July 2024, pp. 238–240. DOI: 10.1109/ICUFN61752.2024.10625618. URL: <https://ieeexplore.ieee.org/document/10625618> (visited on 04/06/2025).
- [123] Wei Fang et al. «SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence». In: *Science Advances* 9.40 (Oct. 2023). Publisher: American Association for the Advancement of Science, eadi1480. DOI: 10.1126/sciadv.adi1480. URL: <https://www.science.org/doi/10.1126/sciadv.adi1480> (visited on 09/05/2024).
- [124] Jens E. Pedersen et al. «Neuromorphic Intermediate Representation: A Unified Instruction Set for Interoperable Brain-Inspired Computing». In: (2023). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2311.14641. URL: <https://arxiv.org/abs/2311.14641> (visited on 09/05/2024).
- [125] Ting-Ting Wang et al. «Efficient Network Architecture Search Using Hybrid Optimizer». en. In: *Entropy* 24.5 (May 2022). Number: 5 Publisher: Multidisciplinary Digital Publishing Institute, p. 656. ISSN: 1099-4300. DOI: 10.3390/e24050656. URL: <https://www.mdpi.com/1099-4300/24/5/656> (visited on 01/31/2024).

- [126] Alireza Ghaffari and Yvon Savaria. «CNN2Gate: An Implementation of Convolutional Neural Networks Inference on FPGAs with Automated Design Space Exploration». en. In: *Electronics* 9.12 (Dec. 2020). Number: 12 Publisher: Multidisciplinary Digital Publishing Institute, p. 2200. ISSN: 2079-9292. DOI: 10.3390/electronics9122200. URL: <https://www.mdpi.com/2079-9292/9/12/2200> (visited on 01/31/2024).
- [127] Zoltan Czako, Gheorghe Sebestyen, and Anca Hangan. «AutomaticAI – A hybrid approach for automatic artificial intelligence algorithm selection and hyperparameter tuning». In: *Expert Systems with Applications* 182 (Nov. 2021), p. 115225. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2021.115225. URL: <https://www.sciencedirect.com/science/article/pii/S0957417421006576> (visited on 03/13/2024).
- [128] Adarsha Balaji et al. «NeuroXplorer 1.0: An Extensible Framework for Architectural Exploration with Spiking Neural Networks». In: *International Conference on Neuromorphic Systems 2021*. ICONS 2021. New York, NY, USA: Association for Computing Machinery, Oct. 2021, pp. 1–9. ISBN: 978-1-4503-8691-3. DOI: 10.1145/3477145.3477156. URL: <https://dl.acm.org/doi/10.1145/3477145.3477156> (visited on 01/29/2024).
- [129] Kurt Marti. *Optimization Under Stochastic Uncertainty: Methods, Control and Random Search Methods*. en. Vol. 296. International Series in Operations Research & Management Science. Cham: Springer International Publishing, 2020. DOI: 10.1007/978-3-030-55662-4. URL: <http://link.springer.com/10.1007/978-3-030-55662-4> (visited on 03/15/2024).
- [130] Fabrizio Ferrandi et al. «A Multi-objective Genetic Algorithm for Design Space Exploration in High-Level Synthesis». In: *2008 IEEE Computer Society Annual Symposium on VLSI*. ISSN: 2159-3477. Apr. 2008, pp. 417–422. DOI: 10.1109/ISVLSI.2008.73. URL: <https://ieeexplore.ieee.org/document/4556831> (visited on 04/03/2024).
- [131] Alessandro Savino, Alessandro Vallero, and Stefano Di Carlo. «ReDO: Cross-Layer Multi-Objective Design-Exploration Framework for Efficient Soft Error Resilient Systems». In: *IEEE Transactions on Computers* 67.10 (Oct. 2018). Conference Name: IEEE Transactions on Computers, pp. 1462–1477. ISSN: 1557-9956. DOI: 10.1109/TC.2018.2818735. URL: <https://ieeexplore.ieee.org/document/8323226> (visited on 04/03/2024).
- [132] Sepide Saeedi, Alessandro Savino, and Stefano Di Carlo. «Design Space Exploration of Approximate Computing Techniques with a Reinforcement Learning Approach». In: *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. ISSN: 2325-6664. June 2023, pp. 167–170. DOI: 10.1109/DSN-W58399.2023.00047.

- URL: <https://ieeexplore.ieee.org/abstract/document/10207095> (visited on 01/31/2024).
- [133] Brandon Reagen et al. «A case for efficient accelerator design space exploration via Bayesian optimization». In: *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. July 2017, pp. 1–6. DOI: 10.1109/ISLPED.2017.8009208. URL: <https://ieeexplore.ieee.org/document/8009208> (visited on 01/31/2024).
- [134] James G. March. «Exploration and Exploitation in Organizational Learning». In: *Organization Science* 2.1 (Feb. 1991). Publisher: INFORMS, pp. 71–87. ISSN: 1047-7039. DOI: 10.1287/orsc.2.1.71. URL: <https://pubsonline.informs.org/doi/10.1287/orsc.2.1.71> (visited on 03/15/2024).
- [135] Antonio Candelieri. «A Gentle Introduction to Bayesian Optimization». In: *2021 Winter Simulation Conference (WSC)*. ISSN: 1558-4305. Dec. 2021, pp. 1–16. DOI: 10.1109/WSC52266.2021.9715413. URL: <https://ieeexplore.ieee.org/document/9715413> (visited on 12/13/2024).
- [136] Youngeun Kim et al. «Neural Architecture Search for Spiking Neural Networks». en. In: *Computer Vision – ECCV 2022*. Ed. by Shai Avidan et al. Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, 2022, pp. 36–56. ISBN: 978-3-031-20053-3. DOI: 10.1007/978-3-031-20053-3_3.
- [137] Rachmad Vidya Wicaksana Putra and Muhammad Shafique. «Q-SpiNN: A Framework for Quantizing Spiking Neural Networks». In: *2021 International Joint Conference on Neural Networks (IJCNN)* (July 2021). Conference Name: 2021 International Joint Conference on Neural Networks (IJCNN) ISBN: 9781665439008 Place: Shenzhen, China Publisher: IEEE, pp. 1–8. DOI: 10.1109/IJCNN52387.2021.9534087. URL: <https://ieeexplore.ieee.org/document/9534087/> (visited on 04/03/2024).
- [138] Chen Li, Lei Ma, and Steve Furber. «Quantization Framework for Fast Spiking Neural Networks». English. In: *Frontiers in Neuroscience* 16 (July 2022). Publisher: Frontiers. ISSN: 1662-453X. DOI: 10.3389/fnins.2022.918793. URL: <https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2022.918793/full> (visited on 04/03/2024).
- [139] Andrea Castagnetti, Alain Pegatoquet, and Benoît Miramond. «Trainable quantization for Speedy Spiking Neural Networks». English. In: *Frontiers in Neuroscience* 17 (Mar. 2023). Publisher: Frontiers. ISSN: 1662-453X. DOI: 10.3389/fnins.2023.1154241. URL: <https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2023.1154241/full> (visited on 04/03/2024).

- [140] E. Bakshy et al. «AE: A domain-agnostic platform for adaptive experimentation». In: 2018. URL: <https://www.semanticscholar.org/paper/AE%3A-A-domain-agnostic-platform-for-adaptive-Bakshy-Dworkin/1c088a1a0cf5e4a7627d92fc1ed0cd97ee362bc5> (visited on 04/06/2025).
- [141] Diederik P. Kingma and Jimmy Ba. «Adam: A Method for Stochastic Optimization». In: *CoRR* (Dec. 2014). URL: <https://www.semanticscholar.org/paper/Adam%3A-A-Method-for-Stochastic-Optimization-Kingma-Ba/a6cb366736791bcccc5c8639de5a8f9636bf87e8> (visited on 04/03/2024).
- [142] Andrew Y. Ng. «Feature selection, L1 vs. L2 regularization, and rotational invariance». In: *Proceedings of the twenty-first international conference on Machine learning*. ICML '04. New York, NY, USA: Association for Computing Machinery, July 2004, p. 78. ISBN: 978-1-58113-838-2. DOI: 10.1145/1015330.1015435. URL: <https://dl.acm.org/doi/10.1145/1015330.1015435> (visited on 04/03/2024).
- [143] Arnon Amir et al. «A Low Power, Fully Event-Based Gesture Recognition System». In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, July 2017, pp. 7388–7397. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.781. URL: <https://ieeexplore.ieee.org/document/8100264/> (visited on 03/11/2024).
- [144] Taras Iakymchuk et al. «Simplified spiking neural network architecture and STDP learning algorithm applied to image classification». In: *EURASIP Journal on Image and Video Processing* 2015.1 (Feb. 2015), p. 4. ISSN: 1687-5281. DOI: 10.1186/s13640-015-0059-4. URL: <https://doi.org/10.1186/s13640-015-0059-4> (visited on 12/12/2023).
- [145] Dario Amodè et al. *Concrete Problems in AI Safety*. arXiv:1606.06565 [cs]. July 2016. DOI: 10.48550/arXiv.1606.06565. URL: <http://arxiv.org/abs/1606.06565> (visited on 12/13/2024).
- [146] Sylvestre-Alvise Rebuffi et al. «iCaRL: Incremental Classifier and Representation Learning». In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. ISSN: 1063-6919. July 2017, pp. 5533–5542. DOI: 10.1109/CVPR.2017.587. URL: <https://ieeexplore.ieee.org/document/8100070> (visited on 12/13/2024).
- [147] Lorenzo Pellegrini et al. «Latent Replay for Real-Time Continual Learning». In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Oct. 2020, pp. 10203–10209. DOI: 10.1109/IROS45743.2020.9341460. URL: <https://ieeexplore.ieee.org/document/9341460> (visited on 12/13/2024).
- [148] Benjamin Ehret et al. «Continual learning in recurrent neural networks». en. In: Oct. 2020. URL: <https://openreview.net/forum?id=8xeBUGD8u9> (visited on 12/13/2024).

- [149] Vincenzo Lomonaco, Davide Maltoni, and Lorenzo Pellegrini. «Rehearsal-Free Continual Learning over Small Non-I.I.D. Batches». In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. ISSN: 2160-7516. June 2020, pp. 989–998. DOI: 10.1109/CVPRW50498.2020.00131. URL: <https://ieeexplore.ieee.org/document/9150818> (visited on 12/13/2024).
- [150] Nicolas Skatchkovsky, Hyeryung Jang, and Osvaldo Simeone. «Bayesian continual learning via spiking neural networks». English. In: *Frontiers in Computational Neuroscience* 16 (Nov. 2022). Publisher: Frontiers. ISSN: 1662-5188. DOI: 10.3389/fncom.2022.1037976. URL: <https://www.frontiersin.org/journals/computational-neuroscience/articles/10.3389/fncom.2022.1037976/full> (visited on 12/13/2024).
- [151] Rachmad Vidya Wicaksana Putra and Muhammad Shafique. «SpikeDyn: A Framework for Energy-Efficient Spiking Neural Networks with Continual and Unsupervised Learning Capabilities in Dynamic Environments». In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. ISSN: 0738-100X. Dec. 2021, pp. 1057–1062. DOI: 10.1109/DAC18074.2021.9586281. URL: <https://ieeexplore.ieee.org/document/9586281> (visited on 12/13/2024).
- [152] Bing Han et al. *Adaptive Reorganization of Neural Pathways for Continual Learning with Spiking Neural Networks*. arXiv:2309.09550 [cs]. Oct. 2024. DOI: 10.48550/arXiv.2309.09550. URL: <http://arxiv.org/abs/2309.09550> (visited on 12/13/2024).
- [153] Michela Proietti, Alessio Ragno, and Roberto Capobianco. «Memory Replay For Continual Learning With Spiking Neural Networks». In: *2023 IEEE 33rd International Workshop on Machine Learning for Signal Processing (MLSP)*. ISSN: 2161-0371. Sept. 2023, pp. 1–6. DOI: 10.1109/MLSP55844.2023.10285911. URL: <https://ieeexplore.ieee.org/document/10285911> (visited on 12/13/2024).
- [154] Alberto Dequino et al. «Compressed Latent Replays for Lightweight Continual Learning on Spiking Neural Networks». In: *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2024, pp. 240–245. DOI: 10.1109/ISVLSI61997.2024.00052.
- [155] Xavier Glorot and Yoshua Bengio. «Understanding the difficulty of training deep feedforward neural networks». In: Mar. 2010. URL: <https://www.semanticscholar.org/paper/Understanding-the-difficulty-of-training-deep-Glorot-Bengio/ea9d2a2b4ce11aaf85136840c65f3bc9c03ab649> (visited on 01/09/2025).

- [156] James Kirkpatrick et al. «Overcoming catastrophic forgetting in neural networks». In: *Proceedings of the National Academy of Sciences* 114.13 (Mar. 2017). Publisher: Proceedings of the National Academy of Sciences, pp. 3521–3526. DOI: 10.1073/pnas.1611835114. URL: <https://www.pnas.org/doi/full/10.1073/pnas.1611835114> (visited on 04/08/2025).
- [157] Friedemann Zenke, Ben Poole, and Surya Ganguli. «Continual learning through synaptic intelligence». In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, Aug. 2017, pp. 3987–3995. (Visited on 04/08/2025).
- [158] Andrei A. Rusu et al. *Progressive Neural Networks*. arXiv:1606.04671 [cs]. Oct. 2022. DOI: 10.48550/arXiv.1606.04671. URL: <http://arxiv.org/abs/1606.04671> (visited on 04/08/2025).
- [159] Jaehong Yoon et al. «Lifelong Learning with Dynamically Expandable Networks». en. In: Feb. 2018. URL: <https://openreview.net/forum?id=Sk7KsfW0-> (visited on 04/08/2025).
- [160] Guillaume Bellec et al. «A solution to the learning dilemma for recurrent networks of spiking neurons». en. In: *Nature Communications* 11.1 (July 2020). Number: 1 Publisher: Nature Publishing Group, p. 3625. ISSN: 2041-1723. DOI: 10.1038/s41467-020-17236-y. URL: <https://www.nature.com/articles/s41467-020-17236-y> (visited on 12/06/2023).
- [161] U. Rajendra Acharya et al. «Heart rate variability: a review». eng. In: *Medical & Biological Engineering & Computing* 44.12 (Dec. 2006), pp. 1031–1051. ISSN: 0140-0118. DOI: 10.1007/s11517-006-0119-0.
- [162] Zetao Zhu et al. «A fitness training optimization system based on heart rate prediction under different activities». eng. In: *Methods (San Diego, Calif.)* 205 (Sept. 2022), pp. 89–96. ISSN: 1095-9130. DOI: 10.1016/j.ymeth.2022.06.006.
- [163] Paul A. Heidenreich et al. «Forecasting the Future of Cardiovascular Disease in the United States». In: *Circulation* 123.8 (Mar. 2011). Publisher: American Heart Association, pp. 933–944. DOI: 10.1161/CIR.0b013e31820a55f5. URL: <https://www.ahajournals.org/doi/10.1161/CIR.0b013e31820a55f5> (visited on 01/09/2025).
- [164] Hannah Davidoff et al. «Unraveling the relationship between heart rate and agitation in people with dementia». en. In: *Alzheimer's & Dementia* 19.S18 (2023). eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/alz.078916>, e078916. ISSN: 1552-5279. DOI: 10.1002/alz.078916. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/alz.078916> (visited on 01/09/2025).

- [165] Sienna Caspar et al. «Nonpharmacological Management of Behavioral and Psychological Symptoms of Dementia: What Works, in What Circumstances, and Why?» In: *Innovation in Aging* 1.3 (Mar. 2018), igy001. ISSN: 2399-5300. DOI: 10.1093/geroni/igy001. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6176983/> (visited on 01/09/2025).
- [166] Elisabetta Chicca et al. «Neuromorphic Electronic Circuits for Building Autonomous Cognitive Systems». In: *Proceedings of the IEEE* 102.9 (Sept. 2014). Conference Name: Proceedings of the IEEE, pp. 1367–1388. ISSN: 1558-2256. DOI: 10.1109/JPROC.2014.2313954. URL: <https://ieeexplore.ieee.org/document/6809149> (visited on 01/09/2025).
- [167] Elisa Donati and Giacomo Indiveri. «Neuromorphic bioelectronic medicine for nervous system interfaces: from neural computational primitives to medical applications». en. In: *Progress in Biomedical Engineering* 5.1 (Feb. 2023). Publisher: IOP Publishing, p. 013002. ISSN: 2516-1091. DOI: 10.1088/2516-1091/acb51c. URL: <https://dx.doi.org/10.1088/2516-1091/acb51c> (visited on 01/09/2025).
- [168] Felix Christian Bauer, Dylan Richard Muir, and Giacomo Indiveri. «Real-Time Ultra-Low Power ECG Anomaly Detection Using an Event-Driven Neuromorphic Processor». In: *IEEE Transactions on Biomedical Circuits and Systems* 13.6 (Dec. 2019). Conference Name: IEEE Transactions on Biomedical Circuits and Systems, pp. 1575–1582. ISSN: 1940-9990. DOI: 10.1109/TBCAS.2019.2953001. URL: <https://ieeexplore.ieee.org/document/8896021> (visited on 01/09/2025).
- [169] Anup Das et al. «Unsupervised heart-rate estimation in wearables with Liquid states and a probabilistic readout». In: *Neural Networks* 99 (Mar. 2018), pp. 134–147. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2017.12.015. URL: <https://www.sciencedirect.com/science/article/pii/S0893608017303003> (visited on 01/09/2025).
- [170] Mohammadali Sharifshazileh et al. «An electronic neuromorphic system for real-time detection of high frequency oscillations (HFO) in intracranial EEG». en. In: *Nature Communications* 12.1 (May 2021). Publisher: Nature Publishing Group, p. 3095. ISSN: 2041-1723. DOI: 10.1038/s41467-021-23342-2. URL: <https://www.nature.com/articles/s41467-021-23342-2> (visited on 01/09/2025).
- [171] Yongqiang Ma et al. «Neuromorphic Implementation of a Recurrent Neural Network for EMG Classification». In: *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. Aug. 2020, pp. 69–73. DOI: 10.1109/AICAS48895.2020.9073810. URL: <https://ieeexplore.ieee.org/document/9073810> (visited on 01/09/2025).

- [172] Antonio Vitale et al. «Neuromorphic Edge Computing for Biomedical Applications: Gesture Classification Using EMG Signals». In: *IEEE Sensors Journal* 22.20 (Oct. 2022). Conference Name: IEEE Sensors Journal, pp. 19490–19499. ISSN: 1558-1748. DOI: 10.1109/JSEN.2022.3194678. URL: <https://ieeexplore.ieee.org/document/9849452> (visited on 01/09/2025).
- [173] Emre Nefci et al. «Synthesizing cognition in neuromorphic electronic systems». In: *Proceedings of the National Academy of Sciences* 110.37 (Sept. 2013). Publisher: Proceedings of the National Academy of Sciences, E3468–E3476. DOI: 10.1073/pnas.1212083110. URL: <https://doi.org/10.1073/pnas.1212083110> (visited on 03/08/2024).
- [174] Dongchen Liang and Giacomo Indiveri. «A Neuromorphic Computational Primitive for Robust Context-Dependent Decision Making and Context-Dependent Stochastic Computation». In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 66.5 (May 2019). Conference Name: IEEE Transactions on Circuits and Systems II: Express Briefs, pp. 843–847. ISSN: 1558-3791. DOI: 10.1109/TCSII.2019.2907848. URL: <https://ieeexplore.ieee.org/document/8675453> (visited on 01/09/2025).
- [175] Misha Mahowald. *An Analog VLSI System for Stereoscopic Vision*. Boston, MA: Springer US, 1994. DOI: 10.1007/978-1-4615-2724-4. URL: <http://link.springer.com/10.1007/978-1-4615-2724-4> (visited on 04/08/2025).
- [176] Giacomo Indiveri et al. «Neuromorphic Silicon Neuron Circuits». English. In: *Frontiers in Neuroscience* 5 (May 2011). Publisher: Frontiers. ISSN: 1662-453X. DOI: 10.3389/fnins.2011.00073. URL: <https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2011.00073/full> (visited on 04/08/2025).
- [177] Shih-Chii Liu and Tobi Delbruck. «Neuromorphic sensory systems». In: *Current Opinion in Neurobiology*. Sensory systems 20.3 (June 2010), pp. 288–295. ISSN: 0959-4388. DOI: 10.1016/j.conb.2010.03.007. URL: <https://www.sciencedirect.com/science/article/pii/S0959438810000450> (visited on 04/08/2025).
- [178] Chiara Bartolozzi and Giacomo Indiveri. «Synaptic Dynamics in Analog VLSI». In: *Neural Computation* 19.10 (Oct. 2007), pp. 2581–2603. ISSN: 0899-7667. DOI: 10.1162/neco.2007.19.10.2581. URL: <https://doi.org/10.1162/neco.2007.19.10.2581> (visited on 04/08/2025).
- [179] Stephen R. Deiss, Rodney J. Douglas, and Adrian M. Whatley. «A pulse-coded communications infrastructure for neuromorphic systems». In: *Pulsed neural networks*. Cambridge, MA, USA: MIT Press, Feb. 1999, pp. 157–178. (Visited on 01/09/2025).

- [180] Ramin Khajeh, Francesco Fumarola, and L. F. Abbott. «Sparse balance: Excitatory-inhibitory networks with small bias currents and broadly distributed synaptic weights». en. In: *PLOS Computational Biology* 18.2 (Feb. 2022). Publisher: Public Library of Science, e1008836. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1008836. URL: <https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008836> (visited on 04/10/2025).
- [181] Wolfgang Maass. «On the Computational Power of Winner-Take-All». In: *Neural Computation* 12.11 (Nov. 2000), pp. 2519–2535. ISSN: 0899-7667. DOI: 10.1162/089976600300014827. URL: <https://ieeexplore.ieee.org/abstract/document/6789357> (visited on 04/10/2025).
- [182] Wolfgang Maass and Henry Markram. «On the computational power of circuits of spiking neurons». In: *Journal of Computer and System Sciences* 69.4 (Dec. 2004), pp. 593–616. ISSN: 0022-0000. DOI: 10.1016/j.jcss.2004.04.001. URL: <https://www.sciencedirect.com/science/article/pii/S0022000004000406> (visited on 04/10/2025).
- [183] Delaram Jarchi and Alexander J. Casson. «Description of a Database Containing Wrist PPG Signals Recorded during Physical Exercise with Both Accelerometer and Gyroscope Measures of Motion». en. In: *Data* 2.1 (Mar. 2017). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, p. 1. ISSN: 2306-5729. DOI: 10.3390/data2010001. URL: <https://www.mdpi.com/2306-5729/2/1/1> (visited on 01/09/2025).
- [184] Shyam Narayanan et al. «SPAIC: A sub- μ W/Channel, 16-Channel General-Purpose Event-Based Analog Front-End with Dual-Mode Encoders». In: *2023 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. ISSN: 2766-4465. Oct. 2023, pp. 1–5. DOI: 10.1109/BioCAS58349.2023.10388815. URL: <https://ieeexplore.ieee.org/document/10388815> (visited on 01/09/2025).
- [185] Marcello Zanghieri et al. «Event-based Low-Power and Low-Latency Regression Method for Hand Kinematics from Surface EMG». In: *2023 9th International Workshop on Advances in Sensors and Interfaces (IWASI)*. ISSN: 2836-7936. June 2023, pp. 293–298. DOI: 10.1109/IWASI58316.2023.10164372. URL: <https://ieeexplore.ieee.org/abstract/document/10164372> (visited on 01/09/2025).
- [186] Rachel Sava, Elisa Donati, and Giacomo Indiveri. «Feed-forward and recurrent inhibition for compressing and classifying high dynamic range biosignals in spiking neural network architectures». In: *2023 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. ISSN: 2766-4465. Oct. 2023, pp. 1–5. DOI: 10.1109/BioCAS58349.2023.10388963. URL: <https://ieeexplore.ieee.org/document/10388963> (visited on 01/09/2025).

- [187] Marcello Zanghieri et al. «Event-based Estimation of Hand Forces from High-Density Surface EMG on a Parallel Ultra-Low-Power Microcontroller». In: *IEEE Sensors Journal* (2024). Conference Name: IEEE Sensors Journal, pp. 1–1. ISSN: 1558-1748. DOI: 10.1109/JSEN.2024.3359917. URL: <https://ieeexplore.ieee.org/document/10422755> (visited on 01/09/2025).
- [188] Alessio Carpegna et al. *Neuromorphic Heart Rate Monitors: Neural State Machines for Monotonic Change Detection*. arXiv:2409.02618 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2409.02618. URL: <http://arxiv.org/abs/2409.02618> (visited on 09/13/2024).
- [189] Nicoletta Risi et al. «A Spike-Based Neuromorphic Architecture of Stereo Vision». English. In: *Frontiers in Neurorobotics* 14 (Nov. 2020). Publisher: Frontiers. ISSN: 1662-5218. DOI: 10.3389/fnbot.2020.568283. URL: <https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2020.568283/full> (visited on 08/19/2024).
- [190] Jingyue Zhao et al. «Learning inverse kinematics using neural computational primitives on neuromorphic hardware». In: *npj Robotics* 1.1 (Oct. 2023), p. 1. ISSN: 2731-4278. DOI: 10.1038/s44182-023-00001-w. URL: <https://doi.org/10.1038/s44182-023-00001-w>.

This Ph.D. thesis has been typeset by means of the T_EX-system facilities. The typesetting engine was pdfL^AT_EX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete T_EX-system installation.