

論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Neural Network Hardware Codesign via Nimble FPGA Implementations and Synchronized Spiking in Edge-AI
著者(和文)	Bartels Jim
Author(English)	Jim Bartels
出典(和文)	学位:博士(学術), 学位授与機関:東京工業大学, 報告番号:甲第12922号, 授与年月日:2024年9月20日, 学位の種別:課程博士, 審査員:伊藤 浩之,岡田 健一,徳田 崇,原 祐子,白根 篤史,Indiveri Giacomo
Citation(English)	Degree:Doctor (Academic), Conferring organization: Tokyo Institute of Technology, Report number:甲第12922号, Conferred date:2024/9/20, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Neural Network Hardware Codesign via Nimble FPGA Implementations and Synchronized Spiking in Edge-AI

Jim Bartels

A dissertation submitted in partial fulfilment of the
requirements for the degree of

Doctor of Philosophy

in the

Department of Electrical and Electronic Engineering

in the

School of Engineering

of

Tokyo Institute of Technology

Supervisor

Professor Hiroyuki Ito

Fall 2024

To the pursuit of knowledge and humanity,

Acknowledgment

First of all, I want to especially thank Professor Hiroyuki Ito for his extensive supervision, Professor Ludovico Minati and Professor Korkut Kaan Tokgoz. I am deeply grateful for their unwavering support, trust and positive criticism that have allowed me to grow as an aspiring researcher and aim for unexplored multidisciplinary research. It was them who opened my eyes to non-linear dynamics, hardware and machine learning.

I want to express my gratitude to Professor Giacomo Indiveri for hosting me for 11 months at the Institute of Informatics (INI) of the University of Zurich and ETH Zurich. My time at INI has completely changed my views about research and I was honored to be able to learn from many talented students and researchers. My deep thanks also goes to Dr. Saptarshi Ghosh with whom I had the pleasure to work together extensively on utilizing DYNAP-SE2 for epileptic seizure detection and various topics concerning non-linear dynamics and neuromorphic computing. Moreover, it was a pleasure to work with Olympia Gallou who I collaborated with on the simulation of DPI circuits for long-term iEEG conversion to events and DYNAP-SE2. It was them who kept lending me their enthusiasm and knowledge that have allowed me to create part of the works in this thesis.

Next to them, I want to sincerely thank Assistant Professor Sangyeop Lee, Professor Katsuyuki Machida and Professor Noboru Ishihara for their guidance and suggestions. I am also deeply grateful to Professor Kenichi Okada, Professor Takashi Tokuda, Associate Professor Atsushi Shirane, Associate Professor Yuko Hara, and Professor Giacomo Indiveri, for their valuable time in examining my thesis.

Furthermore, it was my pleasure to work together with Masamoto Fukawa and Shohei Otsubo on the cow project. I want to thank them and also Dr. Chao Li, who was involved heavily in data gathering, and A Sihan, who has given me invaluable knowledge and insights in cow behavior and animal science. I also want to thank Ikumi Rachi for her dedicated research on the reduction of sensor sampling frequency and decision trees. My thanks also goes to Alexander Varieras and Isabel Heidtmann, it was a pleasure to help supervise and collaborate on topics in Spiking Neural Networks and FPGA implementations. Lastly, I want to deeply thank Hagihara Aran and Maari Kazuki for their enthusiasm and work on the RNN implementations on FPGA. The work done in collaboration with them made me realize about the various interdisciplinary aspects and skills required for the works in this thesis.

Also, I want to thank Chie Mashiko for her role as secretary in arrangement of documents, submission to conferences, significant aid in the graduation process among other administrative tasks and her support.

Finally, I want to express gratitude towards my parents and family who have always been there for me and had to sustain the long distance over many years and my partner who has given me love and never-ending support.

Abstract

Artificial Intelligence (AI) implemented at the edge, involving neural networks or machine learning algorithms close to the source of data, dubbed Edge-AI, is identified as one of the key technologies to enhance efficiency across industries including manufacturing, healthcare, agriculture and transport. As a result, there is an increasing academic interest in advancing this technology, mostly considering AI accelerators that propose tailored hardware architectures of Artificial Neural Networks (ANN) such as Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Transformers. Implemented on Field Programmable Gate Arrays (FPGA) or designed as Application Specific Integrated Circuits (ASIC), these architectures promise significant speedup and efficiency gains over Graphical Processing Units (GPU) that see considerable use in cloud computing. However, this approach to Edge-AI overlooks the significant portion of Internet of Things (IoT) devices that generate time-series data.

These IoT devices often operate in the real world using sensors and are frequently limited by extremely scarce energy budgets. A substantial proportion of such devices are used in combination with processing on the cloud, implementing systems that enable out-of-hospital monitoring of epileptic seizures using neural recordings or classification of the behavior of livestock herds using accelerometers. This processing necessitates raw data transmission from sensor node to the cloud, taking up the majority of the energy budget. Here, Edge-AI that proposes to implement AI close to or on a device itself offers a distinct advantage as data transmission is severely reduced and real-time classification is enabled.

Alongside this issue, conventional AI algorithms significantly underperform compared to the efficiency of natural nervous systems, with the human brain as the most prominent example. Neuromorphic computing adopts the principles and structure of the brain and has given rise to hardware and algorithms that model the complex dynamics of neurons and their sparse, spike-based communication. Unlike conventional AI that relies on ANNs, neuromorphic computing utilizes Spiking Neural Networks (SNN) that have recently come close in terms of performance. However, most existing works adopt conventional ANN network structures and approximated learning rules to achieve comparable performance with higher efficiency. While this approach holds promise for advancing Edge-AI, it fails to recognize and utilize the complex dynamics in SNNs that give rise to emergent phenomena such as partial synchronization and chaos that can also be observed in biology.

The objective of this thesis is to advance the field of Edge-AI in terms of capability, accessibility and efficiency with partial solutions to the above two issues by means of neural network hardware codesign. First, an application-oriented approach is introduced that proposes hardware architectures for a Decision Tree (DT) and RNN that are implemented on an extremely resource-constrained FPGA. Second,

a dynamics-oriented approach is introduced that proposes computation and optimization by employing synchronization in SNNs on a mixed signal neuromorphic processor and FPGA. As a result, this thesis is split up into two main parts, the first part starts with a case study on Precision Livestock Farming (PLF) and cow behavior estimation that builds up to the design of low-power hardware architectures for low-frequency sensors, contributing to the accessibility and efficiency of Edge-AI. The second part starts with a study on the synchronization phenomena within a network of chaotic spiking oscillators that builds up to encoding of epileptic seizures by using partial synchronization in a neuromorphic processor. These two parts culminate into an SNN implemented on FPGA that has substantial reductions in logic size requirements via synchronization. Therefore, this part contributes to the capability and efficiency of Edge-AI.

In Part I, a case study of the environmental issues of livestock farming is described followed by the introduction of PLF and a short literature overview of existing ML implementations that classify livestock behavior. Subsequently, a DT implementation on the Lattice ICE40UP5K FPGA is proposed that uses the horizontal-longitudinal acceleration angle with the two-argument arctangent as the principal feature. The implementation of this DT shows a 87% accuracy on classifying four essential cow behaviors at a power consumption 216 μ W and energy per inference of 557 μ J. This study is followed up by a proposed quantization scheme for RNNs and a Neural Architecture Search (NAS) to find an optimized RNN for cow behavior monitoring. This leads to a 95% accurate integer-only RNN, dubbed "TinyCowNet", that comprises of 4 layers, a sequence length of 35 and 52 neurons, comprising 1330 weights that take up only 11.74 kB of memory. This application-focused codesign approach culminates into an integer-only resource-minimized RNN hardware architecture for IoT devices with low-frequency sensors, suitable for low-power Edge-AI applications. This hardware architecture proposes time-and-layer multiplexing, folding all layers and sequence timesteps of the RNN into a single processing unit, minimizing resource usage. Subsequently, TinyCowNet implemented on the ICE40UP5K FPGA shows resource use of only 716 registers and 3132 Look-Up Tables (LUT). At a clock frequency of 146 kHz up to 12 MHz this implementation measures a power consumption of 360 μ W to 2.3 mW and an energy per inference of 516 to 44.3 μ J, respectively.

In Part II, a study on the synchronization phenomena of a ring network of chaotic spiking oscillators is described and a physical reservoir for non-boolean computation is proposed. This is followed up with a proof-of-concept of utilizing partial synchronization in SNNs to encode epileptic seizures on a mixed-signal neuromorphic processor, the DYnamic neuromorphic Asynchronous Processor (DYNAP-SE2), for long-term always-on out of the hospital monitoring. The codesigned SNN on the DYNAP-SE2 processor is represented by two layers, namely, a non-linear spike-based filter and a subsequent non-local non-global layer that encodes inherent correlation patterns between input channels. Applied to the SWEC-ETH dataset that contains multi-electrode intracranial Electroencephalogram (iEEG) data with time-varying seizures, this SNN encodes seizures by showing emerging synchronization patterns during seizure periods. Finally, this part culminates into a codesigned hardware architecture, dubbed "NimbleSNN", that aims for speed under the constraints of the ICE40UP5K FPGA on classifying handwritten digits of the MNIST dataset. This hardware architecture, comprising of 5 parallel spike engines, is optimized by means of synchronization, i.e., an algorithm that minimizes spiking at equivalent processing

cycles by allocating the weight of synchronized Integrate-and-Fire (IF) neurons in separate memory blocks and rows. Due to this neuron weight allocation, the spike buffer assigned to each spike engine is reduced to hold only 3 spikes within a single processing cycle. As a result, the number of LUTs and registers for implementation are decreased by 45.8%. Using only 4639 LEs and a power consumption of 20.4 mW at a 40 MHz clock frequency, NimbleSNN shows an inference latency of 65.8 μ s, leading to an energy per inference of 1.34 μ J and an effective efficiency of 47.7 GSOPs/s/W.

This thesis describes a coherent exploration of multiple disciplines and applications to codesign neural networks with the goal of enhancing Edge-AI, leading to hardware architectures and methods to perform computation by means of synchronization and chaos. Starting from an application-oriented study within PLF, this thesis culminates in a sensor-optimized RNN hardware architecture and quantization scheme that allows for the first time RNNs to be implemented on extremely resource-constrained tailored hardware platforms. Furthermore, the implementation of TinyCowNet on the ICE40UP5K FPGA shows the first livestock behavior monitoring Edge-AI implementation that could enable years of monitoring without any battery replacements. Followed by a study on synchronization phenomena with dual-transistor chaotic spiking oscillators, this thesis shows a first proof of concept on utilizing partial synchronization to encode epileptic seizures on a mixed-signal neuromorphic processor. The various topics of this thesis synergize with NimbleSNN that is optimized by means of synchronization, leading to an highly efficient multi-layer SNN accelerator with 448 neurons on an extremely-small FPGA. All hardware architectures devised for FPGA implementation described in this thesis are openly available. In conclusion, this thesis provides a hardware design blueprint of neural networks for a wide range of applications in Edge-AI, including a first step towards comprehensive use of emerging physical phenomena within networks of spiking non-linear dynamical systems and sensor-optimized hardware architectures for neural network implementations on FPGA.

Contents

Acknowledgment	v
Abstract	vi
1 Introduction	1
1.1 Foreword	1
1.2 Background: From IoT and Sensors to AIoT and Edge-AI	2
1.3 The Issue with Edge-AI Accelerators	3
1.4 The Promise of Low-Power Edge-AI at Sensor Nodes	6
1.5 The Unsustainable Pace of Conventional Machine Learning	8
1.5.1 The Transformer and LLMs	9
1.5.2 State-of-the-art AI is Growing 7× Faster Than Moore’s Law	11
1.6 The Promise of Neuromorphic Computing	11
1.6.1 Natural Intelligence in Organisms and their Efficiency	11
1.6.2 Neuromorphic Computing as an Alternative	14
1.7 Thesis Rationale and Outline	19
I From Precision Livestock Farming to Edge-AI via Minimized ML Algorithms and Low-Power Hardware Architectures	25
2 Edge-AI for Precision Livestock Farming: A Study on Cow Behavior Estimation	26
2.1 Environmental Impact of Livestock	26
2.1.1 Effects of Global Warming	27
2.1.2 Mitigating the Impact of Livestock	29
2.1.3 Precision Livestock Farming	30
2.2 PLF with ML: a Short Review	31
2.2.1 Anomaly Detection: Disease and Fertility of Cattle	31
2.2.2 Edge-Based Behavior Detection Systems	33
2.3 Approaches in this Part	35
2.3.1 Bottom-Up: Decision Trees	35
2.3.2 Top-Down: Searching for a Weight-Minimized Recurrent Neu- ral Network and Integer Quantization	36
2.3.3 A Resource-Minimized RNN Hardware Architecture for Low- Frequency Sensors	36
2.4 Japanese Black Beef Cow Behavior Dataset	37

3	A 216 μW, 87% Accurate Cow Behavior Classifying Decision Tree on FPGA With Interpolated Arctan2	40
3.1	ML Algorithm and Feature Set	
	Considerations for Hardware	40
3.2	FPGA Measurement Setup	43
3.3	Proposed Decision Tree Architecture	43
3.4	Hardware-Software Comparison	45
4	TinyCowNet: Memory- and Power-Minimized RNNs Implementable on Tiny Edge Devices for Lifelong Cow Behavior Distribution Estimation	50
4.1	Windowing with Classification Algorithms	51
4.2	Labels in Classification Algorithms	51
4.3	Proposed Distribution Regression	53
4.4	Recurrent Neural Network Methods	55
	4.4.1 Recurrent Layers	55
4.5	Hardware Considerations	57
4.6	Proposed Cow Behavior Distribution Estimating Neural Network (Tiny-CowNet)	57
	4.6.1 Target networks	60
4.7	Finding the target networks	60
	4.7.1 Random Search	60
	4.7.2 Training conditions	63
	4.7.3 Candidate Evaluation: Accuracy, MACs and Memory	63
4.8	Post-Training Quantization	64
	4.8.1 Full-Integer Quantization for RNNs	65
	4.8.2 Optimal quantization with grid search	66
	4.8.3 Memory Estimation for the quantized Models	68
4.9	Results of TinyCowNet with Random Search and Quantization	73
	4.9.1 Weight, MAC and Timesteps Analysis	73
	4.9.2 32-bit Models on low-power MCU and FPGA	73
	4.9.3 Minimum memory and quantized arithmetic for ASIC Implementation	74
4.10	Discussion and Future Work	77
5	An Integer-Only Resource-Minimized RNN on FPGA for Low-Frequency Sensors in Edge-AI	79
5.1	Methods	80
5.2	Resource-Minimized Hardware Architecture	81
5.3	A Single Neuron Processing Unit	82
	5.3.1 Neuron Controller	85
	5.3.2 Weight Controller and Structure	87
	5.3.3 Vector Controller	87
	5.3.4 Processing Engine	87
	5.3.5 Hyperbolic Tangent	88
5.4	Measurement Setup and Implementations	88
5.5	Proof of Concept: Cow Behavior Estimation With a 3D Accelerometer	90
5.6	Implementation Results	90
5.7	Discussion	93

II From Neuromorphic Computing to Edge-AI via Synchronization, Chaos and Spiking Neural Networks 95

6	An Experimental Study of Synchronization Phenomena in Chaotic Spiking Oscillators Realized Experimentally Towards Physical Reservoirs	96
6.1	Spiking oscillator	99
6.1.1	Circuit topology	99
6.1.2	Simplified equation model	101
6.1.3	Simulations	104
6.2	Experimental setup	105
6.2.1	Chaotic network board design	105
6.2.2	Control and data acquisition infrastructure	107
6.2.3	Representative recordings	109
6.3	Dynamics and synchronization phenomena	111
6.3.1	Temporal irregularity and synchronization	111
6.3.2	Predictability and permutation entropy	113
6.3.3	Spontaneous pattern formation and effect of long-distance links	116
6.3.4	Effect of coherent noise	119
6.3.5	Effect of patterned node heterogeneity	122
6.4	Discussion and application to reservoir computing	124
6.4.1	From coupled oscillators to reservoirs	124
6.4.2	Features relevant towards physical reservoirs	125
6.4.3	Hypothetical methods for input/output coupling	126
7	A Seizure-Encoding Spiking Neural Network Using Partial Synchronization on a Mixed-Signal Neuromorphic Processor	128
7.1	Results	135
7.1.1	Asynchronous Delta Modulation (ADM) of iEEG signals . . .	135
7.1.2	The DYNAP-SE2 SNN implementation: firing-rate filtering and encoding correlation	135
7.1.3	Encoding seizures with partial synchronization using the hardware SNN	137
7.1.4	Detecting seizures using SNN encoded features	140
7.2	Discussion	141
7.3	Conclusions	142
7.4	Methods	144
7.4.1	AFE and encoding setup	144
7.4.2	Delta modulated EEG events	144
7.4.3	On-chip SNN parameters	144
7.4.4	Statistical measures	145
7.4.5	Kernelizing spike-trains to continuous functions	145
7.4.6	Manhattan norm and synchronization matrices	146
7.4.7	SWEC-ETHZ iEEG dataset	146
7.5	Data availability statement	147
7.6	Code availability statement	147
7.7	Supplementary materials	148

8 NimbleSNN: A 1.34 μJ/Image 4639 Logic Element Single-Spike Accelerator on FPGA Using Synchronization	154
8.1 A Review of Time-To-First-Spike Coding With a single spike per neuron	155
8.1.1 Input encoding	155
8.1.2 Forward Propagation	155
8.1.3 Backward Propagation	156
8.1.4 Setting the target/relative firing times	159
8.1.5 Regularization and dropout	161
8.2 Towards a TTFS SNN Accelerator on FPGA Using Synchronization	162
8.2.1 Introduction	162
8.2.2 A Reduced Binary SNN with TTFS coding	163
8.2.3 Hardware Implementation and Architecture	165
8.2.4 Experimental Results	170
8.2.5 Conclusion	174
9 Conclusion, Discussion and Future Work	177
9.1 Chapters at a glance	178
9.2 Claims	178
9.3 Discussion and Future Work	182
9.3.1 Physics Inspired Spiking Neural Network (PiSNN)	182
9.3.2 Recurrent Spiking Neural Networks (RSNN) for Resource-Constrained Hardware including FPGA	183
A Academic Publications	185
A.1 First Author	185
A.1.1 International Journal Peer-Reviewed	185
A.1.2 International Conference Peer-Reviewed	185
A.1.3 Domestic Non-Peer Reviewed	186
A.2 Co-Author	186
A.2.1 International Journal Peer-Reviewed	186
A.2.2 International Conference Peer-Reviewed	186
A.2.3 Domestic Non-Peer Reviewed	186
B Published Datasets and Source Codes	187
B.1 Datasets	187
B.2 Source Codes (Verilog + Python)	187
C Hardware Platforms	188
C.1 DYnamic neuromorphic Asynchronous Processor (DYNAP-SE2)	188
C.1.1 Neuron and synapse models	188
C.1.2 Analog neurons with digital routing	189
C.1.3 Experimental Setup	189
C.2 Field Programmable Gate Array (FPGA), the Lattice ICE40UP5K .	191
C.2.1 Experimental Setup	191
D Adaptive Threshold for Asynchronous Delta Modulation: Hardware- Based Simulations and Seizure Encoding	192
D.1 Differential Pair Integrator and Source-Follower Circuits Simulation for Adaptive Thresholding	192

D.2 Using AdaADM to Convert iEEG Data to Spikes	194
---	-----

List of Figures

1.1	Power consumption over efficiency in terms of operations per second per watt shows the majority of state-of-the-art Edge-AI works to focus on speed and efficiency for acceleration. However, this approach is incompatible with sensor nodes that have low power and energy budgets.	4
1.2	Conventional cloud- and edge-implementations of AI focus on applications such as object recognition from images to text generation using Large Language models that focus on speed. Both directions have several merits and demerits. However, both approaches are unsuitable for implementation at sensor nodes that commonly have extremely low power and energy budgets.	4
1.3	Edge-AI at sensor nodes require sensor-optimized AI implementations that balance the limited number of resources, speed and power consumption due to being always on. Different from the applications in the high-speed Edge-AI, these locate themselves at the far edge, involving long-term monitoring of epileptic seizures of patients, anomaly detection within machinery such as mining equipment and manufacturing, and behavior estimation of livestock including cows.	7
1.4	Growth of compute and time required for state-of-the-art AI models to be trained in Peta Floating Point Operation (FLOP) days, taken with permission from Ref. [107]. (CC BY-NC 4.0 DEED licence)	10
1.5	Extrapolated economical cost of compute for training state-of-the-art AI models in US dollars, taken with permission from Ref. [107]. (CC BY-NC 4.0 DEED licence)	10
1.6	a) A Chimpanzee family in the wild in Uganda (public-domain). b) A New Zealand honey bee foraging clover (CC BY-SA 2.0 DEED, taken from Ref. [124]).	13
1.7	The realization of the pace of state-of-the-art AI, unsustainable in terms of cost and lacking in terms of intelligence, combined with inspiration from general intelligence in animals has led me to pursue neuromorphic computing as one of the main topics in this thesis.	15
1.8	a) A simulation of a Leaky Integrate and Fire (LIF) neuron, a non-linear dynamical system, with N spike inputs over a short period of time, leading to output spikes when membrane potential crosses a threshold. b) A diagram of an artificial neuron, a representation of the matrix-based approach of conventional AI using multiplications of weights and activation functions (Rectified Linear Unit, ReLU) for non-linearity.	15

1.9	An elementary implementation of a Leaky Integrate and Fire (LIF) neuron using current sources with switches that close whenever a spike arrives. These current sources charge a capacitor that discharges when they close. A spike is output whenever voltage on the capacitor exceeds a pre-set threshold.	16
1.10	Overview of the energy per spike operation in Joules for the most recent neuromorphic hardware platforms including digital and mixed-signal based implementations (Refs. [49,150,152,161–171],). The hardware platform utilized in this Thesis, Ref. [152], in Chapter 7 is also shown here. Size of dot indicates technology node, ranging from 180 nm [152,163,164] to 14 nm [150]	17
1.11	Structural overview of this thesis. Color graded boxes indicate what fields the various topics contained in Chapters are related to, colored borders indicate the hardware platform employed.	24
2.1	Algae blooms lead to large dead zones underneath the surface in the Gulf of Mexico.	28
2.2	Coral reef in Okinawa, Japan, one of the marine ecosystems that will suffer severely and possibly completely disappear because of global warming (CC license, from Okinawa Institute of Science and Technology (OIST) [188]).	29
2.3	Meat consumption per capita for three regions on Earth and the nations including Japan, USA and Australia as well as nations in the European Union.	30
2.4	Large herds of cattle are harder to monitor for individual behavior and diseases (CC license, [204])	32
2.5	Diagram describing two identified types of Precision Livestock Farming systems with Machine Learning in literature.	33
2.6	Method of transforming camera footage with raw accelerometer data to labeled data used for ML algorithms.	37
2.7	Screen captures of recordings that were used to label cow behavior to tri-axial accelerometer data gathered from accelerometers attached to the neck of each of six cows.	38
2.8	Cow behavior dataset with the amount of samples for each human-labeled behavior.	38
3.1	2 argument arctangent approximation using binary search look up table and assignment of constants to regions with small portion of the overall dataset.	43
3.2	Setup used to send accelerometer data and receive classified behavior from Matlab on PC to the implementation on FPGA	44
3.3	Top level view of the in FPGA implemented hardware architecture for each feature set, addition-accumulate block usage is indicated with colors and bit size with brackets, the blue brackets represent a shift operation and reduction to 8-bit where index 0 indicates the MSB.	45
3.4	The macro average F1 score of five different feature sets applied to four cow behaviors on software and 8-bit fixed point hardware.	46

3.5	Scatterplots of quantized FS1 where decision rules occur in the DT for the processed cow behavior dataset containing 4359 samples, (a) and (b) show a less separable scatterplot than (c) which is also part of FS2.	49
4.1	Label processing for pure data labels and raw data labels having a window size of 64. The colored stripes denote the behaviors of Resting (RES), Rumination (RUM), Moving (MOV) and Eating (EAT) with blue, yellow, red and green respectively.	52
4.2	Method of labeling raw data into distribution vectors. Instead of a single label as in classification tasks, a distribution vector is used for regression. The l_s is the sample window size, normalizing the vector.	53
4.3	Architecture overview of (a) Simple Recurrent Layer, sRL, (b) Long-Short Term Memory, LSTM, and (c) Gated Recurrent Unit, GRU.	54
4.4	Layer width n for four different layer counts N plotted vs. total number of weights using simple Recurrent Layers. The gray lines indicate that a doubling of the layer width n for a quadrupling of the numbers of layers N is possible at the same weight cost.	58
4.5	Cow behavior distribution estimating neural network, rolled out over time from left to right. The yellow blocks represent the recurrent layers with outputs, hidden state vector $\mathbf{h}$ and cell state vector $\mathbf{c}$, and contain the trainable weights $\{\mathbf{U}, \mathbf{W}, \mathbf{b}\}$. The green block represents the linear output layer with trainable weights $\{\mathbf{W}_L, \mathbf{b}_L\}$. Each state vector is indicated with entries (p, q) where p is the current timestep and q is the respective layer. The hyperparameters included in the search space are architecture k , width n and layer count N and timesteps T_s . Finally, $\hat{\mathbf{y}}$ denotes the regressed distribution vector containing cow behavior prevalence.	59
4.6	Diagram showing a single iteration i of a random search applied to four hyperparameters of TinyCowNet (Fig. 4.5). These hyperparameters represent layer width n , layer count N , layer type k , which have influence over the complexity of the network and timesteps T_s . For each iteration, a set of hyper parameters θ in the uniformly distributed range R is randomly assigned to TinyCowNet, in this case M_i . Hereafter, a three-fold stratified split training-validation dataset (Fig. 4.7) is used to train the model with Mean Squared Error (MSE) as loss function for 150 epochs. Finally the resulting three model accuracies and weights are stored for later resource-optimized candidate selection.	61
4.7	Stratified splitting of the imbalanced cow behavior dataset.	62
4.8	Example of full-integer quantization for RNNs with a weight matrix and bias matrix for which the range is defined by the activation function at 8-bit bit-width.	66

4.9	Overview of the simple recurrent layer with integer quantization. The red and green blocks represent the input/output and stored weights, respectively. The light blue stripes indicate the bit-width of the buses, output bit-width h_q , weight bit-width w_q and bias bit-width b_q . The dark blue multipliers are equivalent to the matrix multiplication as described in Eq. (4.41). The hyperbolic tangent function is implemented as a look-up where $\mathbf{h}'_t$ are the indices. The orange blocks are element-wise operation blocks where the output of the addition operation is rescaled to h_q bits.	67
4.10	Scatterplots of the 445 TinyCowNets (parametrized RNNs) found using random search. The color denotes layer type k . Furthermore, the top, middle, and bottom row show the mean absolute error vs. the number of weights P_k , the MAC operation count O_k , and timesteps T_s	69
4.11	Scatterplots of the 445 TinyCowNets (parametrized RNNs) found using random search. The colors denote number of layers N . Furthermore, the top, middle, and bottom row show the mean absolute error vs. the number of weights P_k , the MAC operation count O_k , and timesteps T_s	70
4.12	Scatterplot of all TinyCowNets from the random search with an MAE below 0.07. The three models plotted with a cross indicate the candidates subject to full-integer quantization and ASIC resource estimation.	75
4.13	Absolute distribution error vs weight bit-width and output bit-width, which is the error between the full-integer and floating point counterparts for the bsRL, bGRU and sGRU models, described in Table 4.9.	75
4.14	The Receiver Operating Characteristic (ROC) curves of the three models; best sRL, best GRU and smallest GRU from top to bottom.	76
5.1	Mass Parallelized (MP), Time-Multiplexed (TM), and Time- and Layer-Multiplexed (TLM) RNN architectures. MP signifies the most rapid architecture with the minimum number of clock cycles required for inference, allocating an NPU to each layer and timestep. TLM employs a single NPU, minimizing resources while necessitating the greatest number of clock cycles for processing.	83
5.2	Neuron Processing Unit (NPU), containing five modules, Processing Engine, Activation Function and Weight, Vector, and Neuron Controllers. Expanded modules are shown in Fig 5.3.	83
5.3	Architectural overview of the Neuron Processing Unit (NPU) on FPGA at 8-bit bitwidth, implementing model a (parameter values described in Table 5.1). The structure of the weight memory (ROM IP) and details of the Processing Engine are shown in Fig. 5.5 and 5.6, respectively. Clock, reset and enable signals are omitted.	84
5.4	State Diagram of the Neuron Controller for Recurrent Neural Network inference on model a, described in Table 5.1.	85
5.5	Memory composition of the weight and parameters of the RNN, layer-by-layer (RNN _{0:3}). One word in the ROM comprises a single row of the input weight and hidden weight matrices, and the bias, leading to $(2N + 4)q_{\text{bits}}$ bits per word. The final address of a layer, address $(N + 1)$, contains the quantization parameters, i.e., M_{int} , M_{shift} and zeropoints Z_i (eq. 5.4).	86

5.6	Expanded diagram of the Processing Engine and three proposed operations at 8-bit bitwidth, SUB_MAC (OP1), MUL_SAC (OP2), and ADD_ACC (OP3).	89
5.7	a) Measured power consumption over the internal clock frequency. b) Energy per inference over internal clock frequency. c) Maximum sensor frequency for real-time RNN operation, a function of the clock frequency and number of clock cycles (eq. 5.18-5.21).	89
5.8	Single behavior and mixed behavior classification scenarios of cow behavior. $\mathbf{x}_t$, $\mathbf{y}$ and $\hat{\mathbf{y}}$ describe the corresponding accelerometer data, ground truth distribution coefficients and estimated coefficients by the implemented RNN on FPGA respectively.	91
6.1	Spiking oscillator circuit, shown in the form of a pair (oscillators A and B) coupled via a resistor of value R_C . a) Discrete elemental diagram, b) Simplified equivalent diagram underlying the ODE model in Eq. (1), also including two coherent noise current sources i_N . Transistor orientation according to Ref. [334].	100
6.2	Results from integrating the ODE model in Eq. (1). a) and b) Representative time series for v and v_{CE} , and c) Phase portrait of the oscillator dynamics across them, d) Corresponding temporal distribution of spikes over a longer time-span, e) Synchronization level N as a function of the coupling resistance R_C between two oscillators, f) Synchronization matrix for a ring of 10 oscillators, g) Synchronization as a function of the intensity σ of coherent noise applied over an oscillator pair.	102
6.3	Realization of the experimental network. a) Circuit diagrams of oscillator instantiated at each node, b) Topology of the ring network (red) with long-distance links (blue) and data acquisition infrastructure (FE: Front-end, IF: Interface), c) Photograph of the network circuit board. Transistor orientation according to Ref. [323].	106
6.4	Data acquisition infrastructure. a) Architecture of the front-end (FE). ADC: Analog-to-digital converter, DAC: Digital-to-analog converter, CPLD: Complex programmable logic device, FT: Fiber transceiver). b) Architecture of the interface (IF). FPGA: Field-programmable gate array, SPI: serial peripheral interface, LA: logic analyzer, PC: personal computer. c) Photograph of the front-end (FE) circuit board. d) Photograph of the interface (IF) circuit board.	108
6.5	Oscilloscope waveform screenshots. a) Sweep of the supply voltage V_S for a single oscillator in isolation, b) Sweep of the coupling voltage V_C for two arbitrarily-chosen adjacent nodes, given $V_S = 4$ V. Scale: a) 500 mV/div, 20 μ s/div, b) 1 V/div, 5 μ s/div	110
6.6	Distributions of the inter-spike interval Δt as a function of a) supply voltage V_S and b) coupling voltage V_C	112
6.7	Synchronization and dynamical properties under a two-dimensional sweep of the supply and coupling voltages V_S and V_C . a) Average synchronization $\langle N \rangle_{ij}$, b) Average prediction error $\langle \epsilon_{12} \rangle_i$, c) Average prediction error difference with a delay of 1, $\langle \Delta \epsilon \rangle_i = \langle \epsilon_{23} \rangle_i - \langle \epsilon_{12} \rangle_i$, d) Average prediction error difference with a delay of 2, $\langle \Delta \epsilon \rangle_i = \langle \epsilon_{34} \rangle_i - \langle \epsilon_{23} \rangle_i$, e) Average permutation entropy $\langle h_5 \rangle_i$	114

6.8	Representative synchronization matrices formed by spontaneous activity, across three experimental boards α , β and γ for different combinations of the supply and coupling voltages $\{V_S, V_C\}$, namely a) $\{3.8, 1.7\}$ V, b) $\{4.8, 1.9\}$ V, c) $\{5.8, 2.1\}$ V, d) $\{3.7, 1.6\}$ V, e) $\{4.1, 1.5\}$ V and f) $\{4.7, 1.5\}$ V.	117
6.9	Effect of inserting additional long-distance links. a) and b) Synchronization increase with 4 additional links and baseline as a function of the supply and coupling voltages $\{V_S, V_C\}$, c-f) Bar charts of the average synchronization across the three experimental boards α , β and γ , for different parameter settings $\{V_S, V_C\}$, namely, $\{3.75, 1.59\}$ for 1, $\{4.25, 1.66\}$ for 2, $\{4.00, 1.73\}$ for 3 and $\{6, 1.87\}$ for 4, g-i) Representative matrices showing the synchronization increase as a function of adding 1, 3 or 4 links (black dots indicate locations), for $V_S = 6$ V, $V_C = 1.9$ V.	118
6.10	Effect of injecting coherent noise. a) Baseline average synchronization parametric maps as a function of the supply and coupling voltages $\{V_S, V_C\}$, b) Synchronization maps in the presence of noise having an amplitude of $V_N = 10 V_{pp}$ and a bandwidth of $B = 2$ MHz, c) Corresponding difference maps. d) Parametric maps for the average synchronization increase as a function of amplitude V_N and bandwidth B , for $V_S = 4$ V, $V_C = 1.6$ V, e) Corresponding chart as a function of V_N , f) Representative matrices for the difference in synchronization for different parameter settings $\{V_S, V_C\}$, namely, $\{4.91, 1.54\}$ for 1, $\{3.71, 1.68\}$ for 2, $\{5.64, 1.92\}$ for 3 and $\{4.43, 1.96\}$ for 4.	120
6.11	Effect of patterned node heterogeneity according to split in two populations, A and B. a) Average synchronization and b) Permutation entropy as function of the coupling voltage V_C . c) Matrices depicting the route to synchronization as a function of V_C for the 5/5 split. . .	123
6.12	Possible architectures for constructing a physical reservoir. a) Assuming the synchronization values are output, b) Assuming the activity of select nodes is output.	125
7.1	Graphical abstract of this Chapter; an envisioned real-time seizure monitoring system within a bio-compatible enclosure applied to the skull using flexible EEG electrodes and DYNAP-SE2. On DYNAP-SE2 (assigned neuron per core location color coded) an SNN (middle row) runs that encodes epileptic seizures in real-time by mimicking a silicon brain with a biological brain, namely, showing synchronous neuron clusters (bottom) within the chip whenever a seizure occurs.	129
7.2	Overview of existing, presented, and proposed seizure detection systems; a) SOA (state of art) approaches for seizure detection and diagnosis; seizure recording with a seizure diary (right side), seizure detection using Analog-to-Digital Converters (ADC) and digital processing (left side); b) Presented system; seizure encoding on DYNAP-SE2 with analog iEEG signals using on-chip Analog Front-End (AFE) and a Spiking Neural Network (SNN); c) Envisioned system; real-time seizure detection pipeline with DYNAP-SE2 and a seizure recorder within a wearable device.	131

7.3	a) Photographs of the setup, an FPGA board, DYNAP-SE2 board with DYNAP-SE2 chip on the underside, Digital-to-Analog (DAC) converter board with a Teensy microcontroller for interfacing with the AFE; b) Photograph of the DYNAP-SE2 die (right), and the package located on the underside of the DYNAP-SE2 board (left).	132
7.4	Snippets presenting 320-second long DAC converted analog iEEG signals from two channels and the encoded events from the ADM (‘UP’ and ‘DOWN’) on DYNAP-SE2. The event data rate (above and below the DAC-generated signal) is encoded in color with the seizure period denoted by the grey background; a) Oscilloscope traces depicting the incoming DAC-generated signal (trace 4, in pink) and the AFE amplified signal (trace 3, in purple); b) Analog waveform of one EEG channel and ADM events (with inset above showing a close-up of 16 seconds); c) Analog waveform of another EEG channel and ADM events.	134
7.5	Structure of the on-chip SNN; a translation layer containing 3 types of neurons, Feedforward Inhibitory (FF-I), Recurrent-Excitatory (R-E), and Recurrent Inhibitory (R-I) neurons. This layer performs firing-rate-based filtering, followed by a Non-Local Non-Global (NLNG) layer. Synapse types slow excitatory (NMDA like) and slow inhibitory (GABA-B like) are color-coded	136
7.6	a) Firing rate input-output curve as a result of Poisson-generated input spikes to translation layer; b), c) Raster plots of the translation layer for an input firing rate of 200 Hz and 400 Hz respectively; d) In blue, the mean Pearson correlation coefficient of 42 input channels over the standard deviation of jittering in spike time at input firing rates of 50 Hz. In red, the output firing rate of the non-local non-global encoder as a result of the jittered input spike trains; e), f) Raster plots of input (black) and output of the non-local non-global encoder (blue) at standard deviations of $\sigma = 0.005$ s and $\sigma = 0.015$ s.	137
7.7	a) 42 channels of input EEG signals from patient 1, seizure 1; b) Output rasterplots of the on-DYNAP-SE2 implemented network. Top row to bottom: ADM-encoded input of a), Feedforward (FF) inhibitory neurons’ outputs, Inhibitory (R-I) and Excitatory (R-E) neurons’ outputs of the E-I pairs, and outputs of the Non-Local-Non-Global (NLNG) neurons. The firing rate is color-encoded per one-second time-segment;	138
7.8	Normalized synchronization matrices between channel and neuron pairs of AFE input and NLNG output, respectively. From left to right: the pre-ictal, ictal, and post-ictal periods. The synchronization matrices during the ictal period (2nd and 3rd column) are created from the first half and second half of the seizure in time, respectively.	139
7.9	a) The mean synchronization over all pairs of the NLNG layer cumulatively summed over time, colored areas under the curve indicate a hand-selected period of time used to construct synchronization matrices in b); b) Synchronization matrices taken during periods of maximal synchronization.	140

7.10	Synchronization matrices of Pat. 1, 2 and 3. The columns from left to right show synchronization during the pre-ictal, ictal and post-ictal periods respectively.	148
7.11	Synchronization matrices of of Pat. 3, 4 and 5. The columns from left to right show synchronization during the pre-ictal, ictal and post-ictal periods respectively.	149
7.12	Non-local non-global neuron spike train synchronization matrices of patient 2 seizure 1 based on three distinct input channel ordering methods. The first, second, and third rows represent synchronization based on input with the original dataset order, maximum correlation of the original data, and randomized order respectively.	150
7.13	Non-local non-global neuron spike train synchronization matrices of patient 2 seizure 2 based on three distinct input channel ordering methods. The first, second, and third rows represent synchronization based on input with the original dataset order, maximum correlation of the original data, and randomized order respectively.	151
7.14	The synchronization across all non-local non-global layer neurons over time for all patients and seizures. Grey line indicates seizure start. .	152
8.1	a) Sorted synchronization matrix of the hidden layer neurons, diagonal omitted and normalized over number of inferences. b) Probability of neurons firing throughout all inferences. c) Change in output neuron threshold by adding the constant effect of synchronous neurons. [-20pt]	164
8.2	Color coded weight matrix of the synapses between input and hidden layer (left). Resulting mask on example image determined from inputs with >90% excitatory weights assigned across the hidden layer (right). [-20pt]	164
8.3	a) Proposed hardware architecture diagram. b) Diagram of one of four Hidden Engines (HE) that process 448 hidden neurons of the reduced binary SNN, red indicating threshold and a neuron potential exceeding this value (56), leading to a spike. [-20pt]	166
8.4	Example diagram describing conversion of MNIST pixel intensity to TTFS events using Address Event Representation (AER). Time t_o represents the final timestep and integers below address event sets indicate a one-by-one input mechanism. [-20pt]	166
8.5	a) Example of a worst case scenario during inference; each hidden engine produces more than 3 spikes, leading to clogging (lost spikes) and stalling (increase in latency). b,c) Example time diagram of the proposed architecture without neuron scheduling (left) and with neuron scheduling (right). Red and green vertical lines indicate top-to-bottom direction transmission of spikes and hidden layer processing completion respectively. The orange vertical line represents a global signal that interrupts the entire spike core whenever an output neuron spikes. Abbreviations, I-ER: Input Event Router, HE: Hidden Engine, HL: Hidden Layer, H-SR: Hidden Spike Router, OE: Output Engine. [-20pt]	168
8.6	Energy per inference and efficiency as as a function of frequency, the green and red areas represent valid SNN outputs and invalid due to timing errors, respectively. [-20pt]	169

8.7	Iterations 1, 10 and 30 of the neuron assignment algorithm.	176
C.1	a) Photographs of the setup, an FPGA board, DYNAP-SE2 board with DYNAP-SE2 chip on the underside, Digital-to-Analog (DAC) converter board with a Teensy microcontroller for interfacing with the AFE; b) Photograph of the DYNAP-SE2 die (right), and the package located on the underside of the DYNAP-SE2 board (left). .	190
C.2	a Internal components of the Analog Front End (AFE), a Low-Noise Amplifier (LNA) with bandpass filter followed by the Asynchronous Delta Modulator (ADM); b Block diagram of the boards and DYNAP-SE2: the FPGA sends input spikes to the spike cores and configures the AFE and SNN cores; a DAC board interfaces directly with the AFE; c Neuron schematic. Input events are translated to synaptic currents. The synaptic currents are routed to one of four synapse types, each combined with Differential Pair Integrators, and forwarded to an Adaptive Exponential Leaky Integrate & Fire soma.	190
C.3	FPGA measurement setup using Analog Discovery 2 and Lattice ICE40UP5K-B-EVN breakout board.	191
D.1	a) Block diagram of the non-linear adaptive threshold calibration circuits used in simulation. b) Source Follower Integrator circuit. c) Differential Pair Integrator with non-relevant biasing voltages omitted. .	193
D.2	Adaptive ADM threshold system operating on 420 seconds of a single channel of iEEG data from the ETH-SWEC dataset [433] and software-simulated Asynchronous Delta Modulator (ADM) output. .	195

List of Tables

2.1	Details of a literature survey of Animal Behavior Classification with ML intended to be implemented on hardware itself (Edge-AI) using tri-axial accelerometers.	35
2.2	The exact description of each of the labeled data and their respective percentage as part of the overall data set.	37
3.1	3-axis accelerometer based feature sets	42
3.2	Approximation of the two argument arctangent considering percentage of dataset and machine learning application	42
3.3	Resource allocation for FS2 on Lattice ICE40UP5K FPGA	44
3.4	A resource comparison for the hardware implementations on FPGA for 5 different feature sets	47
3.5	A performance comparison for the hardware implementations on FPGA and software applying 5 different feature sets.	47
3.6	Comparison of FS ₂ on FPGA and state of the art	48
4.1	Weight dimensions and Multiply-Accumulate (MAC) operation of the four different neural network layers. Variables i , n and m denote the input size, recurrent layer width and output vector size.	56
4.2	Hyperparameters defined for the search space of the recurrent layers, i.e., the layer width n , the total number of layers N , the timesteps T_s and the cell architecture type k	57
4.3	Four different low-power programmable devices to be considered as targets for TinyCowNet implementation. The Lattice UL640 and Microsemi AGLN250 FPGA represent ultra low power devices only suitable for the smallest networks. The STMicroelectronics MCU and Lattice UP5K represent low-power FPGAs which are also suitable for medium sized models. The cost in dollars of all these devices were retrieved per single piece from Mouser Electronics.	61
4.4	Quantization parameters defining a quantization search space to be used on three TinyCowNets selected from the random search. While the weight bit-width w_q and h_q are selected arbitrarily and independent, b_q is dependent on the former because of the relationship with matrix multiplications in the recurrent layers.	67
4.5	Average Mean Absolute Error (MAE) of the best-performing 10% of found models per decade vs. the model weights P_k and MACs per inference O_k (10^1 - 10^8) with random search. This is considered with the hyperparameters for recurrent layer type k and layer count N as well as the result of the entire random search results regardless of hyperparameter (All (average)).	71

4.6	Maximum number of 32-bit weights and MACs per timestep when assuming a single multiplier unit on three low power FPGAs and one MCU as described in Table 4.3. In light of these resources, the best MAE on the validation dataset of an implementable TinyCowNet (described in Table 4.7) from the random search results is indicated.	72
4.7	Number of weights, total MACs per inference, timesteps, recurrent layer count and recurrent layer type of the best implementable 32-bit floating-point TinyCowNets for each of the four devices considered (Table 4.3, 4.6).	72
4.8	The independent accuracy for each of four cow behaviors, i.e., resting (RES), rumination (RUM), moving (MOV) and eating (EAT) on the test dataset is described as determined by optimal ROC threshold of the best implementable 32-bit floating-point Tiny- CowNets for each of the four devices considered.	72
4.9	The number of weights, MACs, timesteps and layers of three selected TinyCowNets that are smallest or have best performance for a validation dataset accuracy above 93%.	74
4.10	The accuracy per behavior of three selected TinyCowNets that are smallest or have best performance for a validation dataset accuracy above 93%.	75
4.11	Characteristics of the selected best-performing quantized models, i.e., memory in kB and accuracy vs. weight bitwidth w_q and output bitwidth h_q .	77
4.12	Matthews Correlation Coefficient (MCC) and Area Under Curve (AUC) per behavior from the ROC curves of Fig. 4.14	77
5.1	Parameters of models used for power measurements with internal clock frequency in Section 5.4 (a,b,c), and cow behavior proof of concept in Section 5.5 (c).	82
5.2	Number of 4-bit Look up tables (LUT), 1-bit registers and 4-kbit Embedded Block Rams (EBR) and % of total used by model a, b and c on the Lattice ICE40UP5K FPGA.	88
5.3	Comparison of animal behavior estimation studies implementing algorithms in hardware, including power consumption measurements or estimations.	91
7.1	Seizure classification results and Engel scores with a linear SVM that uses the Analog Front End (AFE) and encoded features of the Non-Local Non-Global (NLNG) layer as input.	141
7.2	Measured change in voltage, and rise and decay times of the membrane potential as a result of a single excitatory spike input per neuron type.	145
7.3	The bias generator currents on DYNAP-SE2 for setting network weights and synaptic compartment time constants (excitatory (NMDA like) and inhibitory (GABA-B like)).	145
7.4	Patient characteristics. TLE Temporal Lobe Epilepsy, PLE Parietal Lobe Epilepsy.	146
8.1	Resource utilization of our proposed NimbleSNN on the ICE40UP5K FPGA.	169

8.2	Scaling factors k of technology taken from [463] with respect to 40 nm for ICE40UP5k. The respective technology has to be multiplied by k to reach 40 nm technology.	170
8.3	Comparison with State Of the Art SNN FPGA implementations on MNIST	171
8.4	Comparison with State Of the Art SNN ASICs on MNIST.	172

Chapter 1

Introduction

1.1 Foreword

The progress of humanity is accelerating at an exponential rate. As the offspring of millions of years of evolution, humans began to first use tools around 3.3 million years ago [1], leading to the first controlled use of fire one million years ago [2]. Much closer to the present day, the adoption of wheat and barley in Mesopotamia 10,000 years ago, and that of rice in China around 8,000 years ago, followed by the domestication of animals, led to the first agricultural revolution. This revolution radically changed the social and economic structure of early human societies [3, 4]. From these advancements, along with progress in writing and religion, the Akkadian Empire emerged in Mesopotamia around 4200 BC. This trend of accelerated human progress continued through various golden ages during the 2nd and 1st millennia BC. Civilizations such as the Babylonians in Mesopotamia, the Romans and Greeks in Europe, and ancient India and China, fostered complex schools of thought, philosophy, and the earliest literate recordings of rationalism, logical deduction, and the scientific method [5–8]. The rediscovery of these advancements during the Renaissance and Enlightenment, culminating in the scientific and industrial revolutions, has taken humanity from war, disease, and feudalism to unimaginable prosperity within a span of 600 years.

This narrative illustrates the accelerating pace of technological and societal complexity. Individuals in ancient civilizations might not have observed technological progress across generations. Similarly, someone in Renaissance Italy in the 15th century AD would have experienced relatively slow changes within their lifetime. However, rapid technological advancements, particularly those driven by breakthroughs in semiconductors and engineering over the last century, are now transforming societies and lifestyles within each decade [9, 10]. This phenomenon is evident in various research areas, where trends shift dramatically within a few years, especially with the rapid rise and impact of Artificial Intelligence (AI) in recent times [11]. Coupled with the increasing complexity of technology, these developments call for a multi-disciplinary research approach covering a wide range of areas.

This thesis explores a philosophy of advancing hardware implementations of AI at the Edge, also known as Edge-AI. These approaches encompass progressing from considerations of sensors and applications and from natural intelligence to neuro-morphic computing. Together, these perspectives lead to various studies proposing advancements in the field of Edge-AI by employing codesign across disciplines including physics, electrical engineering, and computer science.

1.2 Background: From IoT and Sensors to AIoT and Edge-AI

Sensors, devices that capture information about the environment and as a result transmit readable impulses, are in nearly every system in use at the time of writing and a vital part of the Internet of Things (IoT). IoT proposes to connect any device by means of the internet, adding value to society by improving overall productivity of systems, services and people [12]. Over the years, new technologies have emerged that act as enablers of this field including Low-Power Wide Area Transmission (LPWA) technology such as Sigfox, LoRa [13] and ELTRES [14], Micro-Electromechanical Systems (MEMS) that can work almost indefinitely [15] to consumer products that are creating 'smart' environments such as smart homes and smart cities that are controllably by the internet [16, 17]. Next to smart environments, IoT has been applied to increase the efficiency of sectors of industry such as manufacturing, logistics and healthcare and mining [18, 19]. As a result, the number of devices connected to the internet will grow to an estimated 100 billion devices by 2025, generating a global economic impact of 11 trillion USD [20].

The rise of IoT has recently been eclipsed by Artificial Intelligence (AI). Since the advent of Alexnet in 2012, the first large Convolutional Neural Network (CNN) utilizing supervised learning by means of backpropagation to achieve high accuracies on a 20,000 class dataset dubbed Imagenet, AI has gone through an exponential growth of interest. As a result, in academics, the number of papers published in this field between 2015-2019 is nearly equivalent to the period 1974-2014 at 20,000. Furthermore, currently the growth in number of papers per year equates to around 20% compared to the global scientific output growth rate of 3% [21]. Due to models including CNNs and Recurrent Neural Networks (RNN), AI has enable autonomously driving cars, vastly improved the field of natural language processing and robotics. Furthermore, the rise of the Transformer [22] has led to breakthroughs in 3D protein folding estimation, the discovery of new drugs and stable plasma in a nuclear fusion Tokamak to Large Language Models (LLM) that seem limitless in applications [23–26]. The rise of LLMs such as Google's Gemini and OpenAI's Generative Pre-trained Transformer (GPT) combined with a freely accessible terminal, led to a further worldwide explosion of interest into AI from the public. This has among other factors led to record sales of Graphical Processing Units (GPU), and NVIDIA stocks on NASDAQ, the leading manufacturer of GPUs in the world, to triple within one year (2023-2024). From all these applications and improvements to efficiency in industries, the global GDP is expected to increase by 14% until 2030 from the employment of AI [27].

This surge in IoT and AI has led to a synergy, dubbed AIoT, where IoT sensor nodes serve as information gatherers and AI models as the decision maker, drawing similarities with sense organs, the nervous system and the brain. Cloud computing is integral to a considerable share of AIoT implementations [28], providing vast compute resources for implementing AI classifiers that process wirelessly transmitted sensor data including images, sound, temperature, acceleration or humidity [29]. Cloud computing benefits from near-unlimited storage and resources, allowing for scalability and flexibility of developing any solution at an effective price due to economies of scale [30]. However, at the sensor node, this transmission, which may involve the utilization of Low-Power Wide Area (LPWA) networks, incurs considerable power consumption that increases with the number of bytes transmitted and

the frequency of transmission intervals [31]. In fact, this raw data transmission comprises one of the major power consuming elements in an IoT system at the sensor node. Other than transmission cost, IoT systems that employ cloud computing face several challenges including privacy issues due to raw data transmission, for example in medical applications, high end-to-end latency due to back-and-forth transmission from the edge to cloud computing centres, and security [30,32].

Edge-AI proposes to implement AI models near or on the sensor-node itself, leading to a key advantage in power consumption by eliminating the necessity of raw sensor data transmission or transmission to far away cloud servers [33]. Consequently, Edge-AI substantially decreases the size and interval of transmitted data, potentially leading to a 1000-fold increase in battery life when utilizing LPWA technology [34]. Furthermore, Edge-AI offers benefits when deployed near end-users, as it enables processing and transmission of class labels to users in near real-time [35]. As a result, this field has been identified as an important alternative to empower low-latency solutions of LLMs, allowing personal AI assistants in cars and near real-time language prompts on mobile phones [36]. Furthermore, offloading of these LLMs across servers at the edge is another growing research topic that could see these models be used applications such as the metaverse and digital twins [37, 38]. The above considerations and societal impact of Edge-AI has led to the following general objective within this thesis,

Objective: *The advancement of Edge-AI in terms of efficiency, capability and accessibility*

1.3 The Issue with Edge-AI Accelerators

Edge-AI is a large established computing approach in use for time-series classification and computer-vision tasks such as keyword recognition and image-based object classification. This has led to the creation of customized Application Specific Integrated Circuits (ASIC) that are benchmarked using energy per inference or energy per operation as a standard quantity of efficiency [39]. A review of 22 works that create ASICs for machine learning accelerators in the leading conference of solid state circuits, the IEEE International Solid State Circuits Conference (ISSCC) of 2023 and 2024, shows the number of operations per unit of power, expressed as tera (floating point) operations per watt (T(FL)OPS/W), to be the most important metric of comparison [40–42, 42–60]. This metric is model-independent and describes the efficiency of the design under the effect of other factors such as area and technology. The applications and data benchmarks of these works include visual question answering, caption-based image retrieval, language modeling with transformers, text-to-image generation using diffusion models with attention and the general convolutional and recurrent neural networks applied to vision and sequence tasks. Nearly half of these works consider the number of operations per area defined as the number of tera (floating point) operations per square millimeter (TFOPS/mm²) [40, 41, 43, 56–59]. As a result, the power consumption described by the above works, except three, range from 1mW to 4.3 W and do not describe any "sleep" mode. On the other hand, Refs. [42, 47, 58] describe ASICs for speech-enhancement in hearing-aids, and VGG-16 and ResNet50 accelerators for vision applications with a power consumption of 740, 567 and 870 μ W, respectively.

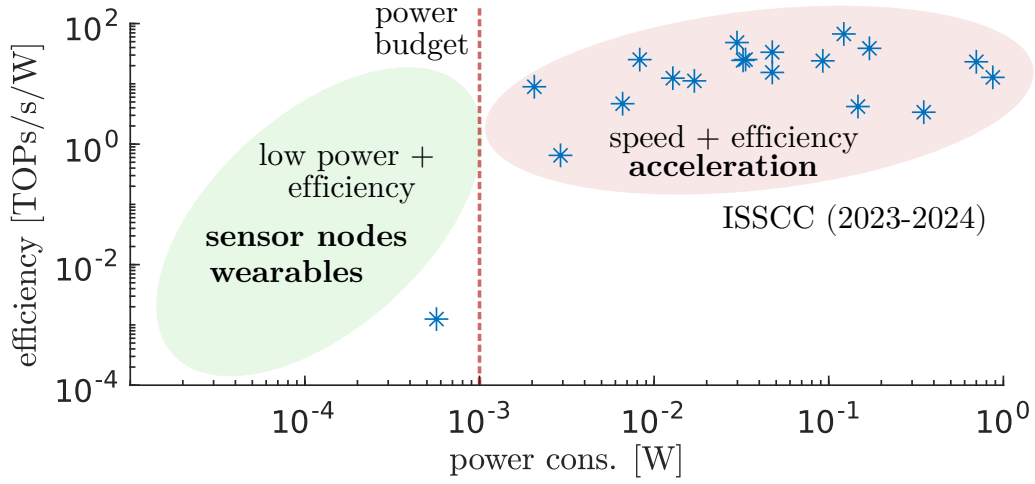


Figure 1.1: Power consumption over efficiency in terms of operations per second per watt shows the majority of state-of-the-art Edge-AI works to focus on speed and efficiency for acceleration. However, this approach is incompatible with sensor nodes that have low power and energy budgets.

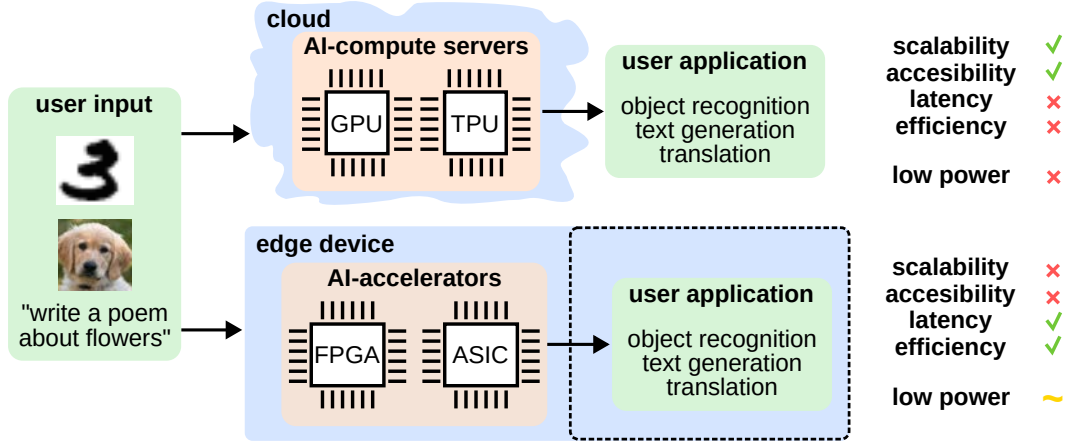


Figure 1.2: Conventional cloud- and edge-implementations of AI focus on applications such as object recognition from images to text generation using Large Language models that focus on speed. Both directions have several merits and demerits. However, both approaches are unsuitable for implementation at sensor nodes that commonly have extremely low power and energy budgets.

Fig. 1.1 plots the power consumption over the efficiency of the above works, expressed as the operations per second per watt. Considering the applications that these works are applied to, it is essential that this efficiency (T(FL)OPS/W) is high since there is no demerit on a high wattage or area that is required as all these applications compete with the GPU. Therefore, it is essential for these works to show approaches that highly outperform the GPU in terms of inference and training efficiency for adoption in existing technologies in industry. In other words, although these works are part of the Edge-AI field, most do not consider application at the far-edge or sensor nodes where there is a limited power and energy budget due to it being finite, i.e., by an on-board battery. Other than Ref. [40], no works provide a description of a power-off mode that is essential for long-term operation with minimum energy consumption. As a result, most of these accelerators are not suitable with AIoT applications where low-frequency sensors are prevalent and energy and power budgets extremely small [29, 61]. Another issue is that the majority of the reported implementations are proprietary and rely on ASICs that have high design costs, restricting public access from adopting novel hardware architectures for custom deployments.

Field Programmable Gate Arrays (FPGA) have emerged as an alternative to ASICs due to their programmable logic and customizable building blocks at significantly lower economic costs. In addition, FPGAs have been proposed as a means to alleviate computing constraints at the edge and accelerate processing, resulting in a significant speedup compared to Central Processing Unit (CPU) based platforms such as microcontrollers [62]. However, similar to ISSCC, the leading works that consider Edge-AI on FPGA, as is exemplified by the proceedings of the 'FPGA' conference in years 2023 and 2024, shows speedup and efficiency to be the main points of advancement [63–72]. However, the emphasis on maximizing processing speed offers limited advantages for sensor node implementations since system latency is inherently constrained by sensor frequency. Therefore, the sacrifice of power consumption over higher efficiency, i.e., scaling up of architectures as these provide more operations per Joule of energy, is unsuitable with advancing the accessibility and adoption of Edge-AI at sensor nodes themselves. Moreover, the FPGAs used for the above works commonly have a purchasing cost between hundreds to thousands of USD. In other words, the majority of described works are not compatible with long-term monitoring applications using IoT systems due to limited power and economical budget.

Fig. 1.2 summarizes the above review on Edge-AI works and compares it to the general method of implementing AI models by using cloud compute solutions. The majority of AI solutions focus on computer vision or natural language processing, involving object recognition, text generation and similar tasks as described with the accelerators above. In the case of the cloud-based approach, the data of a user is sent to the cloud where compute servers implement specific neural networks to perform inference, leading to output such as a generated poem or a class or object description from images. This approach provides near-unlimited resources as compute servers are easily scaled. Furthermore, this approach is highly competitive in accessibility, namely, anyone can use compute servers at highly affordable cost using widely-supported software platforms such as Tensorflow, PyTorch and ONNX. However, cloud systems have several drawbacks, originating from the requirement of back-and forward-transmission between the cloud and user, leading to higher system latency and substantial energy costs. This transmission cost makes this approach

unsuitable for sensor nodes at the edge, for example, in agriculture, where systems have extremely-limited energy budget due to batteries. AI accelerators that are implemented on or near edge devices provide a distinct advantage in latency and efficiency. As a result, this approach is a suitable replacement of GPUs and Tensor Processing Units (TPU). However, the majority of such accelerators have a power consumption (>1 mW) that is too high for AIoT sensor nodes at the edge and utilize hardware that is not economically viable.

1.4 The Promise of Low-Power Edge-AI at Sensor Nodes

Edge-AI at sensor nodes is promising due to the possibility of targeting a yet fairly unknown and small market, allowing for disruptive innovations to emerge [73]. Fig. 1.3 shows the equivalent diagram as before in the case of Edge-AI at the sensor node. Here, sensor-node optimized AI implementations on FPGAs, ASICs or neuromorphic processors with low power consumption allow for long-term monitoring applications within extremely resource-constrained environments. These environments include systems that are used in applications such as epileptic seizure monitoring outside of the hospital, anomaly detection of equipment and production lines within manufacturing, logistics and mining, and estimation of animal behavior in livestock farming. An additional essential difference of these AI-integrated sensor nodes is the prevalence of low-frequency, low-dimensionality sensors compared to the conventional environment where accelerators reign. These sensors include 3-axis accelerometers, humidity sensors, sensors that measure force and medical sensors such as blood pressure, oxygen sensors. Other examples in the medical field include electroencephalogram (EEG), electromyographic (EMG) sensors. Furthermore, the use of small sensor-optimized implementations are fruitful for the far edge or isolated nodes such as in satellites or rovers outside of Earth where energy budgets are extremely small and irreplaceable. As a result of the above considerations, the first question that this Thesis focuses on is

Question 1: *How can efficient Edge-AI hardware architectures be created for extremely resource-constrained sensor nodes?*

In order to study and develop digital hardware architectures for implementing Machine Learning (ML) algorithms on these sensor nodes, this Thesis considers two approaches; a bottom-up approach and a top-down approach. Both these approaches originate from consideration of an actual field-tested application in precision livestock farming, namely, the classification of cow behavior from accelerometers. The bottom-up approach considers classical ML algorithms, i.e., low algorithmic complexity, while the top-down approach considers an RNN, a complex algorithm that is commonly used for language processing but unexplored for time-series Edge-AI applications. Classical ML algorithms require the extraction of features from raw data and represent a large section of existing implementations on sensor nodes for AIoT applications [74–76]. Furthermore, a review of machine learning in agriculture and manufacturing that utilize data from sensors at the edge shows that edge implementations of ML are limited and the majority consider either classical ML algorithms, CNNs or Multi-Layer Perceptron (MLP) / fully-connected networks [75, 76].

The often low dimensionality and frequency of sampling in combination with extremely limited resources on sensor nodes in IoT can be effectively leveraged by

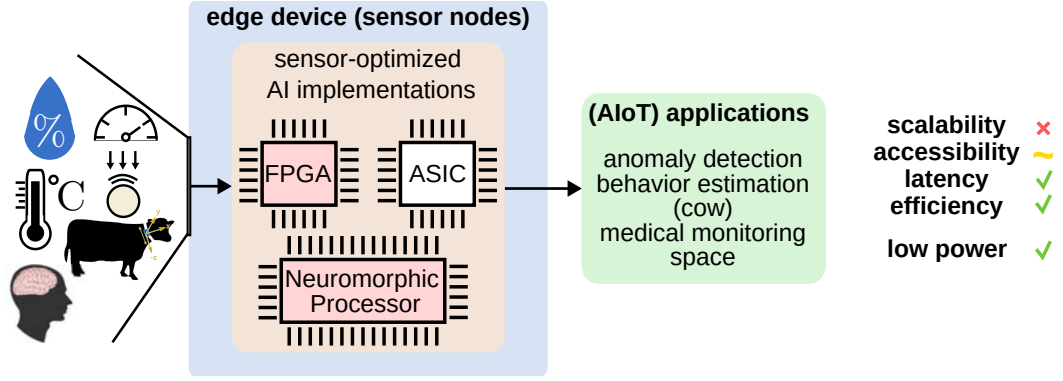


Figure 1.3: Edge-AI at sensor nodes require sensor-optimized AI implementations that balance the limited number of resources, speed and power consumption due to being always on. Different from the applications in the high-speed Edge-AI, these locate themselves at the far edge, involving long-term monitoring of epileptic seizures of patients, anomaly detection within machinery such as mining equipment and manufacturing, and behavior estimation of livestock including cows.

Recurrent Neural Networks (RNN) as these networks process data on an input-by-input basis. In other words, unlike fully connected and convolutional neural networks that require the entire sensor data space during inference, RNNs repeat operations for each new sample over a set number of timesteps. Taking into account also their ability to recognize spatio-temporal patterns within generative processes [77], RNNs seem to be particularly relevant for sensor applications. Other than sensors, RNNs have been extensively used in machine translation, video captioning, and actuator control, where latency is crucial for user-friendliness [78–81]. Similar to the above review, the primary performance metric is system latency and efficiency, which is minimized through techniques such as parallelization, pipelining, quantization, and resource scheduling optimization [82, 83]. However, to the best of my knowledge, no RNN hardware architecture has been proposed in literature that leverages the processing scheme of RNNs to minimize resources, and implements a real-time efficient processing approach with regards to sensor inputs on the node itself.

In Part I, this thesis introduces cow behavior estimation, part of a growing field named Precision Livestock Farming (PLF), an actual application of AIoT at sensor nodes, its significance from a social and climate change perspective followed by a short specific review on ML hardware implementations of animal behavior estimation. Hereafter, Part I builds up with a first DT implementation on the Lattice (Hillsboro, OG) ICE40UP5K FPGA for cow behavior estimation, followed by a study describing a neural architecture search of an RNN on this application and introducing a post-training quantization scheme. As a result of this study, a cow-behavior estimation optimized RNN with integer-only operation is found that can easily be implemented on the prior described FPGA. Finally, a resource-minimized RNN hardware architecture is proposed for the ICE40UP5K FPGA that utilizes an adapted version of the above post-training quantization scheme, leading to one of the most efficient and low-power RNN implementation for IoT applications. This final work, written using Python and SystemVerilog, is freely available [84].

1.5 The Unsustainable Pace of Conventional Machine Learning

Moore's law, coined by Gordon E. Moore in 1965 [85], dictates a doubling of transistors within the same area of an Integrated Circuit (IC) every two years. This law has dominated the semiconductor industry for decades, leading to productivity increases across the world that has played a large role in the economic growth in the latter half of the 20th century and 21st century [86]. The effect of this law is easily noticeable in the computing density of electronic devices in the last decade, for example, the rise of smartphones, tablets and laptops [87, 88]. Recently, signs are indicating the end of Moore's law or at the very least the slowing down of this acceleration, leading to a pivot of focus on the development of unconventional computing architectures [89].

The collapse of Denner's scaling among other factors have led to a standstill of the clock frequency in Central Processing Units (CPU) for further speedup. Instead, the focus has been on the utilization of more area to perform parallel computing, i.e., in the form multi-core multi-threaded CPUs that perform tasks in parallel. This parallelism has culminated in the Graphics Processing Units (GPU) that are constructed with thousands of parallel compute cores. These GPUs have enabled mass parallelism in algorithmic implementations, leading to vast advancements in the graphical capability of computers, allowing for the possibility of real-like graphics in virtual worlds and gaming.

The rise of the GPU and its continued development is on the other hand also the main factor in the rise of state of the art Artificial Intelligence (AI) using neural networks. It is hard to pinpoint exactly when AI took off since AI as a concept is already fairly old. However, many claim it to be the introduction of LeNet, a first Convolutional Neural Network (CNN) that was able to get high accuracy, around 99%, on classifying the Modified National Institute of Standards and Technology (MNIST) database by applying training by means of backpropagation (BP) with gradient descent on CNNs around 1998 [90]. These networks implement convolution, and are therefore able to solve tasks that include shift-invariant datasets. However, at the time it was harder to train bigger networks for solving more complex tasks due to the constraints of hardware. It took until Alexnet in 2012 [91], involving breakthroughs in network and learning methods, for the first larger computer vision classification task to be solved at an accuracy of 84.7%. This task is the classification of upto 20,000 distinct classes from images of a database called Imagenet, containing 14 million hand-annotated images [92].

Similarly, Recurrent Neural Networks (RNN) have their origin in cognitive science and early parallel computation from the 1980s [93, 94] and some of the more complex forms are the Long-Short Term Memory and Gated Recurrent Unit [95, 96]. One issue that comes up RNNs when training such networks is that these require a more complex form of training with gradient descent using an algorithm called backpropagation through time (BPTT), necessitating far more memory and computation than with BP as in the case of CNNs. Therefore, similar to the story described with CNNs, it took until recently for hardware to be ready for large variants of these models to be applied to tasks that involve time-series classification or regression including translation, key-word recognition and other sequence learning problems [97].

One of the most interesting applications of AI, I believe, in this era was the introduction of AlphaGo. AlphaGo, created by combining residual convolutional neural

networks [98], monte-carlo tree search and deep reinforcement learning [99], was the first AI to beat the world champion in one of the most complex strategy board games, Go, in 2016 [100]. Another example is AlphaFold from the same company, Deepmind, that was able to become 1st place in the international competition on the prediction of protein 3D structure using deep residual networks [101]. From this discussion, it can be deducted that AI recently was heavily limited by the advancements of hardware capabilities, dictated by Moore’s law, and continues to be dictated by it, i.e., if the scalability and computational capabilities of hardware does not keep progressing to maintain the speed of expansion of AI, then we will once again come to a standstill.

1.5.1 The Transformer and LLMs

Since the advent of successful large-scale RNN and CNNs on a wide range of tasks, AI has come a long way. The further introduction of more advanced models such as the Transformer, utilizing self-attention [22] and the Generative Adversial Network (GAN) [102] has led to a boom in the field. The original Transformer pushed the task of translation English to German and English to French to Bilingual Evaluation Understudy (BLUE) scores of 28.4 and 41.0, respectively. Here, a BLUE score of 40 represent understandable to good translations and scored higher than 60 represent translations better than humans. As a result, Google translate also switched from a RNN-based model to Transformer. Moreover, the separate fields of computer vision, dominated by CNNs and Natural Language Processing (NLP) that utilized RNNs has been fused with advancements in networks using the vision transformer [103] to further advance accuracy on Imagenet and CIFAR-100, an other large-scale image-based dataset, to 88.55% and 94.55%, respectively.

Although most of the AI models and its progress described above have been motivated by the increase of accuracy and performance on translation and object detection, the Transformer model has led to further breakthroughs wider than the scope of beating the best player in the game of Go. One of the largest news topics during the time of COVID-19 pandemic was the introduction of AlphaFold2 [23]. This model, replacing residual networks and reinforcement learning with a transformer-based approach, can estimate 3D protein structure as good as to or close to those of high-quality experimentally determined structures that cost hundreds of thousands of US dollars. Furthermore, AlphaFold2 contributed to the discovery of the shape of two protein structures of the COVID-19 virus and could help with the development of certain types of vaccines in the future [104]. Due to its performance on single protein structure estimation, it has also had high impact in the field of experimental biology but it is not able to model the interaction between multiple proteins as of yet [105]. Other examples that employ similar approaches are those of new discoveries in drug discovery, for example, the discovery of a new antibiotic [106], and the successful magnetic control of plasmas in a Tokamak nuclear fusion reactor by an AI model employing reinforcement learning [24].

Finally, the most well known AI boom that has occurred recent to the date of writing is the emergence of Large Language Models (LLM). Once more, LLMs employ Transformers to be able to capture extremely long sequences of text efficiently. Two main models that were introduced initially were the Bidirectional Encoder Representations from Transformers (BERT) and OpenAI’s Generative Pre-trained Transformer (GPT) [108,109]. Both these models types are pre-trained on a wide set of diverse language data, after which they are fine-tuned to specific tasks. In other

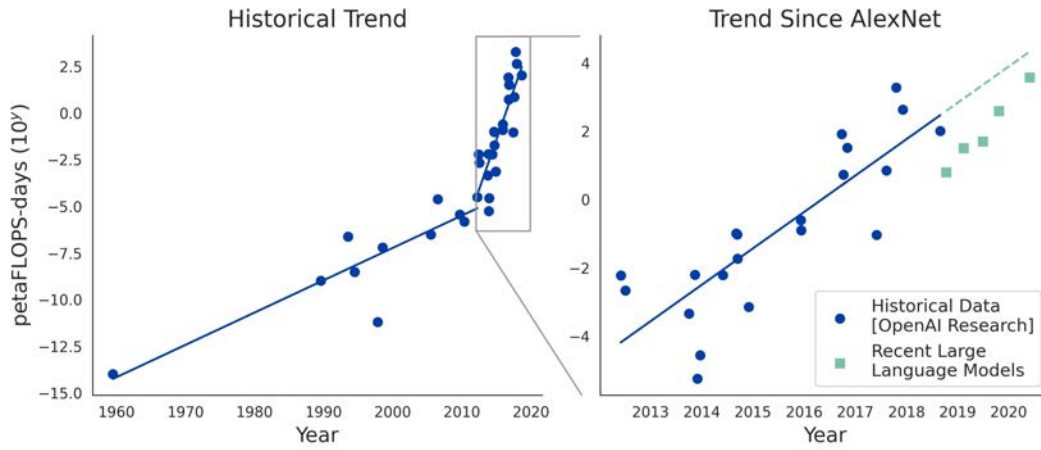


Figure 1.4: Growth of compute and time required for state-of-the-art AI models to be trained in Peta Floating Point Operation (FLOP) days, taken with permission from Ref. [107]. (CC BY-NC 4.0 DEED licence)

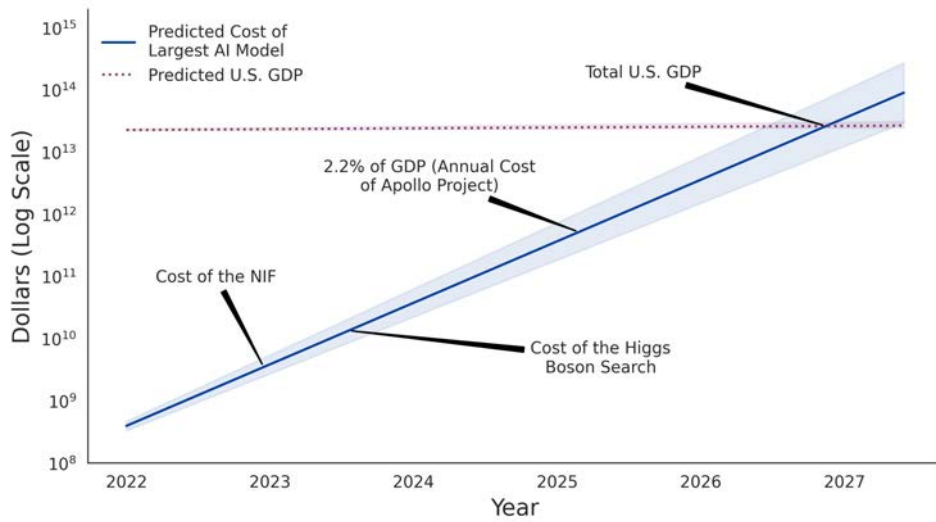


Figure 1.5: Extrapolated economical cost of compute for training state-of-the-art AI models in US dollars, taken with permission from Ref. [107]. (CC BY-NC 4.0 DEED licence)

words, these models are trained in such a way that they can understand any type of task concerning language. The first version of GPT-1 was followed up by iterative improvements up to GPT-3.5 and GPT-4 that have gathered widespread attention around the world. The introduction of GPT-3.5 and GPT-4 combined with a freely accessible terminal has led to a worldwide explosion of interest into AI. This has led to record sales of GPUs and NVIDIA stocks on Nasdaq, the main fabricator of GPUs in the world, to triple within one year (2023-2024).

1.5.2 State-of-the-art AI is Growing $7\times$ Faster Than Moore's Law

The above described developments are coupled with a steep acceleration of increase in the number of Floating Point Operations (FLOP), processed by hardware such as GPUs, required for the training and operation of state-of-the-art neural networks. Before the inflection of Alexnet in 2012, the number of FLOPs to train AI models grew roughly as fast as Moore's law [110]. However, ever since that point in time, the increase in performance has been heavily reliant on the increase of network size, leading to a doubling of FLOPs for training these models every 3.4 months (Fig. 1.4) [110]. At roughly $7\times$ the speed of Moore's law, this acceleration is significantly faster and therefore unsustainable over the long term. The cost of FLOPs per US dollar has halved roughly every 2.3 years since the 1940s in line with Moore's law, slowing down over the last 10 years [111]. Furthermore, due to trends in price and hardware availability such as GPUs and Tensor Processing Units (TPU), the cost for compute has remained nearly constant on various hardware platforms until recently. As a result, SOTA AI models have become increasingly expensive to train in terms of economical costs.

While the number of parameters of the models GPT-1 and BERT equated to 120 and 340 million, respectively, GPT-4 [112] and Gemini [113], two recent SOTA LLM, have around 1.76 and 10 trillion parameters, equating to a 10^6 increase and as a result significantly more FLOPs required for training. The costs for training GPT-1 and BERT equated to hundreds of dollars USD. However, the economical costs to train GPT-4 equated to 100 million US dollars [114] and significantly more for Gemini, but the exact number remains unknown. Figure 1.5 shows the predicted cost in US dollars on a logarithmic scale until past the year 2027 for training SOTA AI models as calculated in Ref. [115]. Compared to this predicted cost, the actual cost of GPT-4 is lower than the curve in the figure predicts for this year, however, Gemini could come close. Even if the pace is slower than Ref. [115] predicts, OpenAI has stated that they are not going to further increase model size for subsequent models, showing that it might indeed be becoming too costly to maintain the current approach for improving the capabilities of LLMs. Another issue with the approach of large scale and compute is that only few institute and researchers will be able to work on it. In other words, only those researchers with the budget for training such LLMs are able to work on SOTA AI topics currently.

1.6 The Promise of Neuromorphic Computing

1.6.1 Natural Intelligence in Organisms and their Efficiency

As described above, AI constructed by large-scale neural networks has been able to attain significant achievements across various domains including those described

prior as well as the creation of artwork, music, and complex planning [100,116,117]. Moreover, it can be said with medium confidence that almost any new application or technological device is equipped with some element of AI or machine learning. However, each of these accomplishments is limited to fairly narrow domains. Although LLMs have emerged using multi-model systems to increase the number of functions [113], as of writing no clear breakthrough model exists that is competent on a wide range of tasks as is the case with natural intelligence.

To dive into what AI currently lacks compared to natural intelligence, a comparison of natural and artificial intelligence is necessary. To build towards this comparison, Ref. [118] defines the characteristics of general intelligence as,

1. reasoning and problem-solving
2. learning
3. inference-making
4. utilization of common sense
5. autonomous defining and adjusting of goals
6. processing ambiguity and ill-defined situations
7. ability to create novel representations of newly-acquired knowledge

With these points in mind, current AI seems to be able to perform well (1)-(3) under certain conditions and (4) seems too undefined to apply to machine intelligence [119]. Moreover, there have been some signs that (5) can be done by SOTA AI models, including LLMs under very special conditions, but in general the defining and adjusting of goals, that can also be termed as planning, are so far deemed impossible [120]. Point (6) and (7) seem to be the main points where AI and natural intelligence diverge. According to Ref. [119] this has to do with the limitation of universal Turing Machines, namely, the output of such machines is determined, albeit in complicated and probabilistic ways, by the input. This input represents the environment by means of sensors. In the absence of input, this system will not perform any task. In other words, on itself the algorithmic system can not come up with an own goal or solution to pursue unless instructed to do so [119].

In Reinforcement Learning, an agent interacts with a changing environment over time over a set of timesteps [121]. For each timestep, this agent receives a certain observable state of the environment and selects an action that is expected to gain the highest reward according to a policy for the next timestep. The behavior of the agent is defined as the mapping from this state to actions and transitioning to next states accordingly, implementable by any Turing machine. Over time, the agent tries to maximize the long term return. With this broad definition, combined with deep learning models, an agent is able to learn how to play games such as AlphaGo [98] but also drive a car or learn how to fly a drone [122,123] over time.

This is per definition similar to organismic agents in nature, which have to traverse a complex environment taking actions according to a goal and reward. However, the difference is that the goal, reward and environment of organismal agents are constantly collapsing and expanding over time [125]. The dynamics of all components of this organism, namely, its environment, the learning, reward and goals are far more complex than that what any algorithm so far can implement [119]. As also pointed

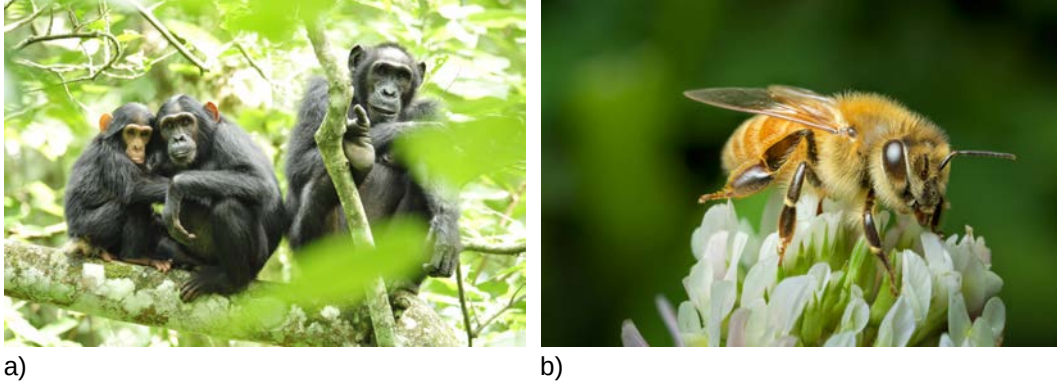


Figure 1.6: a) A Chimpanzee family in the wild in Uganda (public-domain). b) A New Zealand honey bee foraging clover (CC BY-SA 2.0 DEED, taken from Ref. [124]).

to above, some biologists argue that any kind of Universal Turing Machine is incapable of replicating this. Moreover, in many cases, the goal, action and reward are ambiguous, as these are not clearly defined (point (6)). This ambiguity according to biologist and mathematicians is one of the reasons why creativity can emerge, leading to unexpected actions when there is no clear defined goal or situation by humans or any organism [126,127]. A more in depth study of the role of biology and evolution and general intelligence can be read in Ref. [119].

In addition to natural intelligence being far ahead of AI, organisms that show general intelligence have evolved over billions of years under the constraint of natural selection, leading to highly efficient computing paradigms. As a result, vertebrates with large brains emerged, including us humans and chimpanzees (Fig. 1.6a). It is important to note that a brain represents one of the most energetically expensive parts in the body and in addition requires uninterrupted energy supply [128]. In humans the weight of the brain takes up only 2% of body weight but consumes around 20-25% of the total energy expenditure [129]. This is further exacerbated for immature individuals, here the brain consumes around 50% of total energy in take [130]. These high energy costs for brain development in large-brain animals compete with other bodily functions such as reproduction, slowing down its development and leading to later development of maturity [131]. According to the Expensive Brain Hypothesis and related works, the evolution of bigger brains was likely possible due to the reduction in size of other metabolically expensive tissue such as muscle, and the decrease in growth and slowed development of reproduction [132,133]. In addition, animals with large brains tend to have higher diet quality and better capabilities to prevent starvation [134,135]. In other words, humans have large energy-costly brains leading to consciousness and high levels of general intelligence that have evolved due to outweighing of the deficits in energy balance with large improvements in reproduction and survival [131].

Due to the effectiveness of balancing energy deficits and survival benefits by means of evolution, these brains show extremely high energy efficiencies. As stated before, the brain consumes around 20-25% of the total energy expenditure in our bodies, but this equates to only around 20 W of equivalent power, less than an average laptop (around 30-70 W) and far less than a single GPU that is commonly used to train AI models, the NVIDIA A100, that consumes around 250 W of power [136]. As a reference, GPT-4 [112] used 25,000 of these GPUs for its training. Other than

large brains and animals that show general intelligence, similar arguments can be said about the diversity of complex behaviors that emerge from insects that have fairly small brains, for example, the honey bee [137] (Fig. 1.6b). This insect, with a brain that consists of only around one million neurons and a 10 μ W power budget, is able to perform tasks ranging from pathfinding, a wide-range of pattern recognition and complex social activities [138].

1.6.2 Neuromorphic Computing as an Alternative

Due to the efficiency and complexity of nature combined with a possible end to Moore’s law in the near future, a separate field that aspires to replicate natural intelligence has emerged [139], replacing the von-Neumann architectures that implement Turing machines with computing based on the structural principles and information processing of nervous systems in biology including the human brain: Neuromorphic Computing. Neuromorphic computing which is inspired by the efficiency and cognitive ability of the biological neural systems, has emerged as one of the main upcoming computing paradigms [140, 141]. Fig. 1.7 shows the reasoning behind the pursuit of neuromorphic computing within this Thesis.

Neuromorphic computing devices generally implement a distinct non-von Neumann hardware architecture, built with neurons and synapses that are arranged in a highly parallelized fashion. Fig. 1.8 shows the difference between neuromorphic and non-neuromorphic type of computing with neural networks. Non-neuromorphic neural networks, including transformers, RNNs and CNNs, are constructed with neurons represented as matrices that utilize dense computing by means of Multiply-and-Accumulate operations, namely, the multiplications of weights with inputs accumulated over rows and columns, followed by a non-linear activation function. On the other hand, neuromorphic computing utilize the sparseness and asynchronous properties of discrete events (spikes) similar to biological neurons and synapses, allowing for extremely efficient computing architectures [142]. One conventional way to implement brain-like neurons is to use the Leaky-Integrate-and-Fire (LIF) model [143]. Here, weights are added to a leaky membrane potential whenever a spike arrives over time. Whenever this membrane potential exceeds a pre-set threshold, a spike is output. Fig. 1.9 shows a straightforward way of implementing a LIF neuron as a circuit using current sources and switches that close whenever a spike arrives in time, charging the capacitor. An additional block outputs a spike whenever this capacitor is charged beyond the voltage threshold followed by a fast discharge of the capacitance to ground.

The above example serves as an introduction to LIF neurons, a model created by Louis Lapicque in 1907 that represent a first-order representation of neurons in the brain. A more biologically plausible but computationally intensive example is the nobel-prize awarded Hodgkin-Huxley model [144]. Other models include Izhikevich and Adaptive Exponential LIF neurons that represent a balance between computational intensity and biological plausibility as these can be simulated in real-time while showing spiking and bursting behavior similar to cortical neurons in the brain [145, 146]. A balance between bio-plausibility and ease of computation is key for neuromorphic computing and brain-inspired AI as more complex models require extensive circuitry with larger area and higher power consumption [147, 148].

Over the years, various large-scale neuromorphic devices have emerged that allow implementation of networks with the above described neuronal models, ranging from architectures such as LoiHi and SpiNNaker where neurons are implemented

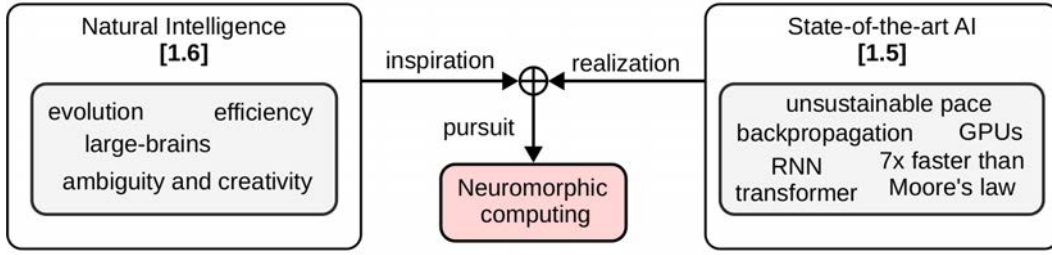


Figure 1.7: The realization of the pace of state-of-the-art AI, unsustainable in terms of cost and lacking in terms of intelligence, combined with inspiration from general intelligence in animals has led me to pursue neuromorphic computing as one of the main topics in this thesis.

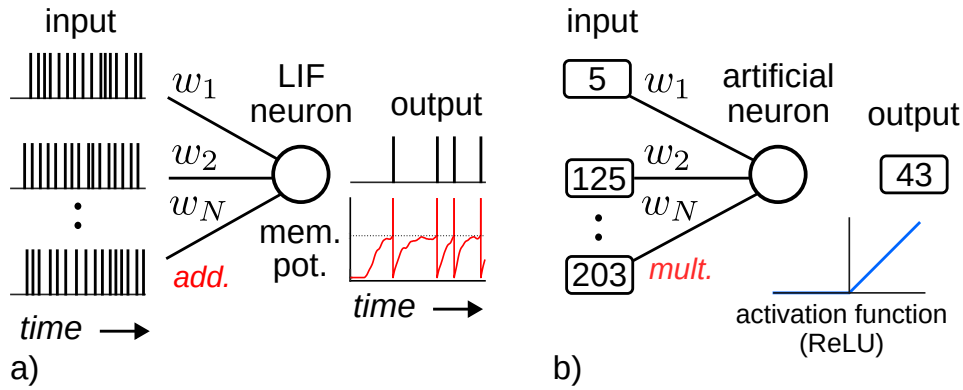


Figure 1.8: a) A simulation of a Leaky Integrate and Fire (LIF) neuron, a non-linear dynamical system, with N spike inputs over a short period of time, leading to output spikes when membrane potential crosses a threshold. b) A diagram of an artificial neuron, a representation of the matrix-based approach of conventional AI using multiplications of weights and activation functions (Rectified Linear Unit, ReLU) for non-linearity.

digitally using discrete timesteps [149, 150] to BrainScaleS and the DYNAP-SE2 where neural primitives are implemented in a custom analog core but with a digital event-routing network [151, 152] to memristor devices such as Mosaic where each neuron is arranged in an analog crossbar array [153, 154]. Digital platforms have allowed 1-to-1 mapping of trained Spiking Neural Networks (SNN) on-chip [155–157] to large-scale brain simulations [158]. However, these platforms remain limited in terms of neuronal dynamics and scaling due to the computational requirements of complex neuronal simulations. On the other hand, memristor devices and analog implementations of neurons using sub-threshold MOSFETs have created complex neuronal dynamics that more closely resemble the brain while maintaining low-power consumption [152, 159, 160]. Such low power operation along with the ability to efficiently process signals in real-time makes the integration of neuromorphic systems in the field of healthcare, particularly bio-signal monitoring, immensely promising.

Fig. 1.10 shows an overview of the state-of-the-art neuromorphic hardware platforms and the amount of energy required per spike operation (J/SOp). In contrast to Floating Point Operations (FLOP) or Multiply-and-Accumulate (MAC) operations that dominate the field of AI, neuromorphic hardware that implement SNNs operate

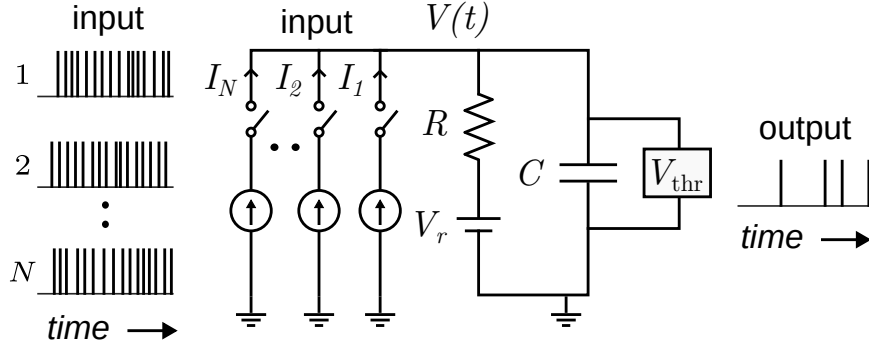


Figure 1.9: An elementary implementation of a Leaky Integrate and Fire (LIF) neuron using current sources with switches that close whenever a spike arrives. These current sources charge a capacitor that discharges when they close. A spike is output whenever voltage on the capacitor exceeds a pre-set threshold.

with so called "Spike Operations" (SOp). These operations are defined by the processing of a single spike arriving at the neuron. The calculation for the total energy consumed for this operation is not universal among all works, sometimes, these only include spike detection while in other cases such as Ref. [152] it comprises the entire neuron processing. Compared to the cost of a Multiply-Accumulate operation in 45nm [172], costing 9.6 pJ, it is only recently that spike operations have surpassed this technology node in terms of efficiency. Furthermore, it can be observed that the majority of efficient solutions are constructed by means of digital hardware at smaller technology nodes. It remains to be seen if large-scale mixed-signal hardware, i.e., analog neurons with digital routing can surpass digital when it reaches lower technology node, similar to Ref. [165].

On the other hand, while the far majority of digital platforms implement simulations of Leaky Integrate and Fire (LIF) neurons, some of the mixed-signal variants implement complexer neuronal models, including the Exponential LIF (ExpLIF) and spike-based adaptation that are closer to actual biological neurons [152, 164]. Compared to digital variants that implement simulations of neuron models using solvers such as Euler's method on hardware, the ExpLIF model and its analog implementation are able to operate with far richer dynamics under extreme conditions of operation including chaos. Furthermore, for better or for worse, the state-of-the-art DYNAP-SE2, employs MOSFETs operating subthreshold region, showing approximately 20-30% inter-neuron heterogeneity and across bias generators. More details can be read about in Appendix C. I believe DYNAP-SE2 and mixed-signal platforms are more suitable for studies about emergent physical properties that are observable in biology and their possible application for computation and task solving. This is because mismatch and heterogeneity, not as present in digital hardware, can have a significant effect on global properties such as the emergence of synchronization patterns [173]. I believe this is a key component in laying down the foundation for computation that is more similar to what we can observe in natural intelligence. On the other hand, digital implementations of neuromorphic computing are I believe more suitable for applied research in Edge-AI including proven benchmarks and tailored electronics that are more easily available such as Microcontroller Units (MCU) or FPGAs, allowing their widespread use in IoT systems in the future. This leads to the second question of this thesis

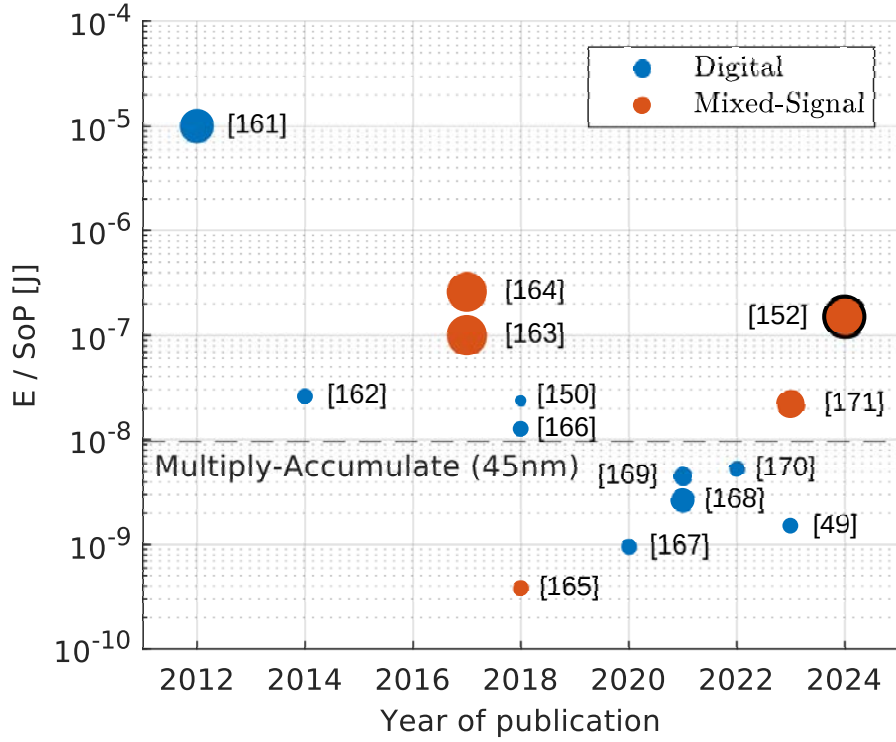


Figure 1.10: Overview of the energy per spike operation in Joules for the most recent neuromorphic hardware platforms including digital and mixed-signal based implementations (Refs. [49, 150, 152, 161–171],). The hardware platform utilized in this Thesis, Ref. [152], in Chapter 7 is also shown here. Size of dot indicates technology node, ranging from 180 nm [152, 163, 164] to 14 nm [150]

Question 2: *How can emergent physical phenomena, namely, chaos and synchronization, be utilized in spiking networks for computation?*

In digital-based neuromorphic hardware, the majority employs leaky or non-leaky Integrate-and-Fire (IF) neurons, employing rate-coding to encode information via spike counts, leading to many-to-many network architectures. However, the intrinsic temporal aspect of spikes opens avenues for leveraging both spatial and temporal dimensions. Notably, Time-To-First-Spike (TTFS) coding, which enforces the generation of a single spike per neuron at the earliest conceivable moment in time, offers a pathway to architectures that are both faster and more sparse than those based on spike-count coding. Recent learning advancements using TTFS have demonstrated competitive performance in both spatial and temporal tasks with significantly reduced number of spike operations [174, 175]. Despite the proven advantages of TTFS, implementations of TTFS-based SNNs in hardware accelerators remain relatively unexplored. This leads to the third question of this thesis

Question 3: *How can synchronization be used to create highly-efficient digital SNN hardware architectures?*

In synopsis, this thesis describes two approaches with regards to neuromorphic computing in Part 2. First, the emergent properties of neuromorphic circuits and processors are explored, namely, synchronization and chaos, and their use in com-

putation, leading to Edge-AI and a proof-of-concept of seizure encoding. Second, synchronization is employed to create an SNN hardware architecture that employs TTFS-coding for the MNIST dataset on an extremely resource-constrained FPGA, the Lattice ICE40UP5K, showing the first SNN accelerator on extremely resource-constrained tailored hardware due to reduction in spike buffer and network size.

1.7 Thesis Rationale and Outline

It is clear that the work described in the following Chapters is highly interdisciplinary, necessitating a wide introduction of the general academic trends in the current literature. Furthermore, it was essential to identify some of the academic issues within these trends and unexplored approaches that could lead to advancements in academic research. In this Section, I will summarize the introduction with a list of academic issues and approaches followed by a description of the main research disciplines within this thesis. Finally I will describe more in detail the structure and an outline of the results of each chapter (Fig. 1.11).

The general objective of this thesis was described by the following statement,

Objective: *The advancement of Edge-AI in terms of efficiency, capability and accessibility*

With this objective in mind, two main issues in the field of Edge-AI and neuro-morphic computing were introduced (see Sections 1.3 and 1.5),

Issue 1: *The majority of Edge-AI hardware focuses on customized architectures for acceleration, promising speedup and efficiency over GPUs that are incompatible with resource-constrained sensor nodes. (Fig. 1.1)*

Issue 2: *Although SNNs recently overcome ANNs in efficiency, they fail to consider and employ the complex dynamics and phenomena that emerge in networks of spiking neurons.*

Due to these academic issues, this thesis introduces three fundamental research questions that are intertwined in the two main parts of this thesis:

Question 1: *How can efficient Edge-AI hardware architectures be created for extremely resource-constrained sensor nodes?*

Question 2: *How can emergent physical phenomena, namely, chaos and synchronization, be utilized in spiking networks for computation?*

Question 3: *How can synchronization be used to create highly-efficient digital SNN hardware architectures?*

This thesis proposes a multidisciplinary approach with a focus on codesigning neural networks, leading to partial answers of the above stated questions by pursuing

- (a) The study and utilization of non-linear *dynamics* within spiking neural networks.
- (b) The research and development of digital *hardware* architectures of machine learning algorithms for Edge-AI.
- (c) The research and development of *algorithms* including quantization for recurrent neural networks.

- (d) The *application* of created *algorithms* and *hardware* within this thesis to precision livestock farming and epileptic seizure detection.

Fig. 1.11 shows the overall outline of this thesis following the above summary of the introduction, and the respective two Parts and Chapters that are to follow. These chapters have assigned a number in the left bottom corner that indicates the particular adapted publication in the list of publications of Appendix A. Furthermore, the disciplines emphasized in points (a)-(d) are color coded per Chapter. In principle, Part I represents an application-oriented approach while Part II represents a dynamics-oriented approach to neural network hardware codesign. The two hardware platforms that were utilized throughout this thesis, the Lattice ICE40UP5K FPGA and DYNAP-SE2, are described in the next Chapter, Chapter C. The border color per Chapter in Fig. 1.11 indicates the utilization of that specific platform. The following gives an extensive outline of the methodology and results described in each Chapter and the respective peer-reviewed publication assigned to it in the form of [J/CX] where "J" indicates a journal paper and "C" indicates proceedings of a conference.

Part I: From Precision Livestock Farming to Edge-AI via Minimized ML Algorithms and Low-Power Hardware Architectures

Chapter 2 Chapter 2 introduces livestock farming followed by an analysis of the increase of livestock and worldwide meat consumption, social issues of livestock farming and its impact on climate change. Hereafter, mitigation strategies are identified including one main strategy reported by the United Nation Food and Agricultural Organisation (UN-FAO), namely, the application of technology, big data and AI to improve the efficiency of livestock farming and reduce green house gas emissions, also dubbed Precision Livestock Farming (PLF). This Chapter continues by identifying that the majority of AI in PLF once again is performed on the cloud, creating equivalent issues of latency, efficiency and also economical costs for farmers. Furthermore, the literature review of previous works of applying Edge-AI to PLF in Section 2.2 points out that the hardware implementations of various algorithms are scarcely optimized for long-term monitoring on sensor nodes, namely showing high power consumption or energy per inference. This review is followed by two approaches to sensor-node optimized hardware architectures, implementing Edge-AI for this specific application.

Chapter 3, 4 [C6,J5] These approaches comprise the implementation of a low-complexity algorithm, a Decision Tree (DT) on FPGA using feature sets in Chapter 3 and the neural architecture search of a special Recurrent Neural Network optimized for cow behavior estimation at the edge and its quantization in Chapter 4. In Chapter 3, the horizontal-longitudinal angle of acceleration of the cow ($\arctan2(\mathbf{y}, \mathbf{x})$) is proposed as a new feature set together with the vertical z -axis of a 3D accelerometer in pipeline with a DT that classifies these features into 4 cow behaviors per a 25 sample window. The implementation of this DT on the Lattice ICE40UP5K FPGA shows an accuracy of 86.8 % with a power consumption of 216 μW and a latency of 2.58 s at a clock frequency of 2.9 kHz, resulting in energy of 557 μJ per classification that is competitive with previous state of the art animal behavior estimation works. Chapter 4 proposes a generalized post-training quantization scheme for RNNs and a random search of three distinct RNN types and their hyperparameters, namely, Long-Short-Term Memory and Gated Recurrent Unit as well as the

least complex form of RNN. This random search is used to find operation-count and memory-minimized RNN architectures that estimate cow behavior distribution from an available cow behavior dataset (described in Section 2.4). The results of this method show highly accurate ($<95.5\%$) distribution estimating RNNs with memory as low as 2.043 kB, utilizing only 42,822 Multiply-Accumulate (MAC) operations per inference. In this way, highly accurate RNNs are trained and optimized without expert knowledge of animal behavior that utilize an extremely small amount of memory and MAC operations. This allows RNNs to be implementable on low-cost tailored hardware that are compatible with the extremely small energy budgets of sensor-node devices at the edge.

Chapter 5 [J3] Part I culminates in the final Chapter, Chapter 5. In this Chapter, the integer-only quantization scheme for RNNs based on Chapter 4 is improved, followed by a specific hardware architecture that is resource-minimized for low-frequency sensor applications in Edge-AI. This hardware architecture applies a time-and-layer-multiplexing processing strategy, namely, resources are shared for the various operations across the computational graph of the RNN in time. As a result, this architecture minimizes resources, leading to minimal power consumption and resource utilization for a sacrifice in speed. Contrary to the majority of Edge-AI where speed and throughput are essential, here speed is not necessary as computation can be set to complete faster than sampling speed for low-frequency sensors that are prevalent in many applications of IoT and Edge-AI. Next to this processing strategy, a custom processing engine is proposed that implements full-integer MAC operations for RNNs. The implementation on the Lattice ICE40UP5K FPGA measured with an experimental setup shows a power consumption between 340 μW at a system clock frequency of 145 kHz using a two layer RNN with 10 neurons to 3.81 mW at 12 MHz with an RNN featuring 20 neuron and 4 layers, respectively. This clock frequency scales with number of sensor inputs or sensor frequency, requiring an internal clock frequency of 200 KHz for sensor frequencies of 10 (large model) to 100 Hz (small model). Finally, a proof-of-concept of this RNN architecture is performed using the cow behavior estimation model found in Chapter 4, featuring a model size of 13 neurons with 4 layers and 3 sensor inputs and accuracy of 95.1% for classifying four cow behaviors. This model consumes a total memory of 11.74 kB on the FPGA with a measured power consumption of 360 μW at an internal clock frequency of 146 KHz and energy per inference of 542 μJ . Furthermore, setting the internal clock frequency to 12 MHz measures a 2.3 mW power consumption (at the limit of power budget) at a clock frequency of 12 MHz, leading to a far lower energy per inference of 44.3 μJ that includes energy consumption from idling in between samples. These results show what I believe to be the lowest power consuming animal behavior estimating Edge-AI hardware implementation at the time of writing. Furthermore, to my knowledge it is the first time that an RNN has been implemented on an FPGA of this size and affordability (FPGA costing typically around 10\$ USD) for any application, leading to a step towards further democratization of AI.

Part II: From Neuromorphic Computing to Edge-AI via Synchronization, Chaos and Spiking Neural Networks

Chapter 6 [J4] This Chapter describes a study on the various synchronization phenomena under experimental conditions of noise and coupling within a ring network of chaotic spiking oscillators and their use in reservoir computing. These conditions include sweeping of the supply voltage, driving each node from periodic to

chaotic behavior, and coupling voltage that increases coupling strength of the node to an equivalent nearest neighbor. Furthermore, an analysis on the effect of coherent noise on synchronization is presented followed by patterned node heterogeneity where the ring network is split into two distinct halves with a separate supply voltage. Finally, a reservoir is proposed that utilizes synchronization phenomena to perform non-boolean computation. As a result, this study is used as the background for utilizing synchronization in the following Chapters.

Chapter 7 [J1] Subsequently, Chapter 7 proposes utilization of partial synchronization to encode seizures using intracranial Electroencephalography (iEEG) of actual patients with a network of Exponential Leaky-Integrate-and-Fire (ExpLIF) neurons on the mixed-signal DYNAP-SE2 neuromorphic processor [152]. Here, a full proof-of-concept system pipeline is proposed that includes an analog front end and an on-chip implemented Spiking Neural Network (SNN) to encode seizures from raw iEEG data. This SNN emphasizes correlation structures of the iEEG data during periods of epileptic seizures using partial synchronous clusters within the network. Consequently, the emphasis in this section lies on the entire system, the proof-of-concept SNN and the phenomena arising from the emergent properties of this network under iEEG input. Both Chapter 7 and 6 utilize the circuitry on the actual substrate, i.e., chaotic spiking oscillators implemented on a Printed Circuit Board and the analog subthreshold MOSFETs that implement Exp LIF neurons with digital routing on DYNAP-SE2. The works described above contribute to the fields of Edge-AI, neuromorphic computing and synchronization with complex non-linear dynamics by showing phenomena of synchronization within analog substrates and employing them for computation. The key point within these Chapters is the importance of investigating emergent phenomena in networks of non-linear dynamical systems and their use in computation, followed by an actual proof-of-concept on encoding epileptic seizures on a neuromorphic processor.

Chapter 8 [J2] Part II culminates in Chapter 8. This Chapter starts with an introduction to Time-To-First Spike (TTFS) coding in Section 8.1, a spike coding type that encodes information using only a single spike per neuron, followed by an introduction of some of the recent works using this coding in SNNs for handwritten digit recognition (MNIST dataset). Afterwards, Section 8.2 unveils a proof-of-concept study of employing synchronous spiking behavior in a Time-To-First-Spike (TTFS) encoded SNN to reduce the number of weights by 49.4% and the spike buffer size by 78.6%. Along with these approaches, I propose a hardware architecture, NimbleSNN, that balances resource use and processing speed, featuring a compact design with four parallel processing engines within a central spike core, processing a total of 28 neurons per cycle. Implemented on a Lattice ICE40UP5K FPGA, this proposed architecture shows a 20.4 mW measured power consumption with an inference latency of 65.8 μ s, resulting in an energy expenditure of 1.34 μ J per image from MNIST. Utilizing a total 4,639 logic elements, this implementation outperforms rate-based SNN implementations and competes closely with another TTFS-based implementation, providing a blueprint for the advancement of SNN accelerators using the internal dynamics, namely, synchronization between spiking neurons.

As described above and shown in Fig. 1.11, Chapter 2 functions as the background study for Chapters 3 and 4. Using the proposed RNN quantization and cow behavior model and hardware architecture of these Chapters, Part I culminates into the first RNN optimized for low-frequency sensors, implementable on FPGA at the

sensor-node, namely, at the edge itself, proposed in Chapter 5. This represents an application-oriented approach to codesigning neural network on hardware. Similarly, Chapter 6 presents a study on synchronization phenomena in a network of spiking oscillators, serving as the background study for utilizing these phenomena for computation in the subsequent Chapters. As a result, Chapter 7 shows a proof-of-concept for a full on-chip pipeline from iEEG analog input from patients to the encoding of epileptic seizures using the emergent synchronization phenomena in a network of ExpLIF neurons on the DYNAP-SE2 neuromorphic processor. This proof-of-concept I believe is a significant step in the direction of advancing the field of Edge-AI by utilizing the physical phenomena of complex dynamical systems that are currently not used in the conventional literature. Moreover, the hardware architecture that was developed in Chapter 5 was adapted and used to create the proposed NimbleSNN of Chapter 8. This Chapter shows another proof-of-concept of using the dynamics of a network of spiking neurons, pre-trained on the MNIST dataset, to propose an enhanced hardware implementation for advancing Edge-AI applications. Therefore, Part II represents a dynamics-oriented approach to the codesign of neural networks on hardware. In other words, the Chapters in this Thesis consider a multitude of disciplines and backgrounds that are employed and combined for proposing new methods in the development of neural networks with a focus on but not limited to hardware. As a result, this Thesis put together, proposes a neural network codesign approach that is believed necessary for the advancement of Edge-AI in terms of accessibility, capability and efficiency for the long-term future.

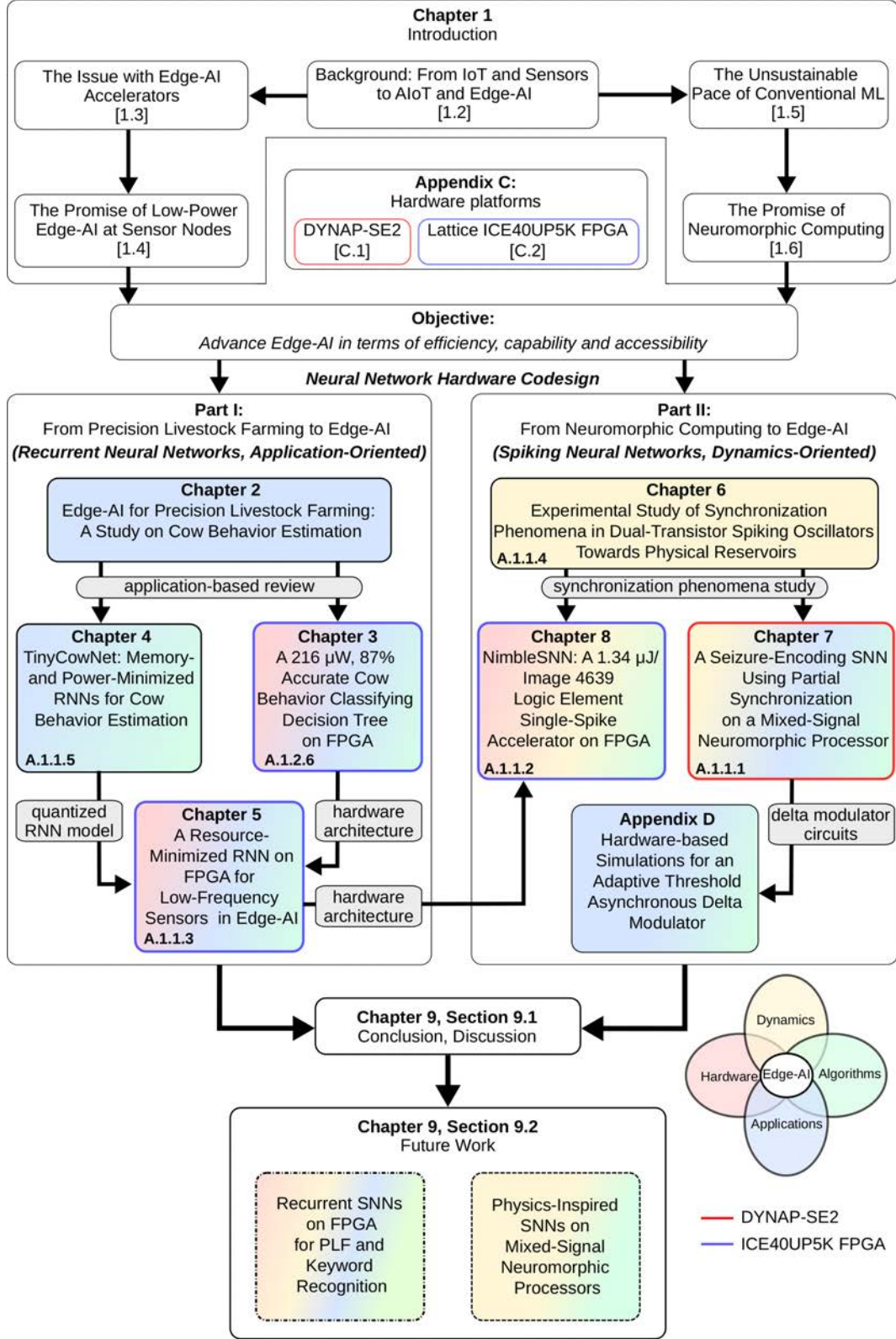


Figure 1.11: Structural overview of this thesis. Color graded boxes indicate what fields the various topics contained in Chapters are related to, colored borders indicate the hardware platform employed.

Part I

From Precision Livestock Farming to Edge-AI via Minimized ML Algorithms and Low-Power Hardware Architectures

Chapter 2

Edge-AI for Precision Livestock Farming: A Study on Cow Behavior Estimation

This Part represents the application-oriented approach of this thesis, namely, application of Edge-AI on what I believe to be one the most important industries for mitigating climate change in the near future, livestock farming. Precision Livestock Farming (PLF) is an upcoming field of agriculture that applies sensory systems and wireless technologies to create real-time monitoring systems for management of production, reproduction, health and welfare of livestock.

This Part starts with an analysis on the impact of livestock on the environment and climate change based on a literature study followed by a short study on the mitigation with an introduction to PLF in Section 2.1. Section 2.2 provides a short review on the existing works that perform livestock behavior classification using cloud-based implementations and a comparison with edge-based implementations. The approaches of this Part with regards to Edge-AI for cow behavior estimation are introduced in Section 2.3 and dataset used throughout the proceeding Sections is introduced in Section 2.4. The results of two approaches to applying Edge-AI for cow behavior estimation is described in Chapter 3 and 4 comprising of a Decision Tree implementation on the Lattice ICE40UP5K FPGA, and a Neural Architecture Search combined with a novel full-integer quantization scheme for a cow behavior distribution regressing Recurrent Neural Network. Finally, Chapter 5 introduces an integer-only resource-minimized RNN hardware architecture, implemented on the ICE40UP5K for low-frequency sensors in Edge-AI.

2.1 Environmental Impact of Livestock

According to a United Nations Food and Agriculture Organization (FAO) report in 2013, 14.5% of all anthropogenic carbon dioxide emissions at 7.5 gigatonnes per annum originate from the livestock industry. Furthermore 65% of livestock industry emissions, which represents 9.6% of total global emissions, arise from cattle farming [176]. More significantly, 44% of all human induced methane, being 100 and 28 times more potent at warming the earth over timespans of 20 and 100 years respectively [177], is emitted from the livestock industry. Once again in this case, cattle is the most significant contributor at 53.4% of all methane emissions from livestock

considering each species, originating from intestinal activities. Next to emissions, the livestock sector contributes to environmental problems worldwide such as fresh water eutrophication, air pollution and over-nutrition of soils [178]. One example of large-scale effects partially from livestock is in the Gulf of Mexico where an overflow of nitrogen from agriculture at and near the exit of the Mississippi river causes a large no-oxygen dead-zone from algae bloom on the surface during a large part of the year [179] (Fig. 2.1). With the fast pace of growth of agriculture and livestock industry in developing countries along with the effects of higher ocean temperatures [180], such dead-zones will increase on a world-wide scale. Environmental problems such as these which are influenced by many factors including global warming are hard to tackle because for one, the affected industry might not be the industry that causes the problem per se and there are many stakeholders involved including enterprise and governments which require multilateral cooperation. In general, the rapid increase of meat consumption in developing countries and simultaneously the pressing necessity to reduce the Greenhouse Gas (GHG) emissions of which livestock farming is a large contributor will require the cooperation of humanity on a global scale. The livestock industry is one of the key areas for humanity to succeed in global sustainability and reduce global carbon dioxide emissions by the 40% aimed for in the Paris climate agreement [181]. This is why I consider this industry one of the most important fields of application of Edge-AI as I hope the reader will discover in the proceeding sections.

2.1.1 Effects of Global Warming

Global warming is an increasingly important and threatening problem in the world. One might argue it is the most threatening challenge humanity has ever faced. The general goal of countries as set out in the Paris agreement is to keep the amount of degrees in temperature well under 2°C and preferably under 1.5°C compared to pre-industrial levels by 2100 [181]. However, currently we are already at a temperature increase of 1.3°C and it is estimated that with current policies and commitments by countries if they are to be followed, there is a 50% chance of the temperature overshooting even 2.1°C by 2100 [182]. The following text will sum up some of the effects for an increase of 2°C of global warming taken from a 2018 United Nations (UN) Intergovernmental Panel on Climate Change report [183].

In oceans, because of ocean temperature increase, 99% of coral reefs (Fig. 2.2) will be lost accompanied with a high risk of irreversible damage to many marine and coastal ecosystems. Other than ocean warming, carbon dioxide increases the acidity of oceans and this reduces calcification of organisms which is an important process for all marine life from shells to plankton and fish. Warming and acidification thus affect the reproducibility and health of marine life, having a large impact on global fisheries expected to reduce the fishing volume by 3 million tonnes a year. Other than marine life, Ref. [184] estimates that by 2100, the amount of category four and five hurricanes will double in frequency resulting in significantly more damage to coastal areas worldwide. On land, 13% of the global land area is expected to change from one climate type to the other. Furthermore, it is estimated that the livable habitat of 18% of insects, 16% of plants and 8% of vertebrates is cut in half. For agriculture, there is a large chance that the crop yield will be reduced for wheat, rice and barley in many regions of the world and the risks for livestock farming is also large with reduction of water availability, onset of more diseases and changes in animal feed quality. These effects are devastating and it is hard to predict what cascading effects



Figure 2.1: Algae blooms lead to large dead zones underneath the surface in the Gulf of Mexico.

will follow from such events biologically but also in societies around the world. A recent work has shown empirical evidence of food insecurity from climate change leading to the risk of societal collapse [185]. We can also look back at the Akkadian Empire among other examples such as the Ming Dynasty [186] and even recently in Syria [187] where on a small scale, climate change effects contributed to the partial collapse of society. Luckily, for a mere reduction of 2°C to 1.5°C Celsius, many of the above effects can be significantly reduced. In order for this to succeed, it is important for livestock farming to become more sustainable in order to reduce its carbon footprint. In any case, it is my belief that far more has to be done by governments worldwide including Japan to reduce the risk of major catastrophe.



Figure 2.2: Coral reef in Okinawa, Japan, one of the marine ecosystems that will suffer severely and possibly completely disappear because of global warming (CC license, from Okinawa Institute of Science and Technology (OIST) [188]).

2.1.2 Mitigating the Impact of Livestock

Mitigation methods that reduce the impact of livestock on GHG emissions and environment are split mostly in two by focusing on one of either sides in the industry; consumption or production.

In 2013, the average meat consumption in the world was at 43.22 kg per person per year. Fig. 2.3 shows the average meat consumption for different countries and regions of the world plotted from FAO statistics [189]. Confirmed also by [190], being a western or western aligned country is one of the main indicators for high meat consumption per capita (Australia, USA, EU and Latin America). On the other hand, there is the influence of GDP per capita which can be seen from the difference in meat consumption of Japan and the rest of Asia and Africa. Other than having a large impact on the world as described before, high meat consumption in many western countries cause health problems leading to higher risks of cardiovascular diseases and several types of cancer. One of the ways to mitigate livestock impact is to reduce this overall consumption. As such, Ref. [191] shows that if the consumption of meat but also dairy products and eggs were to be halved in the EU, at most a 40% reduction of Nitrogen and GHG gas emissions of livestock is achievable. Furthermore, the amount of land used for crops would be reduced by 23% which is important in countries such as the Netherlands where there is lack of build-able land. Although consumption of such products are decreasing in the EU currently, there is another solution to substituting proteins of livestock. This in the form of insect protein which

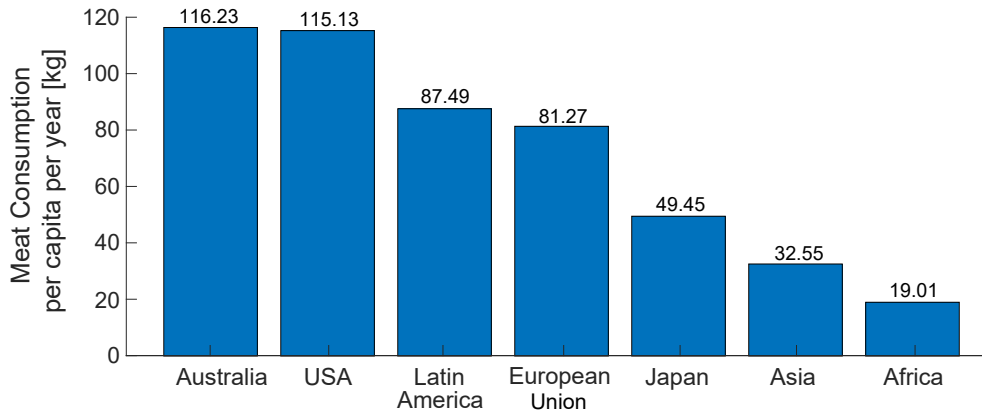


Figure 2.3: Meat consumption per capita for three regions on Earth and the nations including Japan, USA and Australia as well as nations in the European Union.

is more nature-friendly because of its lower need of land and water and less GHG emission [192]. With the use of by-products of agriculture to grow such insects, insect protein can be 2-5 times more environmentally beneficial than meat [193]. However, the consumption of insects is at this moment still a social barrier for humans and it is unclear what the effects are on the human diet and health.

Other than consumption, insect larvae can also be used as feed production for livestock systems which can be produced at low costs and locally on small farms which serves to decrease the environmental impact of livestock feeding systems but is held back by legislative and technological challenges [194]. In the farm itself there are many studies that look at reducing the impact of livestock. Reference [195] concludes that non-technological solutions such as the increase of manure removal, not letting livestock graze on wet grass and additives in feed have the potential of severely reducing the carbon dioxide and methane of animals on livestock farms. However, these strategies only look at increasing productivity of animals on farms which has large potential in developing countries but is already high in developed countries. Furthermore, Ref. [196] argues that the intraspecies diversification of livestock animals and changing to mixed-crop feed systems has the largest potential to mitigate the impact of livestock.

2.1.3 Precision Livestock Farming

The above studies do not consider in depth the potential reduction of environmental impact from the application of recent technological advances in sensory systems and Artificial Intelligence (AI). Sensory systems have become far smaller with significantly lower energy and power consumption with the effects of Moore's law and beyond Moore's law technology in circuits and digital systems as described in detail in previous Chapters. This makes it possible for small sensors in agriculture that can gather large amount of data with considerable device lifetime and the Internet of Things to be developed.

Combining IoT and AI and other advances in technology, Precision Livestock Farming (PLF) was born that offers real-time monitoring systems for management of production, reproduction, health and welfare of livestock. There are multiple reasons this field was established; the necessity to improve efficiency and the increase of livestock per farmer which causes individual monitoring of livestock to become

more challenging [197, 198]. Therefore, reducing the environmental impact and climate change mitigation are not its primary goals. However, Ref. [199] shows that increasing the productivity, reproduction and health maintenance of livestock with PLF contributes to the reduction of methane, ammonia, nitrogen and antibiotics in the environment. As a result, this field is very promising to reduce the GHG emissions and environmental problems that are caused by livestock of which ruminants such as cattle are the dominant contributors. Also, with larger herds of animals per farmer, the monitoring of livestock for behavior, disease detection but also farm management will become more complex and difficult (Fig. 2.4). Similar to a high population density of humans, a disease can spread more easily and cause larger damage in such farm environments. Therefore, it is important that the farmer can recognize disease of livestock fast to prevent overuse of antibiotics, reducing impact on the environment. One such example is early disease detection for cows (mastitis) from milk production using naive Bayes and random forest (RF) algorithms [200] that potentially leads to this reduction of antibiotic substances in the environment. Furthermore, according to agricultural and veterinarian experts, improvement of animal welfare also positively contributes to United Nations Sustainable Development Goals (SDG) and half of the SDGs are related to how humans produce or consume food [201, 202]. At the time of writing, large-scale adoption of PLF is still lacking because of the barriers for farmers i.e. large initial investment costs and low reliability of monitoring systems [203].

2.2 PLF with ML: a Short Review

2.2.1 Anomaly Detection: Disease and Fertility of Cattle

As stated before, accurate estimation of animal behavior enables the monitoring of large herds for early signs of disease, health and reproduction which prevents mortal cases of disease and antibiotic treatment and improves the efficiency and sustainability of livestock farms [199]. There is previous work that show that they can specifically estimate a single disease or fertility by applying ML. Figure 2.5a shows the composition of such systems which require labeling of anomalies that involve diseases and rare events such as oestrus. This is performed by diagnosing the anomaly with traditional methods such as visual observation. This labeled raw data is then used to extract features and apply these to a learning algorithm. Below is a short review of previous works that have applied such methodology to detect anomalies in livestock.

Reference [205] conditioned cows to a metabolic disease called Subacute Ruminant Acidosis (SRA) and showed that K-Nearest Neighbor algorithm is able to detect abnormal behavior when this disease onsets. The raw data that they used was the location of cows near specific parts of the barn such as feeding and drinking areas. Furthermore, pH value sensors were used to detect when SRA onsets. More specifically, Ref. [206] showed a Decision Tree (DT) applied to the classification of sub-clinical Mastitis, a disease that effects the udder, and reported an accuracy of 85.9% by using features related to the milking rate of cows. For lameness detection which affects cow movement, the majority of studies have focused on using the lying and standing behaviors and gait features to achieve high detection accuracy [207] along with camera footage. As for fertility, Ref. [208] applied Principal Component Analysis with labeled time of behaviors such as walking, feeding, drinking and lying

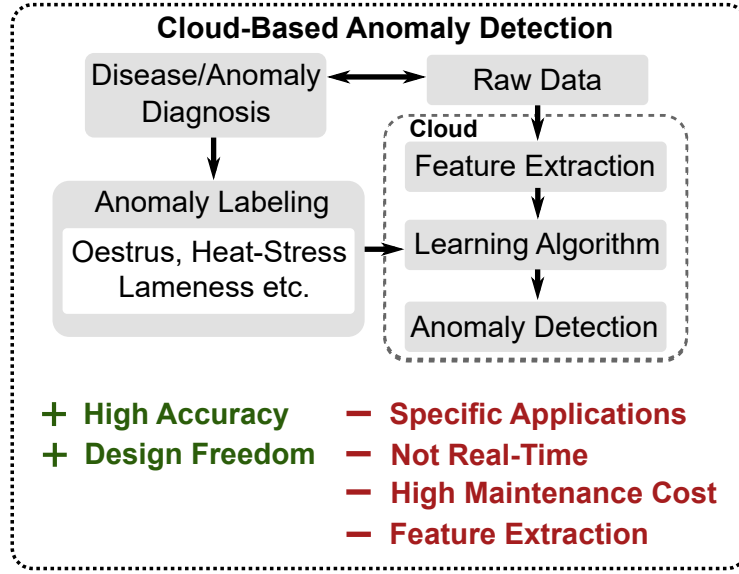


Figure 2.4: Large herds of cattle are harder to monitor for individual behavior and diseases (CC license, [204])

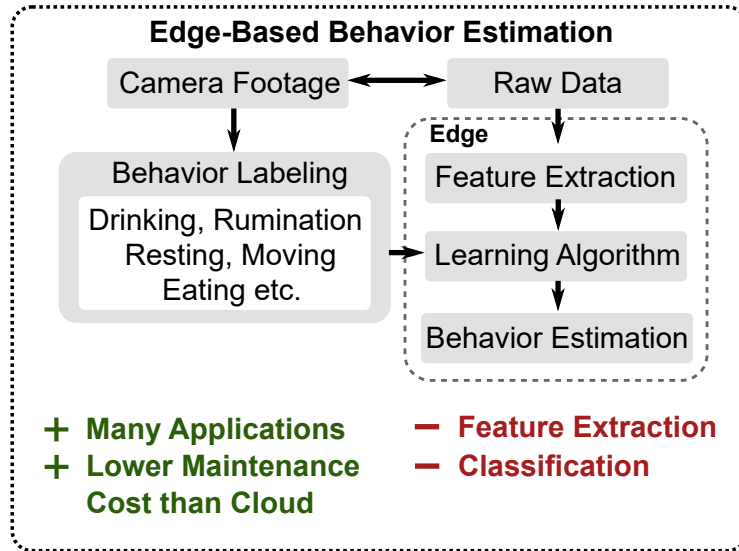
to detect estrus of cows up to an accuracy of 95.4% using Multi-Layer Perceptrons (MLP) for half hour intervals. Oestrus anomalies were diagnosed by using a video observation scoring system that accumulated scored for behaviors such as mounting, sniffing and head resting on other cows. Raw data location and acceleration of each cow was gathered for a frequency of 1Hz. Another type of anomaly frequent in countries such as Japan that effects milk output and health is heat-stress, often determined from temperature and humidity. Reference [209] used an MLP and DT to estimate the skin temperature and respiration rate from heat stress indicators; air temperature, relative humidity and solar radiation. They reported a root mean squared error of 9.695 for the estimated respiration per minute and a 0.335 °C error for estimated skin temperature.

These works focus on detecting specific complex behaviors and events that are essential for maintaining animal welfare at high accuracies. Furthermore, their ML algorithms are implemented on the cloud that results in far higher maintenance cost from frequent raw data transmissions. Also, such cloud based systems have higher latency since the data has to be transmitted over long distances and processed along the pipeline. On the other hand, since on software there is a large amount of library support with platforms such as Tensorflow and Python, the design freedom is large.

Instead of narrowing down on specific applications, I set out to first obtain significantly high accuracy (>90%) to estimate cow behavior i.e eating, moving, resting and rumination in this Chapter as the main application-oriented work of this thesis. Seeing that these are four of the most frequent behaviors of cows and include feeding, I think that the detection of any of the above anomalies such as mastitis and estrus is possible since they all affect certain change in these four behaviors. In such cases, the cow that is diagnosed with a disease could be used to build a new dataset for the development of a small additional pipeline such as a Decision Tree to classify anomalies from the estimated behaviors. Moreover, these estimated four behaviors can be used in different behavioral studies for animal scientists. Therefore this defined system provides for more generalization and future applications.



(a)



(b)

Figure 2.5: Diagram describing two identified types of Precision Livestock Farming systems with Machine Learning in literature.

2.2.2 Edge-Based Behavior Detection Systems

Compared to the enormous amount of literature that focuses on different applications with cloud ML implementation with anomaly detection that was described above, the number of works that concerns Edge-Based ML implementations is severely limited. Figure 2.5b shows the structure of such Edge-Based Behavior Detection systems that are in line with the central approach within this thesis. As described throughout this thesis, Edge systems have a distinct advantage as raw data is transformed to data with higher information content and lower dimensionality. In this case, this describes the number of ML-classified behaviors and therefore allows for significant power consumption reduction because of far lower transmission rates. This lower transmission rate enables Low Power Wide Area (LPWA) technology and

thus larger range than Wi-Fi and Bluetooth. Furthermore, the latency is reduced because the on-device inferred behaviors can be directly sent to end-users such as farmers or health experts. The below describes a short literature review of previous work in this field of edge-based animal behavior estimation using ML.

Reference [210] showed K-Nearest Neighbor (KNN), requiring a large set of points in memory on an ARM Cortex Processor leading to a 96.2 % accuracy for the binary classification of high or low activity of two sows for every 64 samples (1.28s) with 0.542 mJ energy consumption per inference. Similarly, Ref. [211] used heuristic classification, which requires intensive statistical analysis of raw sensor data, to achieve a 90% precision for the classification of three different kinds of lying combined with high and low activities for sows using an FPGA. Although these works show successful Edge-AI systems for sows, the prescribed behaviors do not include any feeding behavior.

On the other hand, there is more recent research of Edge-AI applied to sheep, cattle and horse behavior classification. Reference [212] used a 30 neuron hidden layer Multi-Layer Perceptron (MLP) to achieve a precision of 80.77% for the classification of three horse behaviors; motionless, walking and trotting of horses. This was implemented on a Microelectronic Nucleo Micro Controller Unit (MCU) using the Fast Artificial Neural Network library [213]. A later work, Ref. [214], shows an accuracy of 98.34% on the same three behaviors with an energy consumption of 185.76 mJ using a 163kB, three layer MLP floating point model with Tensorflow Lite [215] on a similar MCU. Reference [216] measured energy consumption on a low power Texas Instruments (MSP430FR5739) MCU for different feature extraction and ML algorithms such as Support Vector Machine, Decision Tree (DT), Random Forest and KNN. With these measurements, different number of features and ML algorithms were combined in a cost function that optimized energy consumption and accuracy. Using Linear Regression with 34 different features, they were able to classify upto five behaviors of sheep including grazing at an accuracy and energy consumption of 89.59% and 2050 μ J respectively. As for cattle-based systems, in Ref. [217], a single-layer MLP network was shown that achieves a Matthews Correlation Coefficient (MCC) value of 0.9340 with leave-one-animal-out cross folding for the classification of four cow behaviors including grazing and rumination from neck-attached tri-axial accelerometers. This MLP with feature extraction was estimated to take a total of 2288 bytes if implemented on an embedded device. This MCC is a good measure next to F1-score for imbalanced dataset classification problems [218].

Table 2.1 shows more details about each of these works such as the window size in samples, the sampling frequency, sensor attachment location etc. Only a single previous work describes a system applied to cow behavior which is also really recent (2021). Furthermore, all works implement either MLP or other learning algorithms that require feature extraction beforehand. This feature extraction has been well documented but it biases data processing with human intuition due to expertise requirements for which every expert can have a different opinion. Furthermore, it is also not known whether averaging which is done for each window is the best way of raw data processing for this application. Also, the lowest energy consumption is at 542 μ J for classification of only two behaviors of sows which do not include any feeding behavior. And, Ref. [217] does not show any energy consumption so it is unknown how far the energy consumption and accuracy can be optimized for edge applications.

Table 2.1: Details of a literature survey of Animal Behavior Classification with ML intended to be implemented on hardware itself (Edge-AI) using tri-axial accelerometers.

Reference	[210] 2018	[211] 2011	[212] 2018	[214] 2021	[216] 2019	[217] 2021
Application	Sow	Sow	Horse	Horse	Sheep	Cow
Sensor position	Ear	Ear	Jaw	Jaw	Neck	Neck
Datasize (hh:mm)	1:25	10:40	00:17	00:13	16:11	01:30
Sample Frequency (Hz)	50	4	30	33	20	50
Window size in samples	64	240	NR	NR	128	500
Algorithm	KNN	Heuristic	MLP	MLP	DT,KNN	MLP,DT
Behaviors	2	3	3	3	6	4
Device	MCU	FPGA	MCU	MCU	MCU	
Features	3	2	3	3	34	3
Value	96.2%*	90% [†]	80.8% [†]	98.3%*	89.6%*	0.934 ^{††}
Energy consumption	597 μ J	N/R	N/R	185.76 mJ	2050 μ J	N/R

* Accuracy

[†] Precision

^{††} Matthews Correlation Coefficient

2.3 Approaches in this Part

In Part I, I investigate training Machine Learning algorithms on the task of cow behavior estimation that can be implemented on extremely resource-constrained FPGAs and their implementations. In this case, I have used the Lattice ICE40UP5K [219]. As also described in C.2.1, This FPGA typically costs around 10 US Dollars which is very low end for an FPGA, containing only 5280 Look Up Tables and registers, and a usable on-chip memory of 1 Mb. This part have been divided in three Chapters, each representing a different approach. These two approaches were performed on different timelines during this study, namely, to try and reach the target of reaching higher than 90% accuracy while going below previous studies in terms of power consumption and efficiency in terms of energy per inference.

2.3.1 Bottom-Up: Decision Trees

The following Chapter, Chapter 3, begins with a bottom-up approach. I argue that in order to create a device that has a long lifetime and can estimate cow behavior well, one of the lowest complexity ML algorithms, the Decision Tree (DT), needs to be analyzed and combined with a minimal amount of features for a high accuracy. Therefore, this Chapter first of all focuses on discovering the limitations of estimating cow behavior on FPGA with a DT for cow behavior applications. In this way, this part considers hardware resources as a starting point, therefore proposing a machine learning algorithm and features with minimal complexity.

2.3.2 Top-Down: Searching for a Weight-Minimized Recurrent Neural Network and Integer Quantization

The second approach, described in Chapter 4, is represented as top-down: starting with a complex algorithm that is so far not represented on extremely resource-constrained hardware, Recurrent Neural Networks (RNN), and finding a network structure that can be implemented. Combined with this, the shortcomings of classification algorithms are exposed which represent significant error in the estimation process for mixed behavior sample spaces. In this way, Chapter 4 describes RNNs applied to cow behavior distribution regression, involving the transformation of class labels into distribution vectors, erasing the error from classification. As part of this approach, the architecture of these RNNs are optimized in terms of memory and accuracy using hyperparameter optimization with a random-search. Furthermore, a new full-integer quantization method is proposed for RNNs that is applied post-training. Subsequently, the memory use is considered to propose multiple embedded platforms for implementation. As a result of the above study, I have discovered an RNN, "TinyCowNet", that has the optimal size and accuracy that can be implemented on the Lattice FPGA described prior.

2.3.3 A Resource-Minimized RNN Hardware Architecture for Low-Frequency Sensors

The culmination of the above approaches is expressed in Chapter 5 where a resource-minimized RNN hardware architecture for low-frequency sensors in Edge-AI is described followed by its implementation on the Lattice ICE40UP5K. This hardware architecture and implementation serves as a general purpose study, namely, not only is it meant for PLF applications but for any Edge-AI applications where energy budget is extremely low that use low-frequency sensors. Moreover, the RNN, "TinyCowNet", that is described in Chapter 4 is also implemented using this hardware architecture, leading to the most efficient PLF implementation for cow behavior monitoring at the time of writing. The implementation of this network, the code describing the implementation on FPGA in Systemverilog and Tensorflow conversion code is available online [84]. I highly recommend the reader to use this code to implement your own PLF solution or any IoT solution with AI where a low-cost platform is essential and low energy consumption critical.

Other than lower energy consumption at 557 μJ per behavior compared to literature, the Decision Tree on FPGA reaches the target power at 216 μW however lacks in accuracy at 86.8%. The methods described in Chapter 4 realize a full-integer quantized RNN model with only 2092 bytes of memory. This model is implemented on FPGA using the RNN implementation described in Chapter 5 and measured using the experimental setup described in Section C.2.1. With an accuracy of 95.1% on the 4 behaviors of feeding, moving, resting and rumination at a measured power consumption of 360 μW at an operation frequency of 146 kHz, this top-down approach leads to the best accuracy at a slightly higher power consumption than the DT. More importantly, setting the operating frequency at the maximum of 12 MHz, "TinyCowNet" implemented on the FPGA consumes only 39.5 μJ per inference. According to my knowledge, the above two works represent some of the most efficient animal behavior estimation implementations using Edge-AI on any commercial hardware in the known literature at the date of writing.

2.4 Japanese Black Beef Cow Behavior Dataset

In this section, the pre-labeled dataset that is used for the training and validation of the proposed neural network is described. It involved the gathering of 16-bit, $\pm 2\text{G}$ tri-axis accelerometer data using KX126-1063 accelerometers [220] which were attached to the neck of six different black beef cows at a cow farm of Shinshu University in Nagano, Japan. During a period of two days, these cows were observed in two locations, farm pens and a grass field while being filmed with Sony FDR-X3000 4K video cameras [221]. Hereafter, the timestamps of the video and accelerometer data were matched while human observers which included behavior experts and non-experts labeled the data from the video footage (Fig. 2.6). The gathering of this data with these cows was reviewed and approved by the Institutional Animal Care and Use Committee of Shinshu University.

Fig. 2.7 shows screen captures of each of the four behaviors in one of the two described locations and an illustration that shows the location of the neck-attached accelerometer and its axes. Next to the screen captures, Table 2.2 shows the exact description of each of the labeled behaviors in the dataset. In the pens with muddy floors, there was access to water and hay, and the behaviors were primarily characterized by walking, ruminating and resting. On the other hand in the grass fields, where the cattle was allowed to graze freely, behavior was primarily observed as grazing and running. In total thirteen different behaviors were labeled for 197 minutes of data, 94.4% of this data represented the behaviors of resting (RES), rumination (RUM), eating (EAT) and moving (MOV). The 5.6% that represented all other behaviors were removed. This is because removing such small part of the dataset only has a small impact on the overall time structure and it considerably reduces the imbalance of the data for this dataset.

Table 2.2: The exact description of each of the labeled data and their respective percentage as part of the overall data set.

Labeled Behavior.	Description	Samples (%)
Resting (RES)	resting while standing up	150130 (50.78%)
Rumination (RUM)	ruminating while standing up	53229 (18.00%)
Moving (MOV)	walking or running	50199 (16.98%)
Eating (EAT)	eating hay at stanchion or grazing	25547 (8.64 %)
Others (Removed)	salt licking, mounting, drinking	16569 (5.60%)

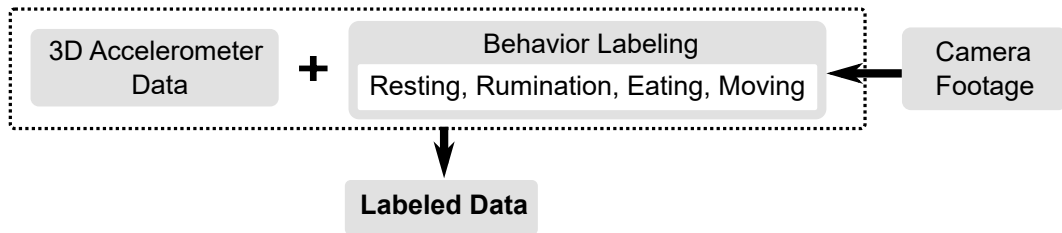


Figure 2.6: Method of transforming camera footage with raw accelerometer data to labeled data used for ML algorithms.

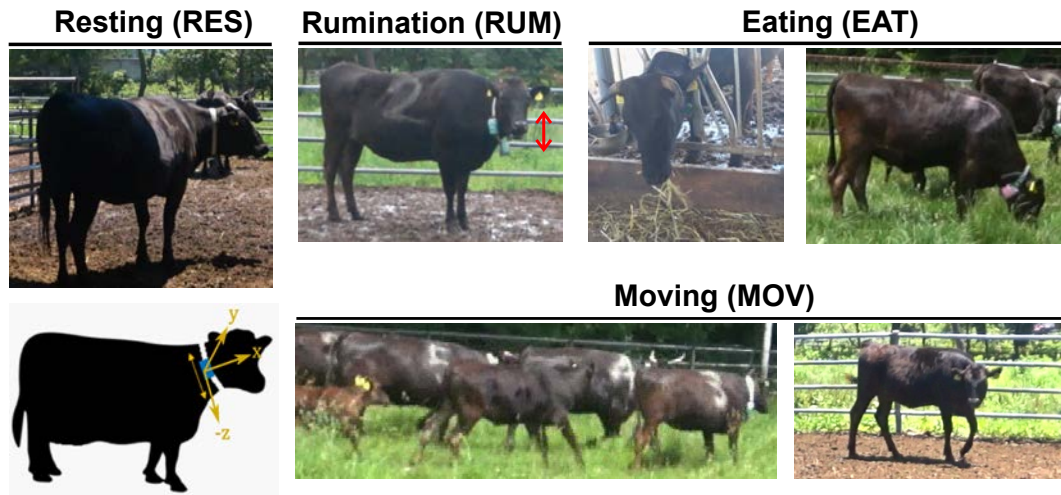


Figure 2.7: Screen captures of recordings that were used to label cow behavior to tri-axial accelerometer data gathered from accelerometers attached to the neck of each of six cows.

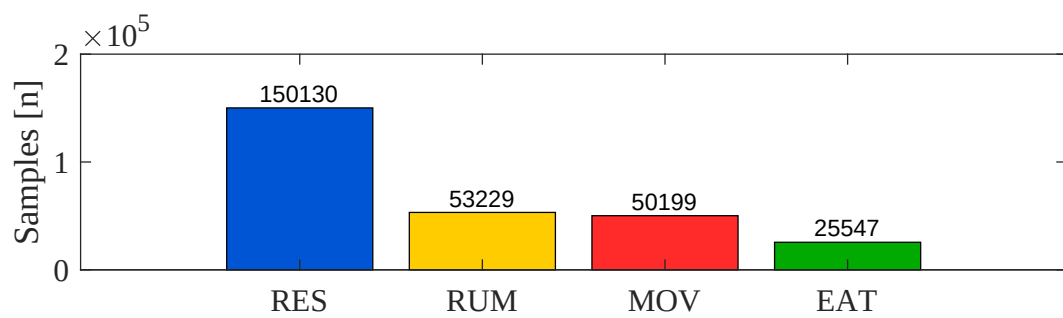


Figure 2.8: Cow behavior dataset with the amount of samples for each human-labeled behavior.

Fig. 2.8 shows the imbalance of the reduced dataset at a distribution of roughly 6:3:3:1, which means that there are roughly six times more resting labels than eating labels in the data set. Multiple ways to solve this imbalance exist such as oversampling minority classes or downsampling majority classes [222]. Specifically applied to cow behavior estimation, [223] uses random rotation of the sensor around the cow's neck to generate additional data, showing accuracy increase range of 2.52-37.05% on Long-Short Term Memory networks. Application of data extension methods is outside the scope of this Thesis.

Chapter 3

A 216 μ W, 87% Accurate Cow Behavior Classifying Decision Tree on FPGA With Interpolated Arctan2

3.1 ML Algorithm and Feature Set Considerations for Hardware

The selection of features and ML algorithm has a large influence on the power consumption and performance of a classification system in hardware. As regards the complexity of ML algorithms, Neural Networks (NN) in hardware extensively use multiply-accumulate operations and therefore have a high power consumption [224]. Moreover, Naive Bayes algorithms require a large number of Look Up Tables (LUT) to determine exponential and logarithmic operations [225]. As the Bayes equation itself is multiplicative, it requires constant multiplications. Besides that, Hidden Markov Models (HMM) require a high amount of states and resources for state calculation [226]. Regarding cow behavior classification, [227] showed the Decision Tree (DT) to be the most accurate among HMM and 2 other ML algorithms. Concerning DTs, decision rules are realizable with functions of any order n . In this case, the 0-th order axis-parallel DT (only constant decision rules) is selected. According to [228], this usually results in bigger trees but requires no multiplications. Also, the Universal Node hardware architecture, which they show to be the lowest power DT architecture is adapted. These rules then only have to be stored in memory and used in a subtraction-bit comparison operation.

Regarding DT creation, the sensor data from the six different cows is split up in a stratified way with 80% for training using Gini index as splitting criterion and 20% for testing whereafter the trained DTs decision rules represent memory in hardware. This Gini index describes the probability that a wrong behavior is chosen in the case of a random selection. Mathematically it is described by the following

$$Gini = 1 - \sum_{n=1}^N (p_n)^2 \quad (3.1)$$

where N is the amount of labels or behaviors which is 4 and p_n is the probability in the case of a random selection. To give an example, if a distribution of 6:3:3:1 is

considered i.e. 6 labels RES, 3 labels RUS and MOV, 1 label EAT, the Gini index has a value of

$$Gini_{distr} = 1 - ((\frac{6}{12})^2 + 2(\frac{3}{12})^2 + (\frac{1}{12})^2) = 0.7456 \quad (3.2)$$

where each of the labels is assumed randomly selected with equal probability. This Gini index is used in the Classification And Regression Trees (CART) algorithm [229] which makes binary Decision Trees, trees that split into two nodes. This CART algorithm can be formulated as the following optimization problem

$$\underset{\mathbf{D}}{argmin}(L(N)) \quad (3.3)$$

where $\mathbf{D}$ are all the decision rules for each node of the Decision Tree and N is the total amount of samples and their respective behavior categories of the training dataset. In this way the decision rules have to be chosen so that the loss function L is minimized. These decision rules are decided by selecting values from the set of features

$$\mathbf{FS} = \{f_1 \dots f_n \dots f_M\} \quad (3.4)$$

where f_1, f_n, f_M represent all the features, real numbers within a range. Each decision rule is decided from the root node to the leaf by assigning a decision criterion

$$D_i = (f_n == \alpha_i) \quad (3.5)$$

where α_i is an arbitrary value that results in the minimum loss for each split. This loss $L(N)$ is then presented as

$$L(N) = \frac{n_l}{N} Gini(f_n \neq \alpha_i) + \frac{n_r}{N} Gini(f_n == \alpha_i) \quad (3.6)$$

where n_l and n_r are the amount of samples on the right and left side of the binary split. The CART algorithm determines the minimum of this loss function by evaluating it for the total range of each feature in feature set $\mathbf{FS}$. When this minimum loss is found it decides the decision rule as in eq. (3.5) and moves to find next optimal split. This continues until no more improvement of this Gini index is possible for another split or if the maximum depth limit is reached that is set manually by the designer. This maximum depth makes sure that overfitting is prevented since too deep trees perform well on the training dataset but underperform on the test dataset. Therefore it is very important to have a suitable feature set that can minimize this loss function well i.e. split the training dataset into respective behaviors in terms of amount of nodes (memory and computation) but also accuracy which involves the Gini index being zero at all leafs.

Conventional features in cow behavior monitoring with 3D accelerometers are such as the static horizontal axis, the vector magnitude and area, the spectral energy and entropy [227, 230–232]. However, none of these consider the horizontal-longitudinal (y-x) acceleration angle with the two argument arctangent. This feature is calculated by

$$\arctan2(y, x) = \begin{cases} \arctan(y/x), & \text{if } x > 0 \\ \arctan(y/x) + \pi, & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan(y/x) - \pi, & \text{if } x < 0 \text{ and } y < 0 \\ +\pi/2, & \text{if } x = 0 \text{ and } y > 0 \\ -\pi/2, & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined}, & \text{if } x = 0 \text{ and } y = 0 \end{cases} \quad (3.7)$$

Table 3.1: 3-axis accelerometer based feature sets

Feature set	Data representation	
	Mean magnitude (a_n)	Mean variation (a_{var})
FS₁	$\arctan2(y, x), z$	$\arctan2(y, x), z$
FS₂	$\arctan2(y, x), z$	z
FS₃	$\arctan(y/x), z$	z
FS₄	$\sqrt{x^2 + y^2 + z^2}$	$\sqrt{x^2 + y^2 + z^2}$
FS₅	$ x + y + z $	$ x + y + z $

Table 3.2: Approximation of the two argument arctangent considering percentage of dataset and machine learning application

Boundaries	%Dataset	Theoretical	Approximation
$0 < \frac{y}{x} < 1$	1.80	$\arctan(\frac{y}{x}), \frac{\pi y}{4x}$	$\frac{\pi}{8}$
$\frac{y}{x} > 1, x > 0$	19.69	$\arctan(\frac{y}{x})$	$k_n(n = 1.....15)^*$
$\frac{y}{x} < 0, x > 0$	0.41	$\arctan(\frac{y}{x})$	$-\frac{\pi}{8}$
$x < 0, y \geq 0$	73.48	$\arctan(\frac{y}{x}) + \pi$	$-k_n(n = 1.....15)^* + \pi$
$x < 0, y < 0$	0.29	$\arctan(\frac{y}{x}) - \pi$	$-\frac{7\pi}{8}$
$x = 0, y > 0$	4.27	$\frac{\pi}{2}$	$\frac{\pi}{2}$
$x = 0, y < 0$	0.01	$-\frac{\pi}{2}$	$-\frac{\pi}{2}$
$x = 0, y = 0$	0	undefined	undefined

* $\mathbf{k} = [0.78, 1.11, 1.25, 1.328, 1.375, 1.406, 1.422, 1.453, 1.469, 1.484, 1.500, 1.516, 1.531, 1.547, 1.563]$, n is calculated by $\text{argmin}(\pm \mathbf{i} - \frac{y}{x})$, $\mathbf{i} = [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 14, 19, 30, 55, 70]$

where $\arctan(y/x)$ is the inverse of the tangent which calculates angle of the opposite, x , and adjacent side, y , of a triangle. This function is a non-linear function and needs some type of approximation for hardware implementation. Table 3.1 shows details of the five feature sets I have considered to train a DT where feature sets **FS₁** – **FS₃** include novelty of the arctangent and **FS₄** and **FS₅** represent the conventional. Each feature’s magnitude and variation (absolute difference for two timesteps) is windowed i.e. averaged for every 64 samples, which acts as an Low Pass Filter [233] and decreases the data size by 98.44%.

Other than data reduction, 87.5% memory size decrease is achieved by reducing 64 floating point (software) to 8-bit fixed point. This results in a quantization error of $\epsilon \approx 0.01563$ G for an acceleration range of ± 2 G. Furthermore, Table 3.2, which describes the proposed approximation of the two argument arctangent, $\arctan2(y, x)$, reduces the function to a binary search lookup which runs in logarithmic time. In the case of look-up vector $\mathbf{i}$, this takes 57.78% less time compared to linear search. Figure 3.1 shows the approximation graphically, regions that represent under 1.8% of the overall dataset are assigned a constant value (region 0,2,4). For the left-over regions, region 1 at 73.48% and region 1 at 19.69%, this binary search vector is

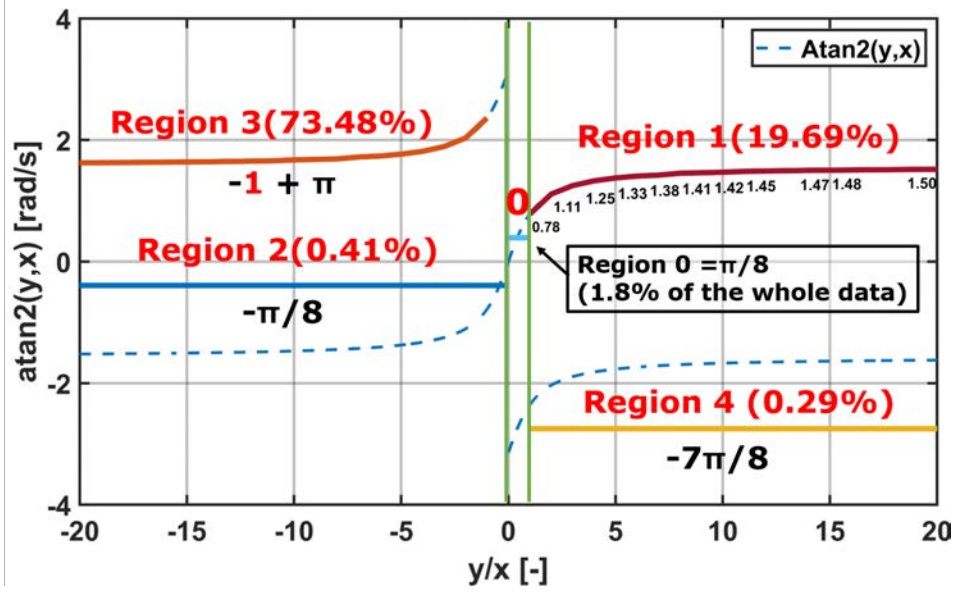


Figure 3.1: 2 argument arctangent approximation using binary search look up table and assignment of constants to regions with small portion of the overall dataset.

either applied directly (region 1) or transformed as in the negative vector added with pi (region 3). This approximation was carefully tuned by considering the DT's performance while assigning constants to data regions. These constant data regions had negligible effect on the accuracy performance of the DT.

3.2 FPGA Measurement Setup

Figure 3.2 shows the setup constructed to compare the DT hardware implementation on the Lattice ICE40UP5K FPGA, that is used for the DT implementation, for each FS with software. After Arduino stores 64 $\{x, y, z\}$ samples from Matlab, the I2C master module in the FPGA is triggered by the reset, requesting all this stored data. Upon reception, the FPGA responds with a 2-bit classified cow behavior. Comparing this system to a cloud-based system, the FPGA serves as a data reducer. For a cloud-based system, 64 samples of tri-axial accelerometer each at 8 bit has to be transmitted to the cloud for ML inference ($64 \times 3 \times 8$). In the case of this Edge-AI system, the FPGA responds only with a 2-bit quantized output which is sufficient to distinguish between four different behaviors. In this way, this system achieves up to a 99.84% or a 768-fold reduction in transmission data. Therefore, this system easily achieves significant power reduction compared to all cloud-based systems. First, because of the utilization of a DT with approximation and only small amount of features. Second, because of implementation on the Edge on a low-power FPGA.

3.3 Proposed Decision Tree Architecture

Figure 3.3 shows the VLSI implementable architecture of each feature set described in Table 3.1. A custom implementation is developed for each **FS**, which changes the number of addition and memory blocks for the data representations as well as the algorithm of the main features. The previously discussed 64 sample average

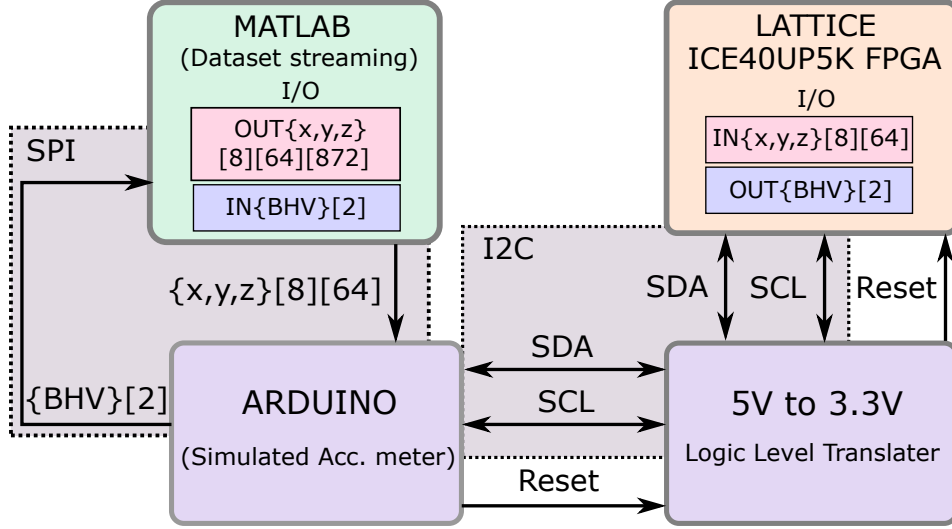


Figure 3.2: Setup used to send accelerometer data and receive classified behavior from Matlab on PC to the implementation on FPGA

and approximated $\arctan2(y, x)$ are applied. Depending on the **FS**, four addition-accumulate (AA) blocks for magnitude and variation calculation, indicated by yellow, blue, green and red are used and mapped on FPGA. After accumulation of 64 samples, the magnitude and variation are divided by 64 and 2 respectively using shift operations. This division by 2 prevents further rounding error since the variation has a relatively small range. The DT module, which contains all the tree nodes in memory, selects an 8-bit feature F with a 2-bit wire, called Select Feature (SF). Repeatedly, a 1-bit wire, $RL = F \leq T$, selects a new node in the tree, which contains SF and decision rule T until a leaf is reached. The behavior stored in this leaf is then forwarded to the I2C module and sent to the peripheral. In consideration of the 25 Hz accelerometer, the clock frequency for the system is set to 2.9 kHz for the I2C to communicate on time. Because the cycle count of the combinational logic for each **FS** is smaller than an I2C message, its frequency is also set to 2.9 kHz, causing dynamic power to be small relative to static power. The resource use of the FPGA implementation of **FS**₂ are shown in Table 3.3. Using 3 AA blocks and 22 tree nodes amounts to 11.31% and 2.25% of the logic elements (LUT4) and 2.75% and 0.66% of the registers for the Feature Extractor and Decision Tree modules respectively.

Table 3.3: Resource allocation for FS2 on Lattice ICE40UP5K FPGA

Module	#LUT4	%FPGA*	#REG	%FPGA [†]
Control	270	5.11	70	1.33
Decision Tree	119	2.25	35	0.66
Feature Extractor	597	11.31	145	2.75
I2C	247	4.68	114	2.16
Total	1222	23.14	364	6.89

*Total LUT4: 5280

[†]Total Registers: 5280

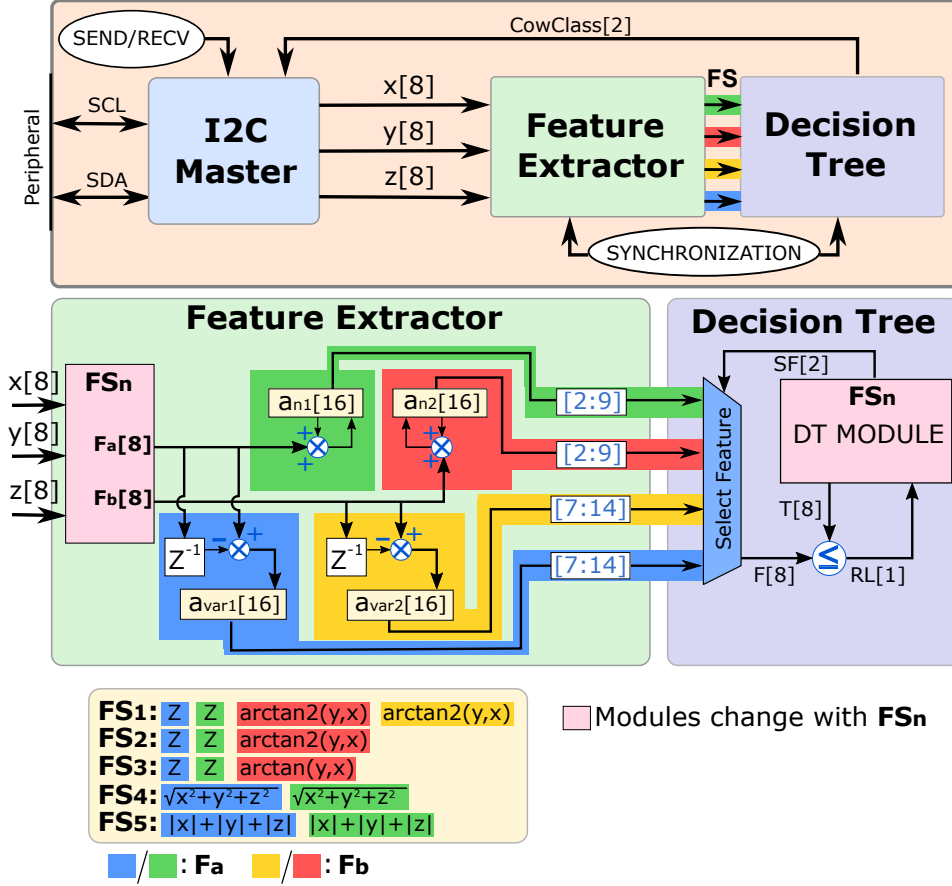


Figure 3.3: Top level view of the in FPGA implemented hardware architecture for each feature set, addition-accumulate block usage is indicated with colors and bit size with brackets, the blue brackets represent a shift operation and reduction to 8-bit where index 0 indicates the MSB.

3.4 Hardware-Software Comparison

Figure 3.4 shows the macro F1 score (the harmonic mean of the precision and recall for each behavior) of four behavior classification obtained from software and the 8-bit fixed point implementations in hardware with the previously stated architecture, setup and conditions. This F1 score is described as

$$F1 = 2 \left(\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \right) \quad (3.8)$$

where precision and recall are m

$$\text{precision} = \frac{tp}{fp + tp} \quad (3.9)$$

$$\text{recall} = \frac{tp}{fn + tp} \quad (3.10)$$

where precision is an indicator for how many of the predicted labels are correct and recall is an indicator of how many of a certain behavior are correctly identified. This relates to the misclassification of one behavior into another. Therefore the F1 score

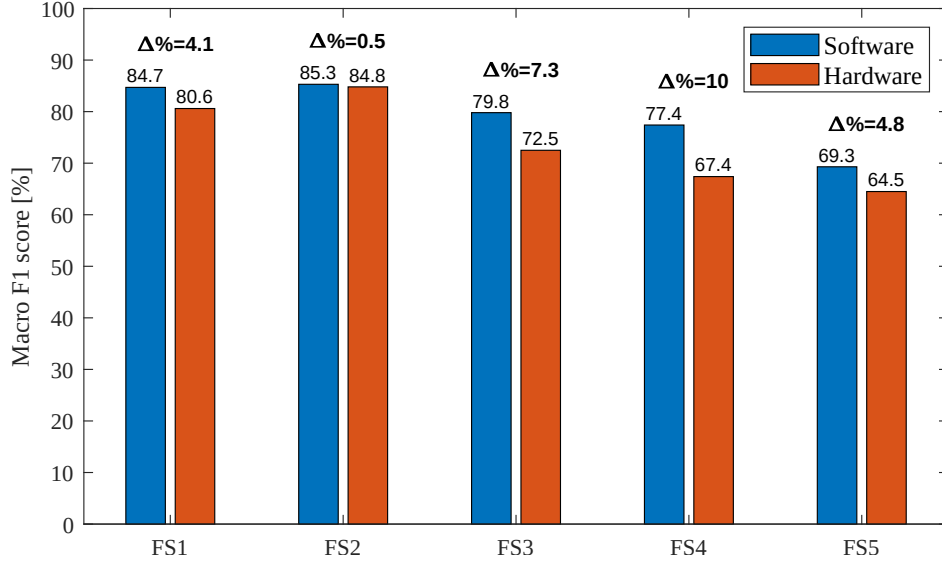


Figure 3.4: The macro average F1 score of five different feature sets applied to four cow behaviors on software and 8-bit fixed point hardware.

is a good performance metric for imbalanced datasets since the accuracy is high if the majority label (Resting) is correctly classified. The percentage difference ($\Delta\%$) is the hardware-software error originating from the quantization of numbers and algorithms. Similarly, Table 3.4 and 3.5 show the precision and recall in hardware (HW), including resources such as the amount of LUTs, registers and static power, and software (SW) for each behavior and **FS**.

While **FS**₁ requires the most hardware resources, its F1 score is 4.2% lower than **FS**₂, which considers only the magnitude of $\arctan2(y, x)$. Inspection of the **FS**₁ DT in software shows that the variation of $\arctan2(y, x)$ as a feature does not improve accuracy. Figure 3.5 supports this by showing less evident axis-parallel boundaries for Fig. 3.5a and 3.5b which are used in **FS**₁, compared to the boundaries in Fig. 3.5c, also used in **FS**₂. Furthermore, F1 score decrement for hardware-software is 0.5% for **FS**₂ while it is 4.1% for **FS**₁ since variation of $\arctan2(y, x)$ results in additional error in hardware. Also, while $\arctan2(y, x)$ uses slightly more resources than $\arctan(y/x)$, the F1 score increases by 12.3% in hardware. Data analysis shows a more distinct division of cow behavior for 4 quadrants of $\arctan2(y, x)$ compared to 2 quadrants of $\arctan(y/x)$. Finally, **FS**₄, similarly used in software in [227, 231] shows an F1 score decrement of 10 % for hardware-software and along with FS5 are inadequate for the classification of moving and eating.

FS₂ is selected since the classification accuracy of four behaviors increases significantly compared to **FS**₃ and conventional **FS**₄ and **FS**₅ while only sacrificing a small amount of static power consumption. Table 3.6 shows a comparison of the hardware implementation of **FS**₂ with other animal behavior classifiers. My implementation performs close to or better than state of the art with a classification latency of 7469 cycles or 2.58 s at the frequency of 2.9 kHz, suitable for cow behavior monitoring, and an average power of 216 μ W resulting in a total energy consumption of 557 μ J for one classification.

Table 3.4: A resource comparison for the hardware implementations on FPGA for 5 different feature sets

Feature set	Hardware Resources		
	#LUT4	#REG	Static power (μ W)
FS₁-HW	1349	473	223
FS₂-HW	1222	364	211
FS₃-HW	1160	358	208
FS₄-HW	724	319	185
FS₅-HW	673	285	179

Table 3.5: A performance comparison for the hardware implementations on FPGA and software applying 5 different feature sets.

Feature set	Classification Performance (%)							
	Resting		Eating		Rumination		Moving	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
FS₁-HW	84.68	92.69	84.52	88.75	97.94	57.23	70.88	80.12
FS₁-SW	90.74	92.69	86.25	86.25	98.25	67.47	72.41	91.30
FS₂-HW	89.79	92.69	83.72	90.00	98.43	75.30	72.07	80.12
FS₂-SW	91.9	92.69	84.15	86.25	98.33	71.08	72.14	90.06
FS₃-HW	89.44	95.36	46.81	83.54	100	68.87	67.92	47.68
FS₃-SW	91.90	95.14	66.28	72.15	100	69.54	69.54	80.13
FS₄-HW	94.69	93.14	37.14	32.50	88.81	71.69	54.17	73.58
FS₄-SW	95.91	90.36	57.58	71.25	89.21	74.70	64.95	79.25
FS₅-HW	82.95	92.04	40.21	48.75	97.44	68.67	59.32	65.22
FS₅-SW	88.98	92.04	40.21	48.75	97.44	68.67	49.30	48.75

Table 3.6: Comparison of **FS₂** on FPGA and state of the art

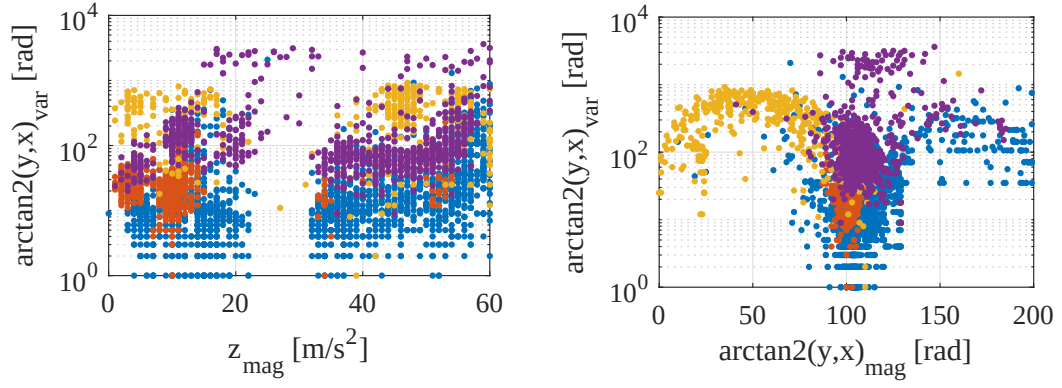
Reference	[212]* 2018	[216] 2018	FS₂ on FPGA
Application	Horse	Sheep	Cow
Sensor	3D Accl.	3D Accl.	3D Accl.
Sensor position	Jaw	Neck	Neck
Sampling Frequency (Hz)	30	100	25
Window size (s)	0.033	6.4	2.56
Algorithm	NN	DT	DT
#Behaviors	3	5	4
Implementation	MCU	MCU	FPGA
Power (μ W)	NR	4380 [†]	216
Energy (μ J)	NR	1259	557
Frequency (kHz)	NR	23722	2.9
Latency (cycle)	NR	1865 [†]	7469
Mean Accuracy (%)	80.77	87.97	86.81

NR: Not Reported

*: The best result [212] reports applies a kalman filter to all 3-axis accelerometer, gyrometer and magnetometer data to obtain a mean accuracy of 95.39%.

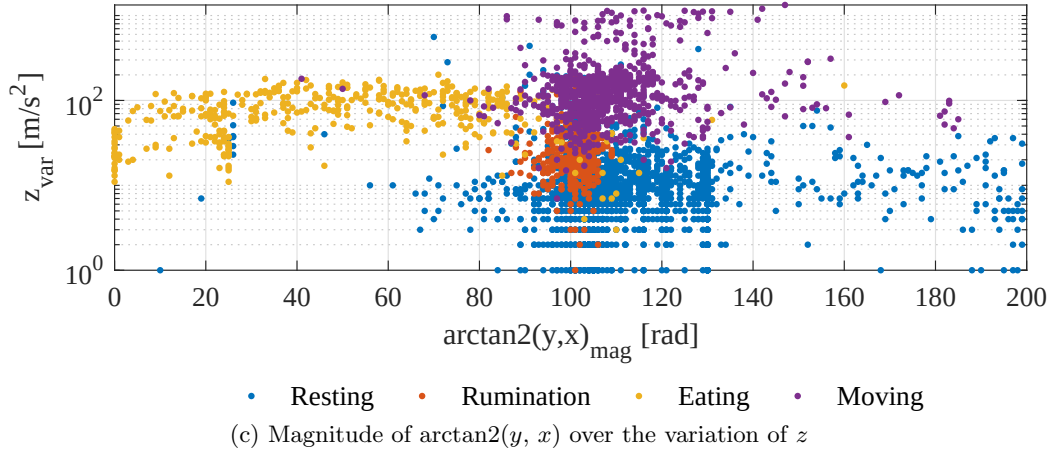
This is a high power application and outside the scope of this implementation.

[†]: This result is only for DT operation without feature extraction and time from window size.



(a) Magnitude of z over the variation of $\arctan2(y, x)$

(b) Magnitude of $\arctan2(y, x)$ over the variation of $\arctan2(y, x)$



(c) Magnitude of $\arctan2(y, x)$ over the variation of z

Figure 3.5: Scatterplots of quantized FS1 where decision rules occur in the DT for the processed cow behavior dataset containing 4359 samples, (a) and (b) show a less separable scatterplot than (c) which is also part of FS2.

Chapter 4

TinyCowNet: Memory- and Power-Minimized RNNs Implementable on Tiny Edge Devices for Lifelong Cow Behavior Distribution Estimation

In this Chapter, a Neural Architecture Search with post-training Integer-Only Quantization applied to Recurrent Neural Networks (RNN) is described with the purpose of finding operation and memory minimized architectures that estimate cow behavior distribution from the cow behavior dataset introduced in Section 2.4. The results of this method show highly accurate ($<95.5\%$) distribution estimating RNNs with memory as low as 2.043 kB and 42,822 Multiply-Accumulate (MAC) operations. In this way, it is shown that highly accurate Deep Neural Networks (DNN) can be trained and optimized without expert knowledge of animal behavior that are implementable on low-cost off-the-shelf hardware devices. The smallest RNN with high accuracy that results from this Chapter is implemented on FPGA using a resource-minimized RNN architecture, implementation described in Chapter 5.

The contribution of this Chapter is as follows:

- An analysis of the error of classification when considering time-windowed data from sensory systems which is dominant in previous literature.
- Development of a novel distribution regression algorithm with parametrized RNNs that are optimized using random search to show implementability of highly accurate cow behavior estimating models on low-power, low cost MCU and FPGAs.
- A variable bitwidth Full-Integer Quantization scheme that maintains similar accuracy

The remainder of this Chapter is structured in the following manner. Sections 4.1-4.3 describes an analysis on the use of windowing and a proposed distribution estimation algorithm. Next, Section 4.4 describes four types of neural networks layers that are

used to establish the proposed parametrized cow behavior distribution estimating RNN that is called TinyCowNet which is shown in Section 4.6. From this RNN, the method of optimizing the hyperparameters to minimize resource use and maximize accuracy is explained in Section 4.7 followed by Section 4.8 which describes post-training Full-Integer Quantization (FIQ). Section 4.9.2 shows the implementability of the 32-bit floating point RNN networks that were found using random search on four low-power embedded devices followed by resource minimized models with FIQ. This Chapter ends with a discussion of the results and possible future work in Section 4.10.

4.1 Windowing with Classification Algorithms

Windowing is the process whereby a data sequence is parcellated over a pre-set sample space, the window size, through time. For a window size l_s , for every l_s -th sample, the features used in an ML algorithm are extracted and averaged. This windowing reduces the amount of data, decreasing transmission size and processing load, and often improves performance when not considering the mixed-label error introduced. Time-based windowing is described by

$$\mathbf{X} = \frac{\sum_{i=1}^{l_s} \mathbf{m}(i)}{l_s} \quad (4.1)$$

where $\mathbf{m}$ is the vector representing the features and $\mathbf{X}$ is the processed data used in training and inference.

4.2 Labels in Classification Algorithms

For a labeled dataset in animal behavior estimation (Section 2.4), each sample has a label assigned by an observer based on video footage. The features are real numbers, but the assigned labels are categorical. Therefore, one cannot apply an averaging method during windowing, and a different approach is necessary. According to our knowledge, most PLF literature does not clearly describe this problem, its impact, and the necessity to consider it in edge applications for farmers [234].

Fig. 4.1 shows windowing over a window size of $l_s = 64$ with the cow behavior dataset described in the next Section. For "pure" mode, the raw data are sorted into behavioral categories and then windowed, forming long homogeneous behavior collations. Nevertheless, this method corrupts the temporal relationships in the data, important for real-life representation. On the other hand, "raw" preserves the original data that may include multiple categories within each window as is, preserving this relationship. However, multiple categories pose a problem for classification because there is a single label output. We consider two ways in which this problem could be solved. In the case of pure data, the label has to match the data input to be correct. However, in the case of raw mixed-label data, there is ambiguity. One way is to consider the majority label as in Fig. 4.1, as the correct classification output. With this method, there is a worst-case error of 75% in cases of equal behavior distribution, and this error depends on the number of behaviors, i.e.,

$$\epsilon_{\max} = 1 - \frac{1}{N_b} \quad (4.2)$$

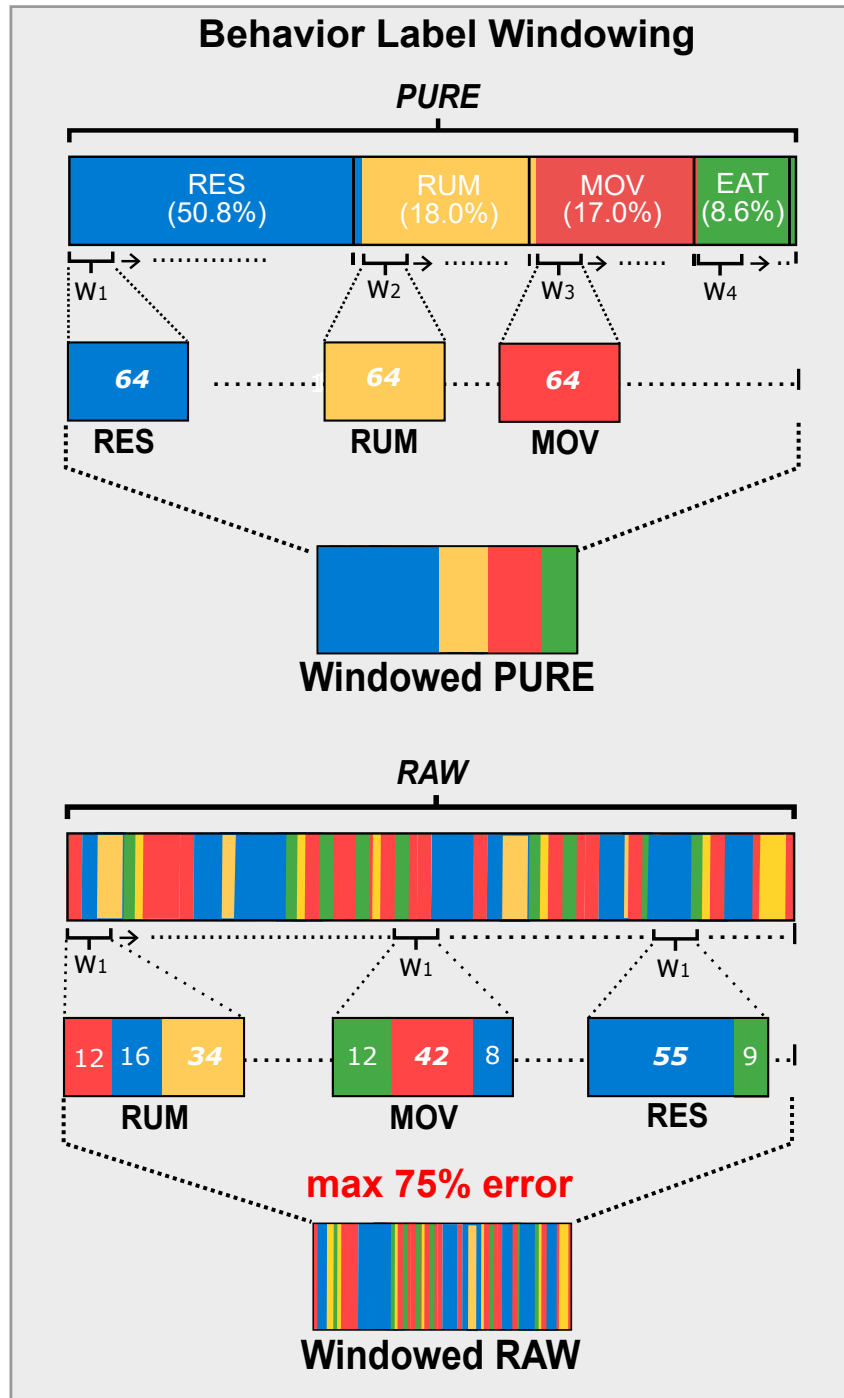


Figure 4.1: Label processing for pure data labels and raw data labels having a window size of 64. The colored stripes denote the behaviors of Resting (RES), Rumination (RUM), Moving (MOV) and Eating (EAT) with blue, yellow, red and green respectively.

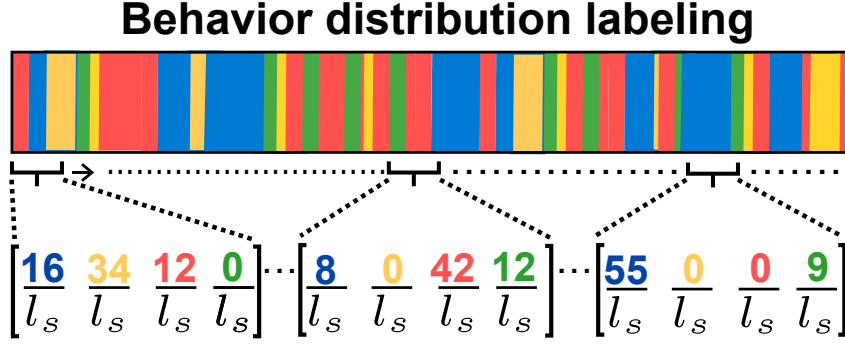


Figure 4.2: Method of labeling raw data into distribution vectors. Instead of a single label as in classification tasks, a distribution vector is used for regression. The l_s is the sample window size, normalizing the vector.

which approaches unity for a large number of behaviors, N_b . Another way is to apply a nearest-neighbor approach and assign a label approaching the mean feature value of each label category. However, this method is sensitive to outliers. If one window contains label categories with largely separated feature values, this assigns a nonsensical label. Therefore, we posit that assigning the majority label as performed in literature, might lead to large error whenever an animal changes behavior. For example, in Table 2.1, across the existing studies, the window size ranges from 1 up to 120 seconds.

4.3 Proposed Distribution Regression

Instead of a classification algorithm that involves mixed label error, we hereafter propose a regression algorithm to estimate a distribution. This regression eliminates the error in Eq. (4.2) as it estimates the label category proportion over an arbitrary-sized window. The output vector of this algorithm is described by

$$\mathbf{y} = [\alpha_1, \dots, \alpha_i, \dots, \alpha_{N_b}] \quad (4.3)$$

where N_b is the number of behaviors, four in this work, $\alpha_i \in [0, 1]$ is the prevalence for behavior i and $\mathbf{y}$ is the output of the regression algorithm. This $\mathbf{y}$ is a vector having unitary L_1 norm, i.e.,

$$\|\mathbf{y}\|_1 = \sum_{i=1}^{N_b} \alpha_i = 1 \quad (4.4)$$

where α_i is calculated as

$$\alpha_i = \frac{m_i}{l_s} \quad (4.5)$$

and m_i is the number of samples for each behavior i in a window. The amount of time per behavior in a window, t_i , with window size l_s is then

$$t_i = \alpha_i \frac{l_s}{f_{acc}} \quad (4.6)$$

where f_{acc} is the sampling frequency of the accelerometer.

Fig. 4.2 shows raw data processed into distribution labels according to Eq. (4.3). This method allows similar data size reduction as classification, while enabling unbiased estimation of the time spent in each behavior.

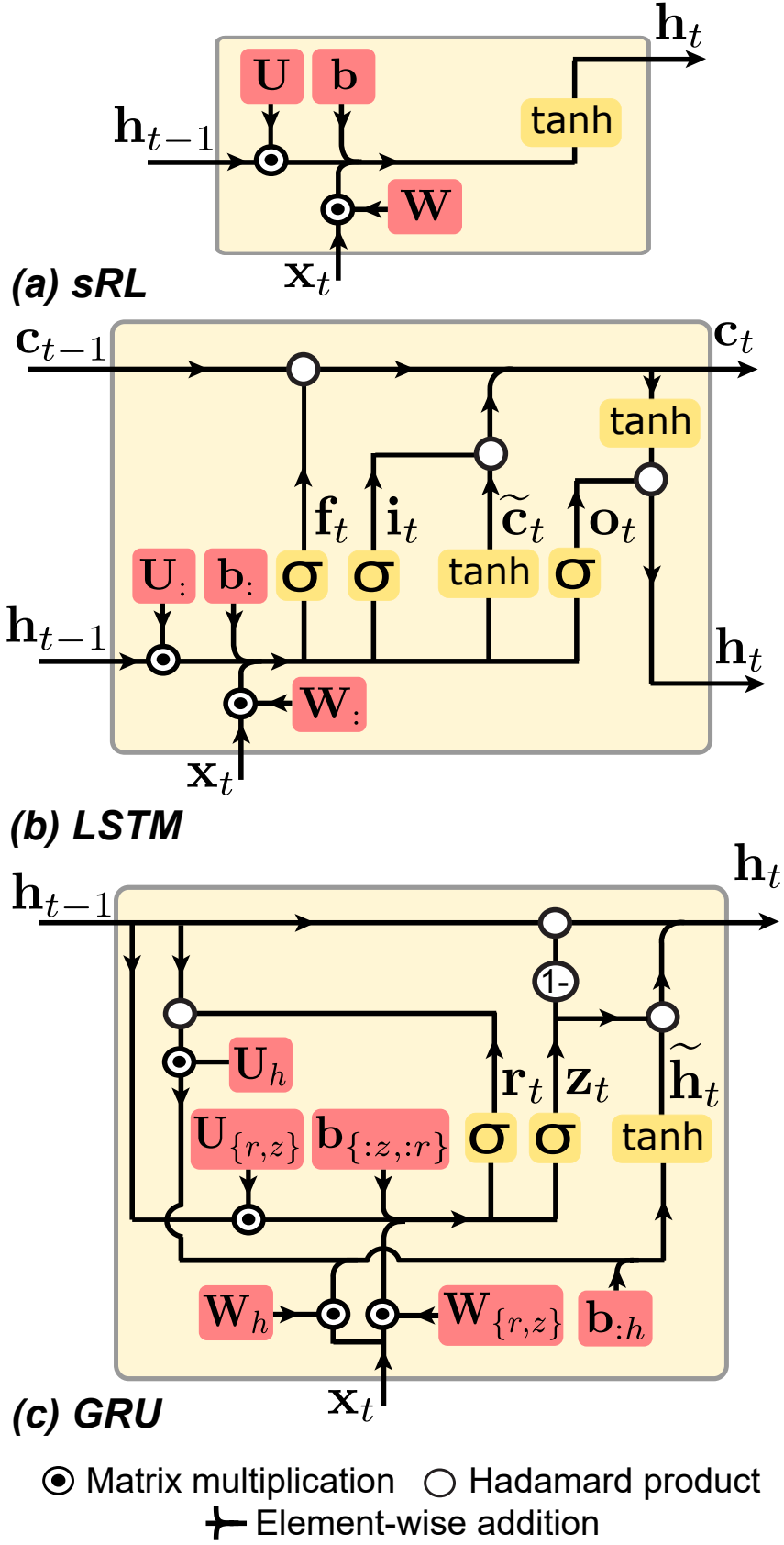


Figure 4.3: Architecture overview of (a) Simple Recurrent Layer, sRL, (b) Long-Short Term Memory, LSTM, and (c) Gated Recurrent Unit, GRU.

4.4 Recurrent Neural Network Methods

This Section presents the methodology of different neural network layers essential for understanding the proposed model architecture in Section 4.6.

4.4.1 Recurrent Layers

Recurrent Layers (RL) are algorithmic units with the following general input and output sequences

$$\mathbf{X} = \{\mathbf{x}_1; \mathbf{x}_2; \dots \mathbf{x}_{T_s}\} \quad (4.7)$$

$$\mathbf{Y} = \{\mathbf{y}_1; \mathbf{y}_2; \dots \mathbf{y}_{T_s}\} \quad (4.8)$$

where $\mathbf{X}$ represents the input sequence and $\mathbf{Y}$ the output sequence with a size of T_s timesteps. Using only the last output of $\mathbf{Y}$, $\mathbf{y}_{T_s}$, is commonly called many-to-one and is considered in this work. RLs can learn the characteristics of time-dependent problem data sets as they recur their outputs for each timestep. Fig. 4.3 shows three RNN architectures, equivalent to those in TensorFlow 2.0 with Keras backend [215, 235], that are used in this work.

Simple Recurrent Layer (sRL)

Recurrent Layers have their roots in cognitive science, and early parallel computation [93, 94]. In its most simplistic form, the simple recurrent layer (sRL) has a trainable bias, two trainable weight matrices, and one activation function. The model equation for this layer is

$$\mathbf{h}_t = \tanh(\mathbf{b} + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t) \quad (4.9)$$

where $\mathbf{h}_t$, $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ represent the recurrent output, the previous recurrent output, and the input vectors. The recurrent, input, and bias matrices are denoted by $\mathbf{U}$, $\mathbf{W}$, and $\mathbf{b}$. Finally, the hyperbolic tangent, $\tanh(x)$, is chosen as activation function. Although least complex, this model suffers from vanishing-exploding gradient, complicating the training of large models. This problem can be solved with gradient clipping [236] or stochastic diagonal approximate greatest descent [237] during training.

Long Short Term Memory (LSTM)

Other than the above, Ref. [95] introduced Long Short Term Memory (LSTM) that addresses the vanishing-exploding gradient by introducing more gates. Since its formulation, LSTM has yielded a multitude of breakthroughs in sequence learning problems [97]. The model equations are

$$\mathbf{f}_t = \sigma(\mathbf{b}_f + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{W}_f\mathbf{x}_t) \quad (4.10)$$

$$\mathbf{i}_t = \sigma(\mathbf{b}_i + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{W}_i\mathbf{x}_t) \quad (4.11)$$

$$\mathbf{o}_t = \sigma(\mathbf{b}_o + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{W}_o\mathbf{x}_t) \quad (4.12)$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{b}_c + \mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{W}_c\mathbf{x}_t) \quad (4.13)$$

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t \quad (4.14)$$

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t) \quad (4.15)$$

where $\mathbf{f}_t$, $\mathbf{i}_t$, and $\mathbf{o}_t$ are the forget, input and output vectors, respectively, the $\circ$ operator indicates the Hadamard product, and $\sigma(x)$ is the sigmoid activation function. Finally, there is an additional output vector, cell state vector $\mathbf{c}_t$, which is recurrent as $\mathbf{h}_t$, but not forwarded to the deeper layers.

Gated Recurrent Unit (GRU)

The GRU, introduced in Ref. [96], also addresses the vanishing-exploding gradient but reduces the gates and outputs. The model equations of the GRU are

$$\mathbf{z}_t = \sigma(\mathbf{b}_{rz} + \mathbf{b}_{iz} + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{W}_z \mathbf{x}_t) \quad (4.16)$$

$$\mathbf{r}_t = \sigma(\mathbf{b}_{rr} + \mathbf{b}_{ri} + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{W}_r \mathbf{x}_t) \quad (4.17)$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{b}_{rh} + \mathbf{b}_{ih} + \mathbf{U}_h(\mathbf{r}_t \circ \mathbf{h}_{t-1}) + \mathbf{W}_h \mathbf{x}_t) \quad (4.18)$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \circ \mathbf{h}_{t-1} + \mathbf{z}_t \circ \tilde{\mathbf{h}}_t \quad (4.19)$$

where $\mathbf{z}_t$, $\mathbf{r}_t$ and $\tilde{\mathbf{h}}_t$ are the update, reset and candidate activation gates. Furthermore $\mathbf{b}_{r\cdot}$ and $\mathbf{b}_{i\cdot}$ represent the recurrent and input biases. The GRU has one more bias term than LSTM but a single recurrent output. An empirical study showed that the GRU outperforms the LSTM on all tasks except natural language processing [238]. However, both GRU and LSTM suffer from gradient decay for deep multi-layer networks, increasing the training time [239].

Linear Layer (ll)

Linear Layers have two trainable weight matrices: the bias and the kernel. Each layer is a linear function, i.e.,

$$\mathbf{o} = \mathbf{W}\mathbf{x} + \mathbf{b} \quad (4.20)$$

where $\mathbf{o}$ and $\mathbf{x}$ are the output and input vectors. Furthermore, $\mathbf{W}$ and $\mathbf{b}$ represent the trainable kernel and bias. This layer is used frequently as an output layer, mapping features to class labels. Alexnet, which revolutionized deep learning for computer vision and Mobilenet, are networks that have leveraged these in the past [91, 240].

Table 4.1: Weight dimensions and Multiply-Accumulate (MAC) operation of the four different neural network layers. Variables i , n and m denote the input size, recurrent layer width and output vector size.

Layer (k)	Weight matrix size			Operations
	$\mathbf{U}$	$\mathbf{W}$	$\mathbf{b}$	N. of MAC
sRL	$n \times n$	$i \times n$	$1 \times n$	$n^2 + ni$
LSTM	$4n \times n$	$i \times 4n$	$1 \times 4n$	$4n^2 + n(4i + 3)$
GRU	$3n \times n$	$i \times 3n$	$2 \times 3n$	$3n^2 + n(3i + 3)$
Linear	-	$n \times m$	$1 \times m$	mn

4.5 Hardware Considerations

Recurrent layers incur a high number of multiplications, proportional to the sizes of the input kernel, recurrent kernel, and bias matrices. Furthermore, activation functions are non-linear functions that require an approximation in digital hardware such as Application Specific Integrated Circuits (ASIC).

Table 4.1 shows the number of weights and Multiply-Accumulate (MAC) operations for each of the layers, i.e., sRL, GRU, LSTM, and Linear. The number of MACs increases linearly or quadratically with the number of weights. Thus, sparsifying the network, that is, minimizing the number of weights can reduce the amount of MAC operations considerably. The majority of dynamic energy consumption of neural networks on hardware is due to memory access [241], which can be reduced by replacing dynamic with static RAM (e.g., $128\times$ on 45nm CMOS [172]). Nevertheless, minimizing n is essential.

Fig. 4.4 shows the relationship of the weights with layer width n and the number of stacked layers N for the sRL. With every doubling of layer width n , layer count can be quadrupled at the same weight cost. Thus, stacking multiple recurrent layers increases the amount of memory and multiplications only linearly, in contrast with increasing the weight dimensions with parameter n .

4.6 Proposed Cow Behavior Distribution Estimating Neural Network (TinyCowNet)

Artificial recurrent neural networks are known to learn time-based relationships of generative processes and are prevalent in applications of natural language processing, machine translation, and sequence modeling [77]. In hardware, over 70 implementations on ASIC, FPGA, or GPU/CPU have been demonstrated in Ref. [78], but only two were applied to action recognition using video footage [242, 243]. To our knowledge, no RNN for animal behavior monitoring on hardware has been described.

The recurrent layers described in Section 4.4 are used to optimize network architecture towards estimating cow behavior distribution in a future low-power hardware implementation. This network is composed of one to four recurrent layers and a linear layer that outputs a behavior distribution vector as in Eq. (4.3). To find an optimal network in terms of accuracy and model size, the parameters, including those of Table 4.1 are optimized with neural architecture search (explained in Section 4.7).

Fig. 4.5 shows the proposed parameterized neural network, TinyCowNet. Each

Table 4.2: Hyperparameters defined for the search space of the recurrent layers, i.e., the layer width n , the total number of layers N , the timesteps T_s and the cell architecture type k .

Hyperparameter	Range	Description
n	$R_n = 3, 4 \dots 250$	Layer width
N	$R_N = 1, 2 \dots 4$	Number of layers
T_s	$R_{L_s} = 2, 3 \dots 300$	Timesteps
k	$R_k = \{\text{sRL,GRU,LSTM}\}$	Recurrent layer type

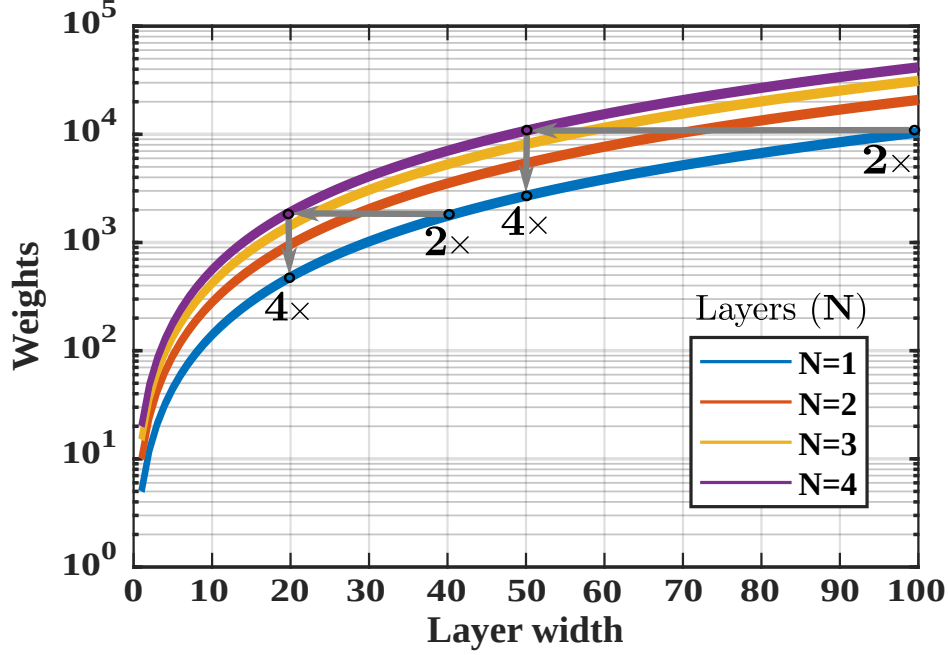


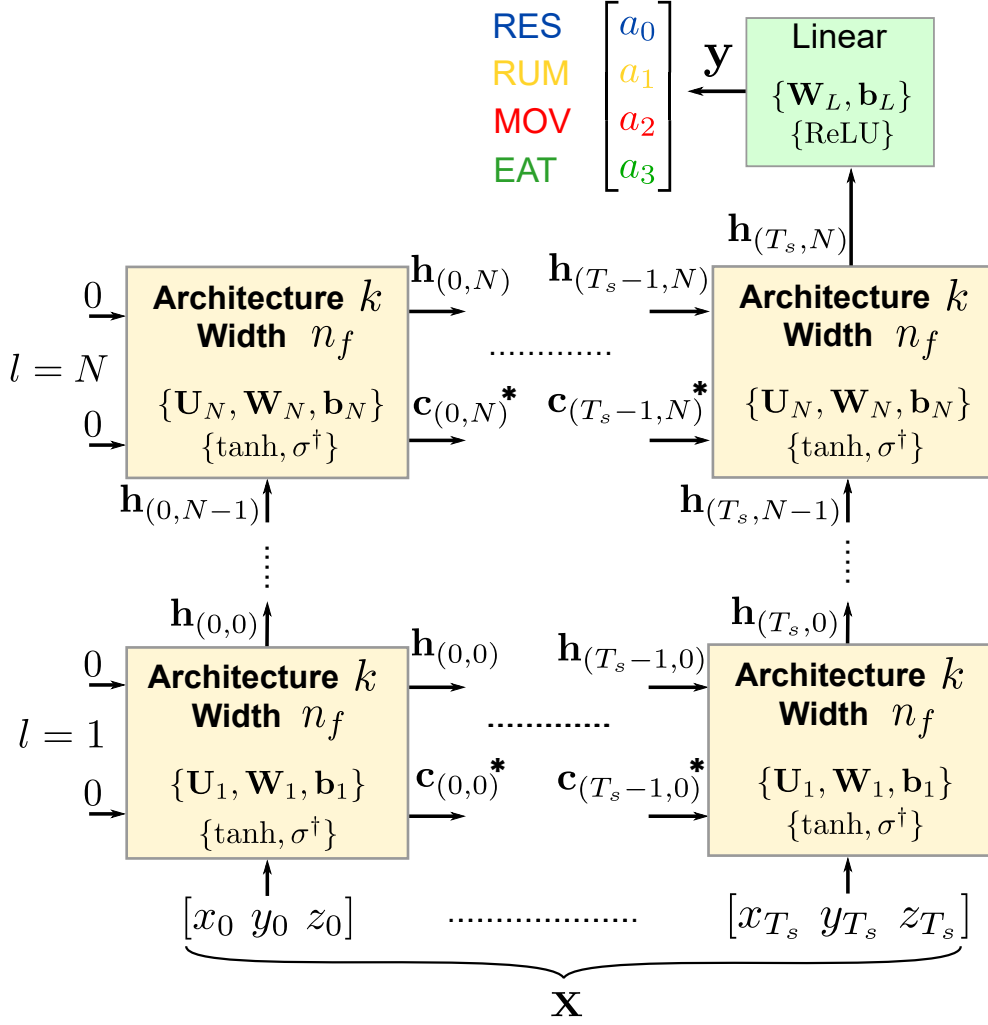
Figure 4.4: Layer width n for four different layer counts N plotted vs. total number of weights using simple Recurrent Layers. The gray lines indicate that a doubling of the layer width n for a quadrupling of the numbers of layers N is possible at the same weight cost.

yellow block represents the architecture k (GRU, LSTM or sRL) with a width n_f , trainable weights $\mathbf{U}_l$, $\mathbf{W}_l$ and $\mathbf{b}_l$, and activation functions $\tanh(x)$ and $\sigma(x)$. This n_f is the layer width n divided by layer count N to prevent the creation of excessively large models, i.e.,

$$n_f = \left\lfloor \frac{n}{N} \right\rfloor \quad (4.21)$$

where the fraction is floored. For each time step, tri-axial accelerometer data are submitted to the network until timestep T_s . Furthermore, each layer recurs the previous hidden state vector $\mathbf{h}_{t-1}$, that is also forwarded to the next layer. At timestep T_s , the final output of the deepest recurrent layer is linearly mapped to output a cow behavior distribution vector using the ReLU activation function, i.e., $\text{ReLU}(x) = \max(0, x)$. In the case of an LSTM network, there is an additional recurrent output, cell state vector $\mathbf{c}_t$, which is not forwarded to the deeper layers.

Table 4.2 shows all parameter ranges of the search space subject to optimization. Because of composite generative processes in nature, it is expected that multi-layer networks will outperform wide single-layer networks [244]. As regards the layer width n , the minimum number (3) is determined by the need to avoid a trivial response, whereas the maximum number (250) reflects the maximum available memory in the devices under consideration. As regards the number of layers N , the minimum number (1) is self-evident, whereas the maximum number (4) was determined based on existing literature indicating that deeper networks are difficult to train when considering the range of timesteps [239]. As regards the timesteps T_s , the minimum number (2) is self-evident, and the maximum number was set to 300 since this is an intermediate level between the limits previously observed: sRLs become increasingly



*: Cell state vector, $\mathbf{c}$, only applies to LSTM networks

† : The Sigmoid activation function, σ , does not apply to sRL networks

Figure 4.5: Cow behavior distribution estimating neural network, rolled out over time from left to right. The yellow blocks represent the recurrent layers with outputs, hidden state vector $\mathbf{h}$ and cell state vector $\mathbf{c}$, and contain the trainable weights $\{\mathbf{U}, \mathbf{W}, \mathbf{b}\}$. The green block represents the linear output layer with trainable weights $\{\mathbf{W}_L, \mathbf{b}_L\}$. Each state vector is indicated with entries (p, q) where p is the current timestep and q is the respective layer. The hyperparameters included in the search space are architecture k , width n and layer count N and timesteps T_s . Finally, $\hat{\mathbf{y}}$ denotes the regressed distribution vector containing cow behavior prevalence.

difficult to train for $T_s > 100$, whereas LSTMs and GRUs become increasingly difficult to train for $T_s > 500$ [239].

4.6.1 Target networks

Table 4.3 describes three low-power FPGAs and one low-power MCU with their total memory, clock speed, board area and purchase price. These are indicated purely as representative examples. Each FPGA has enough resources for the implementation of several multiply-accumulate units and control logic. Furthermore, TensorFlow Lite [215] for MCUs uses only 16kB of runtime memory, therefore we assume a remaining memory of 68kB. The average price of all these devices is 7.2\$. In order to allow for future implementation of TinyCowNets, we have set the target of building networks which have a memory footprint below the smallest device. Furthermore, to be competitive with other works, we aimed to maintain an accuracy $> 93\%$.

4.7 Finding the target networks

Recent advances in neural network design have yielded computer-designed neural network architectures that outperform expert-designed neural networks in computer vision tasks [245, 246]. This field of research, neural architecture search, was first proposed in Ref. [247] using reinforcement learning. Since then, different methods using genetic algorithms and gradient descent have been applied to reduce search time and further increase model accuracy [248, 249]. A comparison of these methods for cow behavior estimation and recurrent neural networks is outside of the scope of this work. We decided to apply random search to determine the recurrent layers in Section 4.6 and Table 4.2 because of its straightforward implementation and because defining suitable heuristics for more advanced methods was beyond scope. For example, determining a suitable fitness function and chromosome structure for efficient optimization using genetic algorithms requires careful consideration of a multitude of aspects, to make sure the evolutionary aspect meaningfully improves the process over a pure random search. Random search generates a model according to randomly assigned values within the allowed ranges of parameters, and as such it is the most parsimonious possibility with respect to the assumptions that have to be introduced. It was shown in Ref. [250] and argued by Ref. [251] that current neural architecture search methods do not significantly outperform this brute-force search method.

4.7.1 Random Search

Each point in the search space as denoted in Table 4.2 is defined as

$$\Theta = \{n_f, N, T_s, k\} \quad (4.22)$$

where n_f is the floored layer width from Eq. (4.21), N is the layer count, T_s is the amount of timesteps and k is the recurrent layer type. There are 295,412 unique possible combinations with the indicated ranges, making a grid search unfeasible. Therefore, we set the hyperparameters to be uniformly-distributed discrete random variables so that each point in the parameter range has an equal probability of being selected. A total of 445 random searches were performed.

Fig. 4.6 shows the method of randomly selecting hyperparameters and training TinyCowNets. For each iteration, one unique subset of the search space denoted

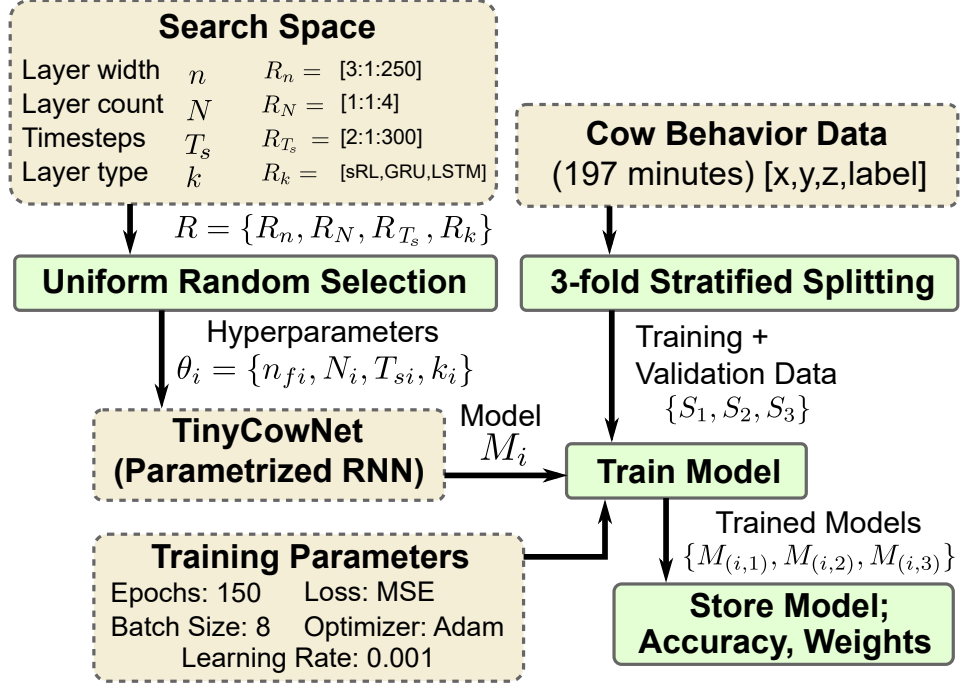


Figure 4.6: Diagram showing a single iteration i of a random search applied to four hyperparameters of TinyCowNet (Fig. 4.5). These hyperparameters represent layer width n , layer count N , layer type k , which have influence over the complexity of the network and timesteps T_s . For each iteration, a set of hyper parameters θ in the uniformly distributed range R is randomly assigned to TinyCowNet, in this case M_i . Hereafter, a three-fold stratified split training-validation dataset (Fig. 4.7) is used to train the model with Mean Squared Error (MSE) as loss function for 150 epochs. Finally the resulting three model accuracies and weights are stored for later resource-optimized candidate selection.

Table 4.3: Four different low-power programmable devices to be considered as targets for TinyCowNet implementation. The Lattice UL640 and Microsemi AGLN250 FPGA represent ultra low power devices only suitable for the smallest networks. The STMicroelectronics MCU and Lattice UP5K represent low-power FPGAs which are also suitable for medium sized models. The cost in dollars of all these devices were retrieved per single piece from Mouser Electronics.

Device	Memory (kB)	Clock (MHz)	Sleep Power (μ W)	Area (mm ²)	Cost (\$)
[A] UL640 FPGA [252]	7.1	48	42	2	1.82
[B] AGLN250 FPGA [253]	4.7	250	10	196	16.43
[C] STM32L071C8 MCU [254]	84	32	37	49	4.60
[D] UP5K FPGA [219]	143	48	90	6.4	5.96

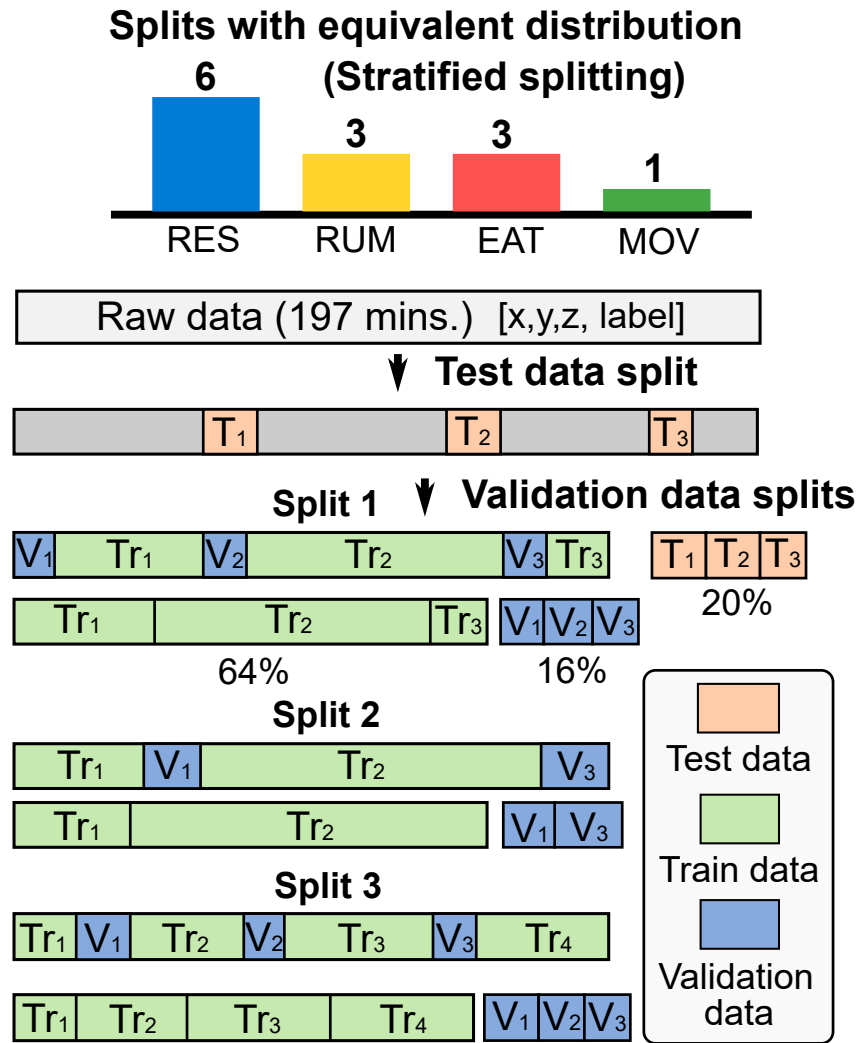


Figure 4.7: Stratified splitting of the imbalanced cow behavior dataset.

as $\Theta_i = \{n_{fi}, N_i, T_{si}, k_i\}$, is randomly assigned from the uniform range R . The parametrized neural network, TinyCowNet (Section 4.6), is adapted with this subset and trained on a supercomputer for 150 epochs and three different splits of the cow behavior dataset using Adam optimizer and a batch size of 8. After training, the model characteristics, weights, subspace Θ_i , and accuracy are stored, and a new iteration is initiated.

4.7.2 Training conditions

Fig. 4.7 shows the data splitting methodology. First, the data are split into 80% and 20%, of which the latter are used as test dataset. The remaining data is then split into 16% and 64% representing the validation and training datasets, repeated over three folds. Each fold's data split is stratified, i.e., the training, validation, and testing dataset have similar distribution of labels as the original dataset (distribution as in Fig. 2.8, i.e., 6:3:3:1). This validation dataset is used to rate each candidate model by its accuracy. For each training iteration, the randomly assigned model is trained and validated with different data, resulting in three different models. Finally, the performance is averaged and stored for both the validation and the test datasets. Stratified splitting ensures that the datasets have comparable distributions and has been used to obtain reliable performance of cow behavior classification [255]. Each of these three data splits represents the original imbalanced dataset with possible overlapping, but performance on each fold is still considered as independent [256].

The evaluated metric during training is the standard Mean Squared Error (MSE) for a four behavior distribution vector which is described by

$$\text{MSE} = \frac{1}{4} \sum_{i=1}^4 (a_i - \hat{a}_i)^2 \quad (4.23)$$

where a_i is the prevalence of the i -th behavior and $\hat{a}_i$ is the estimated prevalence from the selected TinyCowNet. This MSE punishes the algorithm more heavily for error than a L_1 norm vector, decreasing training time.

4.7.3 Candidate Evaluation: Accuracy, MACs and Memory

After training, the Mean Absolute Error (MAE) is used to score each architecture over the validation dataset. This MAE represents the absolute difference between each estimated $\hat{a}_i$ and real prevalence a_i averaged over all behaviors, i.e.,

$$\text{MAE} = \frac{1}{4} \sum_{i=1}^4 |a_i - \hat{a}_i| \quad (4.24)$$

Compared to the MSE, the MAE lends itself to a more intuitive interpretation, in light of the fact that the network output is a distribution of prevalence. On the other hand, from a different perspective, the accuracy of the algorithm in identifying the most prevalent behavior is obtained by considering expectation, i.e.,

$$\text{Accuracy} = \frac{1}{M} \sum_{i=1}^M E[\mathbf{y}(i), \hat{\mathbf{y}}(i)] \quad (4.25)$$

where M is the length of the dataset and the operator $E[\mathbf{y}(i), \hat{\mathbf{y}}(i)] = 1$ if the largest element matches between the two vector, and $E[\mathbf{y}(i), \hat{\mathbf{y}}(i)] = 0$ otherwise. Next to

the MAE and accuracy, we employ the Matthews Correlation Coefficient (MCC) that ranges from -1 to 1, as it is a standard metric used on imbalanced datasets [257, 258]. Besides this, we have observed that the majority of the distribution vectors processed from the dataset contain a single behavior. However, TinyCowNet does not present a binary output (1 or 0) as the final layer is linear and non-ideal. As such, we use the Receiver Operating Characteristic (ROC) curves and calculate the Area Under Curve (AOC) instead of setting an arbitrary threshold to classify each behavior from the output vector independently. This is a standard statistical procedure used for imbalanced datasets [257, 259].

Other than performance metrics, the memory occupation and number of MAC operations of each model are determined. The total number of MACs that a TinyCowNet uses, O_k , for each layer type k including linear layer, layer count N and timesteps T_s is given by

$$O_{\text{sRL}} = T_s(n_f^2(2N - 1) + 3n_f) \quad (4.26)$$

$$O_{\text{LSTM}} = T_s(n_f^2(8N - 4) + n_f(3N + 12)) \quad (4.27)$$

$$O_{\text{GRU}} = T_s(n_f^2(6N - 3) + n_f(3N + 9)) \quad (4.28)$$

$$O_{ll} = 4n_f \quad (4.29)$$

where the contribution of the deeper layers is more significant. This is because the recurrent vector output, $\mathbf{h}_t$, from all layers has a size of n_f , changing the input weight matrix size of the deeper layers to $\mathbf{W} = n_f \times n_f$. Furthermore, the T_s term is necessary to determine the total amount of operations as the networks recur until the final timestep. The number of weights required to store the model for each recurrent layer type k and output layer ll is given by

$$P_{\text{sRL}} = n_f^2(2N - 1) + n_f(N + 3) \quad (4.30)$$

$$P_{\text{LSTM}} = n_f^2(8N - 4) + n_f(4N + 12) \quad (4.31)$$

$$P_{\text{GRU}} = n_f^2(6N - 3) + n_f(6N + 9) \quad (4.32)$$

$$P_{ll} = 4n_f + 4 \quad (4.33)$$

where the GRU has two bias weights instead of one. Overall, the number of MACs in sRL TinyCowNet is between 3-4 times lower than that of LSTM and GRU.

4.8 Post-Training Quantization

TensorFlow models are trained and inferred using 32-bit floating-point representation. Such models on hardware use large amounts of memory and high-complexity arithmetic operations. By contrast, the industry standard in low-power digital processing hardware is using 8-bit fixed point numbers. Quantization of neural networks is generally split into two methods: pre- and post-training quantization. While the former has led to better results than the latter [260], quantization during training adds another dimension to the optimization problem, further complicating decisions on optimal architecture, beyond the scope of this work. Therefore, we decided to create an ad-hoc method of post-training, integer-only quantization aiming to minimize energy consumption and memory occupancy. A similar, although not fully equivalent, approach has been described in a recent preprint report [261]. The novelty in this work is the activation function scaling and dynamic bit-width optimization.

4.8.1 Full-Integer Quantization for RNNs

Neural networks with integer-only arithmetic use lower-power modules in hardware, and show a lower latency for similar accuracy compared to floating-point models [262]. This fixed-bit-width integer-arithmetic-only quantization scheme from Ref. [262] is used in TensorFlow Lite, but limited to convolutional neural networks. Here, we adapt it to RNNs. An analysis of different quantization methods is out of scope and will be systematically performed in a future study. For now, we considered variable bit-widths for different parts of the RNN towards power-accuracy optimization with an in-depth analysis but without comparing different possibilities. This Section describes the proposed quantization scheme.

Consider the mapping of a real number r to an integer q with scale S and zeropoint Z

$$r = S(q - Z) \quad (4.34)$$

the scale is then defined as

$$S = \frac{a}{2^l - 1} \quad (4.35)$$

where a is the range, and l is the bit-width. In the case of a static matrix such as the trainable weights of a neural network, i.e., TinyCowNet, we assign this range to

$$a_w = |\max(\mathbf{W})| + |\min(\mathbf{W})| \quad (4.36)$$

as $\mathbf{W}$ is one of the matrices to be quantized. The zeropoint offsets any negative numbers avoiding the use of two's complement representation, further reducing logic occupancy. We do this by relating the zeropoint to the minimum number in the weight matrix as in

$$Z_w = \left\lfloor \frac{-\min(\mathbf{W})}{S_w} \right\rfloor \quad (4.37)$$

where S_w is the scale of quantized $\mathbf{W}$, and $\lfloor x \rfloor$ denotes rounding. Each weight of the matrix is subsequently quantized by

$$q_w^{(i,j)} = \left\lfloor \frac{r_w^{(i,j)}}{S_w} + Z_w \right\rfloor \quad (4.38)$$

where $r_w^{(i,j)}$ is the floating point number and $q_w^{(i,j)}$ is the quantized number.

In the case of bias $\mathbf{b}$, terms $\mathbf{U}\mathbf{h}_{t-1}$ and $\mathbf{W}\mathbf{x}_t$, the scales and zeropoints have to be equivalent for straightforward element-wise addition. Therefore, the domains of the activation functions are considered to set the quantization range α as in Eq. (4.35). We assume the sigmoid has a range value, $a_\sigma = 12$, and the hyperbolic tangent has a range value of 6, $a_{\tanh} = 6$, considering their asymptotes. The above described terms use the same quantization scheme as in Eq. (4.38), but their scale S_3 and zeropoint Z_3 are determined by the activation function. This is a_σ , for LSTM and GRU networks or $a_{\tanh}$ for the sRL network. Fig. 4.8 shows an example of quantizing a 3×3 matrix and a 1×3 bias matrix with the methods described above.

A matrix multiplication involves two differently-quantized numbers for each entry with different scale and zeropoint. In that case multiplication for each entry with the quantization scheme is described as

$$q_3 = \frac{S_1 S_2}{S_3} (q_1 - Z_1)(q_2 - Z_2) + Z_3 \quad (4.39)$$

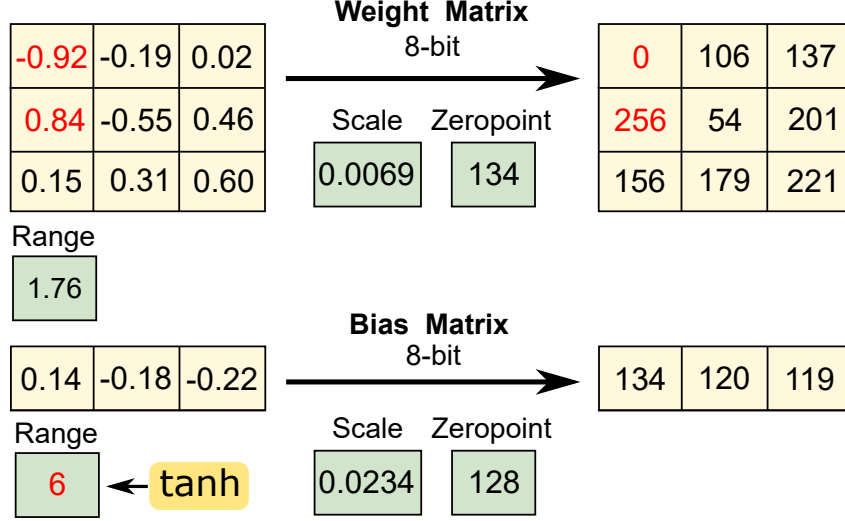


Figure 4.8: Example of full-integer quantization for RNNs with a weight matrix and bias matrix for which the range is defined by the activation function at 8-bit bit-width.

where S_3 and Z_3 are the scale and zeropoint of the resulting number. The results of matrix multiplications with $\mathbf{U}$ and $\mathbf{W}$ are added to the bias matrix $\mathbf{b}$. This means that S_3 and Z_3 have the range of a_σ or $a_{\tanh}$. In turn, S_1 and Z_1 are defined by $\mathbf{U}$ and $\mathbf{W}$. Similarly, S_2 and Z_2 are set to the ranges of the accelerometer data ($\pm 2g$) and the recurrent output vector with range $a_{\tanh}$. Multiplication of an $I \times M$ and an $J \times M$ matrix are performed according to

$$q_3^{(i,m)} = \frac{S_1 S_2}{S_3} \sum_{j=1}^M \left[(q_1^{(i,j)} - Z_1)(q_2^{(j,m)} - Z_2) \right] + Z_3 \quad (4.40)$$

which can be rewritten as

$$q_3^{(i,m)} = \frac{S_1 S_2}{S_3} \left[M Z_1 Z_2 - Z_2 \sum_{j=1}^M q_1^{(i,j)} - Z_1 \sum_{j=1}^M q_2^{(j,m)} + \sum_{j=1}^M q_1^{(i,j)} q_2^{(j,m)} \right] + Z_3 \quad (4.41)$$

where the first two terms in the brackets are known in advance since each entry in weight matrix $q_1^{(i,j)}$ is available [262]. Therefore, these can be stored to speed up the multiplication. However, for large bit-widths, this second term takes up a considerable amount of memory. For this reason, we have decided to store no other than the zeropoints, scales, and column size M . These equations are similar to Ref. [262].

4.8.2 Optimal quantization with grid search

Fig. 4.9 shows the above-described quantization scheme applied to a simple recurrent layer considering the weight bit-width w_q , bias bit-width b_q and output/activation bit-width h_q . These parameters replace l in Eq. (4.35) for each weight matrix. Matrices $\mathbf{U}$ and $\mathbf{W}$ are quantized with w_q , bias matrix $\mathbf{b}$ with b_q and the recurrent output

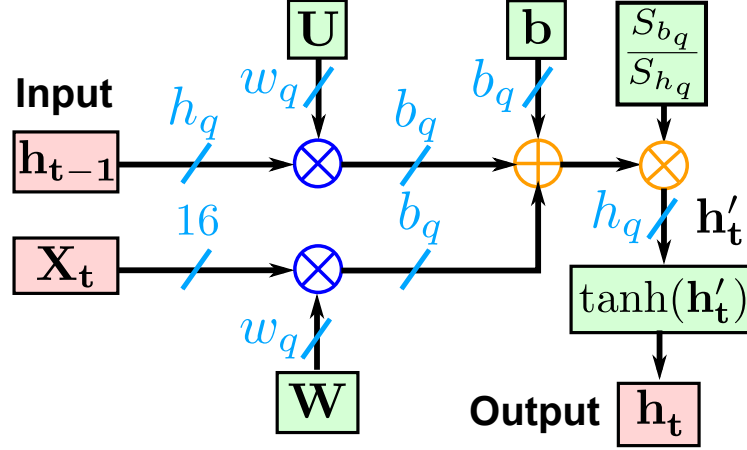


Figure 4.9: Overview of the simple recurrent layer with integer quantization. The red and green blocks represent the input/output and stored weights, respectively. The light blue stripes indicate the bit-width of the buses, output bit-width h_q , weight bit-width w_q and bias bit-width b_q . The dark blue multipliers are equivalent to the matrix multiplication as described in Eq. (4.41). The hyperbolic tangent function is implemented as a look-up where $\mathbf{h}'_t$ are the indices. The orange blocks are element-wise operation blocks where the output of the addition operation is rescaled to h_q bits.

Table 4.4: Quantization parameters defining a quantization search space to be used on three TinyCowNets selected from the random search. While the weight bit-width w_q and h_q are selected arbitrarily and independent, b_q is dependent on the former because of the relationship with matrix multiplications in the recurrent layers.

Param.	Range	Description
w_q	6, 7 ... 10 12,14 ... 16	Weight bit-width
h_q	6, 7 ... 10 12,14 ... 16	Output / Activation bit-width
b_q	$3w_q$	Bias bit-width

vectors $\mathbf{h}_t$ and $\mathbf{c}_t$ are quantized with h_q . The S_{b_q}/S_{h_q} term rescales the added matrices to h_q for activation function $\tanh(x)$. This hyperbolic tangent and the sigmoid use, possibly with interpolation, a Look-Up Table (LUT) with size 2^{h_q} . As such, rescaled $\mathbf{h}'_t$ is used as an index of this LUT. The size and precision of this table are other important factors for performance and memory. The architecture search space already includes hundreds of thousands of possibilities. Therefore, we only consider the three best-trained candidates from the random search for the quantization scheme to limit the compute required for dynamic quantization.

Table 4.4 shows the quantization search space with b_q , w_q and h_q . The bias, $\mathbf{b}$, needs sufficient precision as it is involved with the element-wise addition with the two other terms in Eq. (4.9). Therefore, we select this precision as $b_q = 3w_q$ since the number of weights, especially for the GRU and LSTM layers, is large. This b_q has the smallest influence over the memory occupancy, because it is only involved in the addition of vectors with size n_f (layer width) and rescaled to h_q . Nonetheless, the inclusion of w_q and h_q with ranges of 13 steps in the random search, would increase the involved search space by $169\times$. In other words, performing quantization before random search drastically enlarges the required computation load as the unique combinations increases from 300 thousand to 50 million. In addition, as quantization injects noise into the dataset, the training process would be further complicated. Therefore, we use a separate grid search to find bit-width combinations that yield a minimum loss of accuracy after training. We define this loss as the absolute distribution error, the absolute difference between the quantized model and TensorFlow model output vectors using the test dataset. Next to this, we consider the minimum memory and energy consumption to balance resources and performance. Starting from the three best trained candidates given by the random search, a total of 507 quantized models are generated and tested on the test dataset.

4.8.3 Memory Estimation for the quantized Models

Eqs. (4.30)-(4.33) are transformed to include the two parameters w_q and h_q , and determine the memory in bits for all layers, i.e.,

$$P_{\text{qsRL}} = w_q(n_f^2(2N - 1) + 3n_f) + b_q n_f N \quad (4.42)$$

$$P_{\text{qLSTM}} = w_q(n_f^2(8N - 4) + 12n_f) + 4b_q n_f N \quad (4.43)$$

$$P_{\text{qGRU}} = w_q(n_f^2(6N - 3) + 9n_f) + 6b_q n_f N \quad (4.44)$$

$$P_{\text{qll}} = 4w_q n_f + 4b_q \quad (4.45)$$

w_q and b_q are 32-bit floating-point for vanilla TensorFlow models [263]. Furthermore, the memory required by the hyperbolic tangent and sigmoid LUTs for layer type k is

$$P_{\text{qact},k} = \begin{cases} 2^{h_q-1}, & \text{if } k = \text{sRL} \\ 2 \cdot 2^{h_q-1}, & \text{if } k = \text{LSTM} \\ 2 \cdot 2^{h_q-1}, & \text{if } k = \text{GRU} \end{cases} \quad (4.46)$$

where h_q is the output bit-width and the 2^{-1} term indicates half LUT size since the functions are both odd. In the case of a 32-bit implementation, these functions would use up the majority of resources. However, with small bit-widths, the LUTs are few and only a single SRAM access per activation is required.

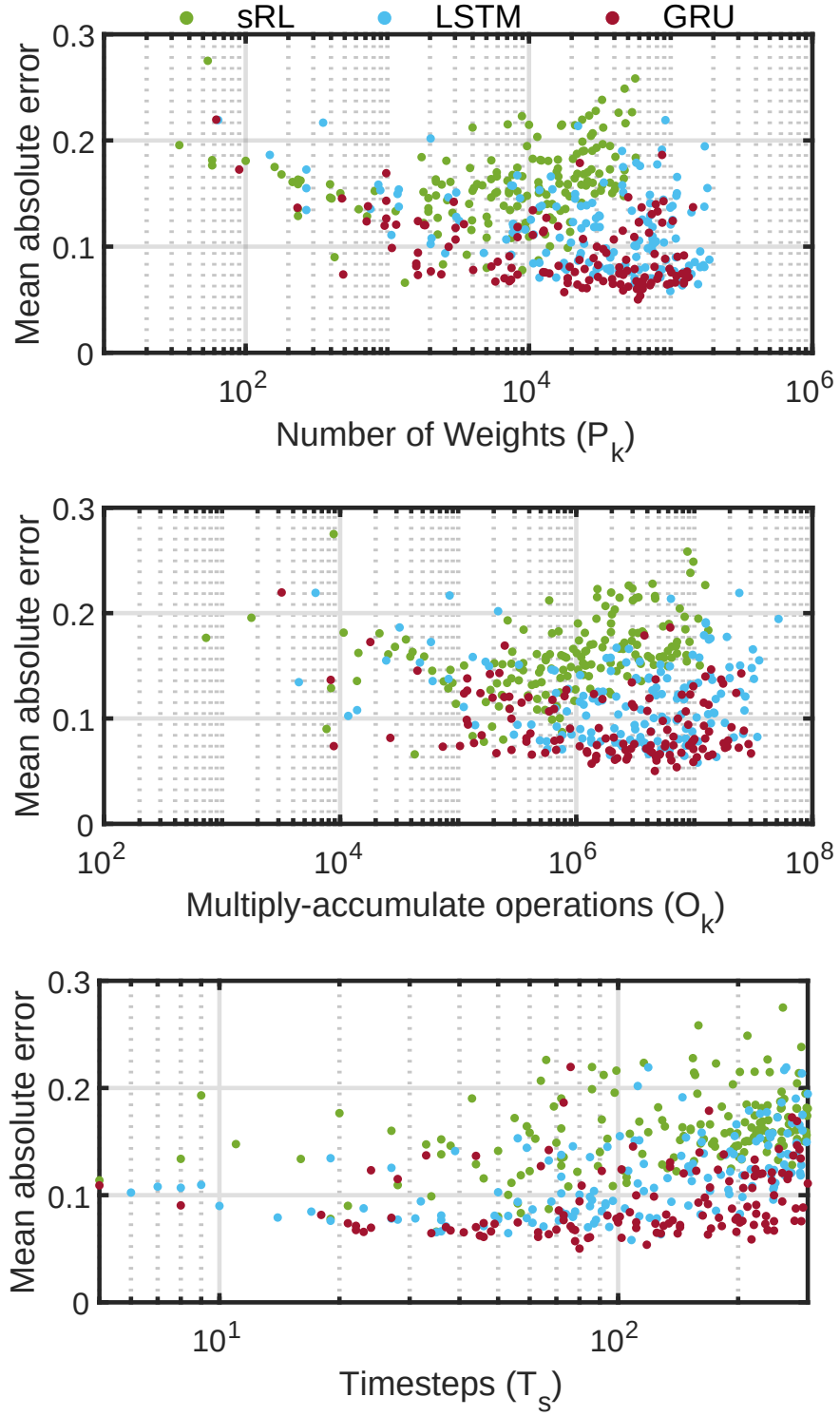


Figure 4.10: Scatterplots of the 445 TinyCowNets (parametrized RNNs) found using random search. The color denotes layer type k . Furthermore, the top, middle, and bottom row show the mean absolute error vs. the number of weights P_k , the MAC operation count O_k , and timesteps T_s .

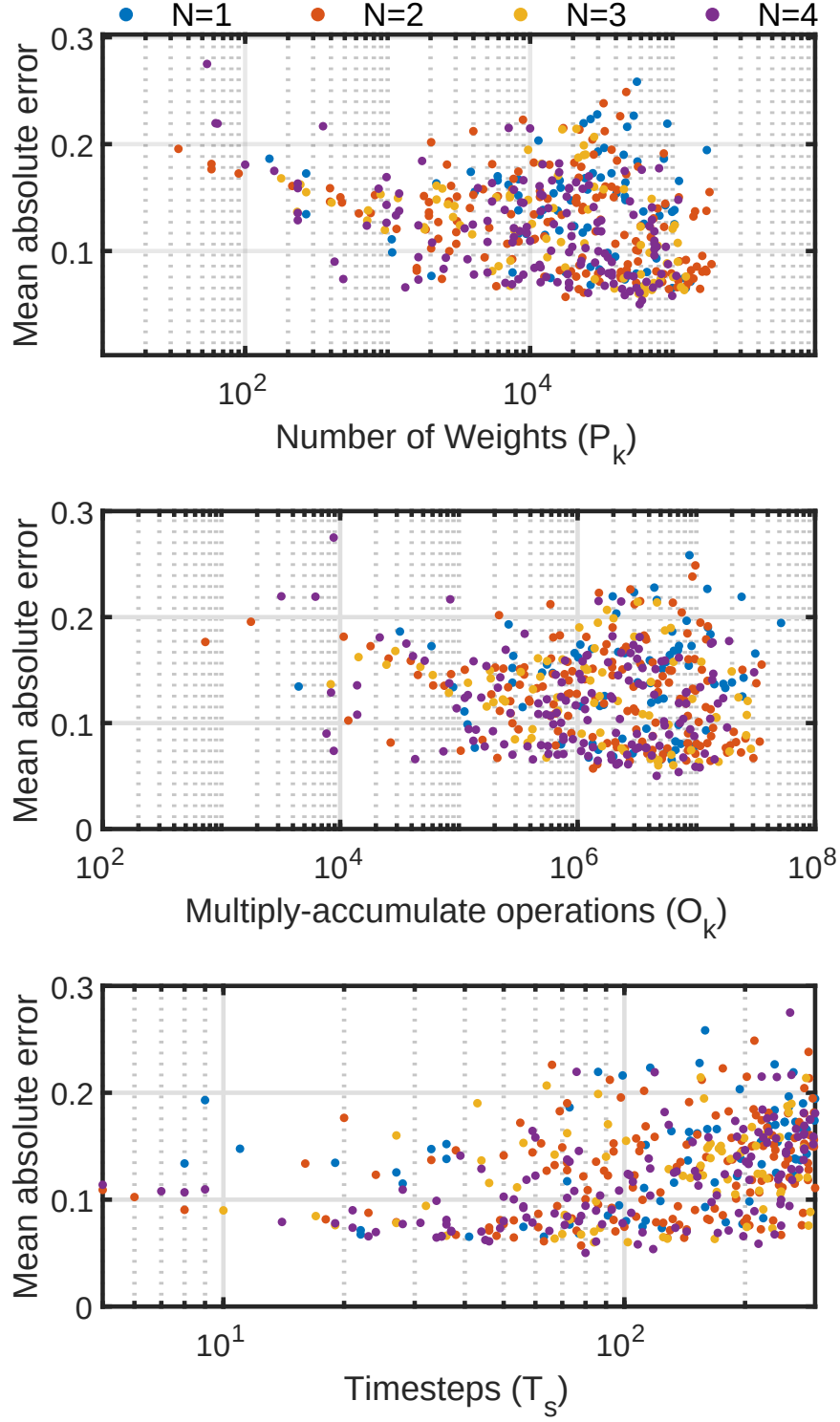


Figure 4.11: Scatterplots of the 445 TinyCowNets (parametrized RNNs) found using random search. The colors denote number of layers N . Furthermore, the top, middle, and bottom row show the mean absolute error vs. the number of weights P_k , the MAC operation count O_k , and timesteps T_s .

Table 4.5: Average Mean Absolute Error (MAE) of the best-performing 10% of found models per decade vs. the model weights P_k and MACs per inference O_k (10^1 - 10^8) with random search. This is considered with the hyperparameters for recurrent layer type k and layer count N as well as the result of the entire random search results regardless of hyperparameter (All (average)).

	Mean Absolute Error (MAE)				
Hyperparameter	Weight (P_k) Decade				
Layer type (k)	10^1	10^2	10^3	10^4	10^5
sRL	0.1765	0.1093	0.0881	0.01135	-
LSTM	0.2192	0.1345	0.0934	0.0704	0.0648
GRU	0.1726	0.0738	0.0678	0.0577	0.0658
Layer count (N)					
1	-	0.1345	0.0783	0.0711	0.0653
2	0.1723	0.1352	0.0767	0.0668	0.0659
3	-	0.1197	0.0782	0.0618	0.0635
4	0.2192	0.0819	0.0696	0.0594	0.0768
Every (average)	0.1726	0.1019	0.0728	0.0627	0.0649
Hyperparameter	MAC (O_k) Decade				
Layer type (k)	10^3	10^4	10^5	10^6	10^7
sRL	0.0900	0.0899	0.08658	0.1361	0.1503
LSTM	0.1345	0.1024	0.0733	0.0702	0.0705
GRU	0.0738	0.0733	0.0676	0.0573	0.0644
Layer count (N)					
1	0.1345	0.1339	0.0783	0.06758	0.1091
2	0.1956	0.0816	0.0789	0.0657	0.0678
3	0.1366	0.1286	0.0743	0.0622	0.0634
4	0.0738	0.0696	0.0695	0.0588	0.0620
Every (average)	0.0738	0.0736	0.0722	0.0625	0.0647

Table 4.6: Maximum number of 32-bit weights and MACs per timestep when assuming a single multiplier unit on three low power FPGAs and one MCU as described in Table 4.3. In light of these resources, the best MAE on the validation dataset of an implementable TinyCowNet (described in Table 4.7) from the random search results is indicated.

Device	Max MACs per timestep	Max weights (32-bit)	Best possible MAE
UP5K FPGA [D]	1,920,000	35,750	0.0571
STM32L07 MCU [C]	1,280,000	17,000	0.0660
AGLN250 FPGA [B]	10,000,000	1,175	0.0738
UL640 FPGA [A]	1,920,000	1,775	0.0660

Table 4.7: Number of weights, total MACs per inference, timesteps, recurrent layer count and recurrent layer type of the best implementable 32-bit floating-point TinyCowNets for each of the four devices considered (Table 4.3, 4.6).

Best Model Per Device	Model Details				
	Weights (P_k)	Total MACs (O_k)	Timesteps (T_s)	Layers (N)	Layer Type (k)
[D]	17,720	1,348,480	78	2	GRU
[C],[A]	1330	42,822	35	4	sRL
[B]	488	8836	21	4	GRU

Table 4.8: The independent accuracy for each of four cow behaviors, i.e., resting (RES), rumination (RUM), moving (MOV) and eating (EAT) on the test dataset is described as determined by optimal ROC threshold of the best implementable 32-bit floating-point Tiny- CowNets for each of the four devices considered.

Best Model Per Device	Accuracy per Behavior on Test Dataset			
	RES	RUM	MOV	EAT
[D]	96.71%	99.72%	99.57%	96.90%
[C],[A]	96.16%	98.87%	98.67%	96.37%
[B]	96.75%	98.67%	99.10%	97.24%

4.9 Results of TinyCowNet with Random Search and Quantization

For each of the 445 iterations of random search, three TinyCowNets were trained for 150 epochs subject to stratified splitting. Overall, this process took a compute time of 859 hours on a Xeon E5-2680 v4 2.4GHz core, averaging 116 minutes per iteration [264].

4.9.1 Weight, MAC and Timesteps Analysis

Fig. 4.10 shows the Mean Absolute Error (MAE) of 445 random search iterations in six scatter plots. The number of weights, P_k , is strongly inversely related to the MAE, which decreases near-logarithmically over the first two decades (10^1 - 10^3). Furthermore, four- and two-layer models seem to be most MAC-efficient. In addition, it seems that sRL networks have increasing error for MACs and number of weights above 10^6 and 10^4 , confirming the vanishing-exploding gradient problem that makes large sRL models harder to train.

Of the in total 445 models, 148 have an MAE equal to or lower than 0.1. Within this error bound, 85 or 59% are of the layer type GRU followed by 55 LSTM and only 8 sRL models. Similarly, 48 and 56 of all the models within this region have two or four layers, representing the majority. The bottom row of Fig. 4.10 shows the best models to maintain similar accuracy with changing T_s . In other words, there is a weak relationship between the timesteps and the MAE.

We statistically analyzed the models within each decade’s 10% lowest average MAE in terms of number of weights and MACs. Table 4.5 describes the results of this analysis. Supporting our earlier claim, the MAE decreases by 0.0707 for the initial weight decade, the highest drop within all decades. Furthermore, the MAE slightly increases over the last decade, indicating a possible lack of training epochs. The best performing models are GRUs at the lowest average MAE of 0.0557 and 0.0573 within 10^4 weights and 10^6 MACs. Also, GRU represents the best models at low decades, i.e., an average MAE of 0.0738 within 10^2 weights and 10^3 MACs. The second block of Table 4.5 shows the layer count vs. MAE. The best-performing models tend to be those with four layers. In other words, the best models in terms of resource and accuracy are four-layer GRUs. This is in line with Ref. [238].

Three TinyCowNets are selected that are smallest or have best performance for a validation dataset accuracy above 93%. The number of weights, MACs, timesteps and layers, the performance on the test dataset for each behavior is described from the optimal Receiver Operating Characteristic (ROC) threshold.

4.9.2 32-bit Models on low-power MCU and FPGA

Table 4.6 shows the maximum number of weights and MACs for each of the devices in Table 4.3. In addition, the MAE of the best implementable model found in the random search is described. The maximum MACs are calculated by dividing the max operating frequency with the accelerometer frequency at 25Hz. In other words, for FPGA, this order-of-magnitude estimate simply assumes that at least one MAC unit fits the device. This MAC count should be at least more than the term factorized by T_s in Eqs. (4.26)-(4.29). We set this limit to ensure that the network can be implemented even with a single multiplier-accumulator module. Table 4.7 shows the model weights, MACs, timesteps. Table 4.8 describes the best accuracy determined

from the ROC curve independently for each behavior with the test dataset. Each of these models is the best model that could be implemented on any of the four devices. The average accuracy as in Eq. (4.25) is 95.7% for model [D], 95.2% for model [C], [A], and 95.3% for model [B].

4.9.3 Minimum memory and quantized arithmetic for ASIC Implementation

The above models use 32-bit floating-point arithmetic, which requires complex logic and is wasteful of memory because of redundant precision. Fig. 4.12 shows a scatterplot of all models below an MAE threshold of 0.07 with number of weights P_k . The three selected candidates subject to quantization are indicated with a cross and described in Table 4.9. The sRL shows an MAE of 0.0669, and the sGRU and bGRU have an MAE of 0.0671 and 0.0502. As these are the best GRU, smallest sRL, and smallest GRU, we shall call these models bGRU, bsRL, and sGRU. The bsRL is the same model for implementation on MCU [C] and FPGA [A]. Below 0.07 MAE, most models are GRU with many MACs, making these impossible to implement on low-power devices with 32-bit floating-point precision. As we shall show, our dynamic full-integer quantization allows all models to be implemented on low-power FPGAs and MCUs.

Fig. 4.13 shows the Absolute Distribution Error (ADE) vs. the weight and output bit-width for integer-quantized bsRL, sGRU, and bGRU. Differently from Section 4.7.3, this ADE is the absolute mean difference of the estimated floating-point and full-integer distribution output vector entries. The output bit-width, h_q , influences this error most, as it controls the precision of the activation functions and recurrent outputs. On the other hand, the weight bit-width w_q , which controls the quantization of $\mathbf{U}$ and $\mathbf{W}$, ceases to improve around a bit-width of 10. The bsRL has the lowest ADE, followed by the sGRU and bGRU. We posit this is because of the difference in size while the quantization search space is equivalent. Therefore, larger models incur more operations, resulting in a higher ADE from weight quantization error. Furthermore, the bias bit-width b_q , set to a value of $3w_q$, might be insufficient for larger models.

Fig. 4.14 shows the ROC curves of these three models that we have selected from the above discussion. These curves show the sensitivity and specificity in false and true positive rates as a function of different thresholds for each behavior versus all others. Although our cow dataset is highly imbalanced (6:3:3:1), all curves approach

Table 4.9: The number of weights, MACs, timesteps and layers of three selected TinyCowNets that are smallest or have best performance for a validation dataset accuracy above 93%.

Models <0.03 MAE	Model Details			
	Weights (P_k)	Total MACs (O_k)	Timesteps (T_s)	Layers (N)
Best/Smallest sRL (bsRL)	1,330	42,822	35	4
Best GRU (bGRU)	58,712	4,630,288	80	4
Smallest GRU (sGRU)	5,788	210,768	38	2

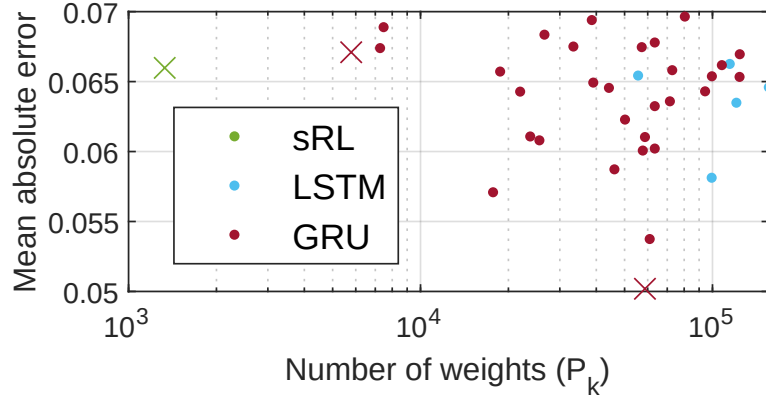


Figure 4.12: Scatterplot of all TinyCowNets from the random search with an MAE below 0.07. The three models plotted with a cross indicate the candidates subject to full-integer quantization and ASIC resource estimation.

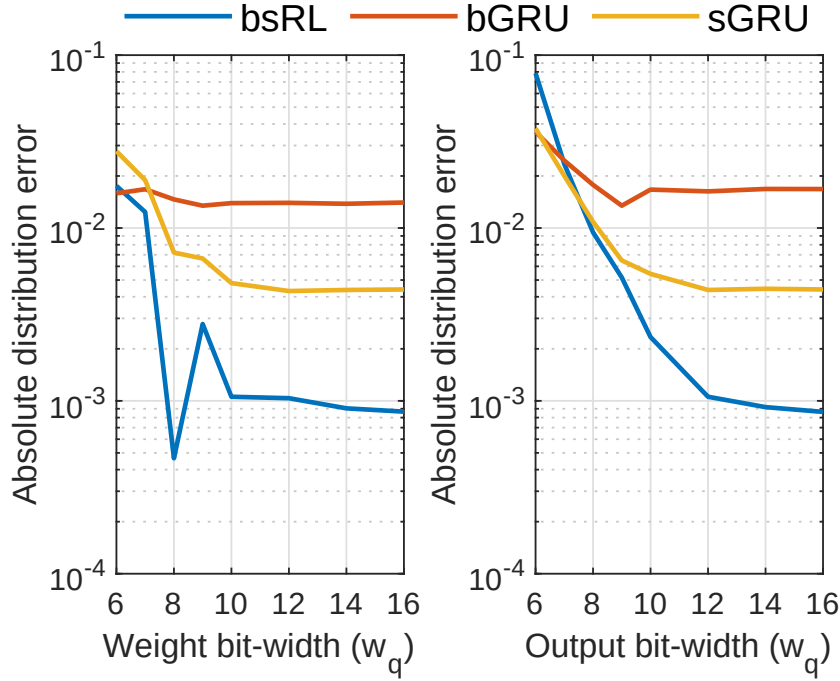


Figure 4.13: Absolute distribution error vs weight bit-width and output bit-width, which is the error between the full-integer and floating point counterparts for the bsRL, bGRU and sGRU models, described in Table 4.9.

Table 4.10: The accuracy per behavior of three selected TinyCowNets that are smallest or have best performance for a validation dataset accuracy above 93%.

	Accuracy per Behavior on Test Dataset			
Models <0.03 MAE	RES	RUM	MOV	EAT
Best/Smallest sRL (bsRL)	92.81%	98.5%	94.5%	98.1%
Best GRU (bGRU)	94.9%	99.3%	96.7%	98.90%
Smallest GRU (sGRU)	92.40%	98.5%	95.2%	98.1%

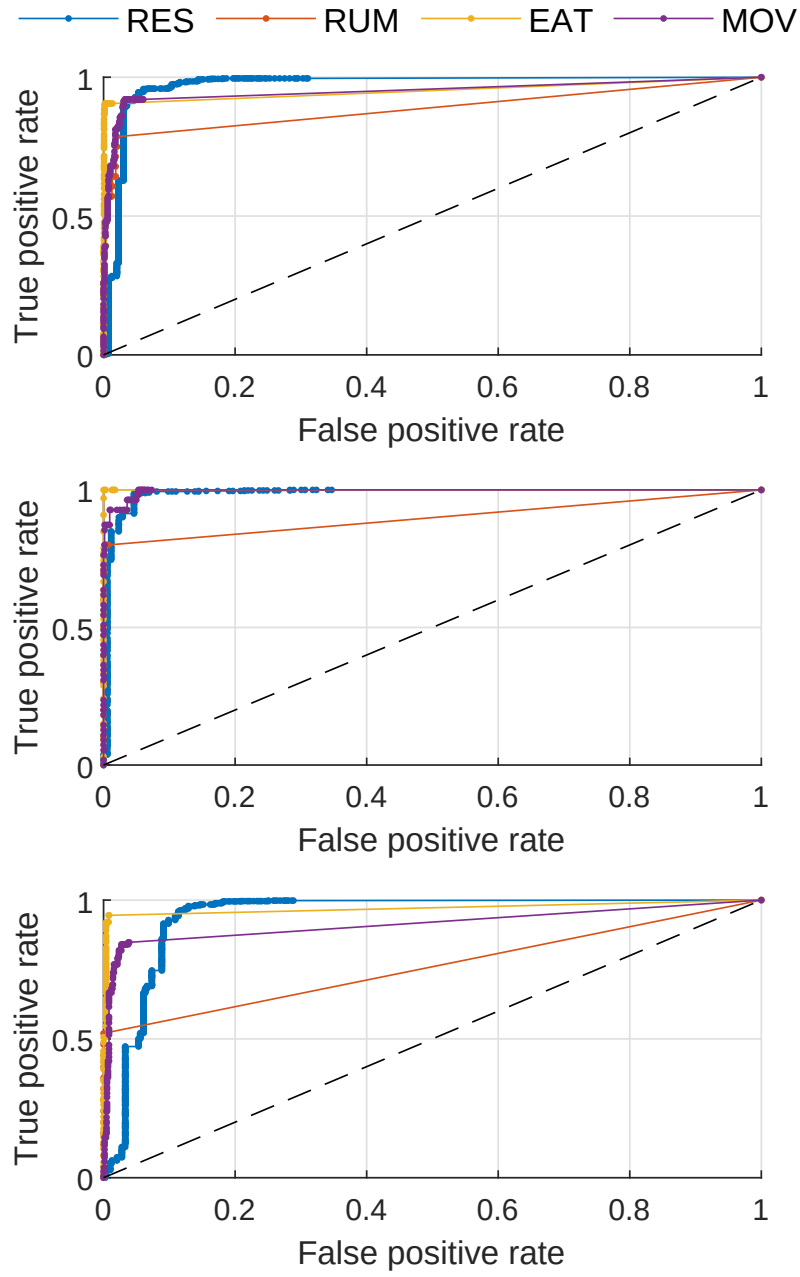


Figure 4.14: The Receiver Operating Characteristic (ROC) curves of the three models; best sRL, best GRU and smallest GRU from top to bottom.

Table 4.11: Characteristics of the selected best-performing quantized models, i.e., memory in kB and accuracy vs. weight bitwidth w_q and output bit-width h_q .

Quantized models	Quantization bit-widths		Memory [kB]	Test Accuracy
	Weight (w_q)	Output (h_q)		
bsRL	10	12	2.043	95.2%
bGRU	9	9	67.47	97.4%
sGRU	10	12	8.351	95.6%

Table 4.12: Matthews Correlation Coefficient (MCC) and Area Under Curve (AUC) per behavior from the ROC curves of Fig. 4.14

Quantized models	Matthews correlation coefficient	Area Under Curve (AUC)			
		RES	RUM	MOV	EAT
bsRL	0.86	0.97	0.89	0.95	0.95
bGRU	0.93	0.99	0.90	1.00	1.00
sGRU	0.88	0.94	0.76	0.92	0.97

the vicinity of the left-upper corner and reach an Area Under Curve (AUC) near unity, indicating a high accuracy across all behaviors.

Tables 4.11 and 4.12 further describe these models. While the bGRU has the highest overall accuracy at 97.4%, the bsRL and sGRU models have an accuracy of 95.2% and 95.6%. However, the estimated memory consumption of the bsRL is roughly 33 times lower than the bGRU and 4.1 times lower than the sGRU. Next to this, the bsRL, bGRU, and sGRU models report an MCC of 0.86, 0.93, and 0.88, showing that these three models distinguish well between behaviors regardless of dataset imbalance. This is further reinforced with the AUC at an average 0.94, 0.97 and 0.90 over all behaviors for the bsRL, bGRU and sGRU models. Because of its low estimated memory, we think that the bsRL is the best option for future ASIC creation or implementation on the embedded devices ([A]-[D]).

4.10 Discussion and Future Work

Future implementation of TinyCowNet on embedded devices could help towards improving farm efficiency by reducing diseases, enhancing feed control, and automating fertility management. These three areas have a potentially significant impact on gas emission reduction of cattle farms [234, 265, 266]. This Chapter identified the issues of current Edge AI animal behavior estimation literature and proposed to solve these with cow behavior distribution regressing RNNs titled TinyCowNet. Random Search and integer quantization were introduced to create minimal but accurate edge-implementable TinyCowNets. As a result, TinyCowNet is implemented in a proof-of-concept on the Lattice ICE40UP5K FPGA using the proposed resource-minimized hardware architecture for RNNs introduced in the next Chapter, Chap-

ter 5. This implementation shows a 360 μW power consumption and an accuracy of 95.2%, verifying the methods of this Chapter in succeeding to create tiny cow nets that can be extended also to other IoT or Edge-AI applications in the future.

Chapter 5

An Integer-Only Resource-Minimized RNN on FPGA for Low-Frequency Sensors in Edge-AI

In this Chapter, I present a resource-minimized RNN hardware architecture based on an extended quantization scheme, described in Chapter 4, for integer-only RNNs that reduces the number of fixed-point multiplications to one third. This architecture, referred to as Time- and Layer-Multiplexed (TLM) RNN enables implementation on heavily-constrained, low-cost FPGAs, such as the Lattice ICE40UP5K (Lattice Semiconductor Inc., Hillsboro OR). While the Lattice ICE40UP5K FPGA serves as a primary example throughout this thesis, it has to be indicated that other low-resource FPGAs, including the Gowin LittleBee (Gowin Semiconductor Corp., Guangzhou Guangdong, CN) and Efinix Trion (Efinix, Inc., Cupertino CA), are equally suitable for implementing the proposed architecture. I employ the ICE40UP5K FPGA to demonstrate the proposed architecture on three models, including a pre-trained model on cow behavior estimation, and assess them using the prior described measurement setup. During the evaluation, the internal clock frequency is adjusted within the range of 145 kHz to 12 MHz and a corresponding substantial increase in power consumption is observed. Based on this observation, I derive an expression that establishes a minimum clock frequency, ensuring that the RNN completes timestep processing just before the arrival of the next sensor input. Applying this expression to the implementation of the cow behavior model (model c) that receives input from a 25 Hz accelerometer, I achieve a power consumption of 360 μ W at a clock frequency of 146 kHz and an energy per inference of 516 μ J. On the other hand, setting this frequency to the maximum of 12 MHz, leads to a power consumption of 2.3 mW and an energy per inference of 44.3 μ J. This effectively demonstrates the applicability of our approach to precision livestock farming with sensors, as I present, to the best of my knowledge, the most efficient implementation of animal behavior estimation, a low-frequency AIoT application, using an RNN (top-1 accuracy >95%) at the time of writing. Furthermore, all of this is performed on an FPGA that is publicly available at a typical cost of roughly 10\$ USD. The code, written with Python and SystemVerilog, and a guide for using this RNN, starting from a Tensorflow RNN model to FPGA implementation, is available in Ref. [84].

Based on an integer post-training quantization for convolutional neural networks [262], Ref. [267] and [261] proposed an approach for quantizing RNNs. This quantization scheme is extended by introducing sharing of quantization scales. In essence, this scheme is based on converting any real number into an unsigned integer

$$r = S(q - Z) \quad (5.1)$$

where r is a real number, q is an integer, and S and Z are the scale and zeropoint, described by

$$S = \frac{|\max(\mathbf{M})| + |\min(\mathbf{M})|}{2^{q_{\text{bits}}} - 1} \quad (5.2)$$

$$Z = \left\lfloor \frac{-\min(\mathbf{M})}{S} \right\rfloor \quad (5.3)$$

where $\mathbf{M}$ is a matrix of any size, and q_{bits} is the bit-width.

5.1 Methods

Let the system equation of the simple RNN be

$$\mathbf{h}_t = \tanh(\mathbf{b} + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t) \quad (5.4)$$

where $\mathbf{b}$, $\mathbf{U}$ and $\mathbf{W}$ are the trained bias, hidden and input weight matrices respectively, and $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ are the hidden vector at time $t - 1$ and the input vector at time t [94]. Using the methods in Ref. [261, 267] and omitting the scale, the above matrix terms are quantized to

$$\mathbf{U}'_q = (\mathbf{U} - Z_U)(\mathbf{h}_{t-1} - Z_h) \quad (5.5)$$

$$\mathbf{W}'_q = (\mathbf{W} - Z_W)(\mathbf{x}_t - Z_x) \quad (5.6)$$

$$\mathbf{b}_q = (\mathbf{b} - Z_b) \quad (5.7)$$

where Z_i is the zeropoint, i representing the corresponding matrix. Including the above in eq. 5.4 with their respective scales leads to

$$\mathbf{h}_{t,q} = \tanh\left(\frac{1}{S_T}(S_b\mathbf{b}_q + S_US_h\mathbf{U}'_q + S_WS_x\mathbf{W}'_q) + Z_T\right) \quad (5.8)$$

where S_i is the scale, i representing the corresponding matrix, and $\tanh$ is the hyperbolic tangent. S_T and Z_T cast the summed up term into a lower bit representation [262, 267].

Scales are decimals, represented with floating point or fixed point numbers. Operations using these representations require more logic to implement than integer-based operations [268]. I propose the sharing of these scales between matrices, and between input-output vectors after discovering that the resulting model error is negligible, as matrix values are bounded within a similar range post-training. Consequently, eq. 5.8 reduces to

$$\mathbf{h}_{t,q} = \tanh(M(\mathbf{b}_q + \mathbf{U}'_q + \mathbf{W}'_q) + Z_T) \quad (5.9)$$

$$M = \frac{S_MS_v}{S_T} \quad (5.10)$$

$$S_M = \frac{|\max(u_{\max}, w_{\max})| + |\min(u_{\min}, w_{\min})|}{2^{q_{\text{bits}}} - 1} \quad (5.11)$$

$$S_v = \max(S_x, S_h) \quad (5.12)$$

$$S_b = S_M S_v \quad (5.13)$$

where $\{u, w\}_{\{\max, \min\}}$ are the maximum and minimum matrix elements of $\mathbf{U}_q$ and $\mathbf{W}_q$, and M is a multiplicative decimal term, reducing non-integer multiplications to one third. This multiplication is performed with a multiply-and-shift operation,

$$M_{\text{shift}} = -\lceil \log_2(M) \rceil + (q_{\text{bits}} - 1) \quad (5.14)$$

$$M_{\text{int}} = \lfloor M \cdot 2^{M_{\text{shift}}} \rfloor \quad (5.15)$$

$$M = M_{\text{int}} \gg M_{\text{shift}} \quad (5.16)$$

where q_{bits} is the bitwidth, and M_{int} and M_{shift} are integers, respectively. Here, $\gg$ represents a bitwise right shift operator. The scales of $\mathbf{x}_t$ and $\mathbf{h}_t$, S_x , S_h , are shared, set based on the domain of the hyperbolic tangent, $D_{\tanh} \in [-2, 2]$. This method can be extended to other RNN types, long-short-term-memory and the gated recurrent unit [95, 96].

Table 5.1 describes the RNN model parameters of three models that are considered for FPGA implementation in subsequent Sections. Using the proposed scheme, each layer undergoes separate quantization, while input-output vector quantization across the model remains constant. Additionally, each RNN layer exhibits an equal width, N . The final RNN layer is followed by a single feed-forward layer, mapping the final hidden output vector at $t = T_s$, $\mathbf{h}_{T_s, q}$ to class prevalence, $\hat{\mathbf{y}}$, in a process referred to as many-to-one classification. Further details regarding the model structure is described in Chapter 4.

5.2 Resource-Minimized Hardware Architecture

Fig. 5.1 shows three RNN architectures, Mass Parallelized (MP), Time-Multiplexed (TM), and TLM, employing a varying number of Neuron Processing Units (NPU). Fig. 5.2 shows this NPU, comprising of five modules that are employed to process a single time step of the proposed RNN. Additionally, the NPU can implement a linear layer, mapping features from the RNN to classified labels. The MP architecture aims to maximize throughput by implementing an NPU at each time step and layer, resulting in the least required NPU cycles per inference. However, this architecture demands a significant amount of resources, rendering it unsuitable for heavily resource-constrained FPGAs. The MP architecture is better suited for large-scale FPGAs, such as the Xilinx Virtex (Xilinx, Inc., San Jose CA) or Intel Stratix series (Intel Corp., Santa Clara CA), frequently utilized in accelerators [269]. In contrast, the TM architecture offers a scaled-down approach, reducing the employed NPUs to the number of layers. Consequently, the speed becomes a function of the number of time steps. This architecture is appropriate for intermediate scenarios, where throughput remains crucial, but the considered FPGA for use contains moderate resource constraints. The focus of this work is the TLM that minimizes resource usage and allows implementation on an extremely-small FPGA. This architecture necessitates a single NPU for constructing any RNN according to the previously discussed architectural setting, namely, an equal RNN layer width followed by a single linear layer. In this case, the processing speed becomes a function of the number of time

Table 5.1: Parameters of models used for power measurements with internal clock frequency in Section 5.4 (a,b,c), and cow behavior proof of concept in Section 5.5 (c).

Parameter	Model			Description
	Experimental		Cow Behavior	
	a	b	c	
N	20	10	13	Layer width
L	4	2	4	Number of layers
I	6	3	3	Number of inputs
O	8	4	4	Number of classes
q_{bits}	8	8	8	Bitwidth
T_s	50	50	35	Timesteps

steps and layers, increasing the number of clock cycles compared to MP and TM architectures. However, as will be demonstrated in Section 5.4, this does not pose a challenge when input data originates from sensors with sufficiently low sampling frequency.

Within this architecture, the single NPU stores the intermediate recurrent hidden output vectors, $\mathbf{h}_{t,q}$ for the next layer and timestep in memory. I propose to implement the TLM architecture with a real-time approach, i.e., the internal clock frequency is set as a function of the architectural parameters that influence the number of clock cycles for a single timestep of the RNN to be completely processed by the NPU, i.e.,

$$f_{\text{in}} = p_{\text{cycles}} \cdot f_{\text{sensor}} \quad (5.17)$$

where f_{in} is the internal clock frequency, p_{cycles} is the number of clock cycles for processing a single timestep (eqs. 5.18-5.20), and f_{sensor} is the sampling frequency of the sensor. This approach leads to a clock frequency on FPGA that is minimized using a clock divider under the constraint that the processing completes before the next sensor sample arrives.

5.3 A Single Neuron Processing Unit

The subsequent sections will present the implementation of the TLM architecture on the Lattice ICE40UP5K FPGA, encompassing Intellectual Property (IP) modules. This FPGA is extremely compact (smallest package measures 5.38 mm²), featuring 5280 4-bit Look Up Tables (LUT) and 1-bit registers, as well as 30 Embedded Block RAMs (EBR) consisting of 4096 bits each at a core voltage of 1.2 V. At a typical price <10 USD, this FPGA is an example of low-cost programmable hardware that is heavily constrained in terms of resources. Other similar examples suitable for implementation include the Gowin LittleBee and Effinix Trion FPGAs. The implementation incorporates five IPs: algorithmic IPs, such as Adders, Subtractors, and Multipliers, and Random-Access Memory (RAM) and Read-Only Memory (ROM) IPs, which are realized using EBRs. The source code, accompanied by a comprehensive guide for adapting the implementation to a customized model on Lattice

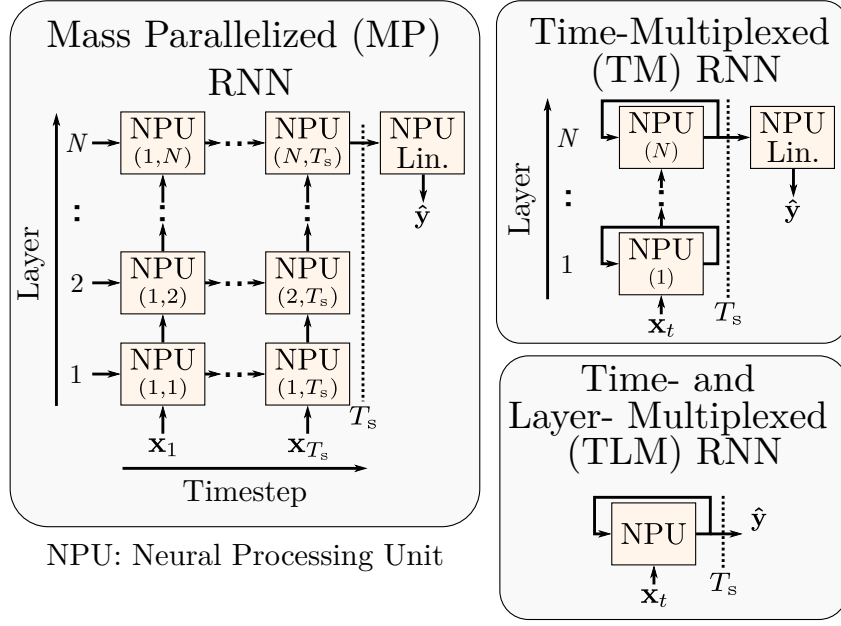


Figure 5.1: Mass Parallelized (MP), Time-Multiplexed (TM), and Time- and Layer-Multiplexed (TLM) RNN architectures. MP signifies the most rapid architecture with the minimum number of clock cycles required for inference, allocating an NPU to each layer and timestep. TLM employs a single NPU, minimizing resources while necessitating the greatest number of clock cycles for processing.

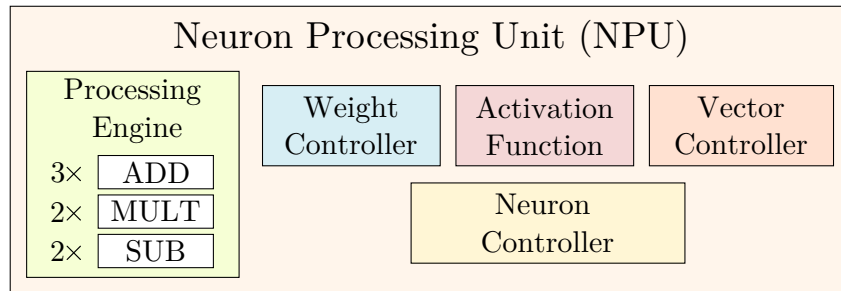


Figure 5.2: Neuron Processing Unit (NPU), containing five modules, Processing Engine, Activation Function and Weight, Vector, and Neuron Controllers. Expanded modules are shown in Fig 5.3.

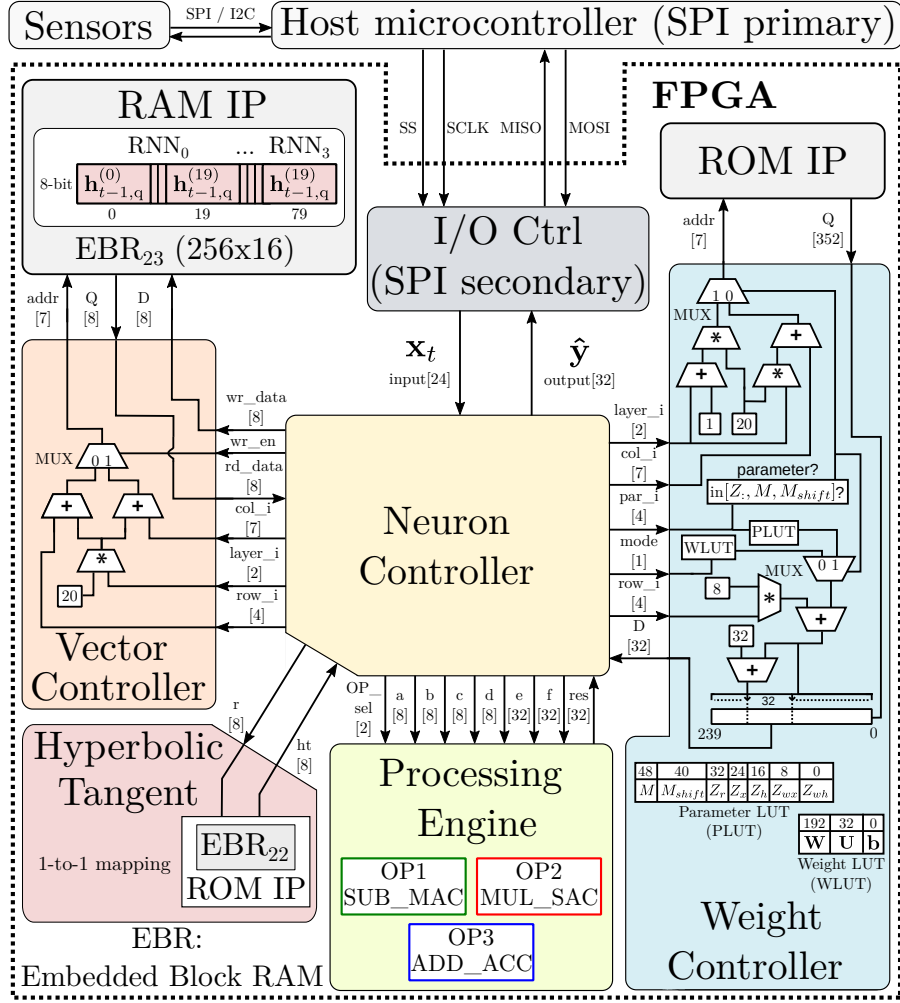


Figure 5.3: Architectural overview of the Neuron Processing Unit (NPU) on FPGA at 8-bit bitwidth, implementing model a (parameter values described in Table 5.1). The structure of the weight memory (ROM IP) and details of the Processing Engine are shown in Fig. 5.5 and 5.6, respectively. Clock, reset and enable signals are omitted.

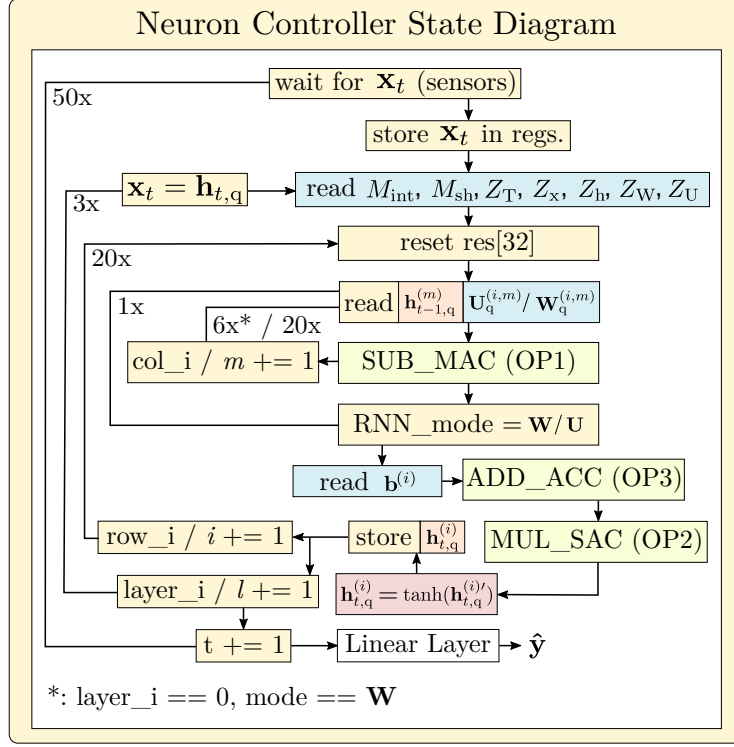


Figure 5.4: State Diagram of the Neuron Controller for Recurrent Neural Network inference on model a, described in Table 5.1.

FPGAs and an overview of IPs to modify in the source code for compatibility with other vendors, can be found in Ref. [84].

Figure 5.3 presents an expanded NPU featuring a system-level overview that incorporates the FPGA, sensor peripherals, and an external microcontroller (parameters of model a, Table 5.1). In this configuration, the FPGA (slave) communicates with a microcontroller (master) to receive sensor data, $\mathbf{x}_t$, and transmit class predictions, $\hat{\mathbf{y}}$, utilizing the Serial Peripheral Interface (SPI). The Neuron Controller, acting as the central component of this architecture, follows a predefined Finite State Machine (FSM) that distributes instructions and data to other modules using the described wires. The following subsections provide detailed descriptions of the key modules and their functions within the proposed architecture: the Neuron Controller, Weight Controller, Vector Controller, Processing Engine, and Hyperbolic Tangent (Activation Function).

5.3.1 Neuron Controller

Fig. 5.4 shows the state diagram of the neuron controller. The states involve instructing the Processing Engine (PE) to commence operations (OP1-3), guiding the memory and vector controller to store and load elements, and incrementing the column, row, layer index and time, denoted as m , i , l and t respectively. Initially, the neuron controller stores incoming sensor data, $\mathbf{x}_t$, in registers, preceded by an instruction to read all quantization parameters M_{int} , M_{shift} and Z . Following this, two distinct modes of operation are selected with wire 'mode'.

In the first mode, $\mathbf{W}_q^{(i)'}$ is processed (eq. 5.6). The controller receives $\mathbf{W}_q^{(i,m)}$

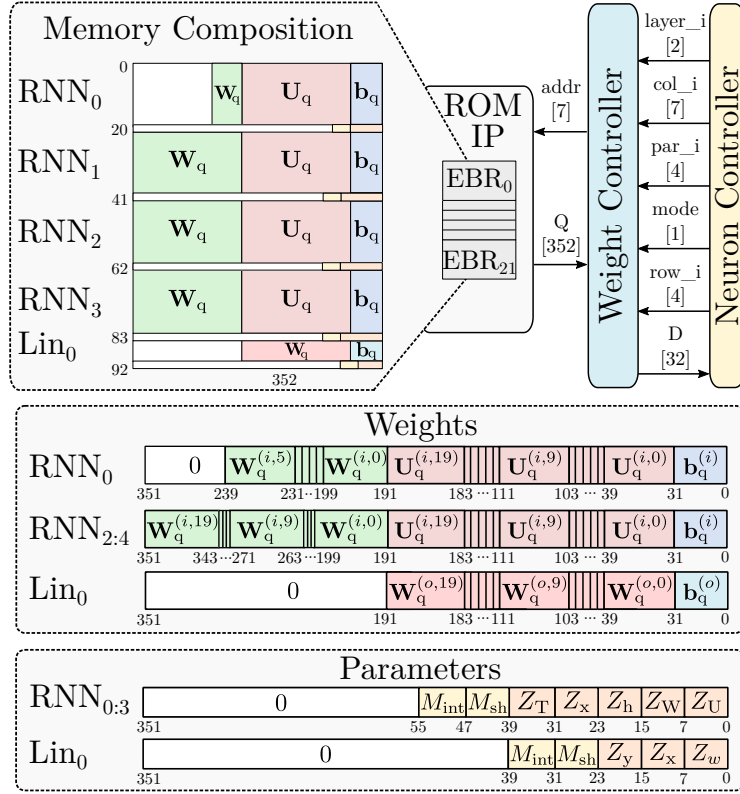


Figure 5.5: Memory composition of the weight and parameters of the RNN, layer-by-layer (RNN_{0:3}). One word in the ROM comprises a single row of the input weight and hidden weight matrices, and the bias, leading to $(2N + 4)q_{\text{bits}}$ bits per word. The final address of a layer, address $(N + 1)$, contains the quantization parameters, i.e., M_{int} , M_{shift} and zeropoints Z_i (eq. 5.4).

from the memory and forwards it along with $\mathbf{x}_t^{(m)}$ at layer $l = 0$, or $\mathbf{h}_{t,q}^{(m)}$ of the previous layer $l - 1$ for $l > 0$ to the processing engine, initiating OP1. After 6 ($l = 0$) or 20 ($l > 0$) increments of m , the second mode of the RNN commences, similarly processing $\mathbf{U}_q^{(i)'} (eq. 5.5)$ using $\mathbf{U}_q^{(i,m)}$ and $\mathbf{h}_{t-1,q}^{(m)}$. Subsequently, OP3 and OP2 are initiated, yielding $\mathbf{h}_{t,q}^{(i)'}$. This scalar is input to the hyperbolic tangent function, mapping $\mathbf{h}_{t,q}^{(i)'} \mapsto \mathbf{h}_{t,q}^{(i)}$. The row, i , is then incremented, and the aforementioned steps are repeated twenty times, resulting in the hidden output vector $\mathbf{h}_{t,q}$ at layer l . This process is repeated for all layers. Accordingly, the hidden output vector of the previous layer is locally stored in registers, i.e., $\mathbf{x}_t = \mathbf{h}_{t,q}$ for layers $l > 0$.

The neuron controller then waits for the next sensor sample to arrive and upon arrival repeats the aforementioned process for the number of timesteps set as $T_s = 50$. Once 50 timesteps have passed, the NPU acts as linear layer with a Rectified Linear Unit (ReLU) activation function, outputting classified vector, $\hat{\mathbf{y}}$.

5.3.2 Weight Controller and Structure

Fig. 5.5 shows the memory structure, encompassing the storage of weight and quantization parameters by means of ROM IP that allocates a specific number of EBRs. An EBR possesses a word length of 16 bits and a total of 256 addresses. As a result, 22 EBRs are required to store the weights of the RNN in accordance with the parameters outlined in Table 5.1, given that the RNN word length amounts to 352 bits.

The weight controller acquires information about the RNN state from the neuron controller, determined by the layer, row, and column indices (l , i , and m), and the selection of the weight matrix ($\mathbf{U}_q$ or $\mathbf{W}_q$). The address to be loaded is determined by the layer and row indices, and whether a quantization parameter or a weight is requested. Upon receiving the 352-bit word from the ROM IP, the weight controller picks a 32-bit segment to return to the neuron controller. In the case of requesting a quantization parameter, no offset is added and the forwarded data is selected based on the index stored in the Parameter Look Up Table (PLUT), selected with wire 'par.i'. In case of a weight, two scenarios transpire: 1) a bias weight, $\mathbf{b}_q^{(i)}$, is picked by taking the initial 32 bits of the 352-bit word and transmitted to the neuron controller; 2) an element of the hidden or input weight matrix, $\mathbf{U}_q^{(i,m)}$ or $\mathbf{W}_q^{(i,m)}$, is requested, and the transmitted element is selected from the word based on column index m .

5.3.3 Vector Controller

To store the hidden vectors of RNN layers, $\mathbf{h}_{t,q}$, each containing N elements, a single EBR is utilized. In this EBR, each address stores a single element, $\mathbf{h}_{t-1,q}^{(m)}$, where m is the column index. During RNN operation, $\mathbf{h}_{t-1,q}^{(m)}$ is loaded for every element multiplication involving the hidden weight matrix (eq. 5.5). Conversely, the resulting scalar $\mathbf{h}_{t,q}^{(i)}$ from a single RNN cycle in eq. 5.4 is stored based on row index i .

5.3.4 Processing Engine

Fig 5.6 shows the Processing Engine that performs matrix multiplications by means of N vector multiplications over row index i , leading to $\mathbf{h}_{t,q}$ of layer l . For a

Table 5.2: Number of 4-bit Look up tables (LUT), 1-bit registers and 4-kbit Embedded Block Rams (EBR) and % of total used by model a, b and c on the Lattice ICE40UP5K FPGA.

Architecture	LUTs	Registers	EBRs
a	3764 (71%)	645 (12%)	24 (80%)
b	2769 (52%)	947 (18%)	14 (46%)
c	3172 (60%)	717 (14%)	17 (56%)
Lattice ICE40UP5K	5280	5280	30

single vector multiplication, I propose three operations using four 8-bit registers, namely, subtract and multiply-accumulate (SUB_MAC, OP1), multiply and shift-accumulate (MUL_SAC, OP2), and add-accumulate (ADD_ACC, OP3). OP1 represents the operation of subtracting the zeropoint from matrix elements and multiply-accumulating these elements over a single row, column by column. Using registers a-b, this operation outputs an accumulated scalar, denoted as 'res'. This operation is performed twice, once per matrix, at a total of $2N$ times, resulting in scalars $\mathbf{U}_q^{(i)'} and $\mathbf{W}_q^{(i)'}$ (eqs. 5.5-5.6). Then OP3 is initiated, summing up the three intermediate vector, $\mathbf{b}_q^{(i)}$, $\mathbf{U}_q^{(i)'}$ and $\mathbf{W}_q^{(i)'}$. Finally, OP2 leads to $\mathbf{h}_{t,q}^{(i)'}$ by means of a shift-and-add of the result of OP3 with M_{int} , M_{shift} and adding Z_T .$

5.3.5 Hyperbolic Tangent

The hyperbolic tangent is a non-linear function, commonly implemented using piecewise linearization or a one-to-one mapping, stored within memory or a Look-Up Table (LUT). Here I propose to perform a 1-to-1 mapping as the range at 8-bit bitwidth is sufficiently small, i.e., $\mathbf{h}_{t,q}^{(i)'} \in [0, 255] \mapsto \mathbf{h}_{t,q}^{(i)} \in [107, 149]$. Consequently, the hyperbolic tangent is implemented using a single EBR, and the address is determined by the value of $\mathbf{h}_{t,q}^{(i)'}$, yielding $\mathbf{h}_{t,q}^{(i)}$.

5.4 Measurement Setup and Implementations

To verify the design, emulated sensor data, is transmitted to the FPGA using the measurement setup shown in Fig. C.3 from a personal computer. After T_s transmissions of $\mathbf{x}_t$, the classified vector, $\hat{\mathbf{y}}$, from the FPGA is compared with the original Python-based model.

To determine the power consumption of the implementation, I measure the voltage drop across a resistor connected in series with the core FPGA voltage of 1.2 V. The voltage measurement is initiated when an SPI message containing $\mathbf{x}_t$ is sent, resulting in a voltage increase due to RNN operation. The design for the FPGA implementation is written in SystemVerilog and parametrized, allowing customization of architectural parameters, sensor frequency, and a customized RNN weights. The code for the conversion from a TensorFlow 2.0 simple RNN model and the design files are publicly available and accompanied by a comprehensive guide [84].

Fig. 5.7a plots the power consumption of the FPGA implementations across a range of internal clock frequencies, f_{int} , from 142 kHz to 12 MHz, while Fig.

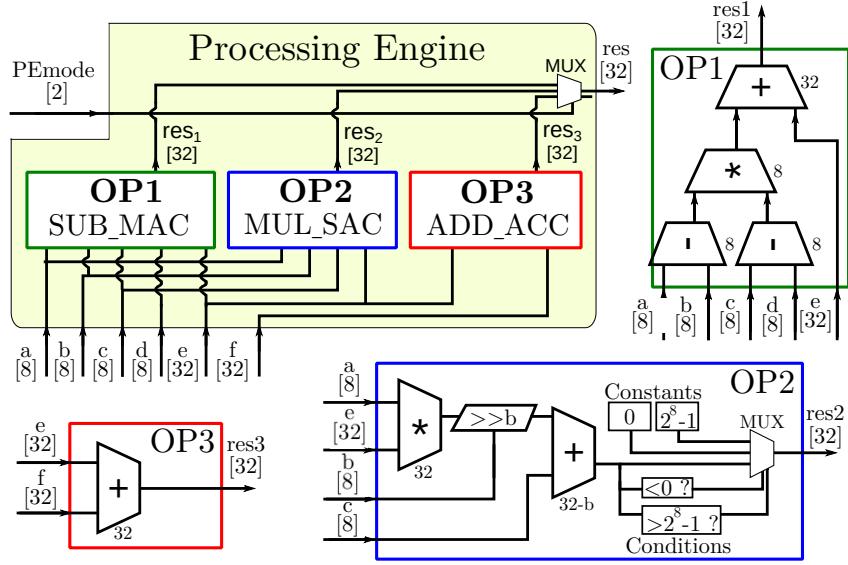


Figure 5.6: Expanded diagram of the Processing Engine and three proposed operations at 8-bit bitwidth, SUB_MAC (OP1), MUL_SAC (OP2), and ADD_ACC (OP3).

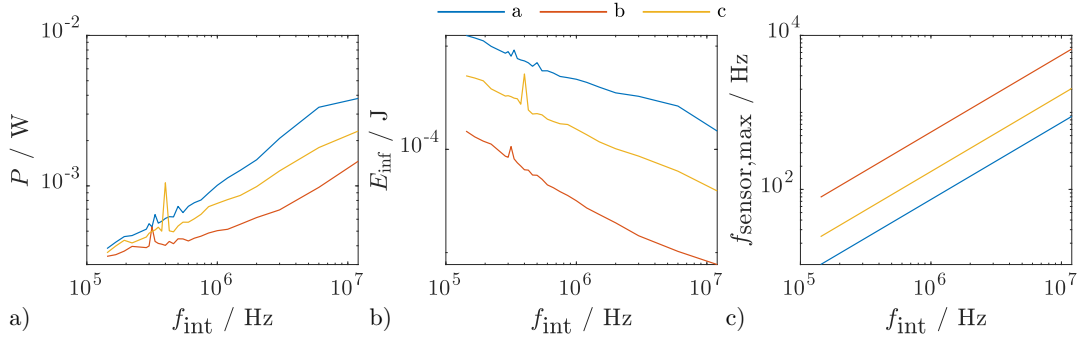


Figure 5.7: a) Measured power consumption over the internal clock frequency. b) Energy per inference over internal clock frequency. c) Maximum sensor frequency for real-time RNN operation, a function of the clock frequency and number of clock cycles (eq. 5.18-5.21).

5.7b plots the energy per inference and Fig. 5.7c plots the maximum supported sensor frequency. The power consumption exhibits a near-logarithmic increase with the internal clock frequency, spanning from 340 μW at 146 kHz (model b) to 3.81 mW at 12 MHz (model a), with verification carried out on model c. Inversely, the energy required per inference drops, spanning from 13 mW (model a) and 150 μJ (model b) at 146 kHz to 150 μJ (model a) and 7.66 μJ (model b) at 12 MHz. Table 5.2 enumerates the resources utilized by the three models. The maximum sensor frequency is determined by the input dimension (I), layer width (N), number of layers (L), and output classes (O). I analytically determined and verified in post-synthesis simulation the number of cycles per timestep as

$$p_{\text{cycles}} = p_{\text{RNN}} + p_{\text{lin}} \quad (5.18)$$

$$p_{\text{RNN}} = N^2(8L - 4) + N(14L + 4I) + I - 2L \quad (5.19)$$

$$p_{\text{lin}} = O(4N + 10) + 3 \quad (5.20)$$

where p_{cycles} represents the number of clocks for the NPU to process a single RNN timestep. Specifically, this represents the number of cycles necessary for the NPU to be able to accept the subsequent input vector from sensors, $\mathbf{x}_{t+1}$. The maximum sensor sampling frequency is determined as

$$f_{\text{sensor,max}} = \frac{f_{\text{in}}}{p_{\text{cycles}}} \quad (5.21)$$

where f_{in} can be any fraction of the 12 MHz oscillator on the breakout board using a clock divider.

5.5 Proof of Concept: Cow Behavior Estimation With a 3D Accelerometer

Utilizing the proposed TLM architecture and its implementation, I demonstrate a proof of concept by employing the RNN, model c, trained and found using the publicly available cow dataset [270] for the task of estimating cow behavior, as introduced in Chapter 4. This RNN, with 8-bit quantization for weights and 32-bit quantization for biases takes up a total of 11.70 kB of memory. This represents the smallest RNN model found by means of the neural architecture search in the previous Chapter with a >95% accuracy on the classification of four behaviors, Chapter 4. In addition, the memory required due to intermediate vectors leads to a total consumption of 11.74 kB.

5.6 Implementation Results

Table 5.1 shows the architectural parameters for the cow behavior RNN model, referred to as model c. With a layer width of $N = 13$, inputs from a 3-dimensional accelerometer at 25 Hz and 4 output coefficients ($O = 4$), model c is smaller than the RNN used for describing the architecture in Section 5.2, model a. The 4 output coefficients represent the prevalence of 4 behaviors, i.e., eating (Eat), resting (Res), rumination (Rum), and Moving (Mov). For comparison purposes with other relevant works, the top-1 accuracy is utilized, taking the behavior that has the largest coefficient in output vector $\hat{\mathbf{y}}$.

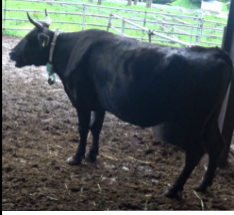
Pens:			Grassfields:		
Resting	Ruminating		Moving	Eating	
					
$\begin{bmatrix} -0.10 \\ 0.88 \\ 0.53 \end{bmatrix}$ $\mathbf{x}_0$	$\begin{bmatrix} -0.10 \\ 0.88 \\ 0.52 \end{bmatrix}$ $\mathbf{x}_1$	$\begin{bmatrix} -0.11 \\ 0.88 \\ 0.53 \end{bmatrix}$ $\mathbf{x}_{34}$	$\begin{bmatrix} -0.07 \\ 1.03 \\ 0.13 \end{bmatrix}$ $\mathbf{x}_0$	$\begin{bmatrix} -0.10 \\ 1.03 \\ 0.15 \end{bmatrix}$ $\mathbf{x}_1$	$\begin{bmatrix} -0.10 \\ 1.01 \\ 0.17 \end{bmatrix}$ $\mathbf{x}_{34}$
$\begin{bmatrix} 0.04 \\ 0.92 \\ 0.19 \end{bmatrix}$ $\mathbf{x}_0$	$\begin{bmatrix} 0.19 \\ 1.12 \\ 0.10 \end{bmatrix}$ $\mathbf{x}_1$	$\begin{bmatrix} -0.10 \\ 1.22 \\ 0.23 \end{bmatrix}$ $\mathbf{x}_{34}$	$\begin{bmatrix} 0.44 \\ 1.00 \\ 0.08 \end{bmatrix}$ $\mathbf{x}_0$	$\begin{bmatrix} 0.45 \\ 0.99 \\ 0.10 \end{bmatrix}$ $\mathbf{x}_1$	$\begin{bmatrix} 0.50 \\ 0.88 \\ 0.00 \end{bmatrix}$ $\mathbf{x}_{34}$
$\mathbf{y}$ $\begin{bmatrix} 1.00 \\ 0.00 \\ 0.00 \\ 0.00 \end{bmatrix}$	$\hat{\mathbf{y}}$ $\begin{bmatrix} 1.00 \\ 0.00 \\ 0.00 \\ 0.00 \end{bmatrix}$		$\mathbf{y}$ $\begin{bmatrix} 0.00 \\ 0.00 \\ 1.00 \\ 0.00 \end{bmatrix}$	$\hat{\mathbf{y}}$ $\begin{bmatrix} 0.00 \\ 0.00 \\ 1.00 \\ 0.00 \end{bmatrix}$	
$\mathbf{y}$ $\begin{bmatrix} 0.00 \\ 1.00 \\ 0.00 \\ 0.00 \end{bmatrix}$	$\hat{\mathbf{y}}$ $\begin{bmatrix} 0.00 \\ 1.00 \\ 0.00 \\ 0.00 \end{bmatrix}$		$\mathbf{y}$ $\begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \\ 1.00 \end{bmatrix}$	$\hat{\mathbf{y}}$ $\begin{bmatrix} 0.00 \\ 0.00 \\ 0.00 \\ 1.00 \end{bmatrix}$	
Resting + Moving			Moving + Eating		
$\begin{bmatrix} -0.09 \\ 0.19 \\ 0.45 \end{bmatrix}$ $\mathbf{x}_0$	$\begin{bmatrix} -0.09 \\ 0.20 \\ 0.45 \end{bmatrix}$ $\mathbf{x}_1$	$\begin{bmatrix} -0.06 \\ 0.21 \\ 0.49 \end{bmatrix}$ $\mathbf{x}_{34}$	$\begin{bmatrix} 0.31 \\ 0.00 \\ 0.69 \\ 0.00 \end{bmatrix}$ $\mathbf{y}$	$\begin{bmatrix} 0.33 \\ 0.00 \\ 0.67 \\ 0.00 \end{bmatrix}$ $\hat{\mathbf{y}}$	
$\begin{bmatrix} -0.22 \\ 0.12 \\ 0.43 \end{bmatrix}$ $\mathbf{x}_0$	$\begin{bmatrix} -0.20 \\ 0.12 \\ 0.43 \end{bmatrix}$ $\mathbf{x}_1$	$\begin{bmatrix} 0.02 \\ -0.48 \\ 0.38 \end{bmatrix}$ $\mathbf{x}_{34}$	$\begin{bmatrix} 0.00 \\ 0.00 \\ 0.31 \\ 0.69 \end{bmatrix}$ $\mathbf{y}$	$\begin{bmatrix} 0.00 \\ 0.00 \\ 0.22 \\ 0.78 \end{bmatrix}$ $\hat{\mathbf{y}}$	

Figure 5.8: Single behavior and mixed behavior classification scenarios of cow behavior. $\mathbf{x}_t$, $\mathbf{y}$ and $\hat{\mathbf{y}}$ describe the corresponding accelerometer data, ground truth distribution coefficients and estimated coefficients by the implemented RNN on FPGA respectively.

Table 5.3: Comparison of animal behavior estimation studies implementing algorithms in hardware, including power consumption measurements or estimations.

Reference	This work			[214]	Ch. 3	[216]	[210]
	2024			2021	2021	2019	2018
Platform	Lattice FPGA			STMicro MCU	Lattice FPGA	TI MCU	ARM MCU
Core voltage [V]	1.2			3	1.2	2	N/R
Technology node [nm]	40			90	40	130	N/R
Application	Cow			Horse	Cow	Sheep	Sow
# of Behaviors	4			3	4	5	2
Algorithm	RNN			MLP	DT	LR	KNN
Bit-width	8			32	8	16	8
Accuracy [%]	95.1			98.3	86.8	89.6	96.2
Core frequency [kHz]	146	2,000	12,000	80,000	2.9	24,000	2,000
Power [μ W]	360	988	2313	31,100	216	4300	N/R
Inf. Energy [μ J]	516	106	44.3	186,000	557	2050	542

Initially, I load the weights of the Tensorflow 2.0 model into our Python-based implementation of the shared-scale RNN. Utilizing sample cow behavior data, the Python model generates a '.mem' file that replicates the model weights in accordance with the memory composition depicted in Fig. 5.5. Subsequently, I adjust the parameters of the HDL design according to Table 5.1, point to the created memory file, and set the sensor frequency to 25 Hz. With these parameters and a sensor frequency of $f_{\text{sensor}} = 25$ Hz, the internal frequency is set to $f_{\text{int}} = 146$ kHz using eq. 5.21.

Fig. 5.8 shows six cow behavior regression scenarios and snapshots of actual camera footage. Here, $\mathbf{x}_t$ is the corresponding accelerometer data. The behavior coefficients are represented by $\mathbf{y}$ and $\hat{\mathbf{y}}$, i.e., the ground truth determined of the cow dataset and the regressed coefficients obtained from the implemented RNN on FPGA with the setup described in Section C.2.1.

Table 5.3 compares the performance of the RNN on cow behavior with works that show hardware implementations with the task of estimating animal behavior. Compared to the original RNN model on Tensorflow 2 that was described in Chapter 4, this RNN implementation shows a minimal top-1 accuracy loss from 95.2% to 95.1%. The measured power consumption of the implemented cow behavior model is 360 μW at 146 kHz, increasing to 2313 μW at 12 MHz. Although the power consumption at the lowest possible frequency, 146 kHz, is $1.66\times$ higher than the DT in Chapter 3, the accuracy increases by 8.3%. Compared to works that show implementations on an STMicro (Geneva, Switzerland) and Texas Instruments (Dallas, TX) microcontroller, our implementation consumes only 1.16% and 8.37% of reported power consumption at 31100 μW and 4300 μW [214, 216]. In contrast, these implementations show an accuracy higher, increase of 3.2%, and lower, drop of 5.5% for a 3 and 5 behavior classification respectively.

More importantly, our implementation shows a significant improvement of efficiency, expressed as the energy consumption per inference over the existing works. At 146 kHz, the energy per inference is represented by 516 μJ , requiring no further sleep or idle mode settings during operation due to clock frequency setting. This energy consumption drops down to 39.5 μJ at 12 MHz. However, additional off-board logic would be required in a real system to set the FPGA to idle or sleep mode until the next sensor sample is ready. Taking this into consideration, a 100 μA standby current of this FPGA is added, leading to a standby energy consumption of 4.8 μJ per inference. As a result, the total energy consumption at 12 MHz is 44.3 μJ . Compared to previous works, 516 μJ represents the most efficient implementation of precision livestock farming in the known literature. This is overcome by the approach of setting a maximal frequency with idle mode, leading to an energy per inference that is roughly $12.2\times$ less than the closest implementation at 542 μJ that only classifies high or low activity behavior of sows. Furthermore, compared to the Decision Tree of Chapter 3 that implements the 2 argument arctangent with a look-up table, the RNN is able to predict behaviors at far higher accuracy and an energy consumption that is $12.6\times$ less.

On the other hand, although the implementation at 12 MHz leads to an energy consumption of 44.3 μJ , the power consumption exceeds 1 mW at 2.3 mW. To balance power consumption and energy consumption, a frequency of 2 MHz can be set, leading to a power and energy consumption of 988 μW and 106 μJ respectively.

The considerable advantage in efficiency when using an FPGA for AI implementation is evident, as our proposed TLM hardware architecture applied to the

shared-scale RNN can be leveraged. As a result, this proof of concept on itself has the potential to significantly impact the field of precision livestock farming by providing a highly efficient and accurate solution for monitoring animal behavior in real-time.

5.7 Discussion

In this Chapter, I have made three significant contributions to the field. Firstly, I introduced a shared-scale, integer-only RNN that reduces the number of fixed-point multiplications to one third. Secondly, I proposed a hardware architecture designed to minimize resource utilization for implementation, referred to as TLM. The key advantage of this hardware architecture is its ability to minimize the required logic for implementation while trading off in throughput. Although this is not ideal for high-throughput applications such as machine translation and audio processing, low-frequency sensor applications remain unaffected. This distinction arises because, unlike machine translation where the input sample space is available from the onset, sensor input arrives on a sample-by-sample basis. Thirdly, I demonstrated a proof of concept by implementing the proposed TLM architecture on a highly compact FPGA, the Lattice ICE40UP5K, for a critical application in precision livestock farming. To the best of our knowledge, there is no existing integer-only RNN with a resource-minimized hardware architecture and a processing scheme tailored for sensor applications at the edge. Consequently, I believe the work described in this Chapter, accompanied by the code and guide made publicly accessible in Ref. [84], substantially advances the fields of IoT and Edge-AI by enabling multi-layer RNNs, neural networks proven to be capable of solving various complex tasks, to be implemented on nearly any FPGA at significantly high efficiencies.

The proposed RNN hardware architecture currently presents several potential improvements that I suggest exploring. At present, the RNN only supports layers with equivalent layer widths. Additionally, the input vector cannot have a size that exceeds the layer width. Adding the possibility of multiple linear classification layers, commonly used in foundation models, could be a valuable improvement. Furthermore, the current RNN can only be employed in a many-to-one setup, where the final layers hidden output vector is used by the linear output layer for classification. Enabling many-to-many, varying layer widths, and incorporating multiple linear output layers may pave the way for implementing encoder-decoder structures and natural language processing models on smaller hardware platforms, albeit with a trade-off in processing speed. For these applications, I believe the TM architecture would be more advantageous in balancing speed and resource use.

Another point of interest is the development of an ASIC from this architecture. In the case of an ASIC low operation frequency, for example, 146 kHz that was set for the proof of concept, could be exploited by allowing subthreshold operation of transistors that perform the digital logic. This could substantially lower the power consumption. An interesting future study could compare the severely diminished energy per inference due to lower power consumption of subthreshold circuits with the low energy per inference due to high frequency operation of non-subthreshold architectures.

In addition to architectural improvements, theoretical or model-based enhancements and modifications can be considered. At present, the only supported type of RNN is its simplest form. However, many applications utilize long-short term

memory or gated recurrent units. These RNN types address the vanishing-gradient problem and maintain longer temporal memory during training. Furthermore, recent advances have been made in training sparse RNNs. Drawing inspiration from the stability of signals in nature, Ref. [271] introduced a delta mechanism that skips matrix multiplications if the difference in hidden or input vector elements between timesteps remains below a threshold Θ , referred to as Delta RNN. In fact, the examination of accelerometer sensor data (Fig. 5.8) and hidden output vector elements confirms this observation on the task of cow behavior estimation. Accelerators on FPGA developed with Delta RNN on GRU and LSTM have demonstrated considerable speed-ups and high percentages of weight sparsity for negligible accuracy loss on high-throughput natural language processing benchmarks such as TIMIT and Librispeech [272, 273].

In light of these advancements, it is conceivable to extend this work to include these sophisticated RNN types and techniques, thereby unlocking greater potential for a wider range of applications and further enhancing the efficiency and performance of the proposed RNN architecture. The integration of this proposed TLM architecture with these theoretical advancements is posited as a subsequent step for the extensive use of RNNs in IoT and resource-constrained Edge-AI applications. Moreover, the synergy between such theoretical and architectural advancements could bridge the gap between resource-limited hardware and large-scale RNN models. Surmounting this barrier would enable the execution of complex tasks in real-time, including machine translation, control, and reinforcement learning, on low-cost, widely available FPGAs, thus contributing to the democratization of AI.

Part II

From Neuromorphic Computing to Edge-AI via Synchronization, Chaos and Spiking Neural Networks

Chapter 6

An Experimental Study of Synchronization Phenomena in Chaotic Spiking Oscillators Realized Experimentally Towards Physical Reservoirs

Synchronization is a widespread phenomenon that occurs in natural and artificial systems wherein the units (e.g., fireflies, neurons, or nodes of a telecommunication or power grid) interact with each other, that is, exchange energy to coordinate their dynamical behavior [274, 275]. Given its universality and fundamental importance, countless studies have advanced our theoretical understanding of the underlying mechanisms promoting or hindering its emergence, as well as of its possible applications in engineered systems and networks [276–279]. One aspect that is of interest is the formation of spatial patterns, whereby the symmetry of structural connections is broken and a more spatially-diversified arrangement appears. While many studies have provided a theoretical analysis alongside an experimental characterization of synchronization for two coupled units only (see, for instance, Refs. [280–288]), the number of experimental investigations considering the general setting of more than two interacting units is considerably more limited. Because, in the presence of multiple units, the mismatches and non-idealities that permeate natural and physical entities can have a profound effect on global properties such as the emergence of synchronization patterns, this represents an important shortcoming [173]. Furthermore, the simulation or modeling of these emergent properties for large networks in software with complex heterogeneous dynamics is a difficult task, requiring a significant amount of compute and often does not translate 1-to-1 to actual hardware substrates. As a result, I believe experimentation with mixed-signal neuromorphic platforms where such heterogeneous dynamics are present is an important step forward to discover more about how complex computation can take place.

Common approaches to the experimental analysis of synchronization make use of mechanical systems such as metronomes or pendulums, [289–291], lasers [292, 293], and analog electronic circuits [294–301]. Experimental setups based on electronic circuits are particularly suited for the purpose, as they often offer a higher degree of reconfigurability of the connectivity and tunability of the control parameters com-

pared to their mechanical or optical counterparts. However, also in electronic systems, the cost and practicality of realizing the interconnections among the oscillators are limiting factors that cap the number of coupled units and the links considered in the experiments. To implement the flexible interconnections among the electronic oscillators, which are needed for characterizing the role of the topology of interactions on synchronization, different solutions have been adopted, including plugins for manual insertion of the coupling elements (e.g., resistors, or capacitors) [302], sets of switchable resistors [294], and digital potentiometers [301]. Similarly, also in large-scale integrated arrays of neuromorphic circuits, the realization of reconfigurable synapses represents a bottleneck, and it can also be noted that the preponderant fraction of the human brain is made up by white matter, that is, most volume is invested in realizing the connectivity between neurons [303,304].

Notably, the use of electronic apparatuses allows querying the influences of parameter mismatches and other non-idealities in the units that, in simulation, may be difficult to model or lead to numerical issues, and are consequently often neglected. The need for an experimental validation is established in the case of global synchronization (i.e., when all the units of the system are synchronized with each other) and becomes even more prominent when non-trivial patterns of synchronization or control techniques for inducing synchronization are investigated under the regime of incomplete synchronization (i.e., when entrainment is detectable but the orbits in phase space remain mostly non-overlapping) [296,297]. In this regard, recent works have provided experimental evidence of the robustness, in networks of non-identical electronic circuits, of global synchronization [298,299], pinning control techniques [294], explosive transitions [295], synchronization patterns in multiplex networks [300,305], and chimera states [301,306]. Finally, synchronization has been experimentally investigated also in presence of large mismatches between the oscillators [307]. This framework yields a network of heterogeneous oscillators, as the units have either largely different parameters or fundamentally diverse dynamics, and makes it possible to study regimes such as remote synchronization [308,309].

In my view, experimental investigation in this area should be intended beyond merely attempting to seek confirmation of numerical findings, but as a generative process in itself, particularly in light of the countless discoveries that can be made by serendipity while manipulating a physical system, later leading to the formulation of simplified models, of which remote synchronization can be taken as an example [173, 297, 310]. It can be said that this, after all, has represented a key mechanism of advancement in physics since the times of Galilei [311]. Furthermore, a step should be taken in the direction of neuromorphic processors that exhibit complex dynamics that are used for large-brain simulations and machine learning applications, but as of my knowledge remain underutilized in experimental studies.

The effect of noise on coupled nonlinear circuits is another aspect that is particularly relevant to investigate experimentally, as noise can either promote or hinder synchronization. When coherent, it can represent an arbitrary external input and provide a globally attractive force. When incoherent, as is normally the case in physical apparatuses, it usually provides a repulsive force that hinders synchronization, potentially enlarging a region of incomplete synchronization, but may also have the non-trivial effect of inducing synchronization at a critical intensity, via effects such as stochastic resonance [312–316]. In general, the study of noise in nonlinear dynamical systems is not trivial because it poses important practical issues, turning ordinary differential equations into stochastic differential equations and therefore re-

quiring the usage of specific solvers, such as the Euler-Maruyama method, which are less desirable than consolidated iterative solvers such as the family of Runge-Kutta methods [317]. These solvers are also not predominantly used in the simulation of LIF neurons in digital neuromorphic hardware platforms. Furthermore, in platforms such as DYNAP-SE2 there is inherent mismatch and noise due to the use of analog MOSFETs at subthreshold region. Therefore, experimental investigation is even more worthwhile in the presence of noise. In recent works, this issue has been dealt with by considering noise acting on a single unit of the system [318,319], a setting that finds applications in power grids [320] and in the theoretical analysis of consensus and synchronization [321]. Owing to the relatively limited number of experimental studies venturing beyond oscillator pairs, many configurations remain to be explored. In particular, concerning single-transistor oscillators, which are minimalist entities potentially well-suited for constructing large networks and exponential leaky-integrate and fire neurons that are more closely related to biology. To my knowledge at the time of writing only one experimental study of a large ring configuration is available [322] and experimentation on networks of exponential LIF neurons using experimental hardware remains underexplored

This Chapter provides an extensive experimental characterization of synchronization in oscillators having dynamics previously considered only in isolation. Namely, I consider an experimental setup based on coupled atypical transistor-based chaotic oscillators. These are a family of recently discovered nonlinear circuits, wherein the nonlinearity consists of one or two bipolar-junction transistors, the dynamics are controllable by the supply voltage together with a series-connected resistor, and a handful of reactive components complete the circuit, leading to particularly simple configurations. Such circuits are discovered via "numerical serendipity" through large-scale simulations and can replicate diverse known attractor shapes, such as funnel-like, double-scroll, and spiking [296,323,324]. In particular, I concentrate on one configuration that can readily generate spiking behaviors, which render it relevant as a possible building block for constructing ensembles having neuron-like dynamics. This configuration represents one of the smallest known types of spike-generating oscillators [323].

Throughout this Chapter, the focus is on presenting the properties of a small network of these circuits and, in the process, discussing its potential suitability as a substrate for the realization of physical reservoir computing. In general terms, reservoir computing is a powerful computational framework wherein an ensemble of coupled dynamical elements is used as means of mapping inputs onto a suitable high-dimensional space wherein a given classification or prediction can be solved efficiently [325]. The architecture comprises an input layer and a non-linear reservoir where all weights are fixed, cascaded by a trained output layer [326,327]. This has shown a computational ability similar to recurrent neural networks in forecasting chaotic dynamics with a significant reduction in training time and memory [328,329]. More specifically, physical reservoir computing is an emerging field wherein one or more advantageous properties of a physical (as opposed to mathematical) system are leveraged to perform the computation: these may include specific dynamics, together with the possibility of low-power operation and very high-density realization, as offered by photonic networks and nanomechanical devices [330]. Physical reservoirs, in fact, can be implemented using various and often elementary apparatuses, allowing them to be realized via small equivalent mechanical systems or electronic circuits [330]. Compelling examples are given by the wave dynamics in a water

bucket [331], and the dynamics of bacterial growth [332]. While reservoir computing based on analog circuits is well-established, and the suitability of experimentally-realized chaotic circuits has been demonstrated, to date, the application of networks of transistor-based chaotic oscillators remains unexplored [333].

Below, I realize a complete experimental setup that I deem to have the required versatility enabling future reservoir computing experiments using the elementary oscillator circuit under consideration. I focus on exemplifying the phenomena that can emerge in the network and relating them to the potential suitability for constructing a physical reservoir. In particular, I investigate the role of two primary control parameters, the influences of long-distance links and noise, and the behavior in the presence of heterogeneous dynamics.

This Chapter serves as an initial study that introduces and describes emergent phenomena, namely, chaos and synchronization that are observed when sweeping parameters of a ring network of dual-transistor chaotic spiking oscillators on an actual PCB. Furthermore, the effect of noise on these phenomena and the necessity of heterogeneity within networks is clearly observed and described. The proposal of a physical reservoir with this network that exploits these phenomena provides a first step towards the subsequent studies of this part of the Thesis. In other words, the study of this Chapter paves the way and is the main inspiration for the subsequent parts, Chapter 7 and Chapter 8, in terms of the approach taken, namely, the use of synchronization for computation and improvement of algorithms and hardware architectures.

6.1 Spiking oscillator

6.1.1 Circuit topology

The topology of the oscillator circuit, shown in Fig. 1a for a hypothetical coupled pair, is atypical in that it cannot be reduced to any known canonical arrangement. It consists of two cascaded NPN bipolar transistors, with the collector of the second (Q_2) connected to the base of the first (Q_1). A string of three inductors, akin to a double-tapped inductor, connects the supply node to the base of the second transistor (via L_1), then reaches the emitter of the same (via L_2) and finally its collector (via L_3). An LC resonant tank forms due to the presence of a single discrete capacitor (C), with possible additional contributions by the transistor junction capacitances. As with other transistor chaotic oscillators of this kind, a series resistor (R) between an external DC voltage source and one end of the string of inductors is provisioned. The same is connected to the collector of the first transistor (Q_1), which is instantiated in a common-emitter configuration.

It should be noted that the circuit configuration considered in this section was described in Ref. [334] and, for reasons of numerical convenience, follows a convention whereby the transistors are flipped compared to the original configuration initially discovered in Ref. [323]. To derive the simplified model used in the simulations in this section, the flipped version is considered, whereas, for realizing the experimental circuit in Section 3, the original topology is used.

A detailed analysis of the functioning of this circuit and the underlying chaos generation mechanisms is left for future work. Nevertheless, an initial level of understanding can be obtained by considering SPICE simulations using realistic transistor models [334]. For suitable component values, the generation of spikes involves the

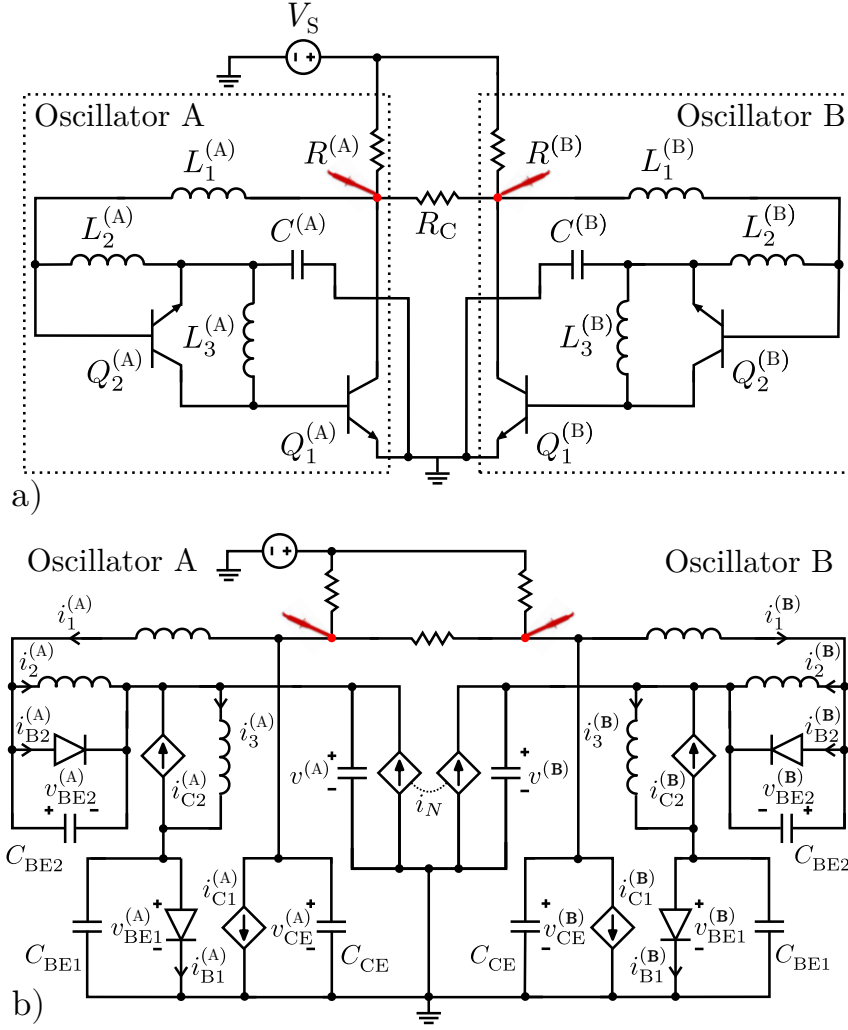


Figure 6.1: Spiking oscillator circuit, shown in the form of a pair (oscillators A and B) coupled via a resistor of value R_C . a) Discrete elemental diagram, b) Simplified equivalent diagram underlying the ODE model in Eq. (1), also including two coherent noise current sources i_N . Transistor orientation according to Ref. [334].

interaction between fast and slow dynamics, which unfold in different ways over the two transistors. During oscillation, Q_1 spends approximately similar time in the saturation and active regions, whereas Q_2 dwells primarily in the saturation and cut-off regions.

The transistor Q_1 is subject to an approximately constant voltage across the emitter junction, with overlapped negative spikes, whereas the collector junction shows fluctuations similar to those observed for Q_2 , with isolated spikes overlapped to them. Differently, Q_2 generates large-scale voltage fluctuations across both junctions centered approximately around zero, with the alternation of slow, phase-coherent dynamics and fast transient oscillations that decay rapidly.

More precisely, between the generation of two spikes, there is an interval wherein slow oscillations predominate and gradually build up in amplitude until a discontinuity occurs. How many cycles are completed before a spike is generated appears to be the variable that most clearly displays chaoticity in this system. Each spike is generated corresponding to the following situation, which hallmarks a fluctuation much sharper than the slow oscillation that more gently leads towards it. For Q_2 , the emitter junction voltage remains approximately constant, whereas the collector junction voltage increases rapidly from a negative to a positive value. On the other hand, for Q_1 , the emitter junction voltage decreases rapidly, tracking in the opposite direction the change in the collector junction voltage of Q_2 , eventually reaching zero. Correspondingly, the collector junction voltage also drops rapidly, becoming negative. It should be clear that this mechanism appears to be different and separate from the fast oscillations mentioned above.

After a spike has been generated, a gradual recovery towards a situation wherein all junction voltages are similar at ≈ 0.8 V follows. After that, a burst of fast oscillations ensues at a frequency that, while depending on the component values, is on the order of 10 times higher, involving primarily Q_2 and decaying rapidly. After these dissipate, the slow oscillations begin again to build up, eventually leading to another spike. These fast and slow oscillations, as reported previously for a single-transistor oscillator of this kind, correspond to different combinations of the inductances and capacitances present in the circuit, possibly also involving the junction capacitances of the transistors [324].

As previously shown for a single-transistor chaotic oscillator belonging to the same family, the identification of a suitable simplified numerical model can be highly instrumental to discovering the precise mechanisms of chaos generation and synchronization [324]. In this case, however, the situation is rendered more complex by the fact that there are two transistors rather than a single one, and large-swing oscillations are present, preventing the recapitulation in terms of a simple non-linearity. In a recent book, an equation model for this circuit was presented, based on piece-wise non-linearities for the transistor junctions [334]. These are described in the next subsection. In the corresponding circuit diagram, visible in Fig. 1b, each transistor is replaced by a diode representing the base junction, a controlled current source representing the collector current, and one or two capacitors. Moreover, the possibility to inject noise coherently across the two oscillators is provided through two identical current sources i_N in parallel with the capacitors.

6.1.2 Simplified equation model

To begin elucidating the behavior of this circuit, an oscillator pair is considered, wherein both oscillators are supplied with the same DC voltage V_S via identical

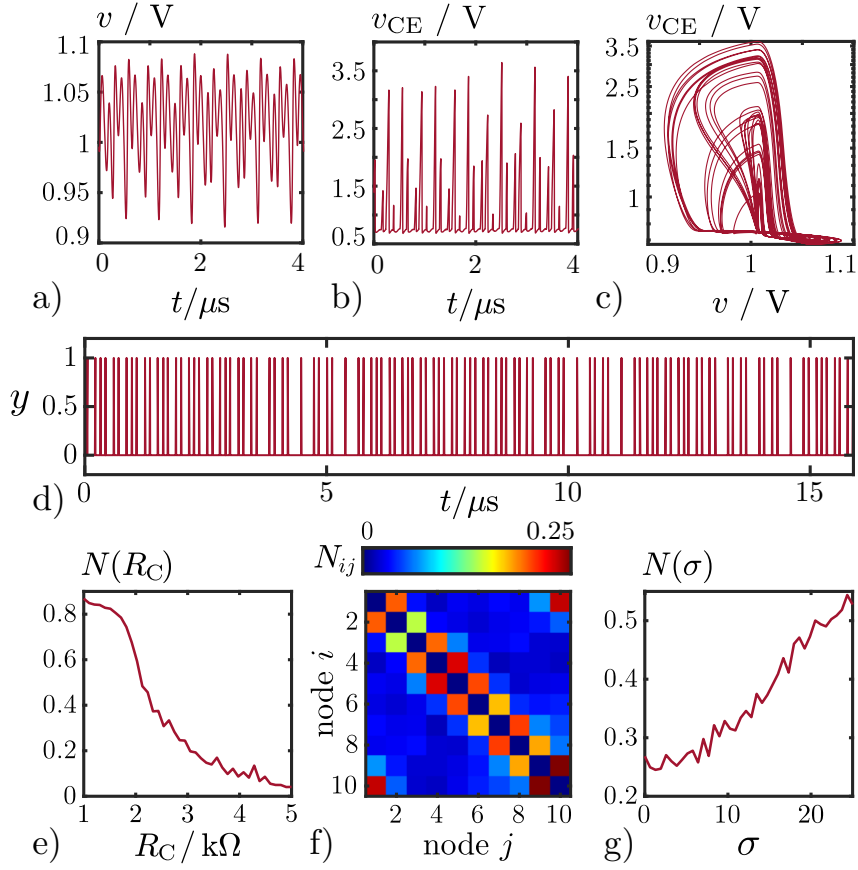


Figure 6.2: Results from integrating the ODE model in Eq. (1). a) and b) Representative time series for v and v_{CE} , and c) Phase portrait of the oscillator dynamics across them, d) Corresponding temporal distribution of spikes over a longer time-span, e) Synchronization level N as a function of the coupling resistance R_C between two oscillators, f) Synchronization matrix for a ring of 10 oscillators, g) Synchronization as a function of the intensity σ of coherent noise applied over an oscillator pair.

resistors of value R . A multitude of coupling schemes can be hypothesized, spanning all the circuit nodes in combination with either bidirectional (e.g., diffusive) or unidirectional (e.g., injection) approaches. Throughout the initial simulations presented in this section, for the sake of simplicity, diffusive coupling is implemented via a resistor R_C connected between the nodes for which the spiking behavior is most evident, as observed in the associated node voltage, v_{CE} of transistor Q_1 (Fig. 1a).

Considering the corresponding simplified model, applying Kirchhoff's laws one obtains

$$\left\{ \begin{array}{l} \frac{dv^{(A,B)}}{dt} = \frac{i_1^{(A,B)} + i_{C2}^{(A,B)} - i_3^{(A,B)}}{C^{(A,B)}} \\ \frac{dv_{BE1}^{(A,B)}}{dt} = \frac{i_3^{(A,B)} - i_{C2}^{(A,B)} - i_{B1}^{(A,B)}}{C_{BE1}^{(A,B)}} \\ \frac{dv_{BE2}^{(A,B)}}{dt} = \frac{i_1^{(A,B)} - i_2^{(A,B)} - i_{B2}^{(A,B)}}{C_{BE2}^{(A,B)}} \\ \frac{dv_{CE}^{(A,B)}}{dt} = \frac{\frac{V_S^{(A,B)} - v_{CE}^{(A,B)}}{R^{(A,B)}} - i_1^{(A,B)} - i_{C1}^{(A,B)}}{C_{CE}^{(A,B)}} \\ \quad + \frac{v_{CE}^{(B,A)} - v_{CE}^{(A,B)}}{R_C C_{CE}^{(A,B)}} \\ \frac{di_1^{(A,B)}}{dt} = \frac{v_{CE}^{(A,B)} - v^{(A,B)} - v_{BE2}^{(A,B)}}{L_1^{(A,B)}} \\ \frac{di_2^{(A,B)}}{dt} = \frac{v_{BE2}^{(A,B)}}{L_2^{(A,B)}} \\ \frac{di_3^{(A,B)}}{dt} = \frac{v^{(A,B)} - v_{BE1}^{(A,B)}}{L_3^{(A,B)}} \end{array} \right. , \quad (6.1)$$

wherein, among the system variables, v denotes the voltage across the discrete capacitor C , v_{BE1} and v_{BE2} denote the voltages across the two base-emitter junction capacitors, v_{CE} denotes the voltage across the collector-emitter of transistor Q_1 , and i_1 , i_2 , i_3 are the currents through the inductors L_1 , L_2 and L_3 , respectively. The values of the discrete reactive components are represented by C , L_1 , L_2 , L_3 , while the parasitic capacitances are assumed to be constant at C_{BE1} , C_{BE2} and C_{CE} . The DC voltage is represented by V_S , with R and R_C corresponding to the supply series resistor and the coupling resistor, respectively. The noise source is omitted for brevity, and introduced in Subsection 2.3.

In the above, the two base and collector currents are determined by piece-wise nonlinearities. Namely, the base currents scale linearly with the corresponding v_{BE} beyond an offset potential V_0 while prohibiting negative amplification, and the collector currents are assumed to be the minimum between the base current scaled by the gain β and that corresponding to the collector-emitter potential across a

fictitious resistor R_D . The corresponding equations are

$$\begin{aligned}
i_{B1}^{(A,B)} &= \max \left(0, \left(v_{BE1}^{(A,B)} - V_0 \right) / R_D \right) \\
i_{B2}^{(A,B)} &= \max \left(0, \left(v_{BE2}^{(A,B)} - V_0 \right) / R_D \right) \\
i_{C1}^{(A,B)} &= \min \left(\beta i_{B1}^{(A,B)}, v_{CE}^{(A,B)} / R_D \right) \\
i_{C2}^{(A,B)} &= \min \left(\beta i_{B2}^{(A,B)}, \left(v_{BE1}^{(A,B)} - v^{(A,B)} \right) / R_D \right) .
\end{aligned} \tag{6.2}$$

Based on Ref. [323] where this circuit was introduced, the supply voltage V_S and associated series resistor R , are considered, respectively, as primary and secondary control parameters, whereas the remaining component values are assumed as fixed. To support the appearance of incomplete synchronization in the results reported below, one component was mismatched, namely by setting $L_1^{(A)} = 100 \mu\text{H}$ and $L_1^{(B)} = 105 \mu\text{H}$, whereas, for the others, $L_2^{(A,B)} = L_3^{(A,B)} = 100 \mu\text{H}$, $C^{(A,B)} = 100 \text{ pF}$ were applied. As regards the components modelling the transistors, the junction capacitances were assumed fixed to $C_{BE1} = C_{BE2} = C_{CE} = 10 \text{ pF}$, whereas the parameters of the piece-wise non-linearities (encompassed within the current source and diode symbols in the schematic, and therefore not shown) were arbitrarily set to $\beta = 100$, $R_D = 100 \Omega$, $V_0 = 1 \text{ V}$. All these values were determined empirically in an attempt to reproduce as closely as possible the qualitative aspects of the behavior of the experimental circuit with the simplified model.

6.1.3 Simulations

For the first two cases below, Eq. (1) was integrated by means of a Runge-Kutta (4,5) solver until $t_{\max} = 500 \mu\text{s}$, rejecting until $t_{\min} = 20 \mu\text{s}$ to allow transient stabilization, starting from fixed and identical initial conditions $v(0) = v_{BE1}(0) = v_{BE2}(0) = v_{CE}(0) = v_0$ and $i_1(0) = i_2(0) = i_3(0) = v_0/R$.

Setting $V_S = 10 \text{ V}$ and $R = 1.25 \text{ k}\Omega$ and considering a single oscillator in isolation (i.e., $R_C = \infty$), it could be seen that dynamics include both sine-like (Fig. 2a) and spike-like (Fig. 2b) components. In combination, these lead to a spiral-like attractor onto which spikes of variable amplitude are superimposed (Fig. 2c). For these settings, a representative spike train shows a strong underlying periodicity, onto which irregularity is imparted by the skipping of spikes; in other words, an underlying near-periodic oscillation is gated into the irregular generation of spikes corresponding to a fraction of, rather than for all cycles (Fig. 2d). As reported in Section 3, the experimental data featured considerably sharper spike profiles and a richer distribution of inter-spike times.

Considering next an oscillator pair, retaining the same settings of V_S and R while sweeping $R_C \in [0, 5] \text{ k}\Omega$, the gradual decay of synchronization with increasing resistance could be observed, revealing a large window of incomplete synchronization suitable for pattern-formation phenomena to take place (Fig. 2e). Here, and throughout the remainder of this paper, for parsimony in terms of assumptions, synchronization is measured by means of the normalized mutual information N_{XY} defined as

$$N_{XY} = I_{XY} / \sqrt{H_X H_Y} , \tag{6.3}$$

where H_X and H_Y denote the entropies of the time series X and Y , and I_{XY} denotes their mutual information [335]. Here, the time series were obtained by binning the spike trains; the bin size was not critical and set to $t_{\text{bin}} = 10 \text{ ns}$.

Then, a ring of 10 oscillators was simulated, trivially extending Eq. (1) to this topology, while setting $R = 1.5 \text{ k}\Omega$ and $R_C = 2 \text{ k}\Omega$. The nodes were mismatched by setting $L_1 = \{100.6, 100.5, 100.1, 100.3, 99.2, 99.6, 100.0, 99.3, 100.7, 100.4\} \mu\text{H}$. Preferential synchronization among some adjacent pairs was observed, with values ranging from $N \approx 0.1$ to ≈ 0.3 , together with a clear decay of synchronization with distance along the ring, and weak traces of cluster formation (Fig. 2f).

Finally, the effect of applying coherent noise across a node pair was considered, given $R = 1.25 \text{ k}\Omega$ and $R_C = 2.5 \text{ k}\Omega$, by adding a term $\sqrt{\Delta t}\xi(t)$ to the equations for $dv^{(A,B)}/dt$, where $\xi(t)$ denotes Gaussian white noise of amplitude σ , namely, $\langle \xi(t) \rangle = 0$ and $\langle \xi(t)^2 \rangle = \sigma^2$. Due to the presence of the stochastic term, the system was integrated by means of the Euler-Maruyama solver until $t_{\max} = 250 \mu\text{s}$ with step $\Delta t = 0.1 \text{ ns}$. Albeit over a more limited range, synchronization gradually increased as a function of noise intensity, resembling the inverse of the effect of elevating the coupling resistor and thus also enabling one to probe the region of incomplete synchronization (Fig. 2g).

Despite only being a superficial characterization, these results demonstrate the viability of the circuit for generating complex dynamics under incomplete synchronization as a function of coupling and external signal injection. They are presented here to provide a basis for interpreting the remainder of this paper, which is grounded on experimental recordings. In particular, they confirm that the predominant aspects of the circuit behavior could be reproduced, albeit in a less developed form, by an elementary numerical model and, therefore, were not fundamentally due to parasitics or other issues in the experimental apparatus.

6.2 Experimental setup

6.2.1 Chaotic network board design

The oscillator used for physical realization had the same component values and circuit topology as in the previous study wherein it was discovered, that is, the transistors were flipped compared to the numerical simulations (Fig. 3a) [323]. To minimize the deviations from an ideal scenario, shielded inductors suitable for high-frequency operation were used, together with correspondingly high-performance transistors. Namely, the inductor values and models were selected as $L_1 = 15 \mu\text{H}$ (type NLFV32T-150K-EF; TDK Corp., Tokyo, Japan), $L_2 = 68 \mu\text{H}$ (type NLFV32T-680K-EF), $L_3 = 150 \mu\text{H}$ (type NLFV32T-151K-EF), the capacitor was $C = 470 \text{ pF}$ (type 101S42E471KV4S; Johanson Technology Inc., Camarillo CA, USA), and both transistors were rated for gigahertz operation (type PRF949; NXP Semiconductor, Eindhoven, The Netherlands). The series supply resistor R was realized with a 100Ω discrete component in series with a 470Ω trimmer, which was adjusted to $120 \pm 10 \Omega$. Because of tolerances on the order of 10% in the inductances and capacitances, respectively, as well as of 30% due to the transistor mismatches, the oscillators were parametrically non-identical.

The realized network consisted of a ring comprising 10 oscillator nodes, equipped with four additional long-distance links between nodes 1-6, 2-8, 3-9, and 4-10 (Fig. 3b). This topology allows studying small-scale pattern formation phenomena, as well as the effect of network topology on overall synchronization, and is similar to the one used previously in Ref. [322] for a non-spiking chaotic oscillator. All links were realized using the same coupling scheme as in the numerical simulations, however, to

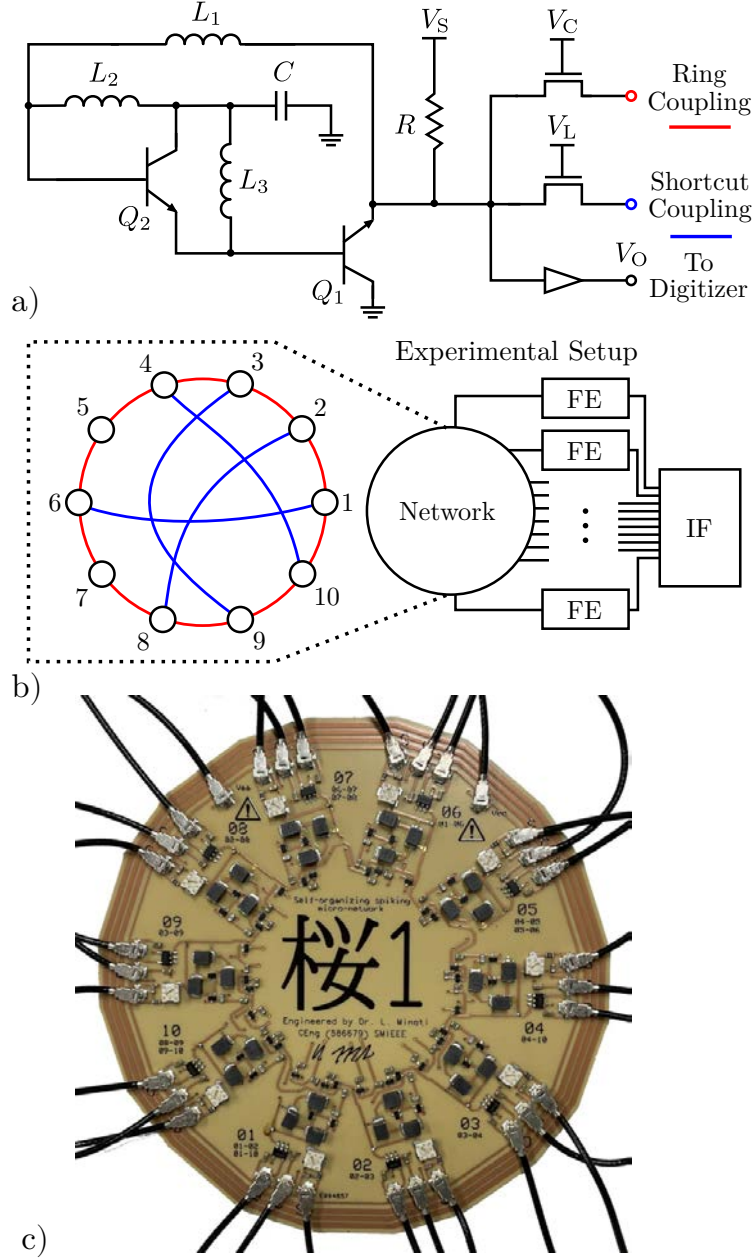


Figure 6.3: Realization of the experimental network. a) Circuit diagrams of oscillator instantiated at each node, b) Topology of the ring network (red) with long-distance links (blue) and data acquisition infrastructure (FE: Front-end, IF: Interface), c) Photograph of the network circuit board. Transistor orientation according to Ref. [323].

allow sweeping the link strengths under software control, resistors were replaced by NMOS transistors (type NTA7002N; On Semiconductor Inc., Phoenix AZ, USA). While the gate voltage does not control the coupling strength in the same linear manner as resistance, a monotonic relationship is expected, which was considered sufficient for the purposes of this work. As the oscillators are parametrically non-identical, it is necessary to avoid mutual influences in the DC bias; for this reason, all links were provided with DC blocking capacitors, not shown for brevity, having a value of 100 nF. To reduce loading effects that could corrupt the dynamics, each oscillator was accompanied by a high-impedance high-speed output buffer amplifier (type MAX4200EUK+T; Maxim Integrated Inc., San Jose CA, USA).

A 2-layer circuit board having an icosagon shape with a corresponding diameter of ≈ 10 cm was designed, containing all oscillators, couplings, and amplifiers (Fig. 3c). The fabrication materials have been made publicly available and are downloadable in Ref. [336], alongside all experimental recordings used in this study. Each oscillator was provided with three U.FL coaxial connectors, offering external access to the supply voltage V_S , ring and long-distance shortcut coupling control voltages V_C and V_L , and output signal V_O . The output buffers were powered at ± 5 V.

6.2.2 Control and data acquisition infrastructure

As elucidated below, the ultimate goal of this research is the realization of a physical reservoir. For this reason, in addition to the main board described above, customized mixed-signal infrastructure for data acquisition and injection of external signals was built. In this paper, this infrastructure was utilized in an elementary manner, whereby no external input data was provided to the network; nevertheless, the architecture and design materials are described and made freely available in full at Ref. [336] to support future work. It should be underlined that, by their architecture, neither the network of analog oscillators in itself (Fig. 3) nor the same in combination with the infrastructure (Fig. 4) could be capable of providing a hardware learning capability, wherein coupling strengths are in any way modulated by previous inputs. In reservoir computing, the coupling strengths (weights) are by definition set to fixed values, therefore, this capability is extraneous to the study purpose [330].

A dedicated front-end (FE) board was designed (Fig. 4a), having the functions of acquiring the spiking output of one oscillator, providing the supply and coupling control voltages to it, and in the future, potentially realizing simple forms of supply voltage control to maintain operation in the vicinity of a chaotic transition. It comprises one analog-to-digital (ADC) converter for spike digitization at 10 bit, 65 MSa/s (type ADC10065CIMTX; Texas Instruments Inc, Austin TX, USA), and one fast digital-to-analog (DAC) converter for adjusting the supply voltage V_S as the primary control parameter at 10 bit, 40 MSa/s (type MAX5181BEEG+; Maxim Integrated), and one slow DAC for setting in a predetermined manner the strength of the corresponding coupling voltage to a fixed value calculated by software (V_C or V_L according to a predetermined mapping) at 12 bit, 180 kSa/s (type DAC121C085C; Texas Instruments). These were interfaced to a complex programmable logic device (CPLD, type XC2C128-7VQG100C; Xilinx Inc., San Jose CA, USA) which, in the present version of the design, only implemented static voltage settings and spike detection based on the first derivative together with refractory period filtering. The circuit was realized as a 6-layer board having a size of $\approx 8 \times 11$ cm (Fig. 4b).

Furthermore, a specific interface board (IF) was designed (Fig. 4c), having the

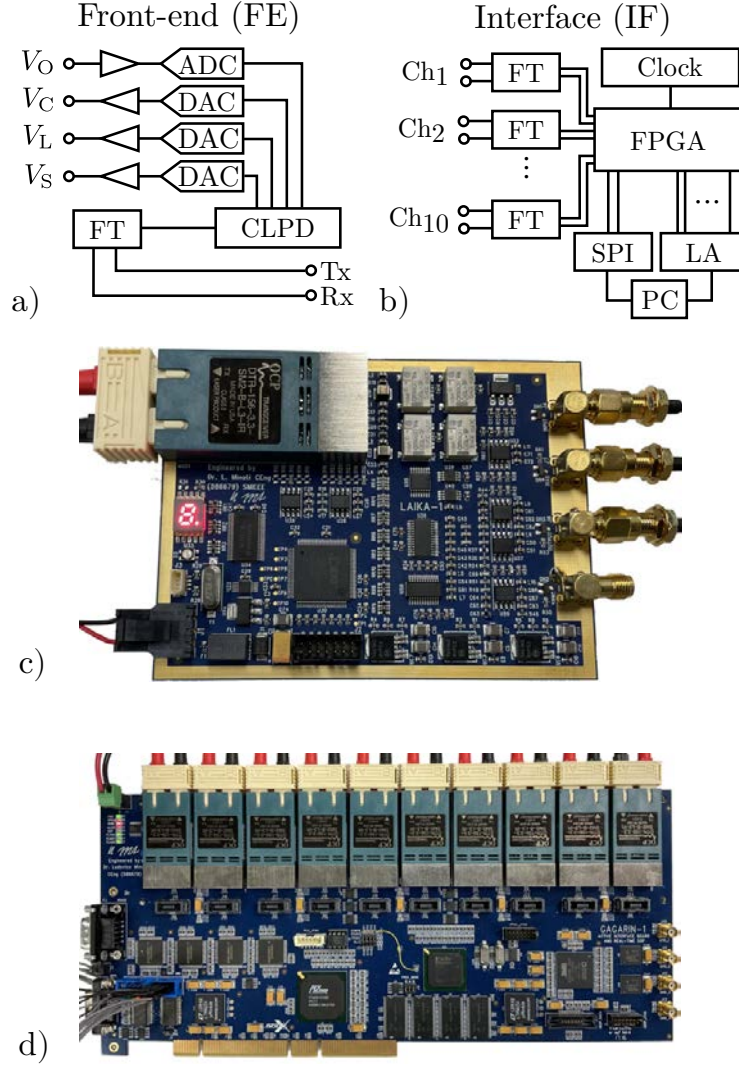


Figure 6.4: Data acquisition infrastructure. a) Architecture of the front-end (FE). ADC: Analog-to-digital converter, DAC: Digital-to-analog converter, CPLD: Complex programmable logic device, FT: Fiber transceiver). b) Architecture of the interface (IF). FPGA: Field-programmable gate array, SPI: serial peripheral interface, LA: logic analyzer, PC: personal computer. c) Photograph of the front-end (FE) circuit board. d) Photograph of the interface (IF) circuit board.

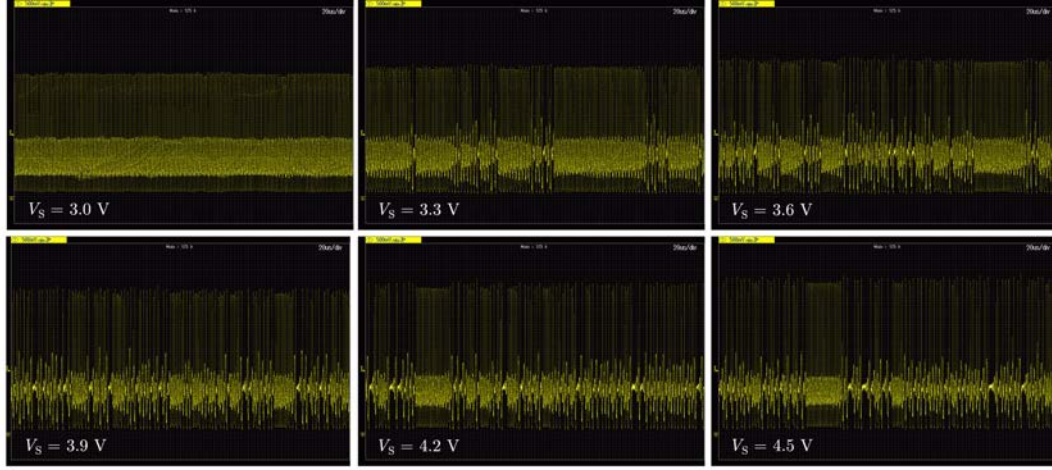
primary function of gathering the spiking inputs from all FE boards through point-to-point fiber optic links (type DTR-156-3.3-SM2-B-L3-IR; Optical Communication Products Inc., Fremont CA, USA). This board was based on a central field-programmable gate array (type XC3S400FG456; Xilinx), connected to 10 links, dynamic memory, a high-speed host bus interface, and a digital signal processor (type ADSP21469KBCZ4; Analog Devices Inc., Norwood MA, USA) for possible signal filtering operations. In the present version of the design, none of these features were supported, and the spike arrival pulses were simply sent out directly to an external logic analyzer interfaced by means of the IEEE488 bus (type 16802A; Keysight Inc., Santa Rosa CA, USA). The experiments were entirely controlled by software code written in C and Matlab language running on a host computer. The circuit was realized as a 10-layer circuit board having a size of $\approx 14 \times 31$ cm (Fig. 4d).

6.2.3 Representative recordings

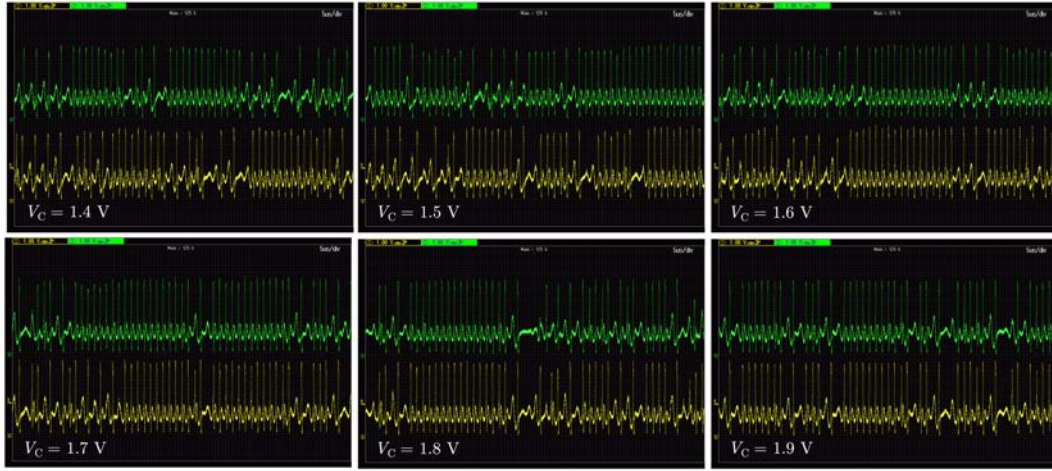
Prior to examining the synchronization states accessible to this experimental network, it is opportune to briefly describe its dynamics through oscilloscope waveform recordings of the output voltage V_O . First, the effect of the primary control parameter, namely the supply voltage V_S , is demonstrated by considering an uncoupled node. Throughout this paper, and in probable contrast with future applications for reservoir construction, this parameter was set homogeneously across the entire network (except in Subsection 4.5); however, it should be kept in mind that this does not yield identical oscillators because of the physical mismatches. A sweep was conducted over the range $V_S \in [3, 4.5]$ V, wherein the transition between periodic and chaotic motion occurs (Fig. 5a); below and above it up to the voltage limits of the transistors, changes are more nuanced. At $V_S = 3$ V, regular period-2 oscillation could be seen, with the alternation of a smaller sine-like wave with ≈ 0.7 V_{PP} amplitude and a considerably larger and sharper spike-like fluctuation on the order of ≈ 2.1 V, having a periodicity of ≈ 1 MHz. At $V_S = 3.3$ V, the onset of turbulent dynamics could be appreciated, with the same irregularly interspersed with shorter laminar time-intervals; the skipping of spikes was well-evident and accompanied by irregular fluctuations in the amplitude of the smaller sine-like activity. Elevating the voltage further, the prevalence of the turbulent phases increased, while the overall spike frequency correspondingly decreased, and the underlying amplitude fluctuations became larger. Notably, while the amplitude of the sine-like fluctuations was considerably irregular, ranging between $V_S \approx 0.2$ -1.2 V, that of the spikes remained approximately constant at $V_S \approx 2.6$ -2.8 V. It is because large-amplitude spikes predominate the output waveform and have an approximately all-or-nothing profile that this circuit has been dubbed a spiking oscillator.

Next, the synchronization between a pair of coupled nodes is demonstrated. A coupling voltage sweep was conducted over the range $V_C \in [1.4, 1.9]$ V (Fig. 5b), while keeping $V_S = 4$ V; below and above it, complete asynchrony and global synchronization prevailed. At $V_C = 1.4$ V, no temporal relationship between the spike occurrences could be seen. Already at $V_C = 1.5$ V, temporal alignment of individual spikes became appreciable, however, without coherence of the spike "gaps". That became detectable at $V_C = 1.6$ V, eventually reaching near-complete overlap for $V_C = 1.9$ V.

At this point, it is worth referring to the notion of liquid state machines, which leverage biologically plausible neuron models with all-or-nothing behavior to construct recurrent networks within reservoirs [326, 337]. One observation, common also



a)



b)

Figure 6.5: Oscilloscope waveform screenshots. a) Sweep of the supply voltage V_S for a single oscillator in isolation, b) Sweep of the coupling voltage V_C for two arbitrarily-chosen adjacent nodes, given $V_S = 4$ V. Scale: a) 500 mV/div, 20 μ s/div, b) 1 V/div, 5 μ s/div

to other computational systems, including natural brains, is that specific advantages are conferred by operation in the vicinity of a chaotic transition [338–342]. This has been observed specifically for a liquid state machine wherein the homeostatic control action of astrocytes is included [343, 344]; in future work, this feedback mechanism could be implemented using the CPLD device within the front-end board of the present setup. On the other hand, it is opportune to point out that the computational ability is not always maximized in the vicinity of a chaotic transition, but this depends on network and problem characteristics [345].

The qualitative features of the transition from regular spiking to chaos are reminiscent, for instance, of observations with the Izhikevich model, which, unlike the present circuit, attempts to present a physiologically-realistic account of neural dynamics [346]. Similar alternations between regular and irregular firing can be observed experimentally in giant squid axons [347]. While only phenomenological, the similarity to the present circuit appears noteworthy. Interestingly, other physical systems can also exhibit similar ordered and disordered pulsing dynamics, including CO₂ lasers and optical microring resonators [348, 349]. While the alternation of laminar and turbulent phases is the defining feature of the intermittency route to chaos, since a rigorous characterization was not performed in this paper, the precise mechanisms need to be clarified in future work [350, 351].

Similarly, the gradual onset of synchronization also recalls observations in other neural and physical systems. In biological neurons, wherein the dynamics are markedly stochastic, the relative frequency of spike coincidence increases with the synaptic strength [352]. This is also the case for physiologically-realistic neuromorphic electronic models of neural dynamics [353]. A similar phenomenon is also found in self-pulsating lasers, and optical microring resonators [349, 354]. In the presence of parametric mismatches, such as in the present case, it is common to observe that, as the coupling strength is increased and the energy exchange rate correspondingly elevated, the system trajectories become aligned gradually, while also exhibiting considerable instability [312]. In the present case, the alignment of individual spikes plausibly required less energy compared to that of bursts and gaps, which is reminiscent of phases synchronizing more easily than amplitudes in other systems, as previously observed, for example, in a ring of non-spiking chaotic transistor oscillators [322].

6.3 Dynamics and synchronization phenomena

6.3.1 Temporal irregularity and synchronization

Having illustrated the combined effects of the supply voltage V_S and coupling voltage V_C , it is opportune to gain deeper insight into the node- and network-level dynamics. Histograms obtained pooling inter-spike times from the entire network are useful for this purpose. First, sweeping V_S at three levels of coupling (Fig. 6a), revealed a situation wherein for low values, the dynamics were essentially period-1 ($\Delta t \approx 0.9 \mu s$, Δt not to be confused with the integration step), hallmarking the generation of a regular spike train. As the supply voltage was elevated, the probability in the corresponding bin could be seen to decrease markedly, while that corresponding to period-2 ($\Delta t \approx 1.9 \mu s$) increased. Period-3 ($\Delta t \approx 2.9 \mu s$), period-4 ($\Delta t \approx 3.9 \mu s$) and period-5 ($\Delta t \approx 4.8 \mu s$) also gradually increased, with a staggered arrangement. The effect of coupling appeared to be non-trivial, with high settings associated with

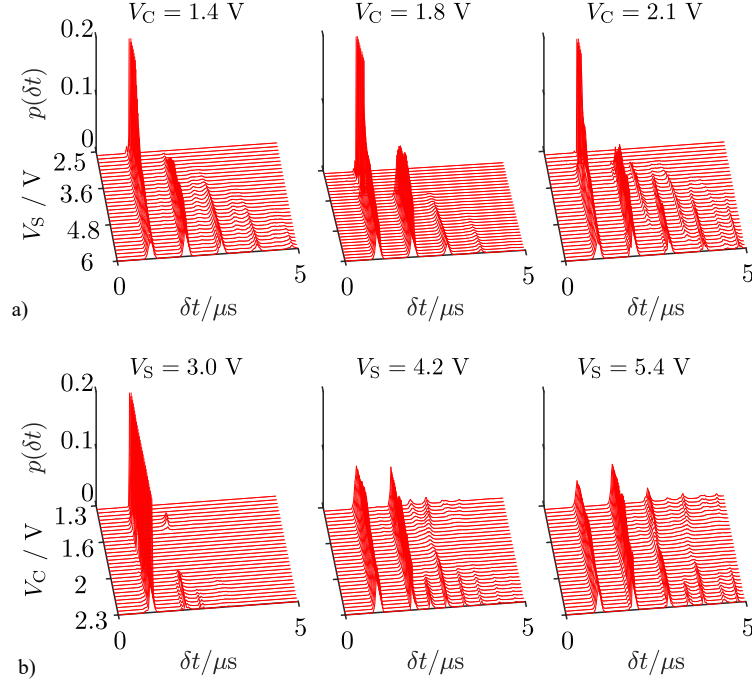


Figure 6.6: Distributions of the inter-spike interval Δt as a function of a) supply voltage V_S and b) coupling voltage V_C .

broadier distribution and the onset of additional peaks not corresponding to integer ratios (e.g., $\Delta t \approx 2.4, 3.4, 4.4 \mu s$). Second, sweeping V_C at three levels of the supply voltage (Fig. 6b), similarly confirmed the non-trivial effect of coupling. Particularly for operation deep in the chaotic region, stronger coupling was associated with a transition to a broad distribution, comprising peaks corresponding to both integer and non-integer ratios (e.g., $\Delta t \approx 0.9, 1.9, 2.8, 3.4, 3.8, 4.4, 4.8 \mu s$).

Altogether, these results indicate that an underlying temporal order, in the form of residual periodicity, is clearly identifiable, and chaoticity manifests itself primarily as jumps between well-separated peaks in the distribution of inter-spike intervals. While many types of forced and unforced systems can generate even and odd period dynamics, chaos manifesting in the form of "cycle skipping" is a more rare occurrence [350, 351]. It is, nevertheless, an established observation in brain dynamics and thought to play an essential role in phenomena such as ocular rivalry and the emergence of grid cells [355, 356]. The dynamics of individual neurons can generate inter-spike interval distributions that are unimodal or multimodal, but, even in the latter case, the peaks generally do not delineate the high regularity of the integer ratios observed for this circuit; there is, therefore, a considerable difference to be borne in mind, and, because of the peculiar aspects of the distribution, the coefficient of variation does not provide a significant measure [357, 358].

Next, it is helpful to consider more in detail the network response to a two-dimensional sweep of V_S and V_C . Here and in the following subsections, three experimentally-built boards, identified with α , β , and γ , were juxtaposed. This comparison was deemed relevant to ascertain the impact of component value tolerances and other non-ideal behaviors, as previously done in another network of non-spiking chaotic oscillators [322]. The parametric map of the average normalized mutual information $\langle N \rangle_{ij}$ (Fig. 7a), revealed an interplay between the supply voltage and coupling. As

expected in an ensemble of non-identical oscillators, stronger coupling was needed to attain synchronization in the presence of more developed chaoticity. A vast region of incomplete synchronization, that is $\langle N \rangle \approx 0.5$, was visible, having a wedge shape extending from between $V_C \approx 1.7$ -1.8 V close to the border with the periodic region, to between $V_C \approx 1.8$ -2.1 V at the highest supply voltage. As previously mentioned, this result is relevant because the regime of incomplete synchronization, wherein the energy exchange rate is sufficient, for example, to lead to the emergence of clusters but not so strong as to cause global synchronization, is a prerequisite for pattern formation, either spontaneously or in response to external inputs [312,322,350,351].

Emergent synchronization patterns have been leveraged in highly diverse ways to implement reservoir computing through coupled oscillator networks. For example, in Ref. [359], a reservoir able to perform XOR operations was created from two relaxation oscillators based on thermally-coupled vanadium dioxide switches. On the other hand, Ref. [360] employed a sparse network of 300 numerically-simulated Wilson-Cowan oscillators set up as inhibitory-excitatory pairs to function as a reservoir. These oscillators represent mesoscopic neuronal models, i.e., the interaction of multitudes of inhibitory and excitatory neuronal populations. Further, Ref. [361] proposed a form of wave-based reservoir computing using the phase dynamics of synchronized limit-cycle oscillators. Using a network of 30 nodes, linear regression of the square root, square, and sine functions could be performed. These studies affirm the potential of using pattern-forming phenomena occurring in coupled oscillators to realize reservoirs.

6.3.2 Predictability and permutation entropy

Having elucidated the principal characteristics of the spiking time series and their synchronization, further insight can be obtained by considering the level of predictability of the inter-spike intervals. Here, it is evaluated by constructing a machine-learning-based model of the dynamics of the real system and measuring the relative error in the approximation of the output variable. In the present context, this parameter provides a good empirical measure of chaoticity, which is easier and more stable to estimate compared to the largest Lyapunov exponent and fractal dimension, particularly starting from the inter-spike times [362–364]. For this purpose, a nonlinear auto-associative network was chosen in virtue of the fact that such networks generally have high performance in learning the dynamics of low-dimensional systems [365]. More specifically, a network with 2 feedback delays and a hidden layer size of 10 neurons was trained using Bayesian regularization backpropagation, using 70%, 15%, and 15% of the data for training, validation, and testing, respectively. The relative prediction error was quantified as $\epsilon = \langle |\hat{y}(t) - y(t)|/y(t) \rangle_t$, where $y(t)$ and $\hat{y}(t)$ denote, respectively, the experimental and estimated time series of inter-spike intervals [363].

Firstly, the error ϵ_{12} for feedback delays of $\{1, 2\}$ (Fig. 7b), markedly depicted the increase of chaoticity while elevating the supply voltage, ranging between $\epsilon_{12} \approx 0.01$ -0.45. In particular, a sharp transition from $\epsilon_{12} \approx 0.01$ to $\epsilon_{12} > 0.1$ could be seen around $V_S \approx 3.5$ V, hallmarking the boundary between periodic and chaotic oscillation. The coupling strength had a more contained effect, however, at the highest supply voltages its effect was markedly non-monotonic, with $\epsilon_{12} \approx 0.45$ at $V_C = \{1.3, 2.3\}$ V, dropping to $\epsilon_{12} \approx 0.25$ at $V_C \approx 2$ V.

Secondly, it is informative to consider the difference in prediction error observed by delaying the temporal horizon, defined as $\Delta\epsilon = \epsilon_{23} - \epsilon_{12}$ (Fig. 7c). A clear gradient

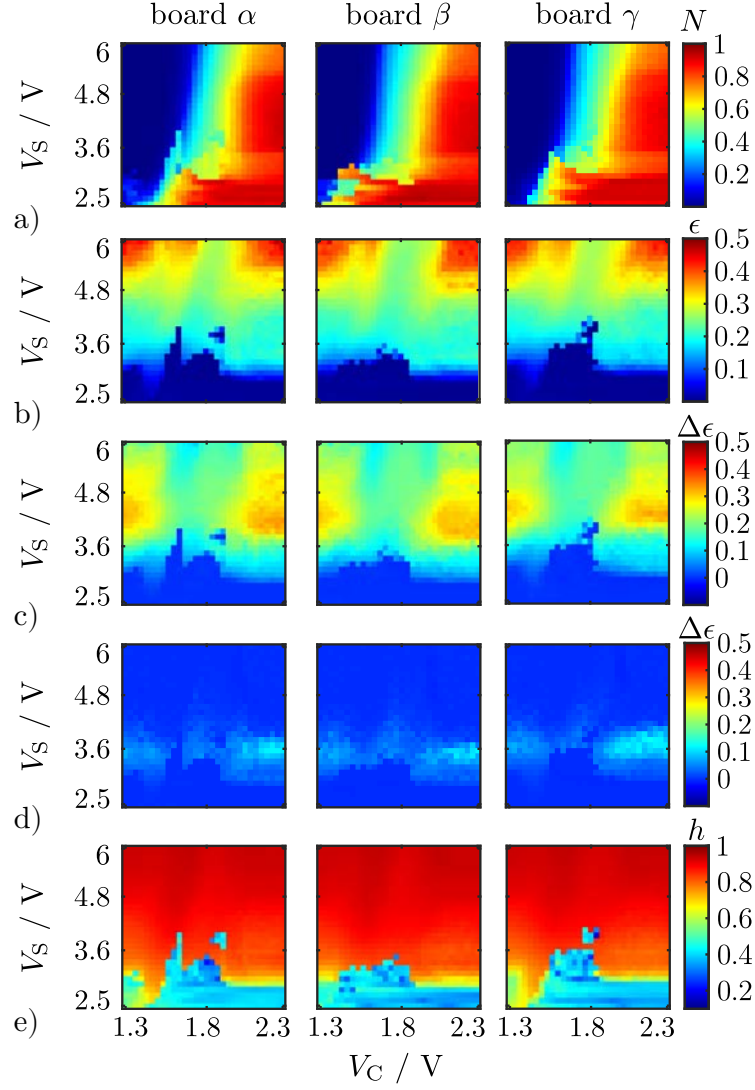


Figure 6.7: Synchronization and dynamical properties under a two-dimensional sweep of the supply and coupling voltages V_S and V_C . a) Average synchronization $\langle N \rangle_{ij}$, b) Average prediction error $\langle \epsilon_{12} \rangle_i$, c) Average prediction error difference with a delay of 1, $\langle \Delta \epsilon \rangle_i = \langle \epsilon_{23} \rangle_i - \langle \epsilon_{12} \rangle_i$, d) Average prediction error difference with a delay of 2, $\langle \Delta \epsilon \rangle_i = \langle \epsilon_{34} \rangle_i - \langle \epsilon_{23} \rangle_i$, e) Average permutation entropy $\langle h_5 \rangle_i$.

as a function of V_S was visible, indicating that the delayed prediction tended to result in a larger error in the chaotic region, particularly at intermediate levels where the original predictor performed satisfactorily.

Thirdly, the difference in prediction by taking a further step of delay was considered as $\Delta\epsilon = \epsilon_{34} - \epsilon_{23}$ (Fig. 7d). In this case, the difference in error was strikingly smaller, that is, $\Delta\epsilon \approx 0.01$ vs. $\Delta\epsilon \approx 0.10$. This result indicates that the system is mainly devoid of long-term memory, so the error difference is considerably more significant between initial delays of 1-2 than 2-3. Although this limits the usable temporal horizon available for computation in a physical reservoir, it plausibly satisfies the requirement of fading memory [326].

Fourthly, it is appropriate to also consider the permutation entropy (Fig. 7e). Compared to the prediction error, this provides a different account of the level of disorder in the dynamics because it is not grounded on the prediction of the dynamics but in information-theoretical arguments based on the ordinal distribution of patterns [366]. This measure was preferred over the correlation dimension or Lyapunov exponent as it is robust and better suited to experimental data, particularly in the presence of slow-fast dynamics, which complicate the choice of embedded settings considerably [363]. In brief, to calculate the permutation entropy, an L -dimensional space is constructed from L -tuples $(u_1, \dots, u_L) = (x_n, x_{n+1}, \dots, x_{n+L-1})$, which are encoded as $(s_1, s_2, \dots, s_L)$, where s_k denotes the ordinal position of u_k : the dynamics are therefore represented by the symbolic sequences of ascending and descending values. Insofar as the phase space is sufficiently filled, the permutation entropy is given by

$$H_m = - \sum_{j=1}^{m!} p_j \log_2 p_j \quad (6.4)$$

wherein p_j are the relative symbol frequencies, and which is customarily normalized to $h_m = H_m / \log_2 m! \in [0, 1]$. The order was empirically set to $m = 5$, ensuring that the coverage criterion $5m! < l$, where l denotes the number of spikes, is met [367]. Compared to the prediction error, the permutation entropy showed a similar influence of the control parameters, but it was considerably more homogeneous within the chaotic region, with $h_5 \approx 0.9$, and emphasized the transition to periodic dynamics, hallmarked by $h_5 \approx 0.4$ (Fig. 7e).

Taken together, these results exemplify a non-trivial relationship between connectivity (herein intended in a limited sense as the changes in link strength under a fixed topology) and dynamics in this system. Similar chaotic dynamics have been extensively leveraged for realizing physical reservoir computing. For instance, in Ref. [368], the transition between periodicity and chaoticity was exploited by influencing the dynamics of a simulated Chua's oscillator through two control parameters [369], obtaining high classification accuracy on an image dataset. Another work realized a physical reservoir experimentally utilizing a Chua's circuit. The circuit was driven by an external signal to encode input data followed by the read-out of the chaotic transients with the reservoir, and high accuracy in polynomial estimation and circle-boundary classification was observed [333]. Although these studies were based on single circuits, they clearly demonstrate the potential of chaotic oscillators as a substrate for reservoir computing, and networks of coupled circuits such as the present one represent a natural extension of that idea. One such network has been proposed in the past, comprising six nonlinear dynamic state spiking neurons based on Rössler dynamics, which were used to perform non-linear transient computation

to discriminate genuine and forged signatures [370].

6.3.3 Spontaneous pattern formation and effect of long-distance links

The average synchronization, prediction error, and permutation entropy revealed a high degree of consistency between the three experimental boards α , β , and γ , with contained differences in the parametric distributions. In previous work on a larger network of coupled chaotic oscillators, a similar situation was accompanied by notable heterogeneity in the underlying synchronization patterns, driven by the component tolerances rendering the boards non-identical [322]. Here, the effect was also appreciable: it was relatively contained far from the order-chaos transition (Fig. 8a-c; pattern correlation $r \approx 0.9$, range 0.7-1), and magnified in the proximity of it, involving a multitude of effects including the desynchronization of individual nodes (Fig. 8d-f). It should be noted that the more limited network size, that is, 10 rather than 30 nodes, effectively prevented the use of formal analyses of community structure such as Louvain’s method [322, 371]. Overall, in the chaotic region an evident synchronization gradient with distance was visible, closely reproducing the earlier simulation results. At higher synchronization levels, the spatial heterogeneity became more evident (Fig. 8e), in the form of preferential synchronization of some structurally-adjacent nodes and the formation of extended groups of non directly-connected nodes having elevated synchronization.

On the whole, these results indicate that the network under consideration is capable of forming non-trivial patterns via phenomena that break the topological symmetry of the connectivity, such as cluster synchronization and perhaps also remote synchronization, however, the effects were tenuous due to the limited network size. The importance of parametric mismatches, present either by design or because of practical factors, for the emergence of these effects has been extensively studied in numerical and experimental settings [372–375]. On the other hand, the fact that individual boards can show different topographical distributions of synchronization while having otherwise similar statistical properties represents both a challenge and an opportunity for physical reservoir computing.

The sensitivity to individual component tolerances implies that the classifier network cascaded to the reservoir would plausibly have to be re-trained for each realized device. However, the situation could also be leveraged to enhance the memory capacity and task performance via cascading or stacking multiple non-identical reservoirs before the classifier. Simulations stacking multiple reservoirs with non-identical parameters (such as leaky-integrator echo state networks) show improvements in memory capacity combined with an increase in inter-layer entropy, potentially pointing to richer dynamics within the overall system [376]. Analogously, the parallelization of two mismatched reservoirs physically realized with optical semiconductor lasers shows improved memory capacity and higher data processing rate [377]. In addition, numerical experimentation on an ensemble of reservoirs consisting of leaky integrate-and-fire neurons shows a better separation and approximation ability on a multitude of tasks [378]. Taken together, these preexisting works underline the potential opportunity offered by the non-identical nature of physical apparatuses, pointing to the possibility of complex computation leveraging multiple physical reservoirs using the network under consideration in this paper.

Ring networks are of interest not only because of their elementary symmetry but also because they lend themselves well to exploring the effect of additional links,

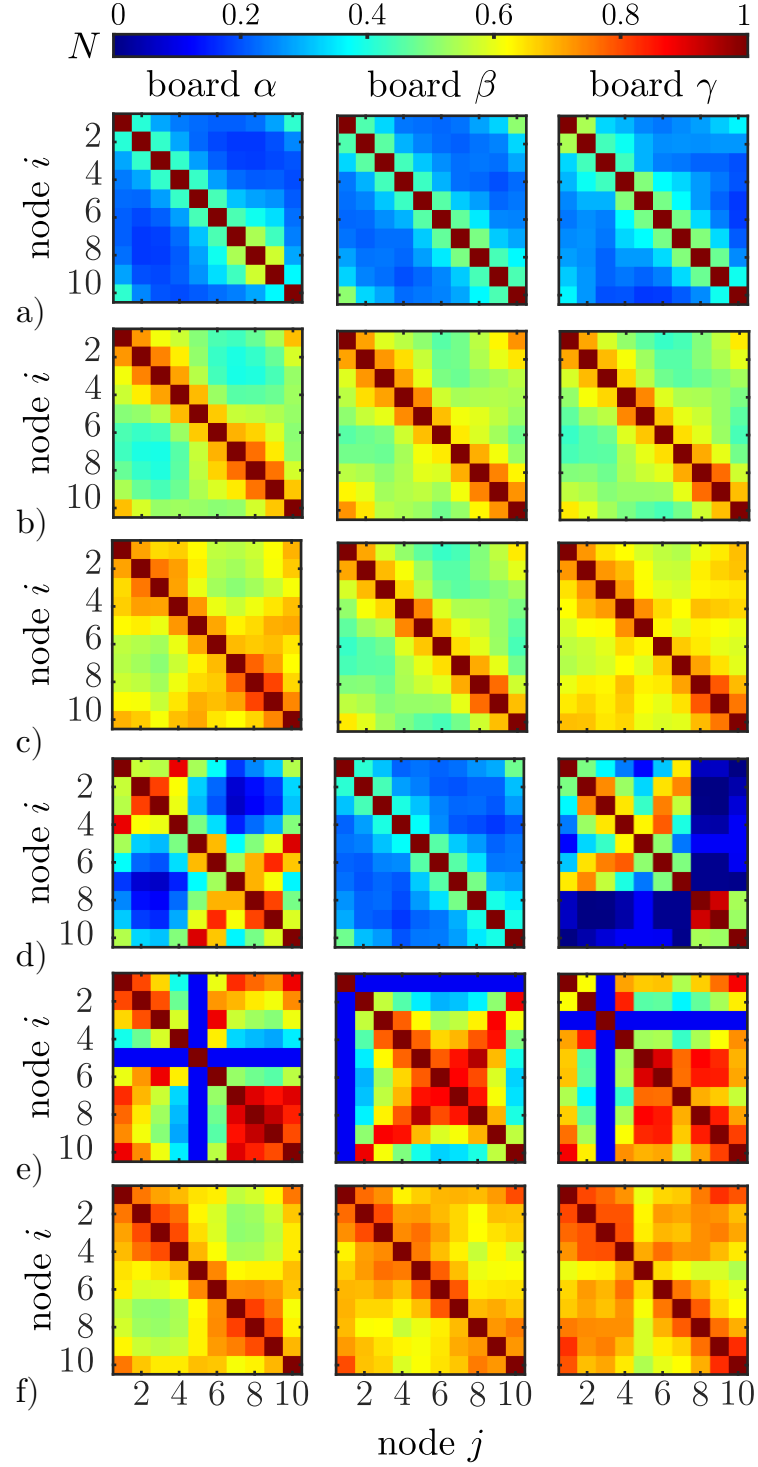


Figure 6.8: Representative synchronization matrices formed by spontaneous activity, across three experimental boards α , β and γ for different combinations of the supply and coupling voltages $\{V_S, V_C\}$, namely a) $\{3.8, 1.7\}$ V, b) $\{4.8, 1.9\}$ V, c) $\{5.8, 2.1\}$ V, d) $\{3.7, 1.6\}$ V, e) $\{4.1, 1.5\}$ V and f) $\{4.7, 1.5\}$ V.

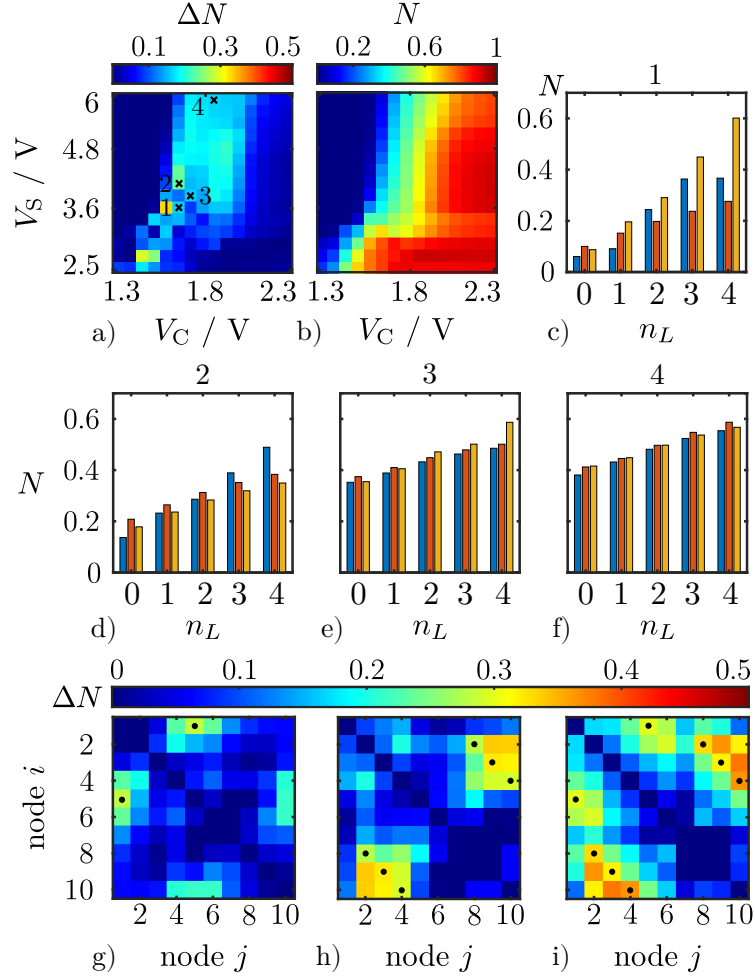


Figure 6.9: Effect of inserting additional long-distance links. a) and b) Synchronization increase with 4 additional links and baseline as a function of the supply and coupling voltages $\{V_S, V_C\}$, c-f) Bar charts of the average synchronization across the three experimental boards α , β and γ , for different parameter settings $\{V_S, V_C\}$, namely, $\{3.75, 1.59\}$ for 1, $\{4.25, 1.66\}$ for 2, $\{4.00, 1.73\}$ for 3 and $\{6, 1.87\}$ for 4, g-i) Representative matrices showing the synchronization increase as a function of adding 1, 3 or 4 links (black dots indicate locations), for $V_S = 6$ V, $V_C = 1.9$ V.

which form long-distance ties and reduce the average path length. For instance, it has been shown that long-distance links which introduce a small-world topology have the effect of increasing the synchronizability and resilience to damage [379, 380]. In our setup, four long-distance shortcut links were provisioned (Fig. 3b). In the hypothetical context of a physical reservoir, these provide an additional degree of flexibility owing to their influence on the collective dynamics. Reservoirs with a ring topology have been used, for example, in experiments on graph processing [381]. In this case, enabling all four links simultaneously considerably enhanced the average synchronization (Fig. 9a) but only within the region of incomplete synchronization (Fig. 9b). In other words, for the regimes of complete asynchrony and complete synchronization, there was no effect, but, at intermediate levels, the impact was substantial. This exemplifies another non-trivial dependency on input parameters that a small network such as the one under consideration can produce and which may be leveraged for reservoir construction; such effects arise because the dynamical range of changes as a function of the energy exchange rate is maximised close to the transition [312].

As regards the effect of the long-distance links, the corresponding synchronization matrices were averaged over all possible combinations of $n_L = 0$ (none), 1, 2, 3, and 4 links, set to have the same strength as the connections forming the ring itself. The average synchronization level consistently tended to increase with the number of links. A notable diversity of effects, apparently magnifying the differences between the three experimental devices, was evident particularly for control parameter settings close to the order-to-chaos transition (Fig. 9c). In the absence of the shortcut links, $\langle N \rangle \approx 0.1$, whereas in the presence of all four links, the attained level was very heterogeneous, namely, $N \approx \{0.4, 0.3, 0.6\}$. Even relatively small parametric changes could lead to a situation of more consistent dependency (Fig. 9d), or reduced slope (Fig. 9e); well into the chaotic region, these effects were considerably attenuated (Fig. 9f).

Visualizing the individual synchronization matrices in terms of the difference between the presence and absence of the long-distance links makes it possible to elucidate their effect. Considering a single shortcut (Fig. 9g), the impact was mainly the increase of synchronization in its immediate vicinity. However, already for three long-distance links (Fig. 9h), the emergence of a cohesive "island" of synchronization could be seen. Increasing the number to four significantly impacted the synchronization level between all node pairs, including those structurally distant from the long-distance links (Fig. 9i). These results are similar to those obtained previously, both numerically and experimentally, using coupled non-spiking single-transistor chaotic oscillators, regarding the possible synergistic effects of adding long-distance links, which lead to a more-than-summative effect [322, 380, 382].

6.3.4 Effect of coherent noise

As mentioned, in physical scenarios, noise, for example having thermal origin, generally manifests itself as an incoherent signal source and hinders synchronization by providing a repulsive force, although more complex effects can also be observed [312–315]. However, throughout this section, the different situation of a globally-coherent noise signal applied identically to all nodes is considered. This represents an arrangement wherein the noise has a dual effect: on the one hand, it strongly perturbs the intrinsic oscillatory dynamics, and, on the other, it promotes synchronization by providing an attractive force. This choice was motivated by the fact that noise, viewed as a

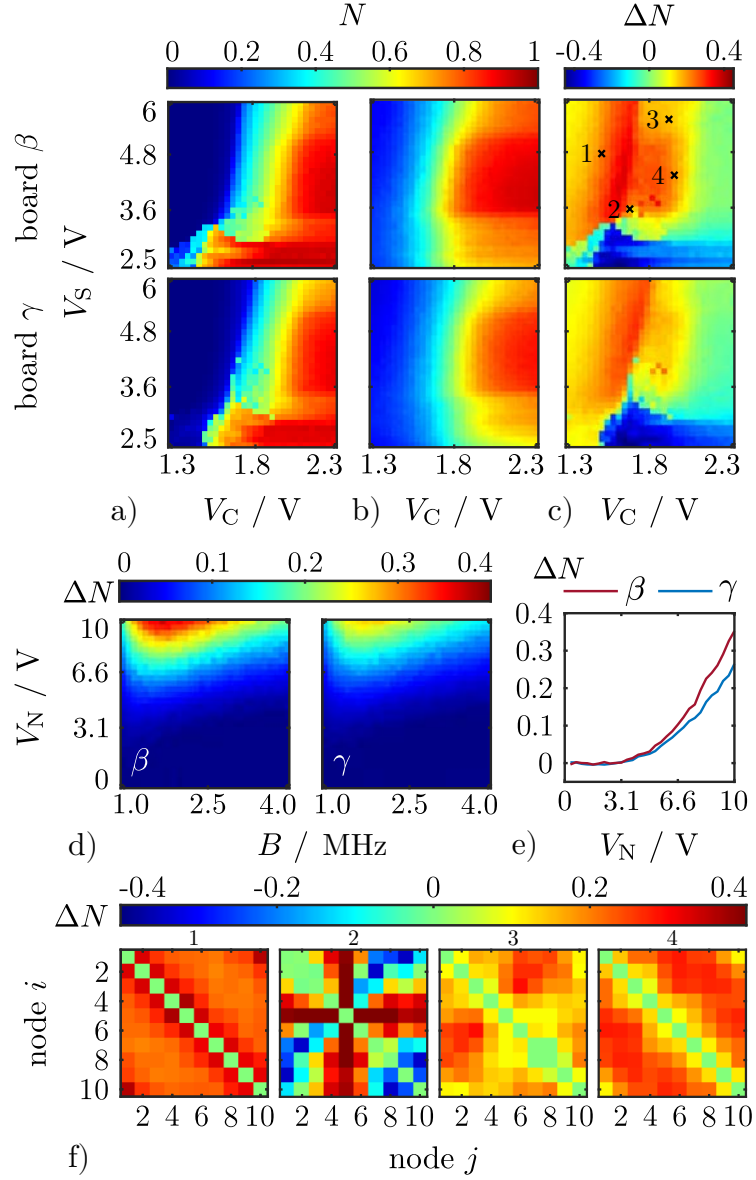


Figure 6.10: Effect of injecting coherent noise. a) Baseline average synchronization parametric maps as a function of the supply and coupling voltages $\{V_S, V_C\}$, b) Synchronization maps in the presence of noise having an amplitude of $V_N = 10 V_{pp}$ and a bandwidth of $B = 2$ MHz, c) Corresponding difference maps. d) Parametric maps for the average synchronization increase as a function of amplitude V_N and bandwidth B , for $V_S = 4$ V, $V_C = 1.6$ V, e) Corresponding chart as a function of V_N , f) Representative matrices for the difference in synchronization for different parameter settings $\{V_S, V_C\}$, namely, $\{4.91, 1.54\}$ for 1, $\{3.71, 1.68\}$ for 2, $\{5.64, 1.92\}$ for 3 and $\{4.43, 1.96\}$ for 4.

bandwidth-limited arbitrary signal, can also be representative of a diffuse arbitrary external input, in a configuration reminiscent, for example, of an afferent connection from a distant brain region or a diffuse innervation by autonomic nervous system terminals into a cortical microcircuit [383, 384].

In the numerical simulations reported in Section 2.3 (Fig. 1a), to minimize model complexity, the noise was applied by injecting a current across the only discrete capacitor present in the circuit. Experimentally, however, to limit the extent of the modifications required to the circuit board, it was injected via an additional 100 pF capacitor installed, individually for each node, between the source output and the node between C and L_3 . An external arbitrary waveform generator (type 33500B; Keysight) was used and controlled via the IEEE488 bus.

Over two representative boards, when the generator was inactive, the same effect of the supply and coupling voltages V_S , V_C was recovered as in Fig. 7a, indicating that the impact of connecting the additional capacitors was negligible (Fig. 10a). By contrast, when enabling the generator with an output of $V_N = 10$ V_{pp} and a bandwidth $B = 2$ MHz, the parametric map was substantially altered (Fig. 10b). Four noteworthy effects could be observed when considering the difference in average synchronization (Fig. 10c). First, in the region where synchronization was initially absent, i.e., $V_C < 1.5$ V and $N \approx 0$, noise injection elevated the synchronization by $\Delta N \approx 0.1$. Second, in the region where complete synchronization was initially observed, i.e., $V_C > 2$ V and $N \approx 0.9$, noise injection had no discernible effect, with $\Delta N \approx 0$. Third, in the region of periodic oscillation, i.e., $V_S < 3$ V and $N \approx 0.9$, noise injection had a strongly desynchronizing effect, with $\Delta N \approx -0.4$. Fourth, in the region of incomplete synchronization, i.e., with $N \approx 0.5$, the most marked increase in synchronization was observed, namely, $\Delta N \approx 0.3$, paralleling the effect of the additional links (Fig. 9a).

Considering next a two-dimensional sweep of the noise amplitude V_N and bandwidth B at a control parameter setting inside the incomplete synchronization region, namely $V_S = 4$ V and $V_C = 1.6$ V, another non-trivial effect could be seen (Fig. 10d). Namely, while the elevation of synchronization increased monotonically with voltage (Fig. 10e), it exhibited a marked non-monotonic dependency on the bandwidth, whereby a peak was observed around $B \approx 2$ MHz. Plausibly, this reflects the fact that it is primarily noise falling within the bandwidth of the spontaneous oscillation dynamics that exerts a synchronizing influence [350, 351].

Lastly, returning to the fixed noise source settings $V_N = 10$ V_{pp} and $B = 2$ MHz, inspection of the synchronization matrices at four representative control parameter settings revealed a notable diversity of the impact of noise, which was plausibly related to the heterogeneity in the physical node parameters (Fig. 10f). Namely, leftwards of the incomplete synchronization region (matrix 1), the effect was a diffuse increase in synchronization with $\Delta N \approx 0.2$. In the proximity of the transition (matrix 2), the richest pattern-generation effects were observed, including the selective synchronization of some nodes (e.g., $\Delta N \approx 0.4$ for node 5) and desynchronization of some node pairs down to $\Delta N \approx -0.3$. Rightwards of the same (matrices 3 and 4), the effect was a clearly non-monotonic influence, wherein the synchronization was enhanced preferentially for nodes that were not structurally adjacent (e.g., $\Delta N_{23} \approx 0.1$ vs. $\Delta N_{26} \approx 0.3$).

Altogether, these results highlight a non-trivial effect of noise on the dynamics of this system, including its interplay with the baseline synchronization state, a non-monotonic influence of bandwidth, and diversified pattern-formation effects.

The corresponding generative potential appears promising in terms of the ability of these oscillators to respond in complex manners to external stimuli, even in the elementary scenario where they are coupled as a ring and the arbitrary input is applied globally, drawing similarity to the perturbation of a reservoir with encoded input data as has been done with reservoirs of single chaotic oscillators. It should be noted, however, that previous examples of photonic and spiking networks used as reservoirs have shown equivalent or even improved computational ability when sparsely perturbing the network, i.e., by providing input to only a minority of the nodes [378, 385]. The notion of a globally-applied noise could, nevertheless, also be relevant as a means of increasing the baseline synchronization level and has not been investigated previously in the context of reservoir computing.

6.3.5 Effect of patterned node heterogeneity

In natural systems such as biological neural networks, there are countless sources of complexity beyond the elementary scenario considered thus far in this work, namely a ring network with homogeneous, albeit weakly non-identical nodes. One aspect is that there are considerable variations in neural phenotype, manifest both as qualitatively different neuron types (e.g., glutamatergic vs. GABAergic) and as diversified soma sizes and degrees of neurite development, which translate into heterogeneous dynamical properties [386, 387]. Furthermore, the study of networks of structurally heterogeneous or strongly parametric mismatched dynamical systems has been gaining increased attention recently. This is due to their relevance in real-world situations and to the fact that highly non-trivial effects are often observed, although they may be difficult to generalize, given that a synchronization manifold may not even exist in some situations [388, 389]. To demonstrate, purely as an example, a first step towards revealing the non-trivial effects of structural heterogeneity, in this section, an arrangement is considered whereby the network is split into two evenly-represented neuron populations, characterized by different supply voltages V_{SA} and V_{SB} . These are laid out according to three different levels of granularity, namely, a 5/5 split corresponding to $A = \{6, 7, 8, 9, 10\}$ and $B = \{1, 2, 3, 4, 5\}$, a 3/2/3/2 split corresponding to $A = \{1, 2, 3, 7, 8\}$ and $B = \{4, 5, 6, 9, 10\}$, a even-odd split corresponding to $A = \{2, 4, 6, 8, 10\}$ and $B = \{1, 3, 5, 7, 9\}$.

Without loss of generality, the populations A and B were set for chaotic and periodic dynamics, respectively, with $V_{SA} = 5.5$ V and $V_{SB} = 2.5$ V. The route-to-synchronization was examined by sweeping $V_C = 1.3$ -2.3 V. Considering at first the average synchronization $\langle N \rangle_{ij}$, a striking diversification appeared (Fig. 11a): while within the chaotic population the synchronization steadily increased for $V_C > 1.5$ V, within the periodic population a non-monotonic effect was evident, with a first peak reaching $N_{BB} \approx 0.8$ at $V_C = 1.5$ V, followed by a trough down to $N_{BB} \approx 0.5$ at $V_C = 1.7$ V, then rising again and eventually reaching $N_{BB} \approx 0.9$ at $V_C = 2.3$ V. The synchronization between the two populations, albeit generally lower, followed a similar trend.

When considering the two finer-grained population splits, this non-monotonic effect gradually vanished up to the point where the corresponding curves were largely overlapping. The corresponding permutation entropy curves revealed additional aspects of the transition (Fig. 11b). In all cases, as anticipated, the baseline levels of the two populations were widely separate in the most weakly synchronized condition at $V_C = 1.3$ V, with $h_A \approx 0.9$ and $h_B \approx 0.6$, and eventually converged to $h_A \approx h_B \approx 0.8$ for the strongest coupling setting at $V_C = 2.3$ V. The trajectory,

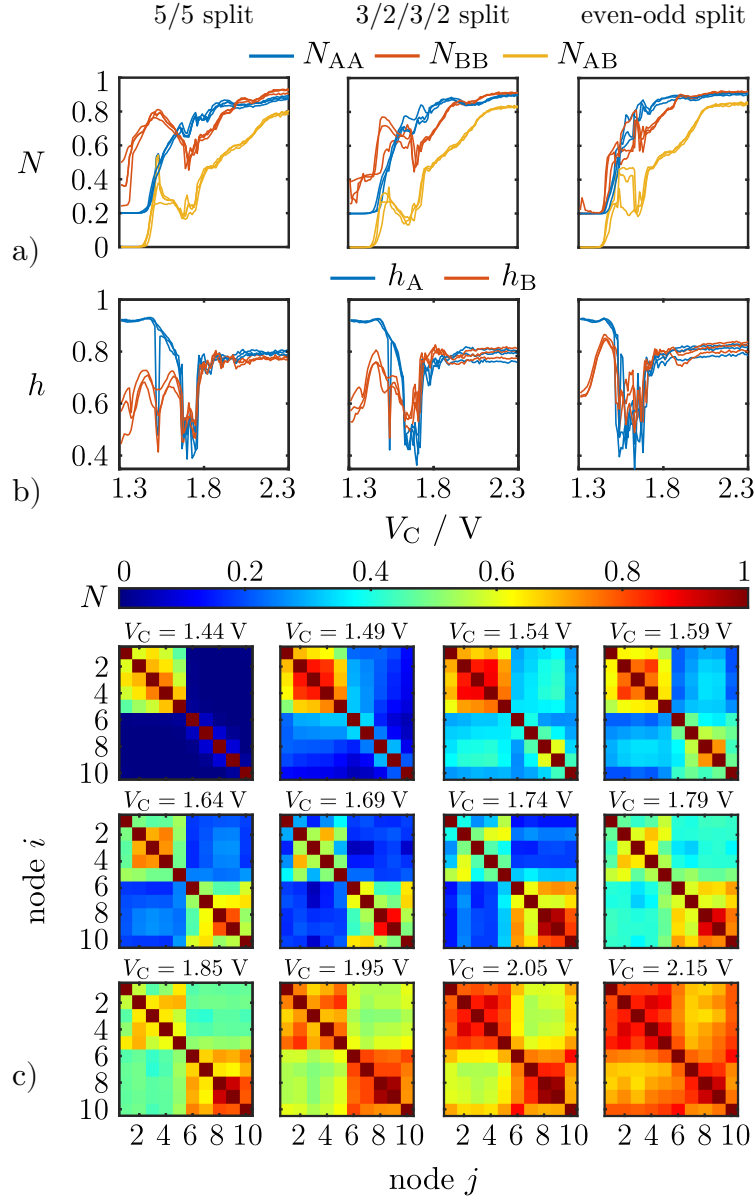


Figure 6.11: Effect of patterned node heterogeneity according to split in two populations, A and B. a) Average synchronization and b) Permutation entropy as function of the coupling voltage V_C . c) Matrices depicting the route to synchronization as a function of V_C for the 5/5 split.

however, was evidently non-monotonic, being associated with a sharp drop around $V_C = 2$ V, signaling a transient suppression of chaos for a critical coupling level.

The richness of the transition to synchronization could be better appreciated by visualizing the patterns formed in the underlying matrices for the 5/5 split case (Fig. 11c). At the weakest coupling setting, $V_C = 1.44$ V, population A was in a state of incomplete synchronization, with $N \approx 0.5$, whereas population B was unsynchronized, with $N \approx 0$. As the coupling was made stronger, at $V_C = 1.54$ V, synchronization between two communities emerged, reaching $N \approx 0.4$. Then, as the coupling was elevated further, at $V_C = 1.64$ V, two well-separated synchronized clusters emerged, corresponding to the populations. Going beyond this level, at $V_C = 1.74$ V, the community corresponding to population A started disgregating, until it was recovered again at $V_C = 2.05$ V, prior to the onset of global synchronization.

Taken together, these results further underline the generative potential of this system, which was manifested as a complex interplay between the two populations on their route to synchronization, and could be attributed to the formation, distortion, and destruction of synchronization manifolds under different rates of energy exchange. "Competitive" dynamics reminiscent of these observations can be seen in the coexistence of multiple attractors [350, 351]. Furthermore, non-monotonous effects of the coupling strength along the route to synchronization have been previously obtained in networks wherein at least one node is parametrically mismatched, particularly in the context of remote synchronization and the emergence of "fading" synchronization in tree networks [309, 390]. Clearly, the situation with a binary split between two populations represents a highly simplified scenario compared to the input patterns that the system could be plausibly exposed to when used as a reservoir. In existing studies, computational improvements have been observed when introducing heterogeneity to reservoir computing, namely, the use of randomized Boolean networks and variations of the non-linear function in nodes of echo state networks [391, 392]. The ability of the present system to generate this kind of complex non-monotonic relationship appears to be a promising substrate for implementing complex computations.

6.4 Discussion and application to reservoir computing

6.4.1 From coupled oscillators to reservoirs

The present work introduces a small network of coupled chaotic oscillators realized experimentally, and discusses its in-principle suitability for realizing physical reservoir computing. It appears interesting to consider a historical perspective briefly. Computation with oscillators originates from the 1950s, when LC-based oscillators were used as phase-dependent logic gates. One early example is the von Neumann oscillatory computer that was further developed into fully-functional Boolean computers using parametrons [393–395]. Apart from Boolean computation, in recent times, networks of coupled oscillators have been proposed in the context of analog, non-linear computation for tasks ranging from stochastic computing to image processing [396, 397]. Furthermore, oscillator networks have been used to create highly diverse computational architectures such as recurrent networks and reservoirs suitable for facilitating pattern recognition [398–403].

As systematically reviewed in Ref. [400], oscillator-based computing has recently been gaining ground as a possible means of breaking through Moore's law while

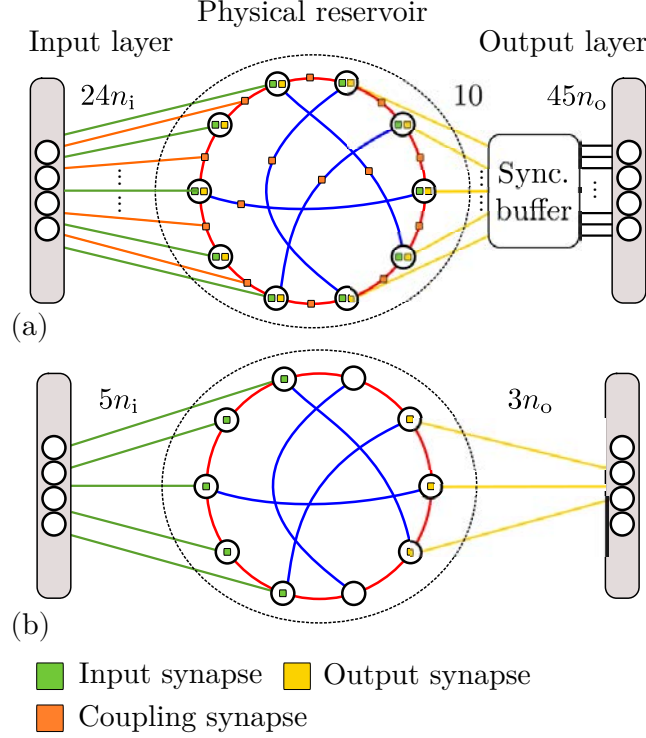


Figure 6.12: Possible architectures for constructing a physical reservoir. a) Assuming the synchronization values are output, b) Assuming the activity of select nodes is output.

leveraging existing mainstream semiconductor technologies. While there is a broad range of potential physical principles and architectures suitable for node realization and coupling, the consensus is growing that this approach may represent an integral part of future artificial intelligence developments. On the one hand, the intent is similar to the one of neuromorphic computing, based on an analog replication of realistic neural dynamics [353]. For example, echo state networks, a realization of reservoir computing, have previously been mapped onto analog neuromorphic hardware to detect abnormalities in ECG signals, showing the plausibility of extending reservoir computing to this field [164, 404]. On the other hand, it is essential to underline that oscillator-based computing in itself does not require physiological plausibility, in fact, the most promising architectures are deemed to be the simplest ones [400]. It is precisely in this regard that the oscillator proposed in the present work, being among the smallest known spiking circuits, attains a high value, while the affinity with neural activity, therefore, is to be intended at the phenomenological rather than structural level [173, 323].

6.4.2 Features relevant towards physical reservoirs

The experimental results reported herein demonstrate the accessibility of a wide variety of synchronization patterns sensitive to the control parameters, namely the coupling and supply voltages, the insertion of long-distance links, the injection of correlated noise, and the patterning of the network into parametrically heterogeneous node populations. As detailed below, these observations plausibly imply that the

system already in its present form meets the requirements for realizing a viable physical reservoir.

In order for a reservoir to be capable of solving complex computational problems, there are several requirements that have been identified and generally agreed upon [326, 327, 330]. First, non-linearity and high dimensionality are necessary properties. These are evidently fulfilled in the present system by construction and as documented through the multitude of behaviors observed in association with the transition to chaos. In addition, separation and fading memory are necessary. Separation requires reservoir states to diverge under input data assigned to separate classes. When perturbed by similar inputs or under the influence of noise, the reservoir has to show proportional reservoir states, allowing the classification of equivalent inputs. Fading memory prescribes that the reservoir states should temporally depend on previous states, according to an association that gradually vanishes over time. This is also commonly denoted as recurrence or memory capacity.

A broad region of incomplete synchronization, maximized for operation in the vicinity of a chaotic transition and widening at high supply voltages, was described in Section 4.1 (Fig. 7a). The location and size of this region appeared to respond in a graded and reproducible manner to the control parameters and the additional links. This points to the possibility of class separation that could be performed by encoding of features by means of control parameter perturbation. A simplified case of perturbation with multiple features was described in Section 4.5, leading to non-monotonic synchronization patterns. Therefore, this network plausibly meets the requirement of separation. Reservoir computing using oscillatory networks under varying degrees of synchronization has accordingly shown high computational ability on a variety of tasks, including XOR operations, classification, and regression tasks using phase-based computing and synchronization metrics [359–361, 405]. In the present case, mismatches between the experimental devices due to manufacturing tolerances would plausibly require re-training of the cascaded linear layer within a reservoir setup. At the same time, they also could be exploited by stacking or parallelizing multiple networks as ensembles, realizing an arrangement that has shown increased separation capability in the context of diverse reservoir types [376–378].

A nonlinear auto-associative network used to predict the time series of inter-spike intervals showed a contained error level in predicting the occurrence time one spike ahead. By contrast, the prediction of more distant future spikes showed a consistently higher error. These findings reasonably indicate that the network possesses the property of fading memory, which is commonly regarded as essential for complex computation using reservoirs [326, 327, 330].

Therefore, based on the experimental findings reported in this work, one can conclude that the present system plausibly meets all four requirements, namely, non-linearity, high dimensionality, separation and fading memory [326, 327, 330].

6.4.3 Hypothetical methods for input/output coupling

Finally, I tentatively propose two methods for realizing a physical reservoir and its input/output couplings starting from the ring network of chaotic spiking oscillators. One possibility is to apply inputs by perturbing the control parameters, i.e., the coupling and supply voltages, potentially diversified across nodes (Fig. 12a). These perturbations could be construed to be proportional, or otherwise univocally related, to the feature values of a problem task as well as to the randomly set weights of the pre-reservoir synapses. The readout layer could be connected to all nodes and

calculate a metric of synchronization between all neuron pairs in the network, using mutual information or coincidence values; an alternative would be to leverage the ratio of sub-harmonics [359]. It is, therefore, proposed to combine the usage of control parameters that perturb dynamics of the reservoir, similar to Ref. [368], with a synchronization metric or another suitable order parameter for state readout, akin to Ref. [359]. In Section 4.5, it was exemplified experimentally that the network undergoes a non-monotonic route to synchronization with increasing coupling strength. Essentially, this setup suggested the possibility of computation within a simplified reservoir where the input is encoded solely into the coupling voltage. The separation is achieved by splitting the network into equal groups with different dynamical properties. On this basis, it appears plausible that the proposed physical reservoir could perform the separation of input classes [359, 391, 392].

An alternative would be to implement a sparse approach involving the perturbation of a single control parameter, such as the supply voltage of a minority of nodes, for example, five nodes (Fig. 12b). The coupling voltages of all nodes would be assigned randomly. Only the states of three nodes on the opposite side of the network would be read out by the linear output layer, for example, by directly sampling their state variables [406]. Perturbing an individual control parameter of a chaotic oscillator reservoir has shown that a single Chua’s circuit may achieve high computational performance in a classification task [368]. Furthermore, sparse input and output connections are helpful for achieving good accuracy in liquid state machines [378]. This sparse setup with well-separated input and readout also appears motivated from a biological viewpoint, where long-distance interactions in the brain are essential for higher-order processing [173].

Comparing the two approaches, reservoir A requires 24 and 45 synaptic weights, whereas reservoir B, requires 5 and 3 synaptic weights per assigned neuron in the input and output layer, respectively. At this stage, it is unclear whether a sparse or a dense approach would be preferable with this network. Future work should leverage the experimental setup described in this paper to compare these configurations as a first step toward tackling concrete classification and prediction problems. As a result, the following Chapters of Part II of this thesis will use synchronization as a foundation. First, a Spiking Neural Network (SNN) is proposed that uses partial synchronization on a mixed signal neuromorphic processor (DYNAP-SE2) for encoding epileptic seizures (Chapter 7). Second, an SNN hardware-software codesign approach is proposed that utilizes synchronized spiking behavior to reduce network size and logic on the ICE40UP5K FPGA substantially for MNIST classification (Chapter 8). In essence, this Chapter serves as the inspiration for these Chapters and the first step towards establishing the codesign of neural networks with a multidisciplinary approach.

Chapter 7

A Seizure-Encoding Spiking Neural Network Using Partial Synchronization on a Mixed-Signal Neuromorphic Processor

Studies on the dynamics and modeling of the brain with non-linear systems have led to valuable insights into its behavior [407]. One particular aspect of such association involves the emergence of synchronous patterns in neural activities [408] and their use in explaining neural disorders, including epileptic seizures [409]. Epilepsy is one of the most common neurological disorders in the world, affecting nearly 1% of all people worldwide in all age groups [410, 411]. It is primarily characterized by chronic seizures which are a localized, highly synchronous brain activity characterized by high amplitude neural signals that can be captured by Electroencephalographic (EEG) recordings in the clinical setting at a high level of confidence [412]. Seizure detection in the EEG is a wide field of scientific interest that relies on current advances in biomedical sensing, intelligent Application Specific Integrated Circuits (ASICs), and advanced machine learning algorithms [413–416]. The EEG dynamics during seizures have recently been associated with a special partially synchronous state, termed “chimera”, where temporarily correlated, synchronous, and non-synchronous groups spontaneously emerge [417, 418]. Such states can provide valuable information in both understanding the disorder as well as help in its treatment.

A common approach to seizure monitoring outside the clinical setting currently utilizes a combination of limited in-person clinical observation and a patient-maintained log known as a “seizure diary” (Fig. 7.2a), which however suffers from over- and underreporting [419]. To improve on seizure monitoring, several devices for automated seizure detection have been validated [420]. These devices record a selection of various biosignals, among them EEG, electrocardiogram, electromyography, electrodermal activity, and accelerometry where EEG remains the most reliable. Two pioneering devices in the domain of classical EEG-based monitoring field are the seizure advisory system (SAS) developed by NeuroVista Corporation [421] and the responsive neurostimulation system (RNS) of Neuropace Corporation [422]. Al-

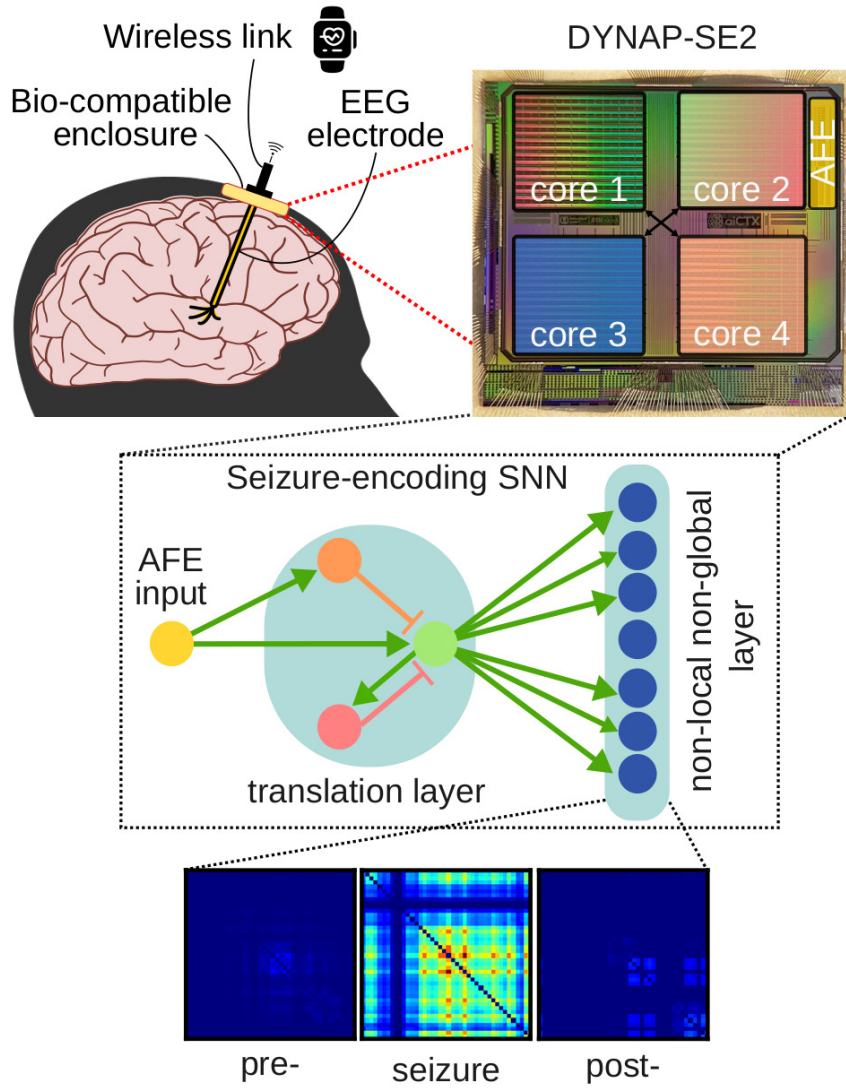


Figure 7.1: Graphical abstract of this Chapter; an envisioned real-time seizure monitoring system within a bio-compatible enclosure applied to the skull using flexible EEG electrodes and DYNAP-SE2. On DYNAP-SE2 (assigned neuron per core location color coded) an SNN (middle row) runs that encodes epileptic seizures in real-time by mimicking a silicon brain with a biological brain, namely, showing synchronous neuron clusters (bottom) within the chip whenever a seizure occurs.

though promising, both of these digital devices show the same constraints in long-term beyond-laboratory monitoring, namely, resource-intensive operation and inefficient A/D conversion with power-limited system clock. Thus, on a system level, these setups may consume a substantial portion of the overall power, limiting their use for long-term monitoring.

The neuromorphic engineering research field, which is inspired by the principles of computation of neural systems [423, 424], represents a promising approach for implementing low-power always-on electronic processing systems applied to biomedical signal processing [425]. neuromorphic computing has enabled the development of extremely efficient architectures [426] that are built with synapse and neuron circuits, arranged in a highly parallelized fashion and operating asynchronously [427]. As a result, various neuromorphic devices have emerged, ranging from *Loihi* and *SpiNNaker*, where neurons are simulated digitally using numerical integration of neuron equations [428, 429], to *BrainScaleS* and *DYNAP-SE* where neural primitives are implemented in custom analog circuits, connected by a digital event-routing fabric [430, 431].

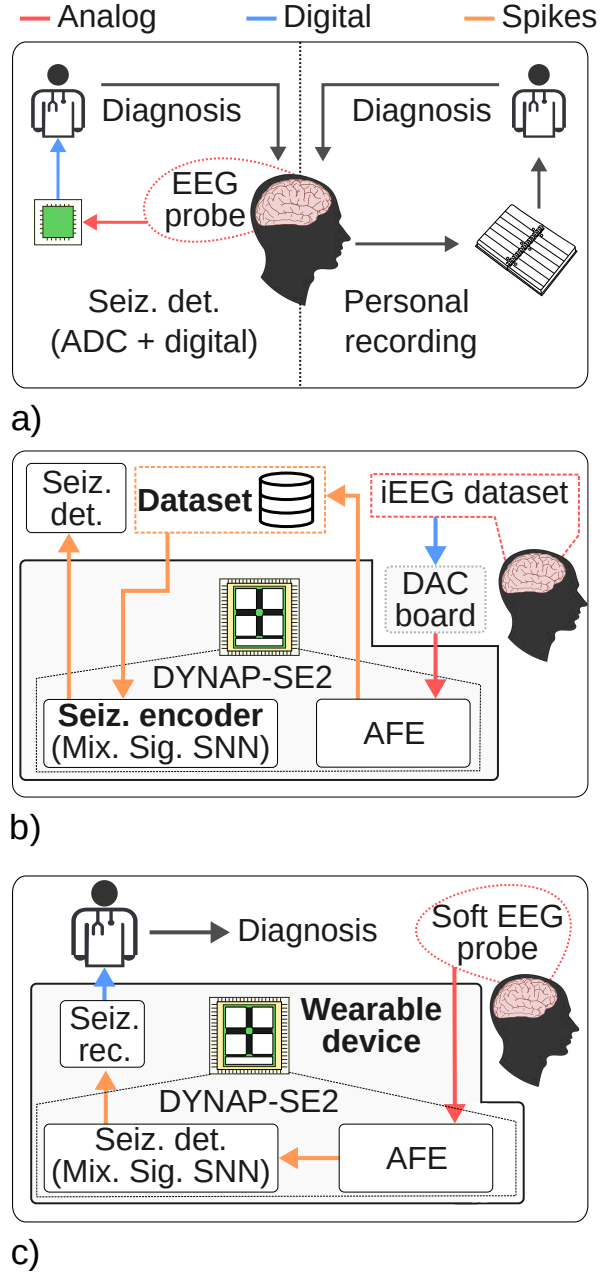


Figure 7.2: Overview of existing, presented, and proposed seizure detection systems; a) SOA (state of art) approaches for seizure detection and diagnosis; seizure recording with a seizure diary (right side), seizure detection using Analog-to-Digital Converters (ADC) and digital processing (left side); b) Presented system; seizure encoding on DYNAP-SE2 with analog iEEG signals using on-chip Analog Front-End (AFE) and a Spiking Neural Network (SNN); c) Envisioned system; real-time seizure detection pipeline with DYNAP-SE2 and a seizure recorder within a wearable device.

In this Chapter, I present an on-chip two-layered Spiking Neural Network (SNN), compatible with mixed-signal analog/digital neuromorphic hardware, that encodes the partial synchronization of brain states during seizure (ictal) periods by means of firing rate-based filtering and a specific connectivity structure. I validate the model

on a full-custom hardware pipeline that does not require any digital compute element in the loop. The model network was implemented in neuromorphic hardware using the spiking neuron circuits of the DYNAP-SE2 chip [304], as shown in Fig. 7.3b.

To convert the EEG signals into streams of spikes that can be sent in as input to the SNN, I used an Analog Front End (AFE) circuit containing low-noise amplifiers, filters, and an Asynchronous Delta Modulator (ADM) circuit [432] directly integrated in the DYNAP-SE2 chip. Utilizing the on-chip AFE, ADM, and a recording setup (Fig. 7.3a), I created an event-based intracranially recorded iEEG dataset from the SWEC-ETHZ iEEG Database [433]. By combining signal conversion and processing stages on the same chip, the SNN proposed in later sections is employed for encoding seizures using network synchronization properties (Figs. 7.2b and 7.3a).

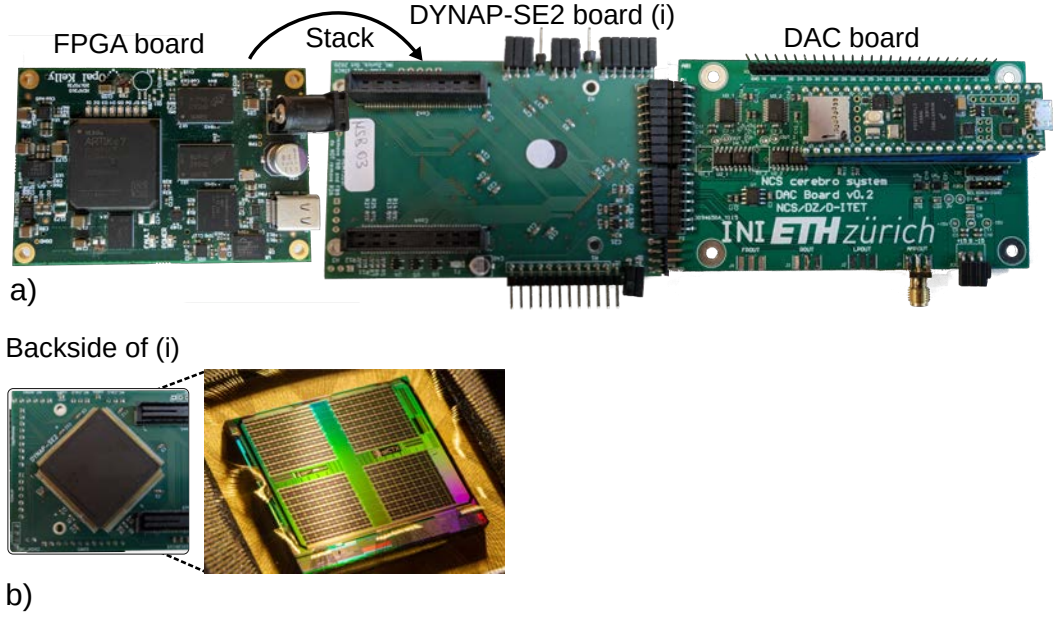


Figure 7.3: a) Photographs of the setup, an FPGA board, DYNAP-SE2 board with DYNAP-SE2 chip on the underside, Digital-to-Analog (DAC) converter board with a Teensy microcontroller for interfacing with the AFE; b) Photograph of the DYNAP-SE2 die (right), and the package located on the underside of the DYNAP-SE2 board (left).

Rather than following the classical Machine learning (ML) approach of training artificial neural networks (ANNs) with large data-sets, I developed a SNN that is at the same time biologically inspired and (as a consequence) compatible with neuromorphic hardware, to amplify and extract the intrinsic partial synchronization (i.e., “Chimera”) patterns that are present (but not prominent) in the input signals. The hardware implementation of this SNN, on the DYNAP-SE2 chip, is able to produce the desired results because it exploits the rich dynamics arising from sub-threshold analog circuits, as well as the heterogeneity present in its synapses and neurons due to device mismatch effects. I show how this mixed-signal hardware setup preserves all of the information present in the original data, by comparing the classification accuracy on the spiking data with that of a Support Vector Machine (SVM) on the original input data. To our knowledge, no prior work exists that encodes in real-time such states with a network on a neuromorphic device to provide

seizure indications.

The main contribution of the work described in this Chapter is the development of a foundational network dynamics-based approach for biomedical signal monitoring using mixed-signal neuronal circuits. The possibility of long-term processing of the data obtained using this approach, combined with its robust signal processing features, will further allow the development of wearable epileptic seizure detection and logging systems in the future (Fig. 7.2c). The general framework of analyzing and processing long-term recordings to detect and possibly predict events of interest has the potential to be useful also for a wide range of different applications, both in the healthcare and medical domain (e.g., for cardiovascular or diabetic diseases) and in other edge computing domains.

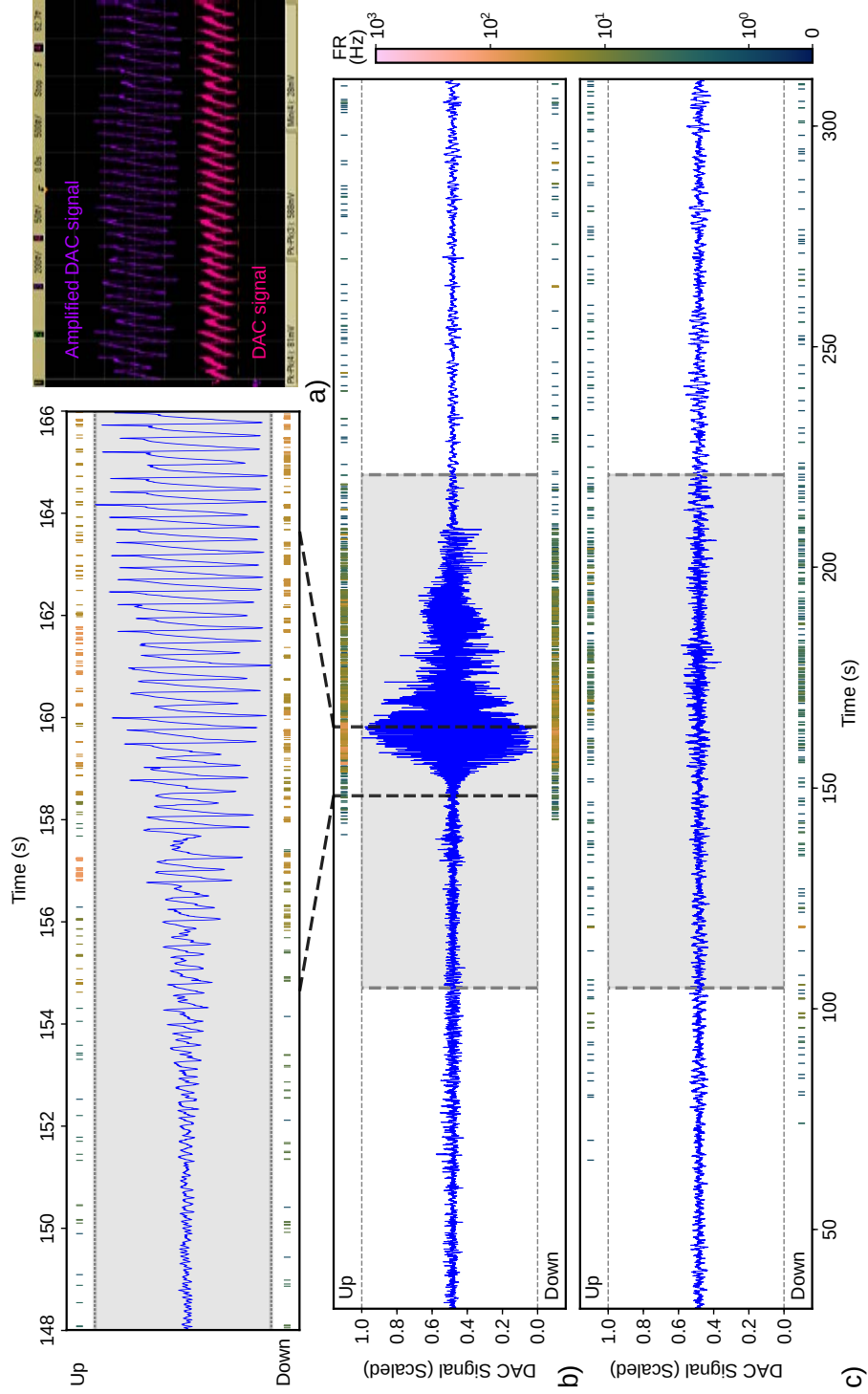


Figure 7.4: Snippets presenting 320-second long DAC converted analog iEEG signals from two channels and the encoded events from the ADM (‘UP’ and ‘DOWN’) on DYNAP-SE2. The event data rate (above and below the DAC-generated signal) is encoded in color with the seizure period denoted by the grey background; a) Oscilloscope traces depicting the incoming DAC-generated signal (trace 4, in pink) and the AFE amplified signal (trace 3, in purple); b) Analog waveform of one EEG channel and ADM events (with inset above showing a close-up of 16 seconds); c) Analog waveform of another EEG channel and ADM events.

7.1 Results

7.1.1 Asynchronous Delta Modulation (ADM) of iEEG signals

The first stage of the processing pipeline is outlined in Fig. 7.2b. It comprises the AFE and ADM blocks, for encoding analog EEG signals into events. Figure 7.4 shows recorded analog waveforms from two EEG input channels (Channel 12 and 13 of a single seizure of patient 1), selected from the SWECD dataset [433]. Figure 7.4a presents the oscilloscope traces of the analog EEG signal. These traces were obtained by converting the digitized dataset waveforms back into analog signals using a DAC setup (Fig. 7.3a, right) and amplified by the initial stage of the AFE. The amplified signal stream of the AFE is then input to the ADM, which produces asynchronous streams of Up and Down pulses, depending on the slope of the analog waveform (See Methods section). The Up and Down events are shown above and below the analog waveforms and the event rate is color-coded as per the colorbar (Fig. 7.4 right). A close-up of 16 seconds of the first channel (zoomed plot above Fig. 7.4b) shows the increase of event-rates and their burstiness during the seizure (ictal) periods. The conversions with the AFE are recorded in a “hardware event-based encoded dataset” (see Sec. 7.5). In total, I encoded and made publicly available [434] data for ten seizures, two per patient for a total of five patients. The events from the ADM circuits are fed as single pulse trains per channel into the SNN core of the DYNAP-SE2.

7.1.2 The DYNAP-SE2 SNN implementation: firing-rate filtering and encoding correlation

The second stage of the pipeline outlined in Fig. 7.2b is the SNN model implemented on the DYNAP-SE2. Figure 7.5 shows the SNN architecture. It comprises two layers: a “translation” layer and a “non-local non-global” (NLNG) encoding layer. The translation layer, adapted from Sava et al. [435], consists of Recurrent Excitatory (R-E) and Inhibitory (R-I) neurons, combined with a Feed-Forward Inhibitory neuron (FF-I) depicted in Fig. 7.5 (left). The FF-I and the R-E neurons receive events from the AFE-converted event streams for iEEG signal channels via a slow-excitatory synapse. Further, the R-E neurons are inhibited by the output spikes from FF-I neurons via a slow-inhibitory synapse. The R-E and R-I neurons represent a neuron pair that both excite and inhibit each other, constructing an excitatory-inhibitory (E-I) balance pair.

Figure 7.6a shows the firing rate input-output curve of the translation layer in the DYNAP-SE2 based on input Poisson trains of varying firing rates. The synaptic time constants of all neuron types (R-E, R-I, and FF-I) have been set heuristically to achieve this frequency transfer curve. All other parameters including the weights were left constant (see Methods for parametric values). The FF-I neurons have a high synaptic time constant, leading to a firing rate that saturates at around 180 Hz for input firing rates >200 Hz. These neurons inhibit low-frequency input spikes (Fig. 7.6b). The slow inhibition of the R-E neurons by R-I neurons results in a near-linear output firing rate between 200 Hz to 600 Hz (Fig. 7.6c). The above represents a spike-based non-linear filter, i.e., the excitation of the R-E, -I neurons is highly sensitive to the 200–600 Hz firing rate band while insensitive to inputs from outside this band.

The second layer, termed the NLNG encoding layer, is excited by the R-E neurons

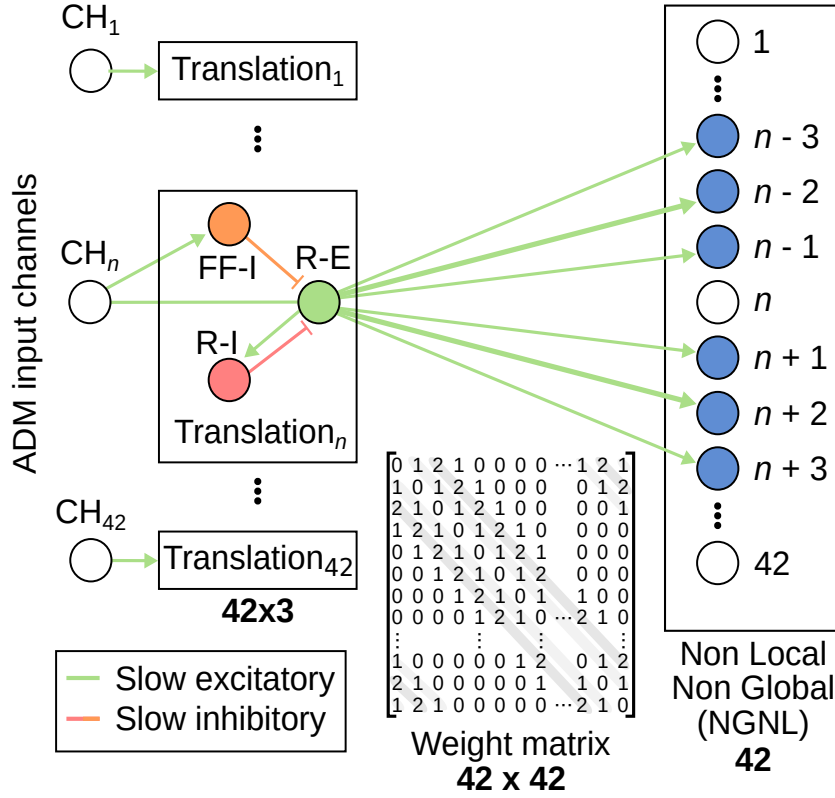


Figure 7.5: Structure of the on-chip SNN; a translation layer containing 3 types of neurons, Feedforward Inhibitory (FF-I), Recurrent-Excitatory (R-E), and Recurrent Inhibitory (R-I) neurons. This layer performs firing-rate-based filtering, followed by a Non-Local Non-Global (NLNG) layer. Synapse types slow excitatory (NMDA like) and slow inhibitory (GABA-B like) are color-coded

of the translation layer with a specific connectivity structure. The connections consist of slow excitatory synapses with constant synaptic weights. The constant weight per synapse is chosen to make sure that the membrane potential of the neurons reaches close to the set threshold. The connectivity structure is described by 1 – 2 – 1 – 0 – 1 – 2 – 1 connectivity between the R-E and NLNG neurons (Fig. 7.5) where 2 represents a synapse with a weight twice the constant weight mentioned above (see Methods for parametric values). To give an example, I consider 42 input neurons (encoding the 42 EEG channels, for the case of a particular patient), producing 42 outputs from R-E neurons as per the frequency response curve shown in Fig. 7.6a. Here, the R-E neuron from a translation block n is connected to the neighboring NLNG neurons with one synapse, two synapses with constant weight (representing one synapse with $2 \times$ constant weight), and one synapse, respectively. This connectivity pattern is mirrored on both sides while ignoring NLNG neuron n itself. The connectivity structure is shown in Fig. 7.5 and is similar to a regular graph. Such connectivity patterns are established for each R-E neuron connected to the NLNG layer.

This proposed connectivity matrix between these layers was determined by a neural architecture search using a *differential evolutionary* algorithm with a cost function defined as the maximum separation between output firing rate from non-correlated input and correlated Poisson input.

Figure 7.6d shows a linearly inverse relationship in the output firing rate of

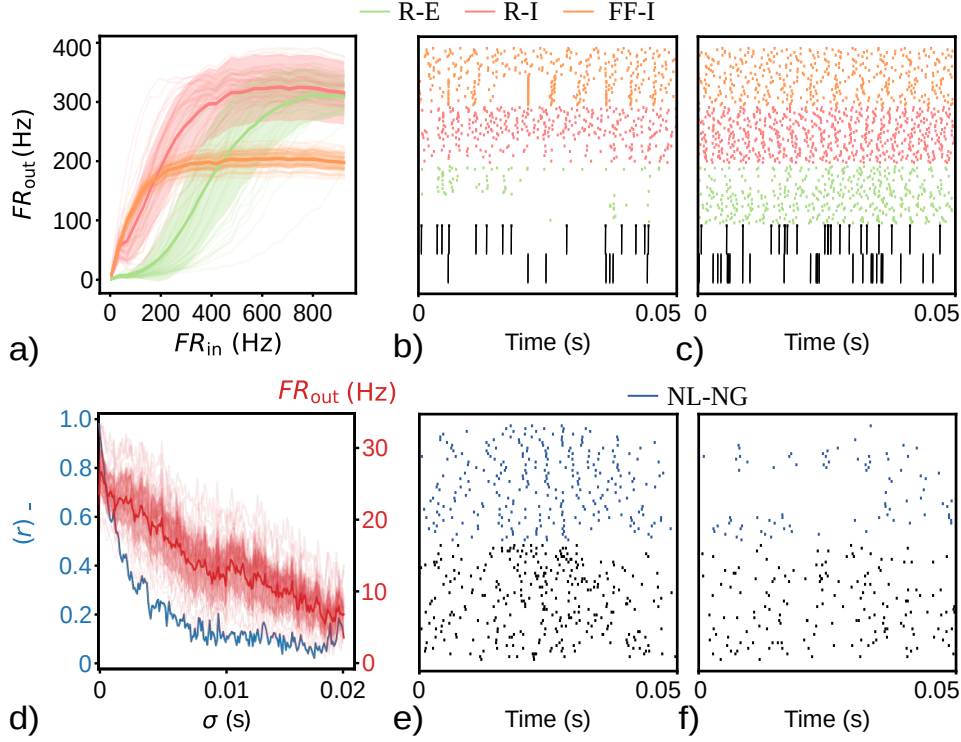


Figure 7.6: a) Firing rate input-output curve as a result of Poisson-generated input spikes to translation layer; b), c) Raster plots of the translation layer for an input firing rate of 200 Hz and 400 Hz respectively; d) In blue, the mean Pearson correlation coefficient of 42 input channels over the standard deviation of jittering in spike time at input firing rates of 50 Hz. In red, the output firing rate of the non-local non-global encoder as a result of the jittered input spike trains; e), f) Raster plots of input (black) and output of the non-local non-global encoder (blue) at standard deviations of $\sigma = 0.005$ s and $\sigma = 0.015$ s.

the NLNG layer from DYNAP-SE2 with the correlation between input spike-trains. These spike trains are generated from a Poisson process and iteratively jittered with jitter noise taken from a normal distribution and input simultaneously across 42 channels to the chip. The NGNL layer exhibits a firing rate near 30 Hz for fully correlated input spike trains, decreasing to 20 Hz and 10 Hz for less correlated spike trains, with mean Pearson correlation coefficient of $r = 0.255$ and $r = 0.114$, arising from spike time jitter noise at standard deviations of $\sigma = 0.005$ s and $\sigma = 0.015$ s respectively (Fig. 7.6e and f). This experiment shows the functionality of this layer and its ability to encode correlation between input channels by means of firing rate for ideal Poisson inputs.

7.1.3 Encoding seizures with partial synchronization using the hardware SNN

The above-described network is used for a proof-of-concept of real-time on-chip seizure encoding using the event-encoded EEG signals proposed in Sec. 7.1.1. As the dynamics of these spike trains are substantially distinct from Poisson-based spike trains (Fig. 7.4), I reduced the time-constant of the NLNG neurons compared to the previous section. Moreover, I further analyze the output of the NLNG neurons by

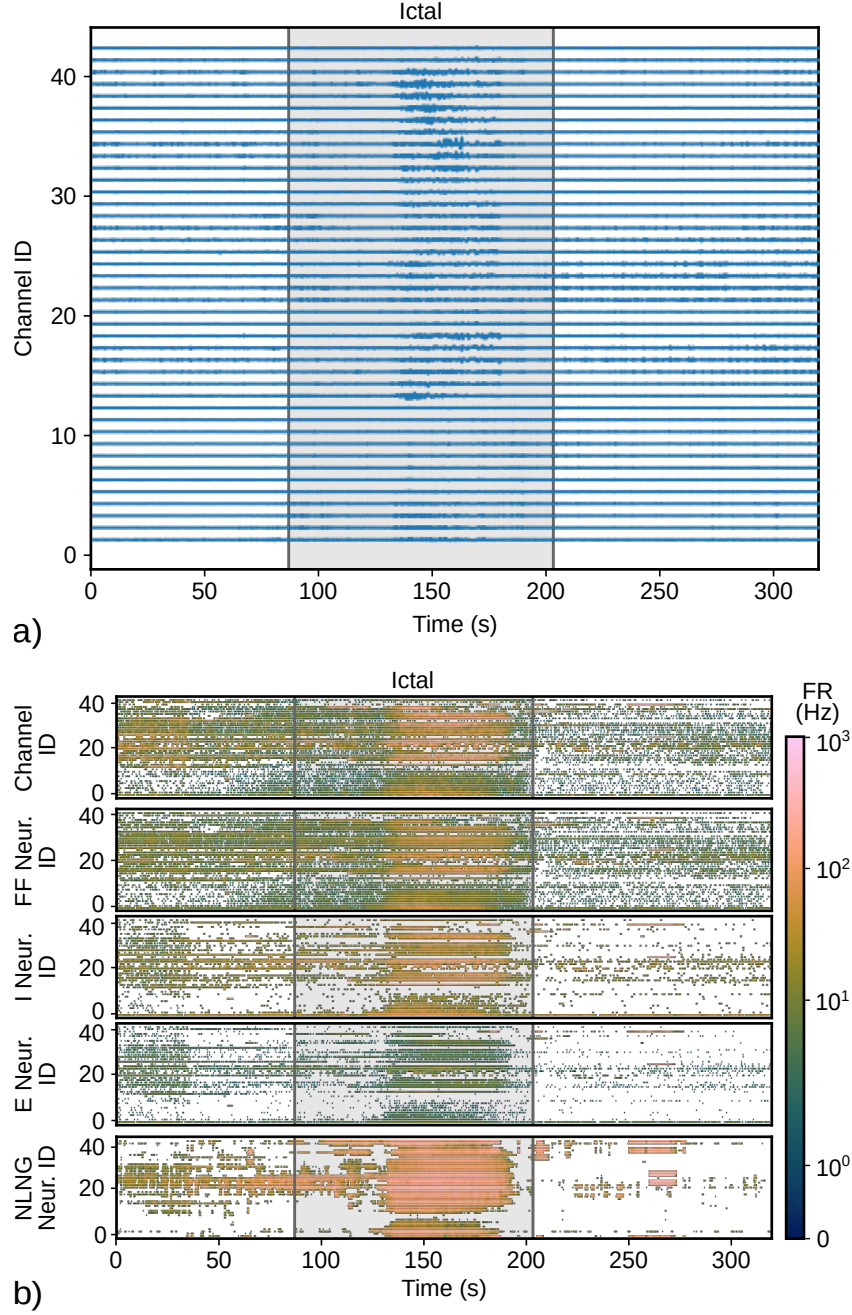


Figure 7.7: a) 42 channels of input EEG signals from patient 1, seizure 1; b) Output rasterplots of the on-DYNAP-SE2 implemented network. Top row to bottom: ADM-encoded input of a), Feedforward (FF) inhibitory neurons' outputs, Inhibitory (R-I) and Excitatory (R-E) neurons' outputs of the E-I pairs, and outputs of the Non-Local-Non-Global (NLNG) neurons. The firing rate is color-encoded per one-second time-segment;

means of synchronization matrices.

Figure 7.7a shows 42 raw iEEG waveforms of the first seizure of patient 1 from the SWEC-ETHZ dataset with annotated ictal (seizure) period (indicated by the grey area). The events of each EEG channel are simultaneously fed to the SNN cores of DYNAP-SE2 in real-time (Fig. 7.7b, row-1). The translation layer rejects

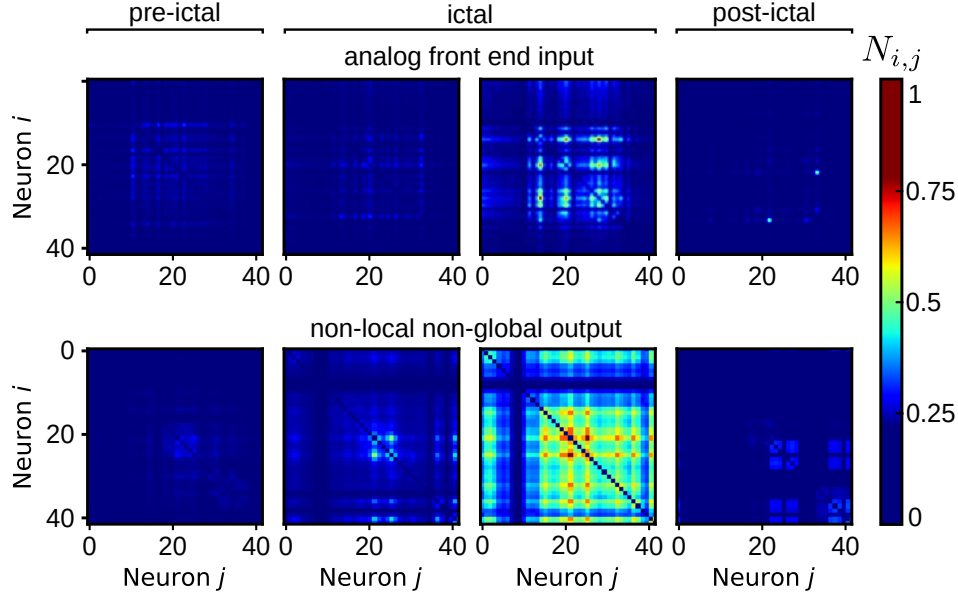


Figure 7.8: Normalized synchronization matrices between channel and neuron pairs of AFE input and NLNG output, respectively. From left to right: the pre-ictal, ictal, and post-ictal periods. The synchronization matrices during the ictal period (2nd and 3rd column) are created from the first half and second half of the seizure in time, respectively.

the lowest firing rate input spikes and reduces high input firing rates (Fig. 7.7b, rows 2,3,4) while keeping a linear relationship between the input and output firing rates similar to Fig. 7.6a.

The NLNG layer shows a high firing rate for the R-E neuron inputs that is correlated locally (Fig. 7.7b rows 4,5), best observed between $t = 125$ s and $t = 190$ s. Furthermore, a firing rate nearing 50 Hz can be observed in the pre-seizure period, between $t = 0$ s and $t = 40$ s across R-E neurons 15 to 30. Here, the NLNG layer does not spike at firing rates > 100 Hz. Moreover, during the period of time $t > 40$ s, most of the firing rate from this group of R-E neurons disappears. Adversely to this reduction, the firing rate of neurons 20 to 25 in the NLNG layer increases substantially to values above 100 Hz and continues to increase until the seizure ends. This points towards the dominant excitation of the NLNG layer, namely correlated input even at low firing rates.

In addition to the above analysis, I have employed kernelized spike trains with causal exponentially decaying functions [436], denoted as $f_i(t)$ where i indicates the NLNG neuron index. Using these kernelized spike-trains, I determine a measure of synchronization by taking the Manhattan (L_1) norm of the inner product between neuron pairs i and j , $N_{i,j} = || \langle f_i, f_j \rangle ||_1$ and average them over a fixed time period; see Methods for a step by step explanation. In essence, I define synchronization as the similarity of spike-trains between neurons over fixed time periods and quantify it with the above.

Figure 7.8 shows the normalized synchronization matrices between each AFE event stream (top) and each NLNG neuron $N_{i,j}$ pair (bottom) for a single seizure. These matrices demonstrate the functionality of the NLNG layer, namely, enhancement of the inherent correlation structure present when seizures occur. Fig. 7.9a shows accumulated Manhattan norm over time which confirms these emergent syn-

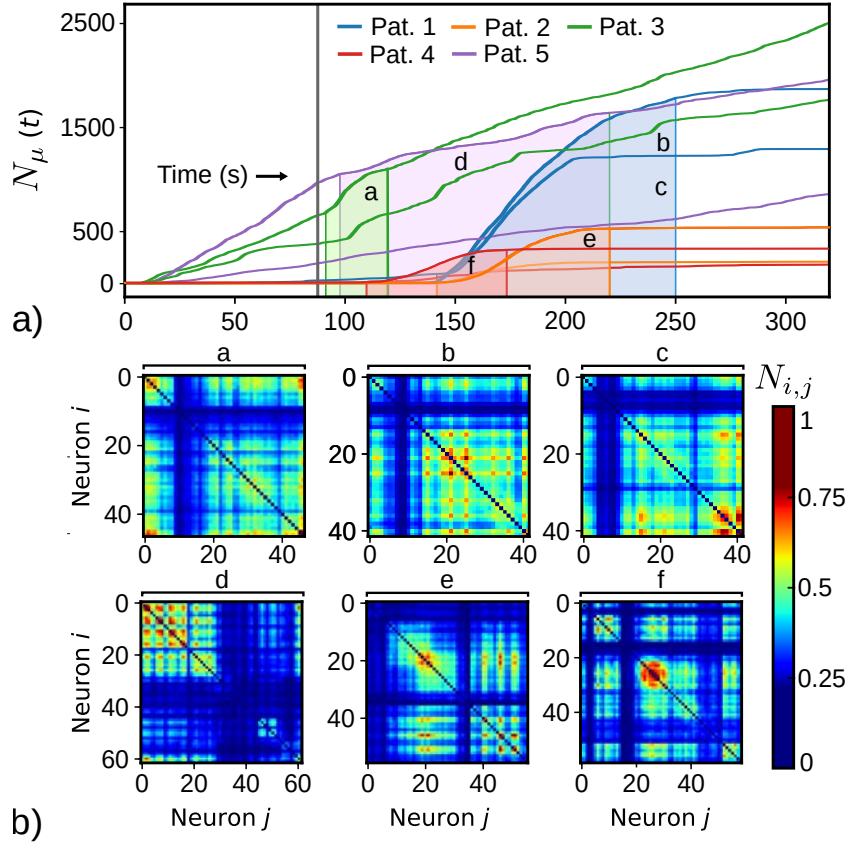


Figure 7.9: a) The mean synchronization over all pairs of the NLNG layer cumulatively summed over time, colored areas under the curve indicate a hand-selected period of time used to construct synchronization matrices in b); b) Synchronization matrices taken during periods of maximal synchronization.

chronization phenomena are replicated by the NLNG layer across all patients during the seizure period. The clusters emerging in Fig. 7.9b for different patients in the seizure period (as labeled in Fig. 7.9a) show emergent localized synchronization.

7.1.4 Detecting seizures using SNN encoded features

The output spike trains of the NLNG layer and the original AFE input are used as features for seizure detection with a linear classifier offline. This empirically shows the ability of the SNN on DYNAP-SE2 to distinguish between ictal and non-ictal periods. In this case, the proposed SNN on DYNAP-SE2 described in Fig. 7.5 functions as a feature encoder, mapping raw input data onto a hyperdimensional plane that can subsequently be read out and decoded. The output spike trains of the NLNG layer are publicly available (see data availability statement) to allow for any follow-up studies without requiring access to a DYNAP-SE2. For the sake of brevity, I show the separability of ictal and non-ictal periods by employing a linear Support Vector Machine (SVM) and firing-rate binning.

The NLNG and AFE spike trains are binned with one-second windows, channel by channel, normalized, and used to classify ictal and non-ictal periods using a linear SVM. Bayesian optimization and 5-fold cross-validation with stratified splitting are employed for training and testing. Table 7.1 describes the result of seizure detec-

Patient	Specificity [%]	Sensitivity [%]	Accuracy [%]
	AFE / NLNG	AFE / NLNG	AFE / NLNG
1	96.0 / 92.2	90.3 / 91.5	92.9 / 91.8
2	95.4 / 94.5	80.3 / 83.0	89.3 / 89.8
3	76.3 / 70.4	87.3 / 90.2	82.8 / 82.3
4	97.0 / 93.1	77.9 / 76.8	91.7 / 88.6
5	85.6 / 83.2	79.9 / 79.4	83.1 / 81.4

Table 7.1: Seizure classification results and Engel scores with a linear SVM that uses the Analog Front End (AFE) and encoded features of the Non-Local Non-Global (NLNG) layer as input.

tion per patient. The SVM is able to achieve accuracies between 81.4–91.8% with a specificity and sensitivity of 70.4–94.5% and 76.8–91.5% respectively using the NLNG layer output. These performance metrics are similar to using AFE input for SVM classification, resulting in an accuracy between 83.1–92.9% and a specificity and sensitivity between 76.3–97.0% and 77.9–90.3% respectively.

7.2 Discussion

This work presents a neuromorphic approach for a real-time seizure monitoring system by employing asynchronous delta modulation based encoding and analog neuronal dynamics on-chip. Instead of the conventional machine learning training methods for solving bio-signal processing tasks with digital computing systems, the approach proposed capitalizes on the inherent dynamics of analog neurons embedded within the hardware. As a major advance over previous applications [437, 438], the spike encoding is realized here with an on-chip AFE, unlike previous simulation-based approaches. Furthermore, by designing a hardware-compatible shallow spiking neural network that couples the encoded input spike-trains in an adjacent but non-global fashion, the prototype neuromorphic processing system developed is able to mimic correlation and form synchronized clusters. This mechanism is indicative of the network’s ability to mirror partial synchronous states observed in the EEG recordings during seizures, representing the first proof-of-concept for encoding seizures through partial synchronization in real-time. Moreover, compared to conventional seizure-detection systems [420], the prototype embedded system developed does not operate based on a global system clock but processes signals in a real-time data-driven mode with asynchronous spike routing and analog neurons [304]. This is a distinct advantage in terms of power consumption and scalability for always-on applications as for most of its operational time, in the absence of high energy inputs, the system is silent, significantly prolonging the monitoring period [439, 440].

The observation that the NLNG layer imitates the ictal state in real-time by forming partially synchronized clusters within itself can be studied to understand the temporal characteristics of seizure patterns. Future work could involve investigating the formation of the synchronized clusters through time and mapping their relation to the location of the recording channels, helping in tracking the progression of the seizure. Such work can help in the identification of the seizure onset zones which are responsible for seizure generation and provide insights into the disorder itself.

Classification with a linear SVM shows that the NLNG neurons are capable of separating ictal and non-ictal periods. However, please note that the accuracy of such an offline linear classifier does not change much between the firing rate of AFE-encoded event streams and NLNG-encoded spike streams, which shows that the firing rate in the current setup is not a good feature for seizure detection. On the other hand, the correlation patterns between neuron pairs in the AFE event streams and the NLNG spike streams varies significantly (Fig. 7.8). This suggests a future investigation avenue to design an event-based output layer that can perform seizure detection based on the correlation patterns of neuron pairs taking advantage of the emergent locally synchronized clusters. This approach can also be expanded for the identification of different temporal zones in a seizure event and also might help with the classification of seizure types based on partially-synchronized cluster patterns. However, please note that the currently utilized linear SVM is limited due to the use of firing-rate encoded features from the NLNG layer. Further work is necessary to determine on-chip neuromorphic approaches for implementing the decoding of the NLNG layer into different classes dealing with either seizure types or the characterization of temporal domains.

This work serves as a gateway to a more universal approach to exploiting the dynamics of analog neurons and their use in classification tasks. The NLNG layer, coupled with event-encoded input channels through a tuned translation layer, showcases the potential of analog computation in encoding non-linear dynamics of complex systems. As is exemplified by the structure of the NLNG weight matrix, I believe network structure, emphasizing topology and reciprocity, appears particularly well-suited to the intricacies of mismatched neuromorphic devices. Given the inherent variability of on-chip weights and parameters in mixed-signal or analog neuromorphic devices [441], integrating the lottery ticket hypothesis [442,443] with the hardware’s dynamical and topological properties could pave the way for additional advancements. I believe this interdisciplinary approach could open up new avenues for the widescale adoption of these devices in a diverse number of applications that are not limited only to bio-medical domains but expand into any complex systems.

7.3 Conclusions

This article presents a first step towards a real-time event-based seizure monitoring system. I introduced an event-based EEG seizure dataset created with an on-chip AFE followed by an SNN with special connectivity to encode seizures with a mixed-signal neuromorphic device. The SNN on-chip shows a proof-of-concept to encode seizures during ictal periods using its inherent dynamical properties.

To our knowledge, this is the first human EEG seizure dataset that has been encoded into events using a real-time delta-modulator. The dataset and synchronization encoded spike trains have been made publicly available [434]. The framework of this investigation provides, on one hand, the detailed specifications for the design of new front-ends and SNNs implemented on neuromorphic hardware, optimized for on-edge seizure monitoring, and on the other creates a foundation for the design of a more general class of always-on embedded and low-power neural network computing systems for a wide range of biomedical signal processing tasks. Inspired by the synchronization phenomena of the previous Chapter, this work provides according to my knowledge the first time that a Spiking Neural Network on-chip is used for encoding seizures with partial synchronization. In essence, this is a first step towards

neural network codesign using the dynamics of a mixed-signal hardware to employ new kinds of algorithms as exemplified with the structure of the SNN.

7.4 Methods

7.4.1 AFE and encoding setup

The dataset chosen in this work consisted of intracranially recorded iEEG signals which after undergoing 16-bit analog-to-digital conversion, and being band-pass filtered between 0.5 and 150 Hz, were recorded digitally at a sampling rate of 512 Hz. To mimic a real-world scenario, we utilized a DAC stage (acting as a virtual sensor) to convert the digitally recorded waveforms of the dataset into analog signals. This DAC stage consists of an 18-bit DAC (AD5781, Analog Devices, Wilmington, MA) and a custom PCB (designed with support from the Microelectronics Design Center of ETH Zurich), allowing regeneration of the digital data into corresponding analog waveforms (Fig. 7.3a). The DAC is controlled by a microcontroller (Teensy 4.1 from PJRC, Sherwood, OR) which transmits a digital signal stored in an SD card to the DAC via an SPI bus. The input signal is generated with a ~ 100 mV peak-to-peak voltage and with a ~ 30 mV offset for all DAC conversions. The EEG signal amplitude is scaled to such values considering the DAC PCB and AFE characteristics. A sample from the oscilloscope is presented in Fig. 7.4a, where the analog signal (trace 4; purple) can be seen.

The AFE signal conditioning stage consisted of a high-fidelity low-noise signal amplifier and a filter bank as depicted in Fig. C.2a. We employed a 20 dB default amplification with no filtering. Note that the source signal (from the database) is pre-filtered to be within the 0.5 to 150 Hz frequency band at the source. This amplified unfiltered signal is then passed on to the ADM. Although both the AFE and SNN cores are located within the same die, we have extracted the output events from the ADM sequentially, i.e., channel by channel, (for limited I/O of the AFE on the die as compared to the number of input EEG channels) before sending these simultaneously from off-chip to the SNN cores (Fig. C.2b).

7.4.2 Delta modulated EEG events

Asynchronous Delta Modulation [444, 445] is a delta modulation approach where the sampling interval is adapted based on the characteristics of the coded signal. In the current context, it represents a level shifter circuit that outputs an event based on the difference in the amplitude of a signal between two sampling points. If two sample points (which might not be consecutively placed, representing asynchronous sampling obeying causality) of a signal show a change in amplitude that is more than a preset delta-threshold, the encoder records an event. An ‘UP’ or ‘DOWN’ tag is attached based on the polarity of the change, i.e. if the signal has increased or decreased at the level crossing. Further details on the circuit schematics of the AFE and the ADM can be found in Sharifshazileh et al. [437]. Due to the initial stabilization of analog components within the AFE circuit, we obtained 320 second snippets of each channel for a given seizure following a discarded two-minute pre-seizure segment.

7.4.3 On-chip SNN parameters

Tables 7.2 and 7.3 describe the relationship between the rise and decay times of the neuron membrane potential as a result of a single spike for all neuron types in this work’s proposed SNN. A high time constant leads to a lower rise and decay time as the decay of the synaptic current is reduced. This confirms that the FF

neurons are more sensitive to low frequency inputs. Similarly, the rise time of the NLNG neuron is the longest: the decay time is not due to fewer connections, as it has the lowest time constant. For each measurement, the number of equivalently weighted synapses, referred to as the "weights" in the SNN, are set to ensure that the membrane potential reaches a value close to the threshold. In the case of the SNN in this work, the number of equivalent connections of the translation layer is set to ten throughout all translation layer synapses and one or two in the NGNL layer. Such heterogeneous neuromorphic processing systems with mixed-signal analog/digital electronic circuits are extremely noisy and imprecise owing to their sub-threshold operation with very small currents. Thus, the values mentioned in the parametric tables are an estimation and more insights into the hardware effects on parametric values can be found in Zendrikov et al. [441].

Neuron type	τ_{rise} (ms)	τ_{decay} (ms)	ΔU (mV)	No. of connections
$U_{\text{mem}}^{E,I}$	46.1	378.7	248.6	5
U_{mem}^{FF}	8.9	153.8	86.2	6
U_{mem}^{NLNG}	74.6	274.3	43.8	4

Table 7.2: Measured change in voltage, and rise and decay times of the membrane potential as a result of a single excitatory spike input per neuron type.

Neuron type	R - E exc. / inh.	R - I	FF	NLNG
$I_{\tau_{\text{exc/inh}}}$ (pA)	44.1 / 356.4	44.1	66.5	17.8
$I_{w_{\text{exc/inh}}}$ (nA)	16.7 / 34.4	16.7	16.7	16.7

Table 7.3: The bias generator currents on DYNAP-SE2 for setting network weights and synaptic compartment time constants (excitatory (NMDA like) and inhibitory (GABA-B like)).

7.4.4 Statistical measures

To quantitatively determine the correlated state of NLNG neurons across long time-scales, namely, synchronization, we apply the inner product between kernelized spike trains of neuron pairs and use the Manhattan (L_1) norm of this product.

7.4.5 Kernelizing spike-trains to continuous functions

Due to the loss of information from spike-train binning, we have transformed spike-trains into continuous functions by means of kernelization [436]. This kernelization is described by a convolution

$$f_i(t) = i * h \quad (7.1)$$

where i represents the spike train of neuron i as a sum of delta dirac functions and h is the kernel, represented by a causal exponential decay [436, 446]. These can be

described by

$$i(t) = \sum_m \delta(t - t_m) \quad (7.2)$$

$$h(t) = \begin{cases} e^{-t/\tau}, & \text{if } t \geq 0 \\ 0, & \text{otherwise} \end{cases} \quad (7.3)$$

where t_m represents the spike time of spike m and τ is the time constant assigned to the kernel h . Here, τ is set to 1 ms to include the long and short-term effects of spikes. Due to the nature of spike trains, eq. 7.2 can be rewritten as

$$f_i(t) = \sum_m h(t - t_m) \quad (7.4)$$

where $f_i(t)$ becomes a continuous exponentially decaying function starting from each spike.

7.4.6 Manhattan norm and synchronization matrices

The inner product between pairs of continuous spike trains, defined by eq. 7.4, is described by

$$\langle f_i, f_j \rangle = \sum_{m,n} \int h(t - t_m^i) h(t - t_n^j) dt \quad (7.5)$$

where $\langle f_i, f_j \rangle$ is the inner product between f_i and f_j representing the continuous forms of the spike trains of neuron i and neuron j respectively, further details in Ref. [436]. Now I take the Manhattan (L_1) norm of this inner product as a measure of synchronization

$$\| \langle f_i, f_j \rangle \|_1 = \frac{1}{T} \sum_t \langle f_i, f_j \rangle \quad (7.6)$$

where $\langle f_i, f_j \rangle \geq 0$ across all i and j . For brevity, I represent this measure as $N_{i,j} = \| \langle f_i, f_j \rangle \|_1$ throughout the paper.

Patient	Age (y)	SWEC label	No. of electrodes	Seizure durations Sez-1/Sez-2 (s)	Epilepsy type
1	19	ID2	42	96 / 260	TLE
2	27	ID9	56	104 / 118	TLE
3	24	ID1	47	125 / 70	TLE
4	31	ID16	59	81 / 67	TLE
5	32	ID4	62	160 / 127	PLE

Table 7.4: Patient characteristics. TLE Temporal Lobe Epilepsy, PLE Parietal Lobe Epilepsy.

7.4.7 SWEC-ETHZ iEEG dataset

The intracranial EEG data used in this study (Table 7.4) were downloaded from a public epilepsy dataset [433], assembled at the Sleep-Wake-Epilepsy-Center (SWEC) of the University Department of Neurology at the Inselspital Bern and the Integrated

Systems Laboratory of the ETH Zurich, available from <http://ieeg-swez.ethz.ch/>. Detailed information about the acquisition methods and patient metadata is contained in Refs. [447, 448]. The iEEG dataset is recorded during pre-surgical evaluation for epilepsy brain surgery. The intracranial EEG was recorded with strip, grid, and depth electrodes. The data was pre-filtered to the 0.5 to 150 Hz frequency band, sampled at 512 Hz, and further preprocessed to remove artifact-corrupted channels. Annotated seizure onsets and offsets were obtained from visual inspection of an EEG board-certified, experienced epileptologist. Each patient has recordings of multiple seizure events consisting of three minutes of pre-seizure segments (system state immediately before the seizure onset), the seizure segment (seizure event ranging from 10 s to 1002 s), and three minutes of post-seizure segment (i.e., system state after seizure event). All the information about the dataset was taken from the dataset website [433] and the associated publications [447, 448].

Due to hardware-related constraints described previously in Sec. 7.4.1, we have chosen a subset of the short-term dataset containing five patients, 2 seizures per patient. Furthermore, considering a constraint on the number of simultaneous input spike trains to the SNN on the event-based processors [304], we have selected patient data with less than 64 electrodes.

7.5 Data availability statement

The event-based dataset, available at Ref. [434], is constructed from a publicly available dataset available at Ref. [433]. The output spike-trains from the AFE and the NLNG layer along with the DAC-generated values of the analog EEG signals are published in <https://doi.org/10.5281/zenodo.10800794>.

7.6 Code availability statement

Data processing and analysis were conducted in Python 3 and the code has been made available at Ref. [434] and published in <https://doi.org/10.5281/zenodo.10800794>.

7.7 Supplementary materials

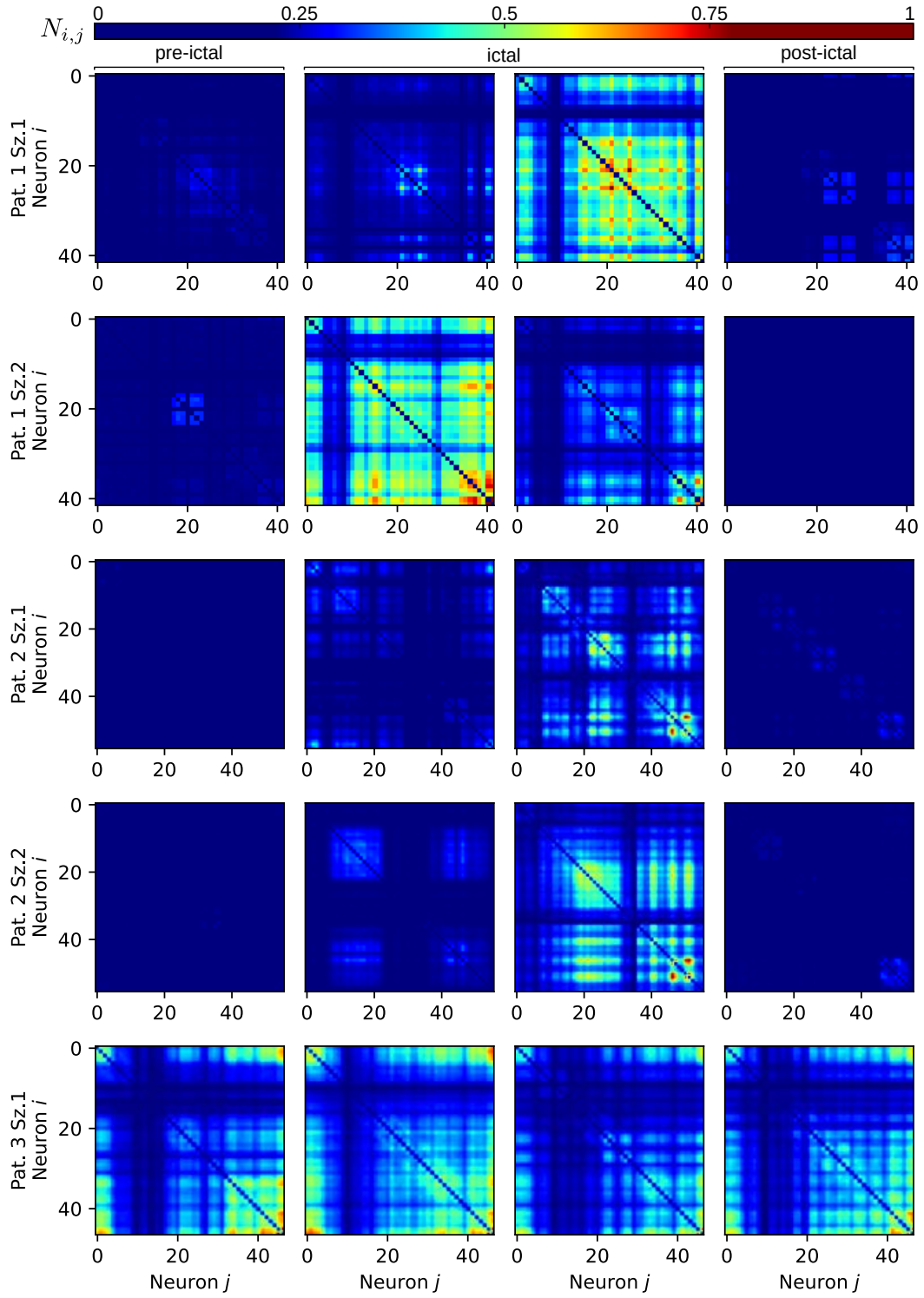


Figure 7.10: Synchronization matrices of Pat. 1, 2 and 3. The columns from left to right show synchronization during the pre-ictal, ictal and post-ictal periods respectively.

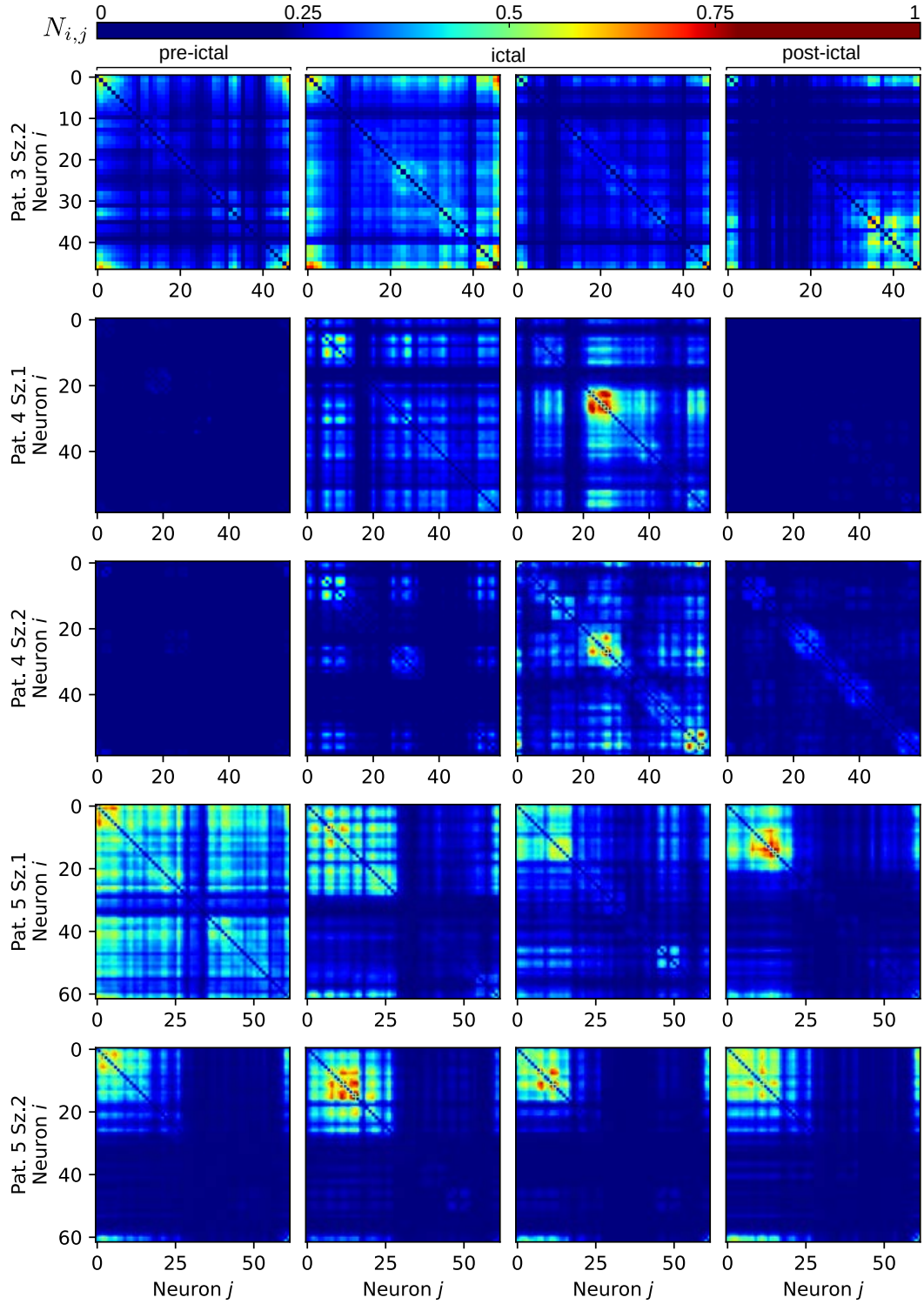


Figure 7.11: Synchronization matrices of of Pat. 3, 4 and 5. The columns from left to right show synchronization during the pre-ictal, ictal and post-ictal periods respectively.

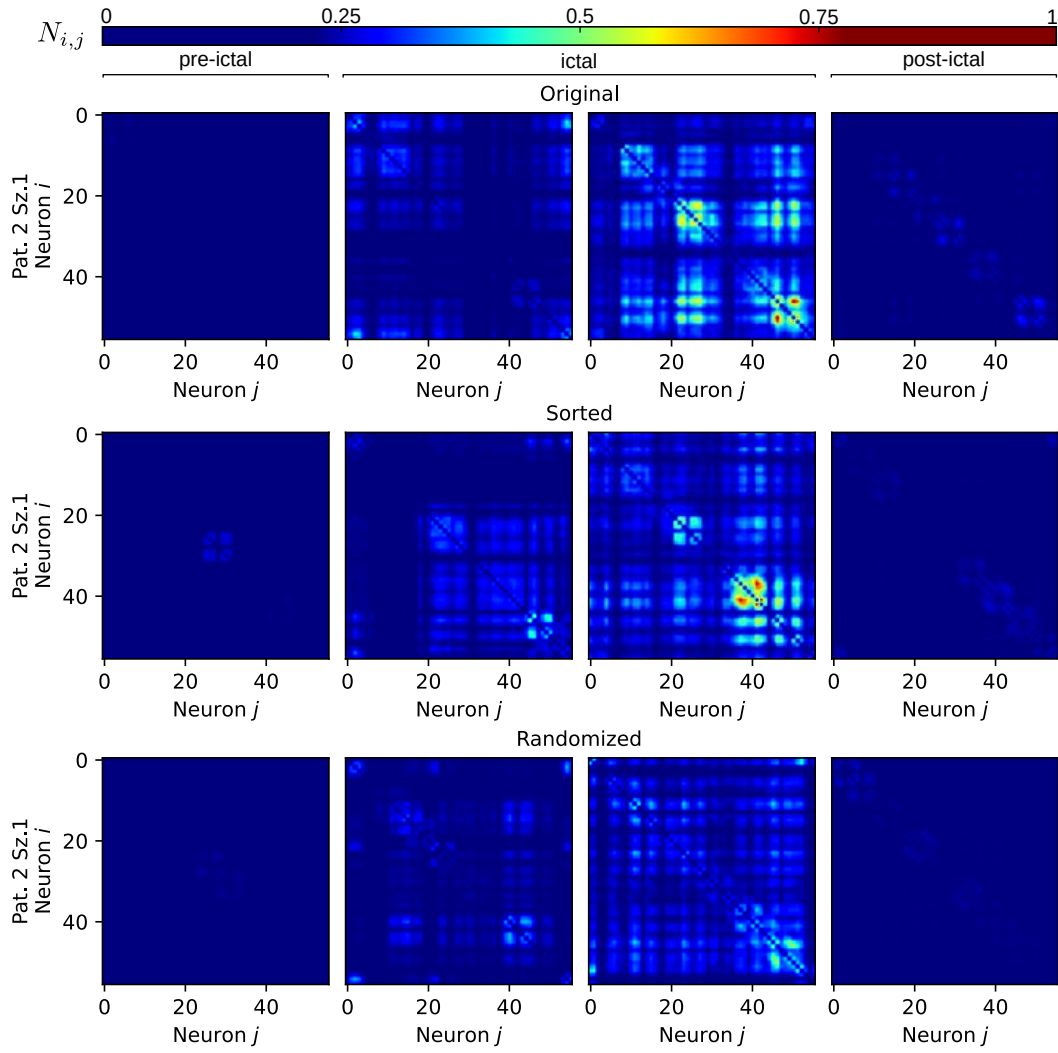


Figure 7.12: Non-local non-global neuron spike train synchronization matrices of patient 2 seizure 1 based on three distinct input channel ordering methods. The first, second, and third rows represent synchronization based on input with the original dataset order, maximum correlation of the original data, and randomized order respectively.

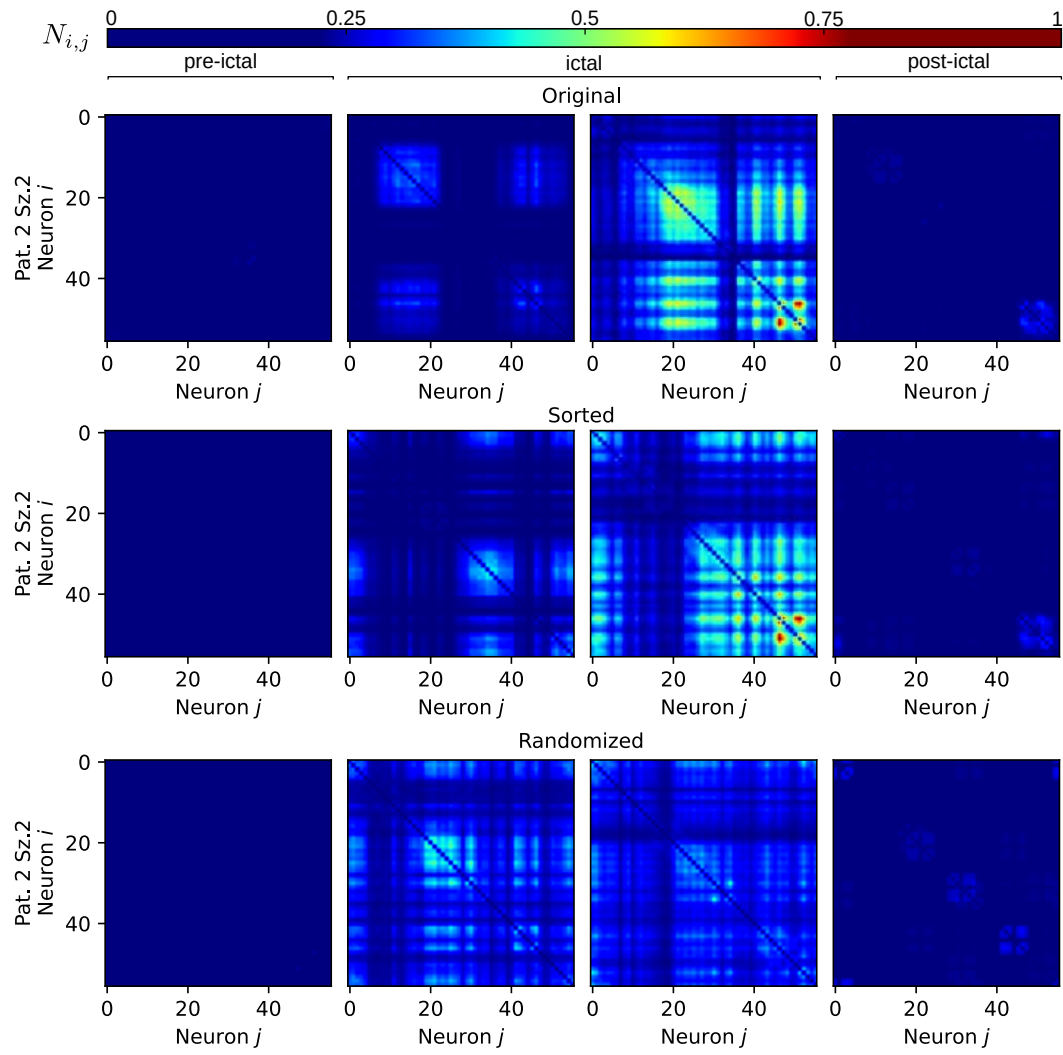


Figure 7.13: Non-local non-global neuron spike train synchronization matrices of patient 2 seizure 2 based on three distinct input channel ordering methods. The first, second, and third rows represent synchronization based on input with the original dataset order, maximum correlation of the original data, and randomized order respectively.

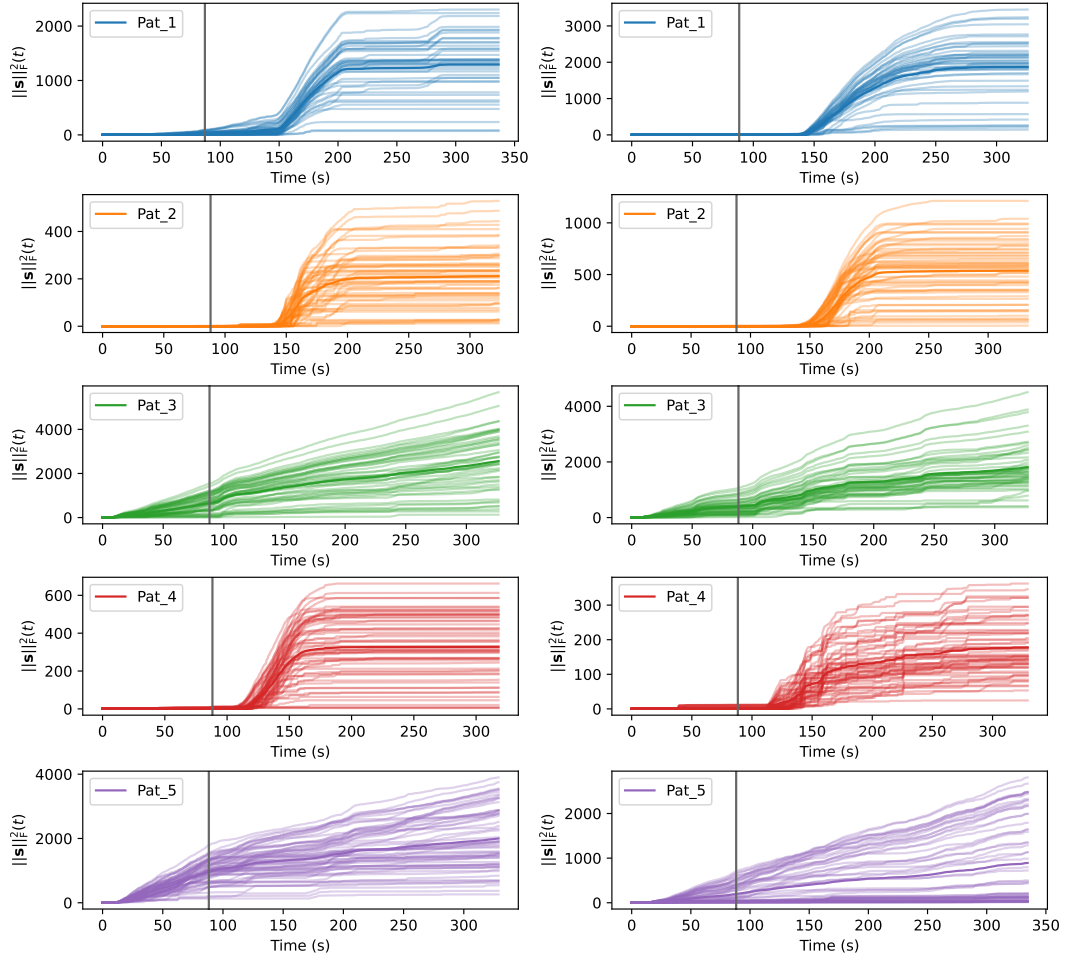


Figure 7.14: The synchronization across all non-local non-global layer neurons over time for all patients and seizures. Grey line indicates seizure start.

+

Chapter 8

NimbleSNN: A 1.34 μ J/Image 4639 Logic Element Single-Spike Accelerator on FPGA Using Synchronization

This Chapter starts with a review in Section 8.1 on three significant works that propose time-to-first-spike (TTFS) coding within supervised SNNs. This is followed by a study on the use of synchronization to create "NimbleSNN" in Section 8.2, a small but highly efficient SNN hardware architecture for an MNIST accelerator that is implemented on the ICE40UP5K FPGA. Employing synchronization to discover synchronous clusters within the Binarized Single Spike Supervised Spiking Neural Network (BS4NN), proposed in Ref. [449], facilitates a 24.9% reduction in the number of weights. This is subsequently followed by a mask around the corners of each MNIST image, leading to a further 33.7% weight reduction. In terms of hardware architecture, the use of synchronization to schedule neurons within memory and thus enabling substantial reduction of the size of spike buffers, an essential element within hardware architectures, leads to a reduction in the number of logic elements on FPGA by 45.81%. As a result, NimbleSNN employs only 4639 logic elements for implementation on the ICE40UP5K FPGA.

The above described methods culminate in the first MNIST-sized SNN, i.e., 448 neurons, on an FPGA with less than 5280 LUTs and registers. Moreover, this implementation shows a measured energy per inference of 1.34 μ J and an effective efficiency of 47.7 GSOp/s/W, therefore possible making NimbleSNN the most efficient MNIST accelerator on FPGA at the time of writing. Furthermore, at an operational efficiency of 21.0 pJ/SOp it comes close (1 magnitude of difference) in terms of efficiency to existing works that shows ASIC accelerators for SNNs on MNIST. The use of synchronization in advancing digital SNN accelerators on FPGA provide a blueprint for future studies in multi-spike recurrent SNNs that are used in complex tasks such as natural language processing and computer vision.

8.1 A Review of Time-To-First-Spike Coding With a single spike per neuron

This Section provides an analysis of the three most significant works that propose time-to-first-spike (TTFS) coding within supervised spiking neural networks. Within time-to-first-spike encoding continuous data is encoded as single discrete spike. The time of the single spike is negatively proportional to the magnitude of the continuous data considered; the maximum value of a real number is mapped to the minimum value in time.

8.1.1 Input encoding

In temporal encoding, input data is encoded by the following description:

$$t_{\text{in}} = \left\lfloor \frac{a_{\text{max}} - a_{\text{in}}}{a_{\text{max}}} \right\rfloor t_{\text{max}} \quad (8.1)$$

where t_{in} defines the time of the input spike, and a_{max} , a_{in} and t_{max} define the maximum input value (255 in 8-bit MNIST images), the input value (pixel intensity), and the maximum time on the time axis in SNNs, respectively. The braces, " $\lfloor \cdot \rfloor$ ", indicate rounding due to quantization.

8.1.2 Forward Propagation

In the case of TTFS coding, the membrane potential is described as

$$U_j^l(t) = \sum_{i \in I} w_{ij}^l \sum_{t_i \in T_I} S_{ij}^{l-1}(t) \quad (8.2)$$

where $U_j^l(t)$ is the membrane potential of neuron j and w_{ij}^l is the synaptic weight assigned to synapse between neuron i of the previous layer $l - 1$, and neuron j of layer l within the set of the connected neurons of the previous layer, I . Furthermore, the sum of $S_{ij}^{l-1}(t)$ represents the summation of spikes at times t_i of all neurons of the previous layer, stored within the set T_I .

One topic of interest is the method of converting digital spike events to an influx in membrane potential. Although many works have considered leakage as this closer to biology, various works that utilize TTFS encoding have disregarded this, reducing computation. Furthermore, the representation is an important consideration as it will have an effect on the method of enabling learning by means of backpropagation. Accordingly, the influx on the membrane potential has been described in the following ways.

In Ref. [175], output spikes are described by

$$S_j^l(t) = \begin{cases} 1 & \text{if } U_j^l(t) \geq \theta_j \text{ \& } S_j^l(< t) \neq 1 \\ 0 & \text{otherwise} \end{cases} \quad (8.3)$$

where a spike is output if the membrane potential exceeds threshold θ_j if and only if there has not been an output spike prior to time t . Moreover, Ref. [450] uses a piece-wise linear approximation, namely,

$$S_j^l(t) = \begin{cases} \frac{t - t_i^{l-1}}{\tau} & \text{if } t_i^{l-1} \leq t < t_i^{l-1} + \tau \\ 0 & \text{otherwise} \end{cases} \quad (8.4)$$

where eq. 8.3 represents an integrate-and-fire (IF) neuron and eq. 8.4 the spike response model neuron. Here, the current and change in voltage according to current is described as

$$I(t) = \sum_{i \in I} w_{ij} \quad (8.5)$$

$$\frac{dU}{dt} = I(t) \quad (8.6)$$

where the membrane potential is formulated the same as eq. 8.2.

8.1.3 Backward Propagation

In order to train networks on classification tasks, these works adapt the backpropagation training method to SNNs. The error is initially defined at the output, namely, the final layer that maps spikes into respective classes, before propagated backwards into the network. This error is defined as

$$e = [e_1, \dots, e_j, \dots, e_C] \quad (8.7)$$

where e is a vector containing all output neuron errors. Each individual neuron error e_j is defined as

$$e_j = \frac{(T_j^o - t_j^o)}{t_{max}} \quad (8.8)$$

where t_j^o is the actual firing time and T_j^o is the expected firing time. In Refs. [175,450], t_{max} is included in the denominator. However in [451] this term is omitted and the nominator is inversed.

The loss is then commonly defined as

$$E = \frac{1}{2} \sum_j e_j^2 \quad (8.9)$$

leading to training, namely, the change in weight according to the above to be described by

$$w_{ij}^l = w_{ij}^l - \Delta w_{ij}^l \quad (8.10)$$

$$\Delta w_{ij}^l = \begin{cases} \eta \frac{\partial E}{\partial w_{ij}^l}, & \text{if } t_j^l \leq t_{max} \\ 0, & \text{if otherwise} \end{cases} \quad (8.11)$$

where η is the learning rate. From hereon, all works diverge, each claiming their own novelty within TTFS coding. This is because the partial derivative in eq. 8.10 has no direct solution for SNNs, requiring expansion with the following chain rule,

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial t_j^l} \frac{\partial t_j^l}{\partial w_{ij}^l} \quad (8.12)$$

this can be further expanded as

$$\frac{\partial E}{\partial w_{ij}^l} = \frac{\partial E}{\partial t_j^l} \frac{\partial t_j^l}{\partial U_j^l} \frac{\partial U_j^l}{\partial w_{ij}^l} \quad (8.13)$$

where each work offers a different approximation described below.

Single Spike Supervised Spiking Neural Network (S4NN), Ref. [175]

Ref. [175] assumes an equivalence between the output of a ReLU neuron and the firing time of a TTFS IF neuron, i.e.,

$$y_j^l \sim t_{\max} - t_j^l \quad (8.14)$$

where y_j^l is the output of a ReLU neuron, namely,

$$y_j^l = \max(0, z_j^l = \sum_i w_{ij}^l x_{ij}^{l-1}) \quad (8.15)$$

The value of y_j^l becomes larger for large inputs with large synaptic weights. Inversely, in IF neurons, an earlier spike corresponds to larger synaptic weights. This description leads to this negative relationship.

Accordingly, the approximation of the partial derivative of firing time with respect to the membrane potential is defined as follows

$$\frac{\partial t_j^l}{\partial U_j^l} \sim -\frac{\partial y_j^l}{\partial z_j^l} \approx -1 \quad (8.16)$$

under the condition that $y_j^l > 0$.

Furthermore, the first term on the RHS in eq. 8.13 is expanded by the chain rule

$$\frac{\partial E}{\partial t_j^l} = \frac{\partial E}{\partial e_j} \frac{\partial e_j}{\partial t_j^l} = \delta_j^l \quad (8.17)$$

this is solvable at the output layer $l = o$, i.e.,

$$\frac{\partial E}{\partial t_j^o} = \frac{-e_j^o}{t_{\max}} \quad (8.18)$$

however in Ref. [175], the $t_{\max}$ term is omitted.

Then, eq. 8.13 reduces to

$$\frac{\partial E}{\partial w_{ij}} = e_j^o \sum_{t_i \in T_I(t \leq t_j^l)} S_{ij}^{l-1}(t) \quad (8.19)$$

at the output layer. Note, that $\sum_{t_i \in T_I} S_{ij}^{l-1} = 1$ only for $t_i^{l-1} \leq t_j^l$. Now backpropagation is applied to train the weights of the previous layers. The first term on the RHS in eq. 8.13 has to be rewritten with the chain rule to arrive at the following equation

$$\frac{\partial E}{\partial t_j^l} = \frac{\partial E}{\partial t_k^{l+1}} \frac{\partial t_k^{l+1}}{\partial t_j^l} = \frac{\partial E}{\partial t_k^{l+1}} \frac{\partial t_k^{l+1}}{\partial U_k^{l+1}} \frac{\partial U_k^{l+1}}{\partial t_j^l} = \frac{\partial E}{\partial t_k^{l+1}} (-1) \left(\sum_k w_{kj}^{l+1} \right) \quad (8.20)$$

where the other terms remain equivalent.

STiDi-BP, Ref. [450]

Similar to eq. 8.11, the change in weight is defined as

$$\Delta w_{ij}^l = \begin{cases} \eta \frac{\partial E}{\partial t_j^l} \frac{\partial t_j^l}{\partial U_j^l} \frac{\partial U_j^l}{\partial w_{ij}^l}, & \text{if } t_i^{l-1} \leq t_j^l \\ 0, & \text{if otherwise} \end{cases} \quad (8.21)$$

the difference emerges in the definition of weight change, namely, setting of the weight change to zero in the case that a neuron in the previous layer fires later, at t_i^{l-1} compared to neuron j at t_j^l . This is similar to eq. 8.11, but there the weight change is set to zero when the previous neuron fires later than time constant $t_{\max}$.

The chain rule is then once again derived according to the following descriptions under the condition of eq. 8.21, namely,

$$\frac{\partial E}{\partial t_j^l} = \frac{-e_j^l}{t_{\max}} \quad (8.22)$$

$$\frac{\partial U^l(t)}{\partial w_{ij}^l} = \epsilon(t_j^l - t_i^{l-1}) \quad (8.23)$$

Here the time is assumed to be at $t = t_j^l$ as this is the time of spiking. The derivative of the firing time over that of the membrane potential is in this case approximated by means of a linear interpolation and as the inverse of this, i.e.,

$$\frac{\partial t_j^l}{\partial U_j^l(t)} = \frac{1}{\frac{\partial U_j^l(t)}{\partial t_j^l}} \simeq -\frac{t_j^l}{U_{th}} \quad (8.24)$$

these then lead to the change in weight to be approximated by

$$\Delta w_{ij}^l = \begin{cases} \eta \frac{e_j^l}{t_{\max}} \frac{t_j^l}{U_{th}} \epsilon(t_j^l - t_i^{l-1}), & \text{if } t_i^{l-1} \leq t_j^l \\ 0, & \text{if otherwise} \end{cases} \quad (8.25)$$

the important aspect to observe from the above equation is that there is no need for recursive gradients. As a matter of fact other than the expected firing time, all parameters are used within the forward pass.

Supervised Spike Time Dependent Plasticity (SSTDTP), Ref. [451]

This paper proposes not to approximate the derivative of the spike time over membrane potential as in the above, rather they propose to use spike-time dependent plasticity (STDP) to approximate the second term of eq. 8.12. This is defined by the following, maintaining the notation within this review,

$$\frac{\partial t_j^l}{\partial w_{ij}^l} = \begin{cases} k_1 \left(\frac{t_j^l - t_i^{l-1}}{\tau} - \delta \right) (w_{\max} - w_{ij}^l)^\mu, & \text{if } t_j^l > t_i^{l-1} \\ k_2 \left(\frac{t_i^{l-1} - t_j^l}{\tau} - \delta \right) (w_{\max} - w_{ij}^l)^\mu, & \text{if } t_j^l \leq t_i^{l-1} \end{cases} \quad (8.26)$$

where $k_{\{1,2\}}$ are the scaling factors, δ is the weight update time interval and τ is the time constant. The second factorized term constraints the weight change to the

max weight w_{max} and a parameter μ . An important point to note is that the effect of a pre-synaptic neuron firing later than a post-synaptic neuron is not neglected as is the case for Ref. [175, 450]. This term can be stored locally at each neuron within the network and does not require back propagation.

Similar to the previously introduced works, the first term of eq. 8.12 is defined as follows

$$\frac{\partial E}{\partial t_j^l} = \frac{\partial E}{\partial e_j} \frac{\partial e_j}{\partial t_j^l} = e_j \quad (8.27)$$

where $e_j = t_j^l - T_j^l$. This term is back propagated through all layers in the following manner

$$\frac{\partial E}{\partial t_i^{l-1}} = \sum_j \frac{\partial E}{\partial t_j^l} \frac{\partial t_j^l}{\partial t_i^{l-1}} \quad (8.28)$$

where the second term is approximated as

$$\frac{\partial t_j^l}{\partial t_i^{l-1}} = \begin{cases} 0, & \text{if } t_i^{l-1} > t_j^l \\ w_{ij}^l, & \text{if } t_i^{l-1} \leq t_j^l \end{cases} \quad (8.29)$$

where once again there is no impact on the weight change if the pre-synaptic neuron fires later than the post-synaptic neuron. This is confused with eq. 8.26 as there later firing does impact one of the terms used in back propagation.

8.1.4 Setting the target/relative firing times

One of the concepts that all of the above works introduce is the expected firing time T_j^l to calculate error e_j within a neuron j . This is defined differently within each of the works according to a set of rules. This subsection delves into the distinct definitions described in Refs. [175, 450, 451].

Output layer ($l = o$)

Starting, from the output layer, in Ref. [175], S4NN, the expected firing time at $l = o$ is described as

$$T_j^o = \begin{cases} \nu & \text{if } j = m_p \\ \max(t_j^o, \nu + \gamma) & \text{if } j \neq m_p \end{cases} \quad (8.30)$$

where m_p is the assigned output neuron of class label p , $\gamma > 0$ is a parameter that ensures separation of neurons with firing times close to parameter ν . ν is the parameter that sets the minimum firing time. A special case considers all neurons to be silent, their firing time set to t_{max} . In such a case the target firing times are calculated by

$$T_j^o = \begin{cases} t_{max} - \gamma, & \text{if } j = m_p \\ t_{max}, & \text{if } j \neq m_p \end{cases} \quad (8.31)$$

to stimulate the assigned class neuron to fire earliest. As recursive gradient is used for backpropagation, no other expected firing time are defined other than the output layer.

In Ref. [450] (STiDi-BP), the expected firing time at the output layer is defined as

$$T_j^o = \begin{cases} t_{\min} - \gamma, & \text{if } j = m_p \\ t_{\max} + \gamma, & \text{if } j \neq m_p \end{cases} \quad (8.32)$$

where the $t_{\min}$ and $t_{\max}$ are defined as the maximum and minimum output firing times, analogous to the notation in this review. The expected firing time for all neurons is calculated as there is no recursive gradient.

In Ref. [451], the expected firing times are once again only defined at the output as a recursive gradient is applied

$$T_j^o = \begin{cases} \min(t_j^o, T_{\text{mean}} - \frac{N-1}{N}g), & \text{if } j = m_p \\ \max(t_j^o, T_{\text{mean}} + \frac{1}{N}g), & \text{if } j \neq m_p \end{cases} \quad (8.33)$$

where

$$T_{\text{mean}} = \frac{1}{N} \sum_{j=1}^N t_j^o \quad (8.34)$$

where N is the amount of classes, T_{mean} is the mean firing time of the output layer and g is the minimum gap to separate neurons firings, similar to γ . The average firing time, although definable for all layers, is only calculated for the output layer as backpropagation is applied recursively.

Hidden layers ($l \neq o$)

Solely for Ref. [450], the expected firing time for all neurons within all layers has to be defined to prevent recursive gradient.

In the hidden layers, $l \neq o$, the expected firing time is defined as

$$T_i^l = t_i^l + \Delta t_i^l \quad (8.35)$$

where Δt_i^l is defined as

$$\Delta t_i^l = -\eta \sum_j \frac{\partial E_j^{l+1}}{\partial t_j^{l+1}} \frac{\partial t_j^{l+1}}{\partial U_j^{l+1}} \frac{\partial U_j^{l+1}}{\partial t_i^l} \quad (8.36)$$

these chain-rule expanded partial derivatives are then set to the following. First, the third term in the RHS is defined by the following

$$\frac{\partial U_j^{l+1}}{\partial t_i^l} = \begin{cases} -\frac{w_{ij}^{l+1}}{\tau}, & \text{if } t_i^l \geq t_j^{l+1} < t_i^l + \tau \\ 0, & \text{if otherwise} \end{cases} \quad (8.37)$$

as the other terms were already derived in eqs. 8.22 and 8.24, Δt_i^l can be written as

$$\Delta t_i^l = \eta \sum_j \frac{e_j^{l+1}}{t_{\max}} \frac{t_j^{l+1}}{v_{th}} \frac{w_{ij}^{l+1}}{\tau} \quad (8.38)$$

where the error e_j^{l+1} can have positive or negative values, allowing the firing time at an epoch to be set back or ahead in time. Although the above proposes to eliminate

the recursive gradient for back propagation, it can be observed that this system of equations does not necessarily decrease the complexity of computation. Therefore, it remains to be analyzed whether this can lead to less computation on an actual hardware.

8.1.5 Regularization and dropout

Spiking neural networks tend to have vanishing-exploding gradients, providing the necessity for layer or batch regularization. Here, I describe some regularization and dropout methods that are proposed in the three works of this review.

S4NN

To avoid vanishing-exploding gradients, normalization is applied at all layers (layer normalization) by all other gradients,

$$\delta_j^l = \frac{\delta_j^l}{\sum_i \delta_i^l} \quad (8.39)$$

where δ_i^l is the backpropagation gradient of all neurons in the layer except neuron j . This would require all gradient to be stored in hardware for every layer. Furthermore the weights are regularized according to a term that is added to the error function of eq. 8.9, namely,

$$E = \frac{1}{2} \sum_j e_j^2 + \lambda \sum_l \sum_{i,j} (w_{ij}^l)^2 \quad (8.40)$$

this term once again requires storage of another term in hardware.

In case a neuron never happens to fire because it cannot reach the required membrane potential, dubbed a dead neuron, the weights of the neuron are reinitialized and set according to a random distribution and fake spikes are fired at the last time step.

STiDi-BP

According to the paper's description, they apply the exact same regularization methods as S4NN. In addition, they randomly disable hidden layer neurons during the backward pass, dubbed dropout.

SSTDP

No dropout or regularization methods are described in this work.

8.2 Towards a TTFS SNN Accelerator on FPGA Using Synchronization

8.2.1 Introduction

The rise of Artificial Intelligence (AI) and Internet of Things (IoT) has catalyzed significant progress in domains such as healthcare, natural language processing, and computer vision, predominantly through the application of Artificial Neural Networks (ANN) [452,453]. ANNs rely on a computationally intensive framework characterized by extensive use of multiply and accumulate operations, and well-established learning algorithms [454,455]. In contrast, their more biologically-inspired counterparts, Spiking Neural Networks (SNN), have not yet achieved comparable levels of performance and scalability due to the complex dynamics of neurons and early challenges in applying learning algorithms [456,457]. Despite these challenges, SNNs offer a unique hardware advantage due to their reliance on binary spike events over time, facilitating multiplication-free sparse computing architectures. This potential has been illustrated in recent hardware accelerators that demonstrate online-learning capabilities and high accuracy across various benchmarks [170,458].

Most of these accelerators utilize SNNs with Integrate-and-Fire (IF) neurons, employing rate-coding to encode information via spike counts, leading to many-to-many network architectures. However, the intrinsic temporal aspect of spikes opens avenues for leveraging both spatial and temporal dimensions. Notably, Time-To-First-Spike (TTFS) coding, which enforces the generation of a single spike per neuron at the earliest conceivable moment in time, offers a pathway to architectures that are both faster and more sparse than those based on spike-count coding. Recent learning advancements using TTFS have demonstrated competitive performance in both spatial and temporal tasks with significantly reduced number of spike operations [174,175]. Despite the proven advantages of TTFS, implementations of TTFS-based SNNs in hardware accelerators remain relatively unexplored.

In this Section, I introduce a FPGA-based hardware architecture for a Binarized Spiking Neural Network (BSNN) with TTFS on MNIST [449,459]. This approach hinges around leveraging synchronous groups of neurons, resulting in a 24.9% reduction of network size. Subsequent analysis of excitatory input channels further decreases the number of input weights by 33.7%. Through the implementation of this reduced network, I propose a streamlined hardware architecture featuring four parallel processing engines that process as single input event at a time, dubbed NimbleSNN. This architecture incorporates neuron scheduling, decreasing the number of Logic Elements (LE) by 45.8% due to smaller spike buffers. Implementation of this architecture on the Lattice ICE40UP5K FPGA (Lattice Semiconductor Inc., Hillsboro OR) demonstrates a measured power consumption of 20.4 mW and an inference latency of 65.8 μ s, utilizing 4639 LEs. With an energy consumption of 1.34 μ J per image, our implementation surpasses spike-count based SNN architectures in performance and aligns closely with a prior TTFS-coding implementation on FPGA. Furthermore, an inspection of the effective efficiency at 47.7 GSops/s/W, shows NimbleSNN to possibly be the most efficient SNN-based MNIST accelerator on FPGA at the time of writing. Next to a comparison on FPGA, prior works on Application-Specific Integrated Circuits (ASIC) describe energy per inference ranging from 0.24 μ J to 26.9 which is competitive to the results described in this Section. Finally, at an operation efficiency of 21.0 pJ/SOp, this work shows promising advancement in the use of synchronization to optimize digital SNN hardware

architectures used for accelerators. The entire code base used for analyses and the FPGA implementation have been made available [460].

The selection of the Lattice ICE40UP5K FPGA is motivated by its constrained resources, including a small number of logic elements (5280 LUTs + PFU registers) and limited memory capacity comprising 4 Single Port Random Access Memory (SPRAM) blocks with 256 kb each and 30 Embedded Block RAMs (EBRs) of 4 kb each. This environment underscores the challenge of accelerator design with severe resource limitations, thereby providing a valuable precursor for the development of Application Specific Integrated Circuits (ASICs). Additionally, its affordability (typically less than \$10 USD) showcases the viability of this FPGA as a cost-effective solution for programmable hardware, potentially facilitating its incorporation into commercial products. While the Lattice ICE40UP5K FPGA is chosen for this study, other resource-constrained FPGAs such as the Gowin LittleBee and Effinix Trion FPGAs are also suitable candidates.

8.2.2 A Reduced Binary SNN with TTFS coding

As also described in the prior section, in Time-To-First-Spike (TTFS) coding, conversion of input data to events is defined as

$$t_{\text{in}} = \left\lfloor \frac{a_{\text{max}} - a_{\text{in}}}{a_{\text{max}}} \right\rfloor t_{\text{max}} \quad (8.41)$$

where t_{in} is the encoded input event time, and a_{in} , a_{max} , and t_{max} are the input value, maximum input value (255 with MNIST images) and maximum event time respectively. The braces, " $\lfloor \cdot \rfloor$ ", indicate rounding due to quantization. Here, I employ Integrate-and-Fire (IF) neurons with TTFS, and the membrane potential of these neurons is represented by

$$U_j^l(t) = \sum_i w_{ij}^l \sum_{t_i \in T_l} S_{ij}^{l-1}(t) \quad (8.42)$$

where $U_j^l(t)$ is the membrane potential of neuron j in layer l , w_{ij}^l is the synaptic weight of synapse ij , connecting neuron i of layer $l-1$, and neuron j of layer l . The sum of $S_{ij}^{l-1}(t)$ represents the summation of pre-synaptic spikes t_i at time t , contained within T_l . Accordingly, I can describe the output spike of neuron j in layer l as

$$S_j^l(t) = \begin{cases} 1 & \text{if } U_j^l(t) \geq \theta_j \text{ \& } S_j^l(<t) \neq 1 \\ 0 & \text{otherwise} \end{cases} \quad (8.43)$$

where $S_j^l(t)$ is the output spike train of neuron j in layer l , equating to unity at time t if and only if the the potential exceeds θ_j and the neuron has not spiked before. The aforementioned points to two pivotal advantages of TTFS encoding:

- Neuronal activity is extremely sparse as a result of the generation of a single spike per neuron during an inference
- In SNNs of IFs with TTFS coding, inference completes whenever an output neuron spikes.

Due to spike sparsity during computation and completion of inference and subsequent shutdown of the network at the earliest output neuron spike in time, TTFS coding inherently contains a significant computational benefit in terms of latency and efficiency.

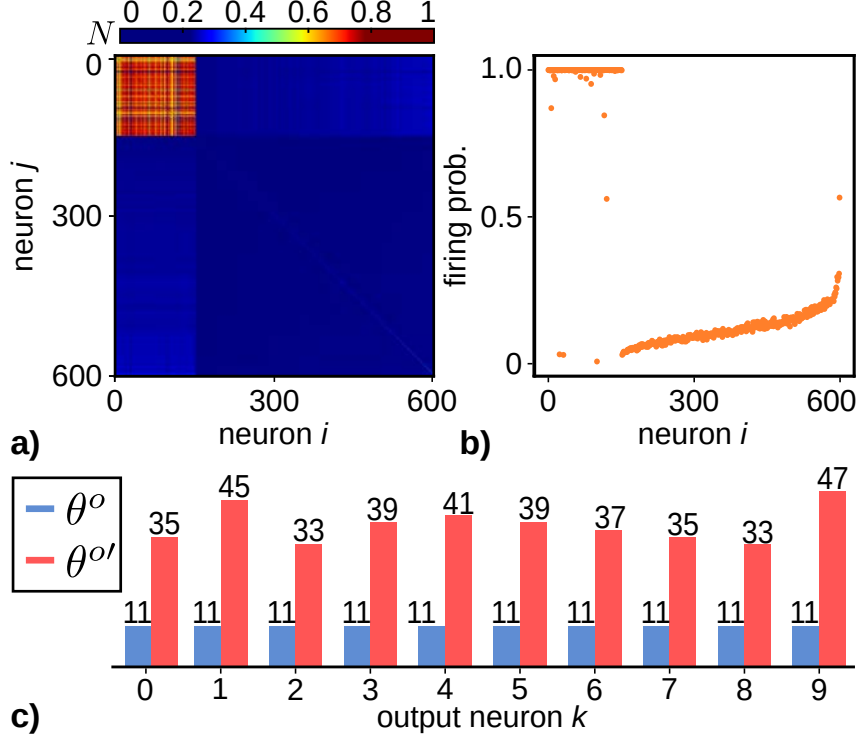


Figure 8.1: a) Sorted synchronization matrix of the hidden layer neurons, diagonal omitted and normalized over number of inferences. b) Probability of neurons firing throughout all inferences. c) Change in output neuron threshold by adding the constant effect of synchronous neurons.

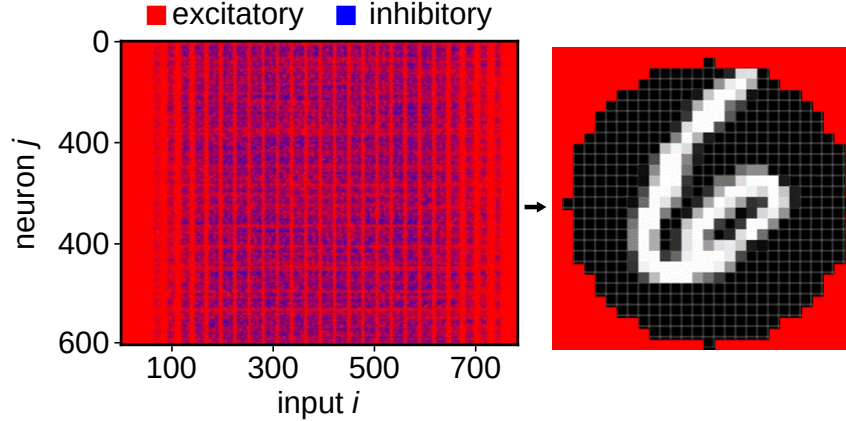


Figure 8.2: Color coded weight matrix of the synapses between input and hidden layer (left). Resulting mask on example image determined from inputs with $>90\%$ excitatory weights assigned across the hidden layer (right).

Reducing Network Size with Synchronization

In this Section, I utilize a Binary Spiking Neural Network (BSNN) that has a 784-600-10 fully-connected structure using the equivalent TTFS coding as described above, trained on MNIST [449] and apply the following analysis. Fig. 8.1a shows a synchronization matrix for 600 neurons within the hidden layer of this BSNN, constructed by assessing the proportion of concurrent spiking events; value of $N_{ij} = 1$ indicates concurrent spiking across all inferences between neurons i and j across the

MNIST dataset. This matrix unveils a distinct cluster of 152 neurons characterized by high proportion of simultaneous firing, showing a resemblance to neural circuits in the brain where synchronous excitatory neuron groups are present and essential for learning [461, 462]. Further examination of the firing probabilities of the neurons in the BSNN, displayed in Fig. 8.1b, reveals that the probability of these neurons to fire approaches unity. Such behavior suggests that these neurons have a near-constant influence on the membrane potential of the output neurons and are essential during the training process. By readjusting the threshold θ_j^o of the output neurons to absorb the effect of these synchronous neurons, I streamline the network without compromising performance for inference. The adjusted threshold $\theta_j^{o'}$ is computed as follows:

$$\theta_j^{o'} = \theta_j^o + \sum_{i \in N_{\text{syn}}} w_{ij}^o \quad (8.44)$$

where θ_j^o and $\theta_j^{o'}$ represent the original and updated thresholds of the output neurons, respectively, and w_{ij}^o denotes the synaptic weight between each synchronous neuron i in the set of synchronous neurons, N_{syn} , and output neuron j . Incorporation of the synchronous neuron group activity into the output neuron thresholds (Fig. 8.1c) facilitates a 24.9% reduction in total weight count. Further analysis could lead to insights in how the training process and dataset affect these synchronous neuron groups, but this is considered out of scope.

Excitatory / Late Input Channels

An overview of the neuron weights is shown in Fig. 8.2, categorizing synapses as either excitatory (red, weight of +1) or inhibitory (blue, weight of -1) for each hidden layer neuron j with respect to inputs i . This showcases a prevalence of excitatory synapses along the image edges. This observation is leveraged to implement an input mask that uniformly applies a constant excitatory stimulus to the hidden layer neurons for all TTFS-encoded pixels located at the image edges. Detailed in Section 8.2.3, this mechanism is constructed with a direct one-to-one mapping in memory. Analysis indicates that in merely 4.9% of inferences input spikes originate from these masked regions prior to the completion of an inference. Furthermore, the implementation of excitatory input masking contributes to an additional 33.7% weight reduction.

8.2.3 Hardware Implementation and Architecture

Architecture overview

Fig. 8.3a shows a block diagram of the proposed SNN architecture for FPGA implementation, that I dub 'NimbleSNN' due to the focus on speed by scaling parallelized spike processors and constructing memory structure that fit within prescribed tailored hardware constraints. This architecture comprises of two spike routers, an Input Event Router (I-ER) for forwarding input event to hidden neurons, processed by Hidden Engines (HE), and a Hidden Spike Router (H-SR) to route spikes from the HEs to the output neurons that are processed by an Output Engine (OE). Next to these elements, an Input Mask implemented with an EBR maps the input events from the corners of the image to inject voltage to all neurons in the HEs or an adjusted event address. This mapping together with the HEs, the I-ER and H-SR

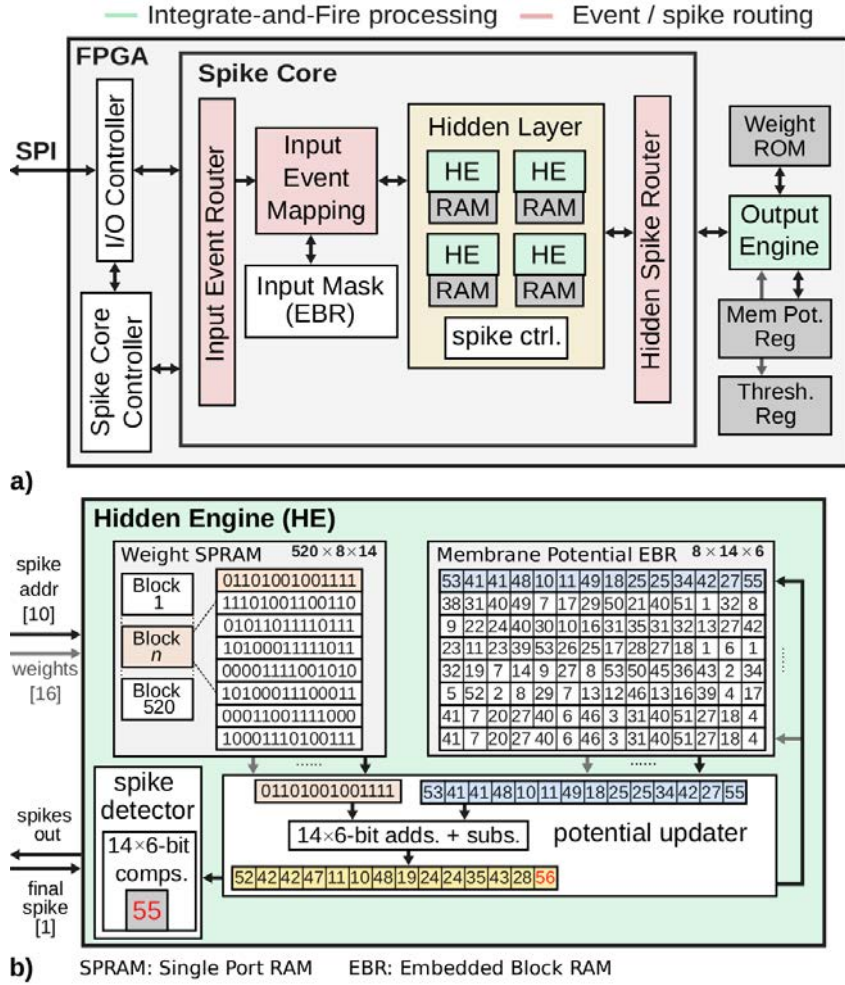


Figure 8.3: a) Proposed hardware architecture diagram. b) Diagram of one of four Hidden Engines (HE) that process 448 hidden neurons of the reduced binary SNN, red indicating threshold and a neuron potential exceeding this value (56), leading to a spike.

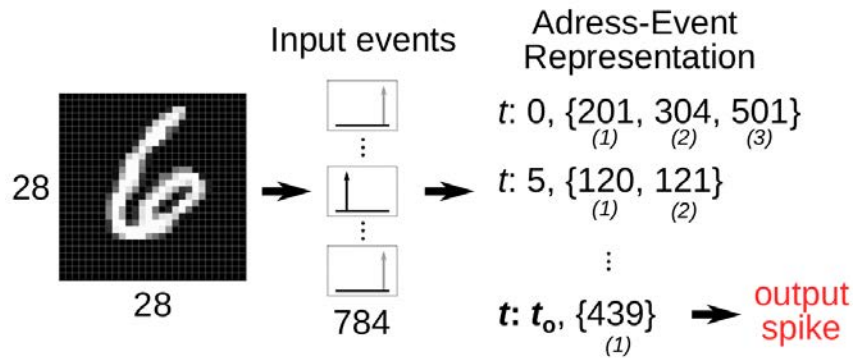


Figure 8.4: Example diagram describing conversion of MNIST pixel intensity to TTFs events using Address Event Representation (AER). Time t_o represents the final timestep and integers below address event sets indicate a one-by-one input mechanism.

construct a spike core that is a fully reusable block employable for multi-layer networks and adjustable with minor edits to support recurrent connections. Moreover, multi-spike event and spike encoding is also compatible with minor edits. However, the neurons implemented in the HE support only IF neurons without any leakage within this Section and the available code.

Pixels from MNIST images are converted to events at time t_{in} in the range of $t \in [0 \ 255]$ according to eq. 8.41. Instead of sending discrete events at time t_{in} , the events are encoded as addresses in the style of Address Event Representation (AER), starting from time $t = 0$ until an output neurons spikes at $t = t_o$ (Fig. 8.4). Each address encodes the location of pixel n in the flattened 784 pixel image (28 by 28). One by one, starting from $t = 0$, these addresses are routed to an input mask that injects potential directly into all hidden neurons if the event address originates from the image edges (see Fig. 8.2) using Embedded Block RAMs (EBR). Event addresses originating from inside the edges are forwarded to all HEs, initiating processing of 112 hidden neurons per HE. In other words, all HEs receive one event address per instance in addition to a control signal that indicates if the event represents the final event of a single timestep. Per event address input, the neurons processed by the HEs accumulate potential. In the case that the final input event of a timestep is incident, the HEs check whether a neuron membrane potential exceeds the threshold (eq. 8.43) in addition to accumulation and store output spikes in a spike buffer. These spikes are then routed to the OE using the hidden spike router.

The OE concurrently processes ten output neurons corresponding to the handwritten digits, each neuron containing a distinct membrane potential threshold, stored in registers (see Fig. 8.1c). This OE introduces one of the inherent features of TTFS coding; the earliest output neuron that exceeds its potential threshold and spikes at time $t = t_o$ determines class. As a result, at time $t = t_o$, the OE resets the entire Spike Core and transmits the spiking output neuron address to the peripheral, initiating the subsequent inference. On average, NimbleSNN requires only 11.5% of events converted from MNIST pixels until an inference is completed due to an output neuron spike. Therefore, the implementation of this functionality leads to a significant advantage in inference latency.

Hidden Engine (HE)

Fig. 8.3b shows the memory structure and processing flow within a single HE following the reception of event address n . A Single Port Random Access Memory (SPRAM) is employed for the storage of weights, structured using input addressing so that each distinct input event or spike is allocated a block of eight rows. Among these rows, one row retains 14 neuron weights assigned to that particular input, resulting in a total of 112 neuron weights per input. Embedded Block RAMs (EBR) store the membrane potential of hidden neurons, mirroring the structure with 6 bits per neuron. This configuration enables rapid processing of input spikes, necessitating only 2 cycles to process 14 neurons per HE and totaling 18 cycles per input address, including 2 cycles for block address transition. Consequently, all 4 HEs together have a peak processing speed of 28 neurons per cycle. The HE continue to process the neurons and checks if and only if the final event within a timestep is incident, the membrane potential of neurons. If this membrane potential exceeds the threshold, a spike is forwarded to the H-SR and subsequently the OE.

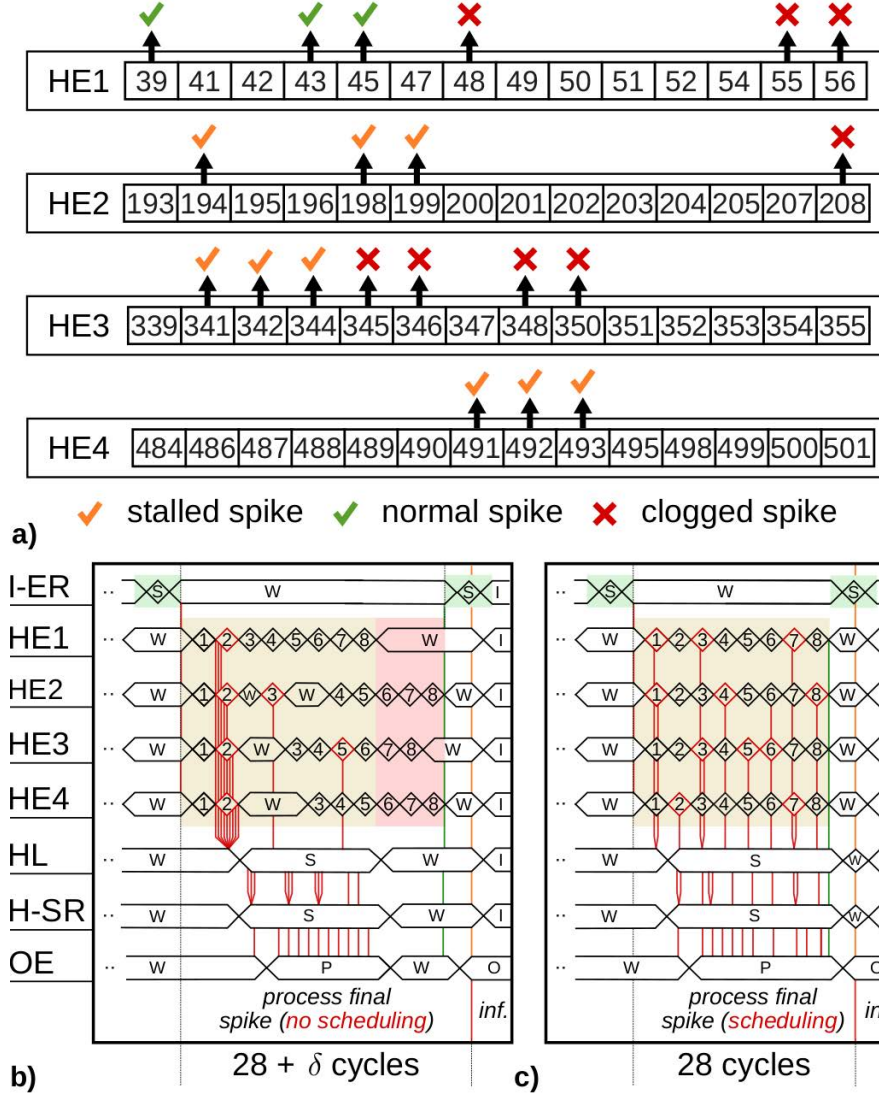


Figure 8.5: a) Example of a worst case scenario during inference; each hidden engine produces more than 3 spikes, leading to clogging (lost spikes) and stalling (increase in latency). b,c) Example time diagram of the proposed architecture without neuron scheduling (left) and with neuron scheduling (right). Red and green vertical lines indicate top-to-bottom direction transmission of spikes and hidden layer processing completion respectively. The orange vertical line represents a global signal that interrupts the entire spike core whenever an output neuron spikes. Abbreviations, I-ER: Input Event Router, HE: Hidden Engine, HL: Hidden Layer, H-SR: Hidden Spike Router, OE: Output Engine.

HE Neuron Scheduling and Spike Buffering

Spike routing poses a significant hardware design challenge, as recognized in existing literature on neuromorphic hardware scaling [154]. For this architecture, the worst spike routing scenario unfolds when all neurons assigned to a single row of an HE spike simultaneously, reaching a total of 14 spikes. If this scenario occurs across all four HEs, the spike count surges to 56 within a single cycle. This necessitates a substantial amount of logic for buffering and parallel processing, a demand unsuitable

Table 8.1: Resource utilization of our proposed NimbleSNN on the ICE40UP5K FPGA.

	LUT	Registers	EBR	SPRAM
Core Controller	314 (5.9%)	0	0	0
I/O Controller	137 (2.6%)	97 (1.8%)	0	0
Spike Core	3717 (70.4%)	374 (7.1%)	29 (96.7%)	4 (100%)
Hidden Layer	2830 (53.6%)	151 (2.9%)	24 (80%)	4 (100%)
<i>Hidden Engine</i>	673 (12.7%)	29 (0.5%)	6 (20.0%)	1 (25%)
Hidden Router	200 (3.8%)	57 (1.1%)	0	0
Input Router	277 (5.2%)	81 (1.5%)	3 (10.0 %)	0
Output Engine	408 (7.7%)	85 (1.6%)	2 (6.7 %)	0
Total	4168 (78.9%)	471 (8.9%)	29 (96%)	4 (100%)
Available	5280	5280	30	4

SPRAM: Single Port Random Access Memory (RAM)

EBR: Embedded Block RAM, LUT: Look Up Table

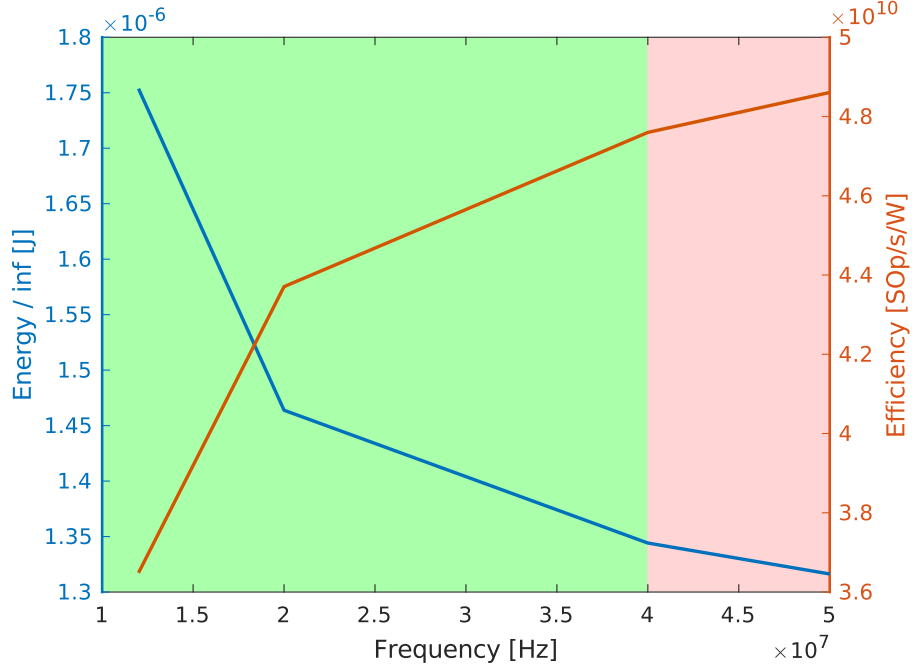


Figure 8.6: Energy per inference and efficiency as as a function of frequency, the green and red areas represent valid SNN outputs and invalid due to timing errors, respectively.

for resource-constrained devices such as the FPGA selected for our implementation. I identify a scenario where a single HE generates more spikes than its buffer can accommodate as 'clogging', and a situation where multiple HEs spike beyond the Hidden Layer (HL) buffer's capacity as 'stalling'. In instances of clogging, any exceeding spike is forfeited, whereas in stalling, HEs are paused until sufficient space is available in the HL buffer.

Fig. 8.5a depicts a worst-case scenario during inference, assuming a buffer capacity of a maximum of 3 spikes. Each Hidden Engine produces at least 3 spikes within a single cycle, causing HE2 to stall until HE1's spikes are processed, followed by HE3, and so on. Moreover, HE1, HE2, and HE3 each generate more than three spikes within one cycle, leading to spike loss due to clogging. To mitigate clogging and stalling, I introduce a neuron scheduling algorithm aiming for an ideal scenario where neuron spikes are temporally separated to avoid simultaneous spiking. This algorithm randomly assigns neurons to one of the 4 HEs, followed by a random exchange of two neurons within the memory based on a cost function representing the average number of cycles per inference that the design is stalled. A random swapping algorithm can be considered as a greedy search and therefore not the optimal algorithm in terms of computation time for finding the best assigned memory structure. However, since the search space is relatively unknown and random search requires the least assumptions, it is considered for the proof of concept in this work. Fig. 8.7 shows three iterations, namely, iteration 1, 10 and 30 of this algorithm. If the random swap of two neurons results in a lower cost, the memory is updated with the swapped neuron locations (green and subsequently yellow colors in the Figure). If not, no swap is performed (red color). After 10,000 iterations of this process, the algorithm achieves a memory configuration that reduces cycles required due to stalling from 10.1 cycles to 5.9 cycles per inference. Additionally, determining a buffer size of 3 for each HE and the HL proves sufficient, with minimal accuracy loss (within 0.1%) due to clogging compared to a 14-spike buffer, decreasing the required logic elements by 45.81%. Fig. 8.5b and c illustrate an example of improvement in processing a single spike from no scheduling to effective scheduling, eliminating stalling.

8.2.4 Experimental Results

The proposed architecture, NimbleSNN, is deployed on the Lattice ICE40UP5K FPGA, with the design coded in SystemVerilog. I used the Lattice Synthesis Engine

Table 8.2: Scaling factors k of technology taken from [463] with respect to 40 nm for ICE40UP5k. The respective technology has to be multiplied by k to reach 40 nm technology.

Technology [nm]	65	28	22	14	10
Area	0.368	2.14	3.41	9.38	25
Delay	0.76	1.13	1.23	1.27	1.33
Energy	0.55	1.49	1.83	2.90	3.67
Power	0.73	1.30	1.52	2.28	2.8

Table 8.3: Comparison with State Of the Art SNN FPGA implementations on MNIST

Reference	This Work 2024	[464] 2024	[465] 2024	[466] 2023	[467] 2023	[468] 2022	[469] 2021	[470] 2021
Platform	Lattice ICE40UP5K 40nm	Xilinx VC7V2000T 28nm	Xilinx XC7Z020 28nm	Xilinx XCZU3EG 16nm	Xilinx KCU115 20nm	Xilinx XC7K325T 28nm	Xilinx XC7Z100 28nm	Xilinx VC7VX485T 28nm
# of Neurons	458	266 ³	138	266 ³	1482	1010	10 ³	512
Area ¹ [mm ²]	5.48	2025	289	152	1600	729	961	1225
Logic elements	4,639	133,802	7,612	27,551	109,000	76,788	156,470	16,324
Clock [MHz]	40	200	100	100	160	200	200	50
Bitwidth	1	8	4	8	8	16	N/R ²	8
Accuracy [%]	95.9	99.1	93.9	99.0	99.4	95.6	99.4	96
Power [mW]	20.4 ⁴	440 ⁵	180 ⁵	280 ⁵	1576 ⁵	535 ⁵	1400 ⁵	400 ⁵
Latency [μ s]	65.8	1956	780	270	N/R ²	2865	26.0	2.8
Energy/ inf [μ J]	1.34	861	140	76.0	N/R ²	1532	36.4	1.13
Effective efficiency* [GSOp/s/W]	47.7	34.7	N/R ²	N/R ²	41.7	N/R ²	28.6	0.340
Norm. [†] Energy/ inf [μ J]	1.34	1283	209	220	N/R ²	2283	54.2	1.68

¹I assume the smallest package size available ²N/R: not reported

³Fully-connected neurons ⁴Measured ⁵Estimated

* Total number of SOps divided by a product of the total latency and power

[†] According to scaling factor retrieved from Ref. [463], Table 8.2

Table 8.4: Comparison with State Of the Art SNN ASICs on MNIST.

Reference	This Work	[471]	[458]	[472]	[473]	[474]	[475]
	2024	2023	2023	2022	2022	2021	2019
Platform	Lattice ICE40UP5K 40nm	TSMC 65nm	N/R ² 65nm	TSMC 28nm	TSMC 28nm	TSMC 65nm	Intel 10nm
# of Neurons	458	N/R ²	1034	384	N/R ²	410	4000
Area ¹ [mm ²]	5.4	10.65	18	1.0	0.89	10.1	1.72
Clock [MHz]	40	100	250	333	500	20	506
Bitwidth	1	8	16	9	8	8	7
Accuracy [%]	95.9	96.1	98.8	93	97.8	97.8	97.9
Power [mW]	20.4	61	182	139	149	23.6	208
Latency [μ s]	65.8	441	2.9	3.6	2.9	10	12.7
Throughput [kFPS]	15.2	2.27	345	278	345	100	78.7
Energy/inf [μ J]	1.34	26.9	0.52	0.50	0.43	0.24	1.7
Norm. [†] Energy/inf [μ J]	1.34	14.80	0.29	0.80	0.68	0.13	6.2
Area efficiency [kFPS/mm ²]	2.81	0.21	19.2	278	388	9.90	45.8
Operation efficiency* [pJ/SOp]	21.0	3.04	N/R ²	N/R ²	N/R ²	N/R ²	8.3

¹I assume the smallest package size available for the FPGA ²N/R: not reported[#]A product of energy per inference and area[†] According to scaling factor retrieved from Ref. [463], Table 8.2

for synthesis and Modelsim for verification. The validation of this implementation utilizes the established setup for conducting SPI communications and power measurements, described in Section C.2.1. Furthermore, the source code for this design is publicly accessible [460].

Table 8.1 describes the resource utilization on the Lattice ICE40UP5K FPGA, highlighting the consumption of LUTs, Programmable Functional Unit (PFU) registers, EBRs, and SPRAMs in realizing the proposed architecture. This implementation leverages 78.9% of the available LUTs and 8.9% of PFU registers, alongside nearly the entire memory resources provided by the device. A significant portion of the logic resources (53.6% of available LUTs) is dedicated to the hidden layer, where each HE employs 673 LUTs and a single SPRAM for storing neuron weights. The EBRs associated with HEs and the output engine are tasked with storing the membrane potential for each neuron. The hidden router utilizes only 3.8% of the deployed LUTs due to the buffer size reduction enabled by the proposed neuron scheduling. Additionally, the input router utilizes 3 EBRs to manage the input mask, facilitating the mapping of inputs to neuron potential injections.

Table 8.3 compares contemporary FPGA implementations of SNNs. The referenced works [465, 466, 476] involve rate-based SNNs, while Ref. [470] uses TTFS coding. Operating at a clock frequency of 40 MHz, our architecture, implementing 458 neurons, achieves a power consumption of 20.4 mW, setting a new benchmark for low power consumption among FPGA-based SNN implementations, with others reporting power usage ranging between 180 mW to 1576 mW. Our design’s logic element count of 4,639 (LUTs + PFU register) is far lower than others and comparable to the 7,612 LEs employed by Ref. [465].

The inference latency presents a large difference between rate-based and TTFS-based models. Our TTFS-based SNN exhibits an inference latency of 65.8 μ s, marking a reduction of 91.6%-97.9% compared to rate-based models, which exhibit latencies between 780 to 3150 μ s. Although the TTFS model on the Xilinx Virtex 7 FPGA [470] demonstrates a significantly lower latency of 2.8 μ s, it comes at the expense of higher resource consumption and power usage, namely, 3.52 times more LEs and 19.6 times greater power consumption, respectively. Despite the advanced capabilities of their FPGA, our model achieves comparable efficiency, with an energy consumption per inference of 1.34 μ J, closely approaching their 1.13 μ J. Table 8.2 shows the scale factor in area, delay, energy and power. These are retrieved from Ref. [463], and I utilize the scale factors closest to the indicated technology to provide a normalized energy efficiency for the comparison provided in this subsection. In other words, these scaling factors are utilized to provide a normalized performance metrics in the respective Tables with respect to various technology nodes.

An analysis of the effective efficiency is necessary in order to compare works in terms of scalability. This performance metric, expressed as the number of spike operations per second per watt (SOps/s/W), is calculated by taking the total number of SOps during inference and dividing it by the total latency and power. NimbleSNN shows an effective efficiency of 47.7 GSOp/s/W, outperforming all other works described in Table 8.3. This metric is a good indication of scalability. For example, an increase in the number of HEs necessitate more logic, leading to a higher power consumption that scales linearly at this size, which would then lead to more spike operations per second. In other words, the energy per inference could be scaled down as this effective efficiency is significantly high. This metric clearly points to the scalability of NimbleSNN to larger networks, indicating the advantages of using TTFS,

binary weights and synchronization for optimization.

Fig. 8.6 shows the energy per inference and effective efficiency as described above as a function of clock frequency. Here, the green area represents valid operation and the red area represents the frequency at which timing errors are presents. For a higher frequency, a higher efficiency is observed leading to the selection of the maximum frequency where valid operation is possible, namely, 40 MHz clock frequency. This is the conventional method to select frequency in neural network accelerators.

Next to FPGAs, Table 8.4 describes a comparison of NimbleSNN with existing works that describe accelerators of MNIST using SNNs on customized ASICs. This comparison comes with additional performance metrics, i.e., the area efficiency described by the throughput per square millimeter, kFPS/mm², and operation efficiency defined by the energy per spike operation as pJ/SOp. In terms of the energy per inference, NimbleSNN performs competitively with existing works on ASIC, namely consuming only 8.93 to $2.19 \times$ more, respectively. This is normal considering that at the time of writing FPGAs typically have an efficiency of a magnitude lower than that of ASICs, showing that this implementation, if transformed into an ASIC, could have comparable performance to the existing literature. This analysis holds similarly when considering the area efficiency and operation efficiency at 2.81 kFPS/mm² and 21.0 pJ/SOp, comparable within a single magnitude of order with existing ASICs.

In terms of accuracy, Refs. [464, 466, 467, 469] show performances on MNIST higher or equal to 99.0%. A deeper inspection of the techniques of these works reveals the use of Spiking Convolutional Neural Networks (SCNN) for Refs. [464, 466, 469]. On the other hand, Ref. [467] employs a 5-layer fully-connected network with 1482 neurons, a network three times the size of the network described in this Chapter. These proposed SCNNs hardware architectures use networks of LIF neurons that are created with one of two identified approaches for solving a range of tasks, namely, native training of SNNs with a surrogate gradient, and conversion of a trained CNN to SCNN [457]. Here, Refs. [464, 466] employs training with a Spatio-Temporal Back-propagation (STBP) algorithm that decomposes the chain rule of gradient descent with SNNs into two separate components, i.e., a spatial component involving output spikes and a temporal component involving the membrane potential of each neuron over time [477]. Moreover, to train large multi-layer SCNNs using this approach, a batch normalization based on the membrane potential threshold of each layer of spiking neurons is utilized that solves the vanishing-exploding gradient problem [478]. On the other hand, Ref. [469], converts a trained CNNs such as VGG-16 into an SCNN by employing a proposed algorithm that scales weights and biases and implements a max-pooling operation by allowing only spikes from neurons that spike maximally [479]. These approaches show accuracies comparable to ANNs at higher or equal effective efficiency on various benchmarks, but, will never be able to overcome conventional ANNs. A future study that investigates synchronization within these networks could be a promising avenue to explore.

8.2.5 Conclusion

This Chapter presents a first proof-of-concept for integrating synchronization in an SNN, tailored for hardware implementation on a highly resource-constrained FPGA. It demonstrates the effectiveness of TTFS coding in reducing inference latency in SNNs, outperforming traditional rate-based encoding under strict resource limitations. The competitive performance of this approach, relative to existing FPGA-

based SNN implementations and ASICs, underscores its potential for broader application, particularly in multi-spike SNNs where challenges such as clogging and stalling are intensified. Future research could extend this work to explore TTFS-based SNNs with complex architectures, including recurrent connections or convolutional layers [174], necessitating advanced techniques for training with weight binarization or ternarization [480]. According to my knowledge, this is the first time that synchronization is employed to increase spatial sparsity within an SNN hardware architecture as an approach. Effectively, this Chapter proposes the employment dynamics within an single-spike SNN to optimize hardware implementations, an additional unconventional approach to codesigning neural networks on hardware.

Chapter 9

Conclusion, Discussion and Future Work

In the Introduction, this Thesis identified the advancement of the following objective to be essential,

Objective: *The advancement of Edge-AI in terms of efficiency, capability and accessibility*

Accordingly, two fundamental academic issues were identified for pursuing this objective,

Issue 1: *The majority of Edge-AI hardware focuses on customized architectures for acceleration, promising speedup and efficiency over GPUs that are incompatible with resource-constrained sensor nodes. (Fig. 1.1)*

Issue 2: *Although SNNs recently overcome ANNs in efficiency, they fail to consider and employ the complex dynamics and phenomena that emerge in networks of spiking neurons.*

Using these two issues, three questions were formulated that are the central research questions of this Thesis, namely,

Question 1: *How can efficient Edge-AI hardware architectures be created for extremely resource-constrained sensor nodes?*

Question 2: *How can emergent physical phenomena, namely, chaos and synchronization, be utilized in spiking networks for computation?*

Question 3: *How can synchronization be used to create highly-efficient digital SNN hardware architectures?*

To answer these questions, this Thesis introduces two neural network codesign approaches. The first, described in Part I, starts with a focus on an actual application in Precision Livestock Farming (PLF), cow behavior estimation, and builds up towards algorithms with a DT and quantization of RNN leading to hardware

implementations on FPGA. The second, described in Part II, focuses on dynamics within hardware, starting with an experimental study of synchronization phenomena in a network of spiking oscillators before moving towards utilization of synchronization within SNN on a neuromorphic processor for epileptic seizure encoding and optimization of an SNN accelerator using synchronization on FPGA.

The following and final parts of this Thesis are organized as follows. Section 9.1 will describe each Chapter’s key contribution within a single sentence followed by a discussion on how the various works of this Thesis have partially answered the above stated questions using a neural network codesign approach and subsequent claims in Section 9.2. Finally, Section 9.3 will introduce some of the caveats of each Chapter, leading to a pair of potential future works using the research paradigm of neural network codesign that has been proposed.

9.1 Chapters at a glance

Chapter 2 A study on the effects of livestock farming on the global environment, Precision Livestock Farming (PLF) as a mitigation strategy, and a literature review of existing Edge-AI in PLF.

Chapter 3 [C6] A 216 μ W, 87% accurate cow behavior classifying decision tree on FPGA using interpolated arctan2 as a feature.

Chapter 4 [J5] An integer-only, 11.74 kB, 95% accurate cow behavior distribution regressing RNN, dubbed "TinyCowNet", discovered by means of random search.

Chapter 5 [J3] An integer-only resource-minimized RNN hardware architecture using time-and-layer multiplexing and a customized processing engine for low-frequency sensors in Edge-AI. (0.98 mW, 106 μ J / inf. @TinyCowNet)

Chapter 6 [J4] An experimental study of synchronization phenomena in a ring network of spiking chaotic oscillators for physical reservoir computing.

Chapter 7 [J1] An SNN on a mixed-signal neuromorphic processor, DYNAP-SE2, that encodes seizures using partial synchronization.

Chapter 8 [J2] A 1.34 μ J/Image 4639 logic element single-spike accelerator, dubbed "NimbleSNN", implemented on the Lattice ICE40UP5K FPGA that is optimized using synchronization.

9.2 Claims

Question 1 is answered by means of an approach that starts from Precision Livestock Farming (PLF) and builds up towards introducing low-power efficient Edge-AI hardware architectures that can be applied to any type of application utilizing sensors in Part I. With the consideration of an actual application, namely, cow behavior classification using 3D accelerometers, this part first introduces a classical machine learning algorithm, a Decision Tree, that shows extremely low-power consumption of 216 μ W and an energy per inference of 557 μ J when implemented on the Lattice ICE40UP5K FPGA. Albeit at an accuracy of 87% on four classified cow behaviors, i.e., resting, grazing, rumination and moving, already the work described in this Chapter shows the most efficient implementation of animal behavior monitoring. This is followed up by Chapter 4 that introduces a neural architecture search and

full-integer quantization for minimizing Recurrent Neural Networks (RNN) on this specific application. This leads to "TinyCowNet", a 11.74 kB, integer-only simple RNN (eq. 4.9), that shows an accuracy of 95.1%. Finally, Chapter 5 introduces a resource-minimized RNN hardware architecture that employs time-and-layer multiplexing. This approach in combination with a custom processing engine that supports integer-only multiply-accumulate operations, multiplexes the required operations of the RNN system equation during processing on hardware, leading to minimal use of resources in terms of registers and logic. Implemented on the Lattice ICE40UP5K, this hardware architecture enables extremely low-power and efficient RNN processing that can be used by any Edge-AI application using low-frequency sensors. Further validated on TinyCowNet with a power consumption of 360 μ W at 146 kHz to 2.3 mW at 12 MHz and an energy per inference at 516 and 44.3 μ J, respectively, the following claim can be made with respect to the first question, namely

Claim 1: *Application- and platform-aware codesign of Machine Learning algorithms and hardware architectures lead to efficient Edge-AI implementations that are suitable for extremely resource-constrained sensor nodes (efficiency and accessibility).*

The above claim exemplifies the importance of selecting the proper platform according to application-specific constraints and then taking this platform as a baseline for designing efficient Edge-AI hardware architectures. The approach of Part I is unique in the way that it considers deeply the benefits and constraints of Precision Livestock Farming at the edge, culminating in an RNN hardware architecture that extends the field of Edge-AI for extremely resource-constrained environments. In addition, the use of a resource-constrained FPGA before moving on to the design of ASIC might prove to be an important methodology for designing low-power and resource-efficient hardware architectures for any application. Therefore, one can say that Part I starts from considering an application, moves towards algorithms and at the end ends up at hardware in terms of codesign space.

Question 2 and 3 are partially answered in Part II. This part begins with an extensive study on the synchronization phenomena of a ring network of chaotic spiking oscillators on a Printed Circuit Board (PCB) in Chapter 6. This network is subjected to various experimental conditions, namely, sweeping of coupling and supply voltage, injection of coherent noise and heterogeneity across network nodes. This study leads to the proposal of utilizing the emergent phenomena of synchronization and chaotic spiking behavior for computation within physical reservoirs. This initial study serves as an inspiration and foundation for the subsequent Chapters, Chapter 7 and Chapter 8.

Chapter 7 builds on this foundation by proposing the use of partial synchronization to encode epileptic seizures on a mixed-signal neuromorphic processor, DYNAP-SE2. This processor implements Exp-LIF neurons using MOSFETs in subthreshold region that operate in real-time combined with an analog front end that implements an asynchronous delta modulator. Within this Chapter, a full proof-of-concept system is outlined that employs the analog front end and the neurons on-chip to implement an SNN that encodes seizures from raw iEEG data. This asynchronous delta modulator on chip converts analog iEEG data from actual patients of the SWEC-ETH dataset [433] into events that are forwarded to the SNN. This SNN is constructed by two layers; a layer that functions as a channel-by-channel non-

linear filter attached to the subsequent layer with a specific connectivity structure, named the non-local non-global (NLNG) layer. This SNN and analog front end on chip is employed on actual iEEG snippets of roughly 320 seconds per patient with time-varying lengths of seizures. During this operation, the SNN shows no signs of synchronization between neurons during periods where normal brain activity is recorded, i.e., before and after the epileptic seizures. On the contrary, during seizure onset, partial synchronous neuron clusters start to form within the NLNG layer that strengthen across the entire layer and diminish over time until the seizure ends. In essence, this shows a first proof of concept of using the underlying network phenomena to mimic the complex dynamics of another system using a neuromorphic processor. As a result, the following claim can be made

Claim 2: *The rich dynamics within mixed-signal neuromorphic processors can be employed to encode anomalies arising from the complex network dynamics of another system such as the human brain undergoing a state of seizure (capability).*

The work described in Chapter 7 shows a proof of concept of the above claim. However, it is not a sufficient study to fulfill the conditions necessary for the claim to hold fully true. More work has to be performed to show that this is the case across multiple applications and cases. Furthermore, the proposed SNN is a first iteration, namely a regular graph that shows it is able to encode long-term correlation structures defines as partially synchronous clusters. However, more underlying experimental studies with multiple types of graphs on-chip, drawing similarity to the study in Chapter 6, have to be performed to provide substantial evidence for the above claim.

Finally, Question 3 is partially answered within the study described in Chapter 8. Here a binarized SNN is employed that implements Time-to-First-Spike (TTFS) coding with Integrate-and-Fire neurons to classify images of handwritten digits (MNIST) at an accuracy of 96%. The discovery of a synchronous cluster, namely, neurons that spike at equivalent timesteps for every inference, allows the introduction of floating output neuron thresholds, reducing the network size by 24.9%. This is combined with an analysis on the excitatory and inhibitory input weights that leads to an additional network size reduction of 33.7%. Furthermore, an SNN hardware architecture is proposed, dubbed NimbleSNN, intended for implementation on the ICE40UP5K FPGA that balances speed and resource constraints. To optimize this architecture, an algorithm is introduced that considers the synchronization between neurons during inference in hardware, leading to neuron scheduling in memory that allows reduction of spike buffers from 14 spikes per hidden engine to 3 spikes without comprise in accuracy. At four hidden engines that process 14 neurons per two cycles, this algorithm leads to a reduction of logic elements for implementation by 45.81%. NimbleSNN implemented on the ICE40UP5K FPGA uses a total of 4,693 Logic Elements (LE) (number of registers + look up tables), leading to a power consumption of 20.4 mW and an inference latency on the MNIST dataset of 65.8 μ s at 40 MHz clock frequency. These lead to performance metrics of 1.34 μ J per inference and an effective efficiency of 47.7 GSOp/s/W, outperforming all rate-based SNN implementations on FPGA and showing possibly the most efficient SNN FPGA accelerator on MNIST available at the time of writing with the least number of LEs. As a result, this Chapter leads to the following claim

Claim 3: *Exploiting the sparsity of spike-efficient algorithms and separation of syn-*

chronous neurons in memory during processing leads to highly-efficient SNN accelerators (efficiency and accessibility).

This claim is partially fulfilled by Chapter 8, namely, a special case is shown where TTFS coding is employed, the most sparse spike approach within SNNs as neurons only fire once, leading to a minimized hardware architecture. Moreover, the separation of synchronous neurons during processing, i.e., allocating neurons to specific memory locations so that these never spike within the same processing cycle is another type of sparsity that allows reduction of spike buffer size. These spike buffers are involved in storing and routing output spikes of neurons from one layer to another. Although NimbleSNN shows high efficiency on the small ICE40UP5K FPGA platform, a large contribution is due to the exclusive use of binary weights. An open question remains how this memory allocation and spike buffer reduction could be exploited in the case where each neuron is free to spike according to its internal dynamics. In other words, future work is necessary to study similar methods for optimizing rate-based and recurrent SNNs that are more prevalent and find wider application on a range of temporal and sequential tasks.

To conclude Part II, a paradigm of neural network codesign is utilized that considers the dynamics, algorithms, hardware and applications within SNNs to advance the field of Edge-AI in terms of efficiency, capability and accessibility. While Part I moves from applications to algorithms and ends up at hardware in a linear fashion, this is not the case for Part II. The original inspiration of Part II, Chapter 6 moves from spiking oscillator circuits to non-linear dynamics with synchronization and ends up at an algorithm, namely, a physical reservoir. This physical reservoir then inspires the SNN on DYNAP-SE2 in Chapter 7 leading to encoding of seizures from the inherent dynamics on chip. This represents going backwards from algorithm to hardware and dynamics ending up at an application. Finally, synchronization is used to optimize a SNN accelerator on MNIST in Chapter 8. This Chapter represents the back and forward movement between algorithm, dynamics and hardware, leading to one of the most efficient SNN accelerators out there.

Putting Part I and II together with the presented claims, it is possible to conclude that the codesign of neural networks is a promising and legitimate approach for advancing Edge-AI in terms of capability, efficiency and accessibility. Here codesign indicates the consideration of a large number of disciplines. I believe this approach serves as a plausible but not conclusive antithesis to the specialist-oriented approach to research, design and development. I invite the reader to consider stepping out of their comfort zones by considering and studying unknown disciplines that can then be subsequently applied to familiar research fields once more.

Other than the paradigm of neural network codesign, it is interesting to note that all works use natural phenomena as an inspiration or foundation for the subsequent methodologies that are proposed. Part I starts with a study on livestock farming, its environmental impact and introduces PLF with Edge-AI as a possible mitigation strategy. Indeed, it is the potential of monitoring livestock behavior with high precision by means of Edge-AI that inspires these works. On the other hand, Part II focuses on advancements in Edge-AI by studying the use of an emergent phenomena prevalent in nature, synchronization, in SNNs to encode seizures and reduce the logic size of accelerators. Indeed, it is the potential of computation performed by organisms in nature, namely, consideration of synchronization that can for example also be observed in groups of fireflies [481], that inspires these works. The above ap-

proaches are unconventional in that their inspiration originates from fields located far away from the objective at hand, namely, the advancement of Edge-AI. Synchronization is a widely-studied phenomenon in the field of physics and network science and livestock is a field on itself. These fields have limited overlap with Edge-AI at the onset. As a result, it is possible to formulate another possible claim

Claim 4: *The study and consideration of natural phenomena, including but not limited to synchronization, neuromorphic computing and cow behavior, serves as an inspiration for advancing Edge-AI.*

In conclusion, this Thesis describes a wide number of proposed methodologies and investigations that span disciplines, including the use of synchronization phenomena for computation, and the creation and optimization of various Edge-AI hardware architectures. These methods are used as proof of concepts for cow behavior estimation as well as seizure encoding and detection on actual hardware. Many of these concepts represent a first initial step into the direction of a different approach to AI that is more natural-like, and a step towards the democratization of Edge-AI by employing affordable hardware. The reader is invited to build upon any of these works or use them as sources of inspiration. As much as I could, I have made the digital hardware designs, algorithms and converted datasets publicly available.

9.3 Discussion and Future Work

Although it is clear that the impact and contributions of this thesis are wide, there are some caveats that were touched upon and should be discussed with future works that link into these limitations. In this section I will introduce future works that could build upon the works described in this Thesis. The following two proposed works were also displayed in Fig. 1.11.

9.3.1 Physics Inspired Spiking Neural Network (PiSNN)

First of all, Chapter 6 introduces a wide number of synchronization phenomena originating from the influence of coherent noise, links, coupling strength and heterogeneity under chaotic and periodic dynamics. Furthermore, a method is proposed to perform computation using these phenomena, supported by a wide range of existing literature. However, no actual proof-of-concept is shown using this proposed method with the PCB board, this is one limitation. On the other hand, Chapter 7 shows an actual proof-of-concept of employing partial synchronization to encode seizures using a full system pipeline that includes analog-to-spike conversion and a Spiking Neural Network (SNN) on a mixed-signal processor, DYNAP-SE2. This work falls short due to the lack of a comprehensive analysis on the non-linear dynamics of networks of neurons on DYNAP-SE2. Instead, the implemented SNN is a hand-tuned network that was partially determined using a Differential Evolutionary algorithm that tunes the parameters and weights. These two shortcomings lead to a proposed future work that I dub as a Physics Inspired Spiking Neural Network (PiSNN) considering a deep experimental investigation including random graphs and various types of small-world graphs [482] reapplied to the application of encoding seizures. In principle, the envisioned goals of this work are twofold:

1. To study the experimental phenomena and dynamics of Exponential Leaky-

Integrate-and-Fire (ExpLIF) neurons within various types of graphs in-silico.

2. To propose theoretical methods for the use of the emerging phenomena of networks of ExpLIF neurons in computation with a proof-of-concept on an actual application

Here, point 1 focuses on the lack of experimental works that studies these phenomena. According to my knowledge, a wide study on the non-linear dynamics within large-scale networks of ExpLIF on a mixed-signal neuromorphic processor does not exist at the time of writing. It is known that these neurons show rich dynamics and behaviors with similarities to the brain, but the emerging properties of brain-like networks that arise from point 1 and their use in computation remains unexplored.

9.3.2 Recurrent Spiking Neural Networks (RSNN) for Resource-Constrained Hardware including FPGA

Chapter 5 proposes a Recurrent Neural Network (RNN) that employs integer-only quantization and a hardware architecture that minimizes resources by means of time-and-layer multiplexing for low-frequency sensors in Edge-AI. I believe this work, combined with open-sourced code, a proof-of-concept on cow behavior estimation and its implementation on the Lattice ICE40UP5K makes it highly impactful. On the other hand, Chapter 8 shows a proof-of-concept on employing synchronization combined with a time-to-fire-spike coding SNN on MNIST. However, this SNN does not include recurrent connections, highly limiting the impact of this SNN for a wide number of time-series applications prevalent in AIoT and edge computing.

The hardware architectures of the proposed SNN implementation and RNN can be combined to create a Recurrent SNN that employs the sparsity of spike-based encoding as well as the power of recurrent connections. Although the above proposed PiSNN also includes recurrency within its network, it represents a distinct approach that focuses on emerging phenomena arising from the non-linear dynamics within networks of ExpLIF neurons on physical substrate. Instead, I propose to start with a literature study on the available Recurrent SNNs that have no available hardware implementations followed by the formulation of a hardware architecture that can both be optimized for low-frequency application as well as speed. In addition, recent algorithms that employ online-learning with SNNs are promising for IoT applications and resource-constrained FPGA implementations and provide a starting point for this review [483–485]. Furthermore, applying the approach of optimizing spike buffers within the hardware architecture by using synchronization is another promising avenue to explore.

Application-oriented work, tactile sensing, medical and robotics

Other than focusing on benchmarks in time-series classification, there are a large number of applications that have not yet been widely used in combination with SNNs. One example is Precision Livestock Farming. Here, cow behavior estimation could serve as a proof of concept once again. Other than this, tactile sensing is a wide field where more advanced AI methods including neuromorphic computing could have a benefit as these are often combined with robotics [486] as well as agriculture [487]. I believe this is as important as advancing the field with new hardware architectures. In the end, if SNNs can persevere on extremely resource-

constrained FPGAs as Chapter 8 shows while being applied to a diverse number of applications, a lot more attention could go to advancing neuromorphic computing.

Appendix A

Academic Publications

A.1 First Author

A.1.1 International Journal Peer-Reviewed

1. **J. Bartels**, O. Gallou, H. Ito, M. Cook, J. Sarnthein, G. Indiveri, and S. Ghosh, “An Event-Driven Neural Network for Long-Term Monitoring of Epileptic Seizures on Low-Power Neuromorphic Hardware”. (under review) [J1]
2. **J. Bartels**, I. Heidtmann, A. Hagihara, L. Minati, and H. Ito, “NimbleSNN: A 1.34 μ J/Image 4639 Logic Element Single-Spike Accelerator on FPGA Using Synchronization”. (under review) [J2]
3. **J. Bartels**, A. Hagihara, L. Minati, K. K. Tokgoz, and H. Ito, “An Integer-Only Resource-Minimized RNN on FPGA for Low-Frequency Sensors in Edge-AI,” *IEEE Sensors Journal*, vol. 23, no. 15, pp. 17 784–17 793, 2023. [J3]
4. L. Minati*, **J. Bartels***, C. Li, M. Frasca, and H. Ito, “Synchronization phenomena in dual-transistor spiking oscillators realized experimentally towards physical reservoirs,” *Chaos, Solitons & Fractals*, vol. 162, p. 112415, 2022. [J4]
5. **J. Bartels**, K. K. Tokgoz, S. A. M. Fukawa, S. Otsubo, C. Li, I. Rachi, K.-I. Takeda, L. Minati, and H. Ito, “Tynecownet: Memory- and Power-Minimized RNNs Implementable on Tiny Edge Devices for Lifelong Cow Behavior Distribution Estimation,” *IEEE Access*, vol. 10, pp. 32 706–32 727, 2022. [J5]

*: shared first authorship

A.1.2 International Conference Peer-Reviewed

6. **J. Bartels**, K. K. Tokgoz, M. Fukawa, S. Otsubo, L. Chao, I. Rachi, K. Takeda, and H. Ito, “A 216 μ W, 87% Accurate Cow Behavior Classifying Decision Tree on FPGA with Interpolated Arctan2”, in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp.1–5. [C6]

A.1.3 Domestic Non-Peer Reviewed

7. **J. Bartels**, K. K. Tokgoz, M. Fukawa, S. Otsubo, L. Chao, I. Rachi, K. Takeda, and H. Ito, “LBCE: A Low Power Behavior Classification Engine using Time-Multiplexed GRU”, 電子情報通信学会 集積回路研究会 学生・若手研究会, Dec. 2020.

A.2 Co-Author

A.2.1 International Journal Peer-Reviewed

1. L. Minati, C. Li, **J. Bartels**, P. Chakraborty, Z. Li, N. Yoshimura, M. Frasca, and H. Ito, “Accelerometer time series augmentation through externally driving a non-linear dynamical system,” *Chaos, Solitons Fractals*, vol. 168, p. 113100, 2023.
2. L. Minati, B. Li, **J. Bartels**, Z. Li, M. Frasca, and H. Ito, “Incomplete synchronization of chaos under frequency-limited coupling: Observations in single-transistor microwave oscillators,” *Chaos, Solitons Fractals*, vol. 165, p. 112854, 2022.
3. C. Li, K. K. Tokgoz, A. Okumura, **J. Bartels**, K. Toda, H. Matsushima, T. Ohashi, K.-i. Takeda, and H. Ito, “A data augmentation method for cow behavior estimation systems using 3-axis acceleration data and neural network technology,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 105, no. 4, pp. 655–663, 2022.
4. C. Li, L. Minati, K. K. Tokgoz, M. Fukawa, **J. Bartels**, A. Sihan, K.-I. Takeda, and H. Ito, “Integrated data augmentation for accelerometer time series in behavior recognition: Roles of sampling, balancing, and fourier surrogates,” *IEEE Sensors Journal*, vol. 22, no. 24, pp. 24 230–24 241, 2022.
5. C. Li, K. K. Tokgoz, M. Fukawa, **J. Bartels**, T. Ohashi, K.-i. Takeda, and H. Ito, “Data augmentation for inertial sensor data in cnns for cattle behavior classification,” *IEEE Sensors Letters*, vol. 5, no. 11, pp. 1–4, 2021.

A.2.2 International Conference Peer-Reviewed

6. P. Chakraborty, K. Maari, **J. Bartels**, A. Varieras, A. T. Narayan, L. Minati, S. Dosho, and H. Ito, “Towards digital synthesis of variable q-factor direct-conversion for low-power edge sensing,” in *2023 IEEE SENSORS*, 2023, pp. 1–4.
7. C. Li, K. K. Tokgoz, **J. Bartels**, T. Ohashi, H. Ito, A. Okumura, K. Takeda, K. Toda, H. Matsushima, “Data Augmentation for cow behavior Estimation Systems Based on Neural Network Technology”, in *2020 International Workshop on Smart Info-Media System in Asia*. IEICE, 2020, pp. 95–100.

A.2.3 Domestic Non-Peer Reviewed

8. S. Otsubo, M. Fukawa, **J. Bartels**, K. K. Tokgoz, S. Lee, K. Takeda, H. Ito, “放牧に対応したウシ行動推定デバイスの試作”, 電子情報通信学会 集積回路研究会 学生・若手研究会, Dec. 2020.

Appendix B

Published Datasets and Source Codes

B.1 Datasets

1. H. Ito, K. Takeda, K. K. Tokgoz, L. Minati, M. Fukawa, L. Chao, **J. Bartels**, I. Rachi, and A. Sihan “Japanese Black Beef Cow Behavior Classification Dataset,” Zenodo, January 17, 2022.
<https://doi.org/10.5281/zenodo.5849025>.
2. **J. Bartels**, O. Gallou, H. Ito, M. Cook, J. Sarnthein, G. Indiveri, and S. Ghosh. “Spiking Seizure Classification Dataset,” Zenodo, March 9, 2024.
<https://doi.org/10.5281/zenodo.10800794>.

B.2 Source Codes (Verilog + Python)

1. **J. Bartels**, A. Hagihara, L. Minati, K. K. Tokgoz, and H. Ito. “An Integer-only Resource-minimized RNN on FPGA for Low-frequency Sensors in Edge-AI,” Zenodo, April 5, 2023.
<https://doi.org/10.5281/zenodo.7800728>.
2. **J. Bartels**, I. Heidtmann, A. Hagihara, L. Minati, and H. Ito. “NimbleSNN: A 1.34 μ J/Image 4639 Logic Element Single-Spike Accelerator on FPGA Using Synchronization,” Zenodo, May 1st, 2024.
<https://doi.org/10.5281/zenodo.11096897>

Appendix C

Hardware Platforms

This Chapter provides an overview of the two main hardware platforms that are used throughout this Thesis for implementing algorithms and constructing Edge-AI systems. There is an additional hardware circuit that is studied in Chapter 6 that is not able to implement any kind of algorithm from the onset. Instead the focus there is on studying the emergent physical phenomena that occur under various conditioning of the underlying circuit in a network setup and its application for Edge-AI. Therefore, this hardware has been omitted in this chapter as it is not a platform as such.

This chapter is divided into two sections. First, I introduce the DYNamic neuromorphic Asynchronous Processor Scalable 2 (DYNAP-SE2) that is a custom-designed neuromorphic chip, implementing Adaptive Exponential Leaky-Integrate-and-Fire neurons (AdExpLIF), developed in the Neurocognitive Systems group of the Institute of Neuroinformatics of the University of Zurich and ETH Zurich. Further details about this hardware platform can be found in Ref. [152]. This platform is used throughout Chapter 7 and one of its main building blocks, the Differential Pair Integrator (DPI) circuit, is simulated in Chapter D and can be used to encode the same intracranial electroencephalogram (iEEG) dataset as in Chapter 7. Second, I introduce the Lattice (Hillsboro, OG) ICE40UP5K Field Programmable Gate Array (FPGA) that is used as a hardware implementation platform for digital designs throughout Chapters 3, 5 and 8, and as a reference in Chapter 4.

C.1 DYNamic neuromorphic Asynchronous Processor (DYNAP-SE2)

The DYNAP-SE2 neuromorphic processor contains analog exponential integrate and fire (ExpIF) neuronal and synaptic circuits with spike-frequency adaptation. The DYNAP-SE2 also hosts an Analog Front End (AFE) that includes a level-crossing Asynchronous Delta Modulator (ADM) circuit on the same die. Here, I describe the equations of the ExpIF neurons and synapses derived from the circuit equations of DYNAP-SE2. This is followed by an explanation of the AFE and a description of the neuron cores of DYNAP-SE2.

C.1.1 Neuron and synapse models

The neurons employed for the SNN in Chapter 7 consist of current-mode Exponential Integrate-and-Fire (IF) neuron circuits. The detailed description of the circuit

equations in the sub-threshold regime and their deduction leading to this simple form is described by Ref. [424]. Omitting the adaptation term, this equation is shown by

$$\tau(1 + \frac{I_{th}}{I_{mem}})\frac{d}{dt}I_{mem} + I_{mem} = \frac{I_{th}}{I_{\tau}}(I_{in} - I_{\tau}) + f(I_{mem}) \quad (C.1)$$

where I_{mem} is the subthreshold current depicting the membrane potential of the silicon neuron. Other terms including the time constant and the threshold can be defined as follows:

$$\begin{aligned} \tau &\triangleq \frac{C_{mem}U_T}{\kappa I_{\tau}} \\ I_{\tau} &\triangleq I_0 \exp \frac{\kappa}{U_T} v_{lk} \\ I_{th} &\triangleq I_0 \exp \frac{\kappa}{U_T} v_{thr} \end{aligned} \quad (C.2)$$

U_T is the thermal voltage, κ is the subthreshold slope factor stemming from the capacitive coupling ratio from gate to channel and I_0 is the transistor dark current which are constants of the n-type MOSFETs used in the circuit [488]. The non-linear term comes from $f(I_{mem})$ which has been experimentally shown to be an exponential function of I_{mem} [489].

The synapses of DYNAP-SE2 are modeled after the DPI circuit [490] and can be represented in simplified form as

$$\tau \frac{d}{dt}I_{out} + I_{out} = I_{in} \frac{I_w}{I_{\tau}} \quad (C.3)$$

where I_{out} is the output current, the time constant is $\tau \triangleq \frac{C_{syn}U_T}{\kappa I_{\tau}}$, where I_{τ} is a bias current, and $I_{in} \triangleq I_0 \exp \frac{\kappa}{U_T} v_{in}$ is the input current.

C.1.2 Analog neurons with digital routing

Figure C.2b shows the block diagram of DYNAP-SE2 comprising four neuron cores and the AFE. Each core contains 256 Adaptive Exponential Leaky-Integrate-and-Fire (AdExpIF) neurons, totaling 1024. All neuron parameters, i.e., time-constants of the synaptic compartments and soma, gain, threshold, and refractory period are shared within a single core. In other words, only four sets of parameters can be assigned within the entire chip. Each neuron has 64 synapses that can receive spikes from any neuron throughout the whole chip and can be connected to up to 1024 post-synaptic neurons. In the event of an incoming spike, up to four synaptic currents are generated, their magnitudes based on 4-bit weights, that are added and fed into one of four synapse types: fast excitatory (AMPA-like), slow excitatory (NMDA-like), slow inhibitory (GABA-B like) (Fig. C.2c) and fast inhibitory (GABA-A like) that acts as a shunt to the soma. The latter synapse type is not considered in this thesis. Throughout the chip, there is a 20–30% variation on all parameters as a result of the subthreshold MOSFET design and the 180 nm manufacturing process [441]. Circuit diagrams of the DPI circuits and a description of additional features can be found in Ref. [304].

C.1.3 Experimental Setup

Fig. C.1 shows the experimental setup that was constructed to implement any Spiking Neural Network (SNN) on DYNAP-SE2 and stream in real-time data using a

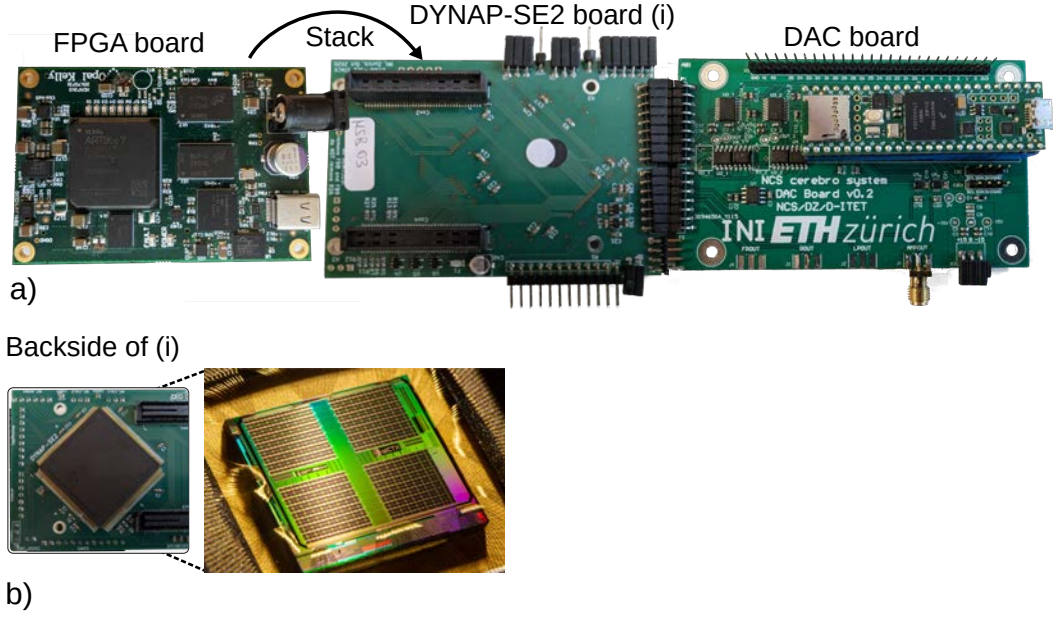


Figure C.1: a) Photographs of the setup, an FPGA board, DYNAP-SE2 board with DYNAP-SE2 chip on the underside, Digital-to-Analog (DAC) converter board with a Teensy microcontroller for interfacing with the AFE; b) Photograph of the DYNAP-SE2 die (right), and the package located on the underside of the DYNAP-SE2 board (left).

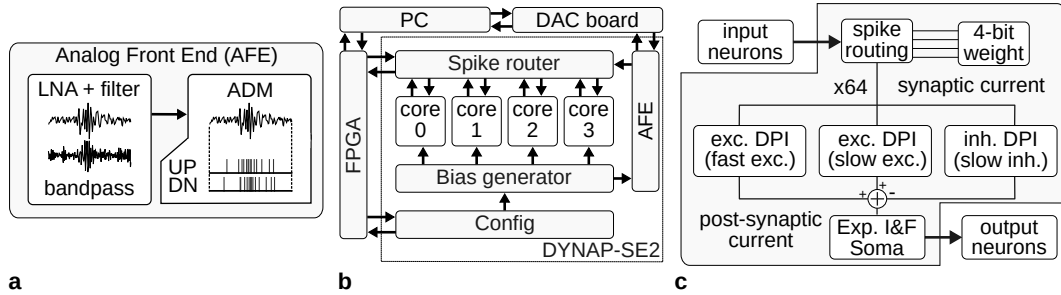


Figure C.2: a Internal components of the Analog Front End (AFE), a Low-Noise Amplifier (LNA) with bandpass filter followed by the Asynchronous Delta Modulator (ADM); b Block diagram of the boards and DYNAP-SE2: the FPGA sends input spikes to the spike cores and configures the AFE and SNN cores; a DAC board interfaces directly with the AFE; c Neuron schematic. Input events are translated to synaptic currents. The synaptic currents are routed to one of four synapse types, each combined with Differential Pair Integrators, and forwarded to an Adaptive Exponential Leaky Integrate & Fire soma.

Digital-to-Analog Converter (DAC) board. The FPGA board programs DYNAPSE-2 according to the set neuron parameters per core, enables connections between neurons to create a user-defined network and reads out the spikes of all configured neurons. This setup was preconstructed by members of the Institute of Neuroinformatics at the University of Zurich and ETH Zurich and employed in the study of Chapter 7. More information can be found in Ref. [152].

C.2 Field Programmable Gate Array (FPGA), the Lattice ICE40UP5K

The Lattice ICE40UP5K is part of the ICE40 UltraPlus FPGA family that can be described as extremely resource-constrained, low-power FPGAs. The ICE40UP5K contains 5280 Look Up Tables (LUT) and registers and 30, 256x16 Embedded Block RAM (EBR) memory blocks that can be initiated with a preset ".mem" file and changed during operation. Other than EBRs, the main memory blocks of the FPGA consist of four Single Port RAMs (SPRAM), a single block containing 16k addresses of 16 bit words (256 kbits). These SPRAMs have to be initialized during operation itself. In other words, data has to be streamed to these SPRAMs during operation instead of compilation with a preset file as with the EBRs. Other than memory, this FPGA contains hard-core modules that are packaged as IP such as 8 Digital Signal Processors (DSP) that can be employed for multiplication as well as two oscillators and a Phase-Locked Loop (PLL). While the two on-board oscillators have frequencies of 48 MHz and 10 KHz, the PLL has a maximum output frequency of 275 MHz. At a typical price 10 USD and a minimum package size of 5.48 mm², this FPGA represents one of the most resource-constrained and economically affordable tailored hardware on the market.

Accordingly, the motivation for using this FPGA lies in its affordability and resource-constrained environment. On one hand, hardware implementations that fit on this device can be used by a majority of the population worldwide. On the other hand, this FPGA environment requires resource-optimized digital design to be able to implement any algorithm on it. Therefore, it can also serve as an optimal staging or initial development platform for small-area, low-power Application Specific Integrated Circuits (ASIC).

C.2.1 Experimental Setup

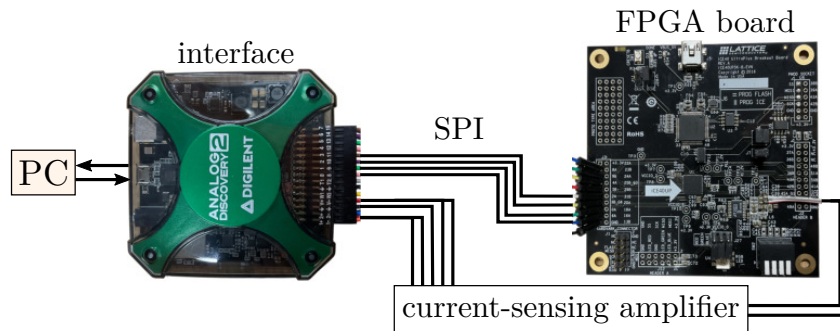


Figure C.3: FPGA measurement setup using Analog Discovery 2 and Lattice ICE40UP5K-B-EVN breakout board.

Fig. C.3 shows the experimental setup utilized to assess power consumption and validate the Edge-AI hardware implementations within this thesis (Chapters 5 and 8) on the Lattice ICE40UP5K FPGA, employing the Lattice ICE40UP5K-B-EVN breakout board. The RNN I/Os, comprising 4 SPI pins for the SPI are connected to the digital I/O of the Digilent Analog Discovery 2 (Digilent Inc., Pullman WA).

Appendix D

Adaptive Threshold for Asynchronous Delta Modulation: Hardware-Based Simulations and Seizure Encoding

Previous studies have shown approaches for encoding biosignals of different modalities into Up and Down event streams with Asynchronous Delta Modulators (ADM) using fixed delta thresholds, followed by neuromorphic processor that implement Spiking Neural Networks (SNN) [432, 437, 491]. In these studies, the delta threshold of the ADM is selected based on properties of input signals. However, in real applications that include biosignals retrieved from patients over a long period of time, this is suboptimal. Especially in applications that utilize electrodes to retrieve EEG signals, inevitable electrode placement differences and intersubject variability changes the properties of each signal. As a result, the encoding of these signals into events with ADM using a fixed threshold could lead to a low signal-to-noise ratio that contributes to false alarms and missed detection in seizure detection systems on neuromorphic processors.

Instead of having a pre-set threshold that is tuned heuristically, automatic calibration of the delta threshold during the initial period of recording leads to better encoding of the analog signals to events. Here, I propose a 10 second initial time window where a threshold is automatically set based on the signal properties. During the calibration window, I adopted an approach of simulating a proposed system used for determining this threshold based on the frequency characteristics of the signal [492]. This approach presents a system using the fundamental building blocks of the DYNAP-SE2 neuromorphic processor, the Differential-Pair Integrator (DPI) and a Source-Follow Integrator (SFI) circuit.

D.1 Differential Pair Integrator and Source-Follower Circuits Simulation for Adaptive Thresholding

Here, I describe differential equations of subthreshold circuits and their simulations based on the previously proposed adaptive ADM system [492] to perform long-term

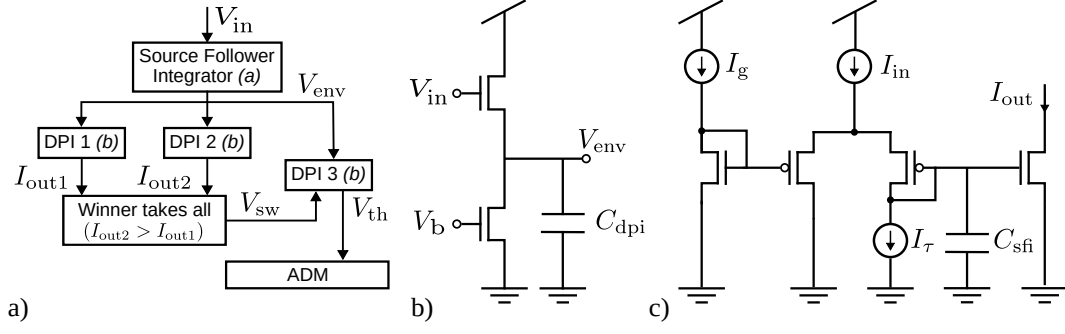


Figure D.1: a) Block diagram of the non-linear adaptive threshold calibration circuits used in simulation. b) Source Follower Integrator circuit. c) Differential Pair Integrator with non-relevant biasing voltages omitted.

conversion of analog signals into events on iEEG. The objective of the adaptive ADM (adaADM) approach is to dynamically adjust the delay threshold to amplitude changes in background activity while retaining information related to large fluctuations. I follow this adaADM encoding technique to convert the iEEG signals into UP and DOWN events streams after the first 10 seconds of calibration and create an integrated simulation that can convert data of any time length. Fig. D.1a shows an overview of this adaptive system, comprising of a Source Follower Integrator (SFI) (Fig. D.1b) circuit that acts as a non-linear envelope detector followed by 3 Differential Pair Integrator (DPI) circuits (Fig. D.1c). The Source Follower Integrator circuit is described by the following differential equations

$$\frac{dV_{out}}{dt} = \begin{cases} \frac{I_0}{C_{sfi}} (e^{\frac{\kappa V_{in} - V_{env}}{U_T}} - e^{\frac{\kappa V_b}{U_T}}) & \text{if } V_{in} > V_{env} \\ -\frac{I_0}{C_{sfi}} (e^{\frac{\kappa V_b}{U_T}}) & \text{if } V_{in} < V_{env} \text{ and } V_{env} > 0 \end{cases} \quad (D.1)$$

where V_{in} is the input voltage of the circuit, the voltage of the EEG channels in this case, V_{env} is the output envelope voltage and C_{sfi} is the capacitance of the circuit. κ , U_T , V_b and I_0 are the mean slope factor at subthreshold, the thermal voltage, the bias voltage and the dark current, respectively. This circuit transforms the input voltage into an envelope using this non-linear transfer function. As for the DPI circuit, I use the equation derived in Ref. [493] where the DPI is approximated as a low-pass filter with non-unitary gain, i.e.,

$$\tau \frac{dI_{out}}{dt} + I_{out} = \frac{I_g}{I_{\tau}} I_{in} \quad (D.2)$$

where I_{out} is the output current and I_{in} is the input current. Furthermore, I_g is the virtual p-type subthreshold current and I_{τ} is another bias current. The time constant τ is described by $\tau = \frac{C_{dpi} U_T}{\kappa I_{\tau}}$ and the input current is set by $I_{in} = I_0 (e^{\frac{\kappa V_{env}}{U_T}})$ where V_{env} is the output voltage of the SFI circuit. The output current of DPI3 is transformed to the threshold voltage using the current-to-voltage equation $V_{th} = \frac{U_T}{\kappa} \ln(\frac{I_{out3}}{I_0})$. Note that both the DPI and SFI circuits operate in the subthreshold regime, where the current through a transistor has an exponential relationship with the gate to source voltage.

Using the circuits described above, the adaptive threshold is set as follows. During the first 10 seconds of input, the blocks in Fig. D.1a are enabled. The input voltage is transformed with the source follower integrator to form an envelope, V_{env} .

This envelope is forwarded to three DPI circuits, DPI1, DPI2 and DPI3. The time constants of DPI1 and DPI2 are set such that rise times for step input are slower for one than the other for a fast changing input. In other words, the cutoff frequency of DPI1 is lower than that of DPI2, causing output voltage of DPI2 to cross that of DPI1 when the frequency of the input voltage is higher than the cutoff of DPI1 in the signal. As a result, $\tau_1 > \tau_2$ where $\tau_1 = 0.36$ and $\tau_2 = 0.11$. Whenever the output current of DPI1 crosses that of DPI2, namely, $I_{out2} > I_{out1}$, the switch voltage, V_{sw} , switches to high, enabling DPI3. Consequently, the delta threshold, V_{th} , is taken as the output voltage of DPI3. The time constant of DPI3 is set such that $\tau_1 > \tau_2 > \tau_3$ so that it passes most of the frequency components of the envelope signal to set this threshold. The threshold voltage is thus only held during the calibration phase if and only if the switch voltage is high. The final voltage resulting from the above described system is used as the "UP" and "DOWN" thresholds of the ADM. For simulation, I have used the laplace domain variant of the DPI transfer function for filtering the input current and first-order Euler to solve for the source follower circuits.

D.2 Using AdaADM to Convert iEEG Data to Spikes

The system presented in Fig. D.1 is utilized for converting an iEEG dataset that contains segments where patients are undergoing clinically annotated epileptic seizures. This is the ETH-SWEC dataset [433]. In this Section, an example of the system is presented based on a single channel, single patient segment of iEEG of this dataset during a seizure. The time length of his segment is approximately 420 seconds and the seizure starts at 180 seconds.

Fig. D.2 top row shows input voltage, V_{in} , representing the iEEG signal and output voltage of the Source-Follower Integrator circuit denoted as V_{env} over time on the x-axis. The second row shows the input current, I_{in} to all DPI circuits that is derived from the input voltage. The third row shows the output currents of DPI1 and DPI2, respectively, followed by the threshold voltage determined by DPI3 that is held only when the switch is high (green signal) on the fourth row. The last row shows the "UP" and "DOWN" events as a result of the adaptive threshold calibration, firing rate per second encoded by color. The ADM output shows a very low number of events for the initial 180 seconds during the iEEG signal followed by a significant increase in events from 180 seconds to around 400 seconds. The seizure starts at 180 seconds, showing a clear distinction between non-seizure and seizure periods encoded by the ADM. This figure shows a single channel of a single seizure of a single patient for explaining the operation of the adaADM system and simulation results.

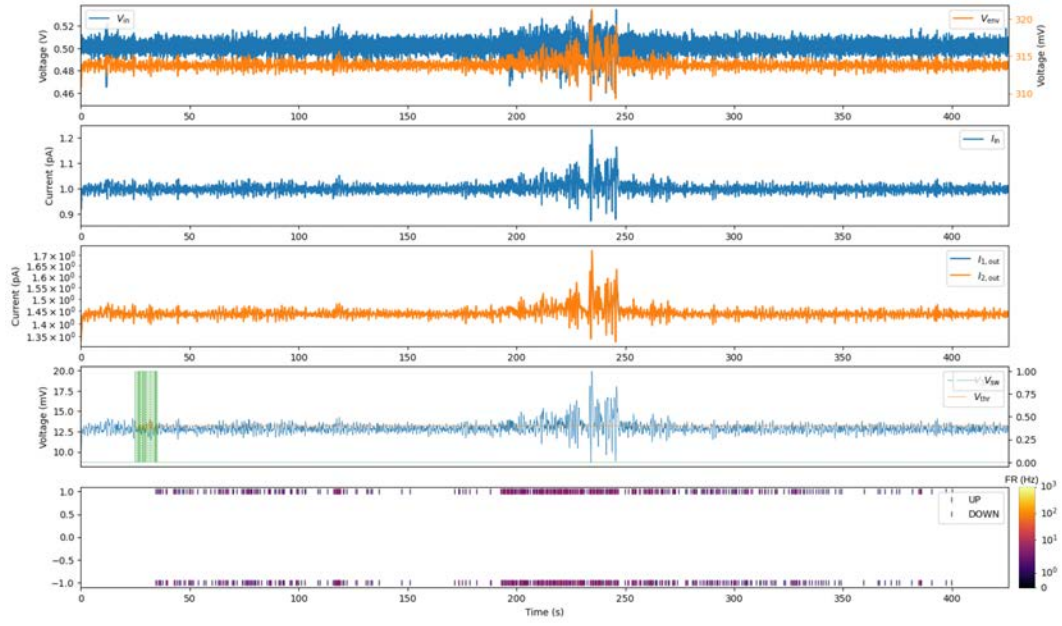


Figure D.2: Adaptive ADM threshold system operating on 420 seconds of a single channel of iEEG data from the ETH-SWEC dataset [433] and software-simulated Asynchronous Delta Modulator (ADM) output.

References

- [1] S. Harmand, J. E. Lewis, C. S. Feibel, C. J. Lepre, S. Prat, A. Lenoble, X. Boës, R. L. Quinn, M. Brenet, A. Arroyo *et al.*, “3.3-million-year-old stone tools from lomekwi 3, west turkana, kenya,” *Nature*, vol. 521, no. 7552, pp. 310–315, 2015.
- [2] F. Berna, P. Goldberg, L. K. Horwitz, J. Brink, S. Holt, M. Bamford, and M. Chazan, “Microstratigraphic evidence of in situ fire in the acheulean strata of wonderwerk cave, northern cape province, south africa,” *Proceedings of the National Academy of Sciences*, vol. 109, no. 20, pp. E1215–E1220, 2012.
- [3] T. D. Price and O. Bar-Yosef, “The origins of agriculture: new data, new ideas: an introduction to supplement 4,” *Current Anthropology*, vol. 52, no. S4, pp. S163–S174, 2011.
- [4] J.-D. Vigne, “The origins of animal domestication and husbandry: a major change in the history of humanity and the biosphere,” *Comptes rendus biologies*, vol. 334, no. 3, pp. 171–181, 2011.
- [5] S. Berryman, “Ancient atomism,” 2005.
- [6] P. Plato *et al.*, *The republic*. Wiley Online Library, 2008, vol. 7.
- [7] A. S. Altekar, *State and government in ancient India*. Motilal Banarsidass Publ., 2002.
- [8] D. L. Hall and R. T. Ames, *Thinking through confucius*. State University of New York Press, 1987.
- [9] L. Tredinnick, *Digital information culture: the individual and society in the digital age*. Elsevier, 2008.
- [10] I. Levin and D. Mamlok, “Culture and society in the digital age,” *Information*, vol. 12, no. 2, p. 68, 2021.
- [11] R. Gruetzemacher, F. E. Dorner, N. Bernaola-Alvarez, C. Giattino, and D. Manheim, “Forecasting ai progress: A research agenda,” *Technological Forecasting and Social Change*, vol. 170, p. 120909, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0040162521003413>
- [12] M. Attaran, “The internet of things: Limitless opportunities for business and society,” *Journal of Strategic Innovation and Sustainability Vol*, vol. 12, no. 1, p. 11, 2017.
- [13] I. Khan, “Suitability of lora, sigfox and nb-iot for different internet-of-things applications,” Master’s thesis, 2019.

- [14] “Eltres technology whitepaper,” 2024. [Online]. Available: https://www.sony-semicon.com/files/62/eltres/ELTRES_Technology_Whitepaper.pdf
- [15] R. Want, B. N. Schilit, and S. Jenson, “Enabling the internet of things,” *Computer*, vol. 48, no. 1, pp. 28–35, 2015.
- [16] B. N. Silva, M. Khan, and K. Han, “Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in smart cities,” *Sustainable Cities and Society*, vol. 38, pp. 697–713, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2210670717311125>
- [17] D. Mocrii, Y. Chen, and P. Musilek, “Iot-based smart homes: A review of system architecture, software, communications, privacy and security,” *Internet of Things*, vol. 1-2, pp. 81–98, 2018. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2542660518300477>
- [18] L. Da Xu, W. He, and S. Li, “Internet of things in industries: A survey,” *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.
- [19] R. T. Yarlagadda, “Internet of things & artificial intelligence in modern society,” *International Journal of Creative Research Thoughts (IJCRT)*, ISSN, pp. 2320–2882, 2018.
- [20] K. Rose, S. Eldridge, and L. Chapin, “The internet of things: An overview,” *The internet society (ISOC)*, vol. 80, pp. 1–50, 2015.
- [21] S. F. Wamba, R. E. Bawack, C. Guthrie, M. M. Queiroz, and K. D. A. Carillo, “Are we preparing for a good ai society? a bibliometric review and research agenda,” *Technological Forecasting and Social Change*, vol. 164, p. 120482, 2021.
- [22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [23] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.
- [24] J. Degraeve, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas *et al.*, “Magnetic control of tokamak plasmas through deep reinforcement learning,” *Nature*, vol. 602, no. 7897, pp. 414–419, 2022.
- [25] K.-K. Mak, Y.-H. Wong, and M. R. Pichika, “Artificial intelligence in drug discovery and development,” *Drug Discovery and Evaluation: Safety and Pharmacokinetic Assays*, pp. 1–38, 2023.
- [26] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy, “Challenges and applications of large language models,” *arXiv preprint arXiv:2307.10169*, 2023.

- [27] A. S. Rao and G. Verweij, "Sizing the prize: What's the real value of ai for your business and how can you capitalise," *PwC Publication, PwC*, pp. 1–30, 2017.
- [28] B. Dong, Q. Shi, Y. Yang, F. Wen, Z. Zhang, and C. Lee, "Technology evolution from self-powered sensors to aiot enabled smart homes," *Nano Energy*, vol. 79, p. 105414, 2021.
- [29] D. Sehrawat and N. S. Gill, "Smart sensors: Analysis of different types of iot sensors," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 2019, pp. 523–528.
- [30] J. Pan and J. McElhannon, "Future edge cloud and edge computing for internet of things applications," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, 2017.
- [31] J. Finnegan and S. Brown, "An analysis of the energy consumption of lpwa-based iot devices," in *2018 International Symposium on Networks, Computers and Communications (ISNCC)*. IEEE, 2018, pp. 1–6.
- [32] M. Carroll, A. Van Der Merwe, and P. Kotze, "Secure cloud computing: Benefits, risks and controls," in *2011 information security for South Africa*. IEEE, 2011, pp. 1–9.
- [33] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE communications surveys & tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.
- [34] R. S. Sinha, Y. Wei, and S.-H. Hwang, "A survey on lpwa technology: Lora and nb-iot," *Ict Express*, vol. 3, no. 1, pp. 14–21, 2017.
- [35] V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge, "Real-time apple detection system using embedded systems with hardware accelerators: An edge ai application," *IEEE Access*, vol. 8, pp. 9102–9114, 2020.
- [36] Q. Dong, X. Chen, and M. Satyanarayanan, "Creating edge ai from cloud-based llms," in *Proceedings of the 25th International Workshop on Mobile Computing Systems and Applications*, 2024, pp. 8–13.
- [37] Z. Lin, G. Qu, Q. Chen, X. Chen, Z. Chen, and K. Huang, "Pushing large language models to the 6g edge: Vision, challenges, and opportunities," *arXiv preprint arXiv:2309.16739*, 2023.
- [38] L. Dong, F. Jiang, Y. Peng, K. Wang, K. Yang, C. Pan, and R. Schober, "Lambo: Large language model empowered edge intelligence," *arXiv preprint arXiv:2308.15078*, 2023.
- [39] J.-s. Seo, J. Saikia, J. Meng, W. He, H.-s. Suh, Y. Liao, A. Hasssan, I. Yeo *et al.*, "Digital versus analog artificial intelligence accelerators: Advances, trends, and emerging designs," *IEEE Solid-State Circuits Magazine*, vol. 14, no. 3, pp. 65–79, 2022.
- [40] W.-H. Huang, T.-H. Wen, J.-M. Hung, W.-S. Khwa, Y.-C. Lo, C.-J. Jhang, H.-H. Hsu, Y.-H. Chin, Y.-C. Chen, C.-C. Lo *et al.*, "A nonvolatile al-edge

- processor with 4mb slc-mlc hybrid-mode reram compute-in-memory macro and 51.4-251tops/w,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 15–17.
- [41] S. Moon, H.-G. Mun, H. Son, and J.-Y. Sim, “A 127.8 tops/w arbitrarily quantized 1-to-8b scalable-precision accelerator for general-purpose deep learning with reduction of storage, logic and latency waste,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 21–23.
 - [42] J. Yue, C. He, Z. Wang, Z. Cong, Y. He, M. Zhou, W. Sun, X. Li, C. Dou, F. Zhang *et al.*, “A 28nm 16.9-300tops/w computing-in-memory processor supporting floating-point nn inference/training with intensive-cim sparse-digital architecture,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 1–3.
 - [43] S. Kim, S. Kim, S. Hong, S. Kim, D. Han, and H.-J. Yoo, “C-dnn: A 24.5-85.8 tops/w complementary-deep-neural-network processor with heterogeneous cnn/snn core architecture and forward-gradient-based sparsity generation,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 334–336.
 - [44] C.-Y. Du, C.-F. Tsai, W.-C. Chen, L.-Y. Lin, N.-S. Chang, C.-P. Lin, C.-S. Chen, and C.-H. Yang, “A 28nm 11.2 tops/w hardware-utilization-aware neural-network accelerator with dynamic dataflow,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 1–3.
 - [45] W. Sun, X. Feng, C. Tang, S. Fan, Y. Yang, J. Yue, H. Yang, and Y. Liu, “A 28nm 2d/3d unified sparse convolution accelerator with block-wise neighbor searcher for large-scaled voxel-based point cloud network,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 328–330.
 - [46] T. Tambe, J. Zhang, C. Hooper, T. Jia, P. N. Whatmough, J. Zuckerman, M. C. Dos Santos, E. J. Loscalzo, D. Giri, K. Shepard *et al.*, “A 12nm 18.1 tflops/w sparse transformer processor with entropy-based early exit, mixed-precision predication and fine-grained power management,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 342–344.
 - [47] S. Park, S. Lee, J. Park, H.-S. Choi, and D. Jeon, “A 0.81 mm² 740 μ w real-time speech enhancement processor using multiplier-less pe arrays for hearing aids in 28nm cmos,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 21–23.
 - [48] Y. Gong, T. Zhang, H. Guo, X. Liu, J. Zheng, H. Wu, C. Jia, L. Que, L. Zhou, L. Chang *et al.*, “22.7 dl-vopu: An energy-efficient domain-specific deep-learning-based visual object processing unit supporting multi-scale semantic feature extraction for mobile object detection/tracking applications,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 1–3.

- [49] J. Zhang, D. Huo, J. Zhang, C. Qian, Q. Liu, L. Pan, Z. Wang, N. Qiao, K.-T. Tang, and H. Chen, “22.6 anp-i: A 28nm 1.5 pj/sop asynchronous spiking neural network processor enabling sub-o. 1 μ j/sample on-chip learning for edge-ai applications,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 21–23.
- [50] F. Conti, D. Rossi, G. Paulin, A. Garofalo, A. Di Mauro, G. Rutishauer, G. marco Ottavi, M. Eggimann, H. Okuhara, V. Huard *et al.*, “A 12.4 tops/w@ 136gops ai-iot system-on-chip with 16 risc-v, 2-to-8b precision-scalable dnn acceleration and 30%-boost adaptive body biasing,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 21–23.
- [51] J. Ryu, H. Kwon, W. Park, Z. Li, B. Kwon, D. Han, D. Im, S. Kim, H. Joo, and H.-J. Yoo, “Neugpu: A 18.5 mj/iter neural-graphics processing unit for instant-modeling and real-time rendering with segmented-hashing architecture,” in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 372–374.
- [52] J. Jung, S. Kim, B. Seo, W. Jang, S. Lee, J. Shin, D. Han, and K. J. Lee, “Lspu: A fully integrated real-time lidar-slam soc with point-neural-network segmentation and multi-level knn acceleration,” in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 370–372.
- [53] S. Kim, S. Kim, W. Jo, S. Kim, S. Hong, and H.-J. Yoo, “C-transformer: A 2.6-18.1 μ j/token homogeneous dnn-transformer/spiking-transformer processor with big-little network and implicit weight generation for large language models,” in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 368–370.
- [54] Y. Ju, G. Xu, and J. Gu, “A 28nm physics computing unit supporting emerging physics-informed neural network and finite element method for real-time scientific computing on edge devices,” in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 366–368.
- [55] K. Nose, T. Fujii, K. Togawa, S. Okumura, K. Mikami, D. Hayashi, T. Tanaka, and T. Toi, “20.3 a 23.9 tops/w@ 0.8 v, 130tops ai accelerator with 16 $\times$ performance-accelerable pruning in 14nm heterogeneous embedded mpu for real-time robot applications,” in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 364–366.
- [56] R. Guo, L. Wang, X. Chen, H. Sun, Z. Yue, Y. Qin, H. Han, Y. Wang, F. Tu, S. Wei *et al.*, “20.2 a 28nm 74.34 tflops/w bf16 heterogenous cim-based accelerator exploiting denoising-similarity for diffusion models,” in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 362–364.
- [57] M.-E. Shih, S.-W. Hsieh, P.-Y. Tsai, M.-H. Lin, P.-K. Tsung, E.-J. Chang, J. Liang, S.-H. Chang, C.-L. Huang, Y.-Y. Nian *et al.*, “Nve: A 3nm 23.2 tops/w 12b-digital-cim-based neural engine for high-resolution visual-quality enhancement on smart devices,” in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67. IEEE, 2024, pp. 360–362.

- [58] G. Desoli, N. Chawla, T. Boesch, M. Avodhyawasi, H. Rawat, H. Chawla, V. Abhijith, P. Zambotti, A. Sharma, C. Cappetta *et al.*, “A 40-310tops/w sram-based all-digital up to 4b in-memory computing multi-tiled nn accelerator in fd-soi 18nm for deep-learning edge applications.” in *ISSCC*, 2023, pp. 260–261.
- [59] S. Liu, P. Li, J. Zhang, Y. Wang, H. Zhu, W. Jiang, S. Tang, C. Chen, Q. Liu, and M. Liu, “A 28nm 53.8 tops/w 8b sparse transformer accelerator with in-memory butterfly zero skipper for unstructured-pruned nn and cim-based local-attention-reusable engine,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 250–252.
- [60] F. Tu, Z. Wu, Y. Wang, W. Wu, L. Liu, Y. Hu, S. Wei, and S. Yin, “Muitcim: A 28nm 2.24 μ J/token attention-token-bit hybrid sparse digital cim-based accelerator for multimodal transformers,” in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2023, pp. 248–250.
- [61] T. Sipola, J. Alatalo, T. Kokkonen, and M. Rantonen, “Artificial intelligence in the iot era: A review of edge ai hardware and software,” in *2022 31st Conference of Open Innovations Association (FRUCT)*. IEEE, 2022, pp. 320–331.
- [62] C. Xu, S. Jiang, G. Luo, G. Sun, N. An, G. Huang, and X. Liu, “The case for fpga-based edge computing,” *IEEE Transactions on Mobile Computing*, vol. 21, no. 7, pp. 2610–2619, 2020.
- [63] H. Chen, J. Zhang, Y. Du, S. Xiang, Z. Yue, N. Zhang, Y. Cai, and Z. Zhang, “A comprehensive evaluation of fpga-based spatial acceleration of llms,” in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2024, pp. 185–185.
- [64] R. Xu, J. Luo, and Y. Liang, “Hermes: Enhancing extensibility in high-level synthesis through multi-level irs,” in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2024, pp. 186–186.
- [65] S. Xu, W. Peng, W. Huang, Q. Liu, and Y. Huang, “Hr-gcn: An efficient gcn accelerator for heterogeneous graph data and r-gcn model,” in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2024, pp. 186–186.
- [66] Q. Wu, L. Zhao, Y. Gui, H. Liang, X. Wang, and X. Jin, “Efficient message passing architecture for gcn training on hbm-based fpgas with orthogonal topology on-chip networks,” in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2024, pp. 187–187.
- [67] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang *et al.*, “Flightllm: Efficient large language model inference with a complete mapping flow on fpgas,” in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2024, pp. 223–234.
- [68] J. Zhuang, J. Lau, H. Ye, Z. Yang, Y. Du, J. Lo, K. Denolf, S. Neuendorffer, A. Jones, J. Hu *et al.*, “Charm: Composing heterogeneous accelerators

- for matrix multiply on versal acap architecture,” in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2023, pp. 153–164.
- [69] A. Khataei, G. Singh, and K. Bazargan, “Approximate hybrid binary-unary computing with applications in bert language model and image processing,” in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2023, pp. 165–175.
 - [70] L. Cai, J. Wang, L. Yu, B. Yan, Y. Tao, and Y. Yang, “Accelerating neural-ode inference on fpgas with two-stage structured pruning and history-based stepsize search,” in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2023, pp. 177–183.
 - [71] M. Huang, Y. Liu, S. Huang, K. Li, Q. Wu, and H. Yu, “Multi-bit-width cnn accelerator with systolic-in-systolic dataflow and single dsp multiple multiplication scheme,” in *Proceedings of the 2023 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2023, pp. 229–229.
 - [72] Z. Susskind, A. Arora, A. T. Bacellar, D. L. C. Dutra, I. D. Miranda, M. Breternitz, P. M. Lima, F. M. França, and L. K. John, “An fpga-based weightless neural network for edge network intrusion detection.” in *FPGA*, 2023, p. 232.
 - [73] C. M. Christensen, *The innovator’s dilemma: when new technologies cause great firms to fail*. Harvard Business Review Press, 2013.
 - [74] S. Neethirajan, “The role of sensors, big data and machine learning in modern animal farming,” *Sensing and Bio-Sensing Research*, vol. 29, p. 100367, 2020.
 - [75] K. G. Liakos, P. Busato, D. Moshou, S. Pearson, and D. Bochtis, “Machine learning in agriculture: A review,” *Sensors*, vol. 18, no. 8, p. 2674, 2018.
 - [76] T. Wuest, D. Weimer, C. Irgens, and K.-D. Thoben, “Machine learning in manufacturing: advantages, challenges, and applications,” *Production & Manufacturing Research*, vol. 4, no. 1, pp. 23–45, 2016.
 - [77] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent trends in deep learning based natural language processing,” *ieee Computational intelligence magazine*, vol. 13, no. 3, pp. 55–75, 2018.
 - [78] S. Mittal and S. Umesh, “A survey on hardware accelerators and optimization techniques for rnns,” *Journal of Systems Architecture*, p. 101839, 2020.
 - [79] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, “Google’s neural machine translation system: Bridging the gap between human and machine translation,” *arXiv preprint arXiv:1609.08144*, 2016.
 - [80] B. Zhao, X. Li, and X. Lu, “Cam-rnn: Co-attention model based rnn for video captioning,” *IEEE Transactions on Image Processing*, vol. 28, no. 11, pp. 5552–5565, 2019.
 - [81] C. Gao, R. Gehlhar, A. D. Ames, S.-C. Liu, and T. Delbruck, “Recurrent neural network control of a hybrid dynamical transfemoral prosthesis with

- edgedrnn accelerator,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 5460–5466.
- [82] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, “A Survey of Accelerator Architectures for Deep Neural Networks,” *Engineering*, vol. 6, no. 3, pp. 264–274, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2095809919306356>
 - [83] P. Dong, S. Wang, W. Niu, C. Zhang, S. Lin, Z. Li, Y. Gong, B. Ren, X. Lin, and D. Tao, “Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.
 - [84] J. Bartels, A. Hagihara, L. Minati, K. K. Tokgoz, and H. Ito, “An Integer-Only Resource-Minimized RNN on FPGA for Low-Frequency Sensors in Edge-AI,” Apr. 2023. [Online]. Available: <https://doi.org/10.5281/zenodo.7800728>
 - [85] G. E. Moore, “Cramming more components onto integrated circuits,” *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.
 - [86] G. Hutcheson, “The economic implications of moore’s law,” *Into the nano era: moore’s law beyond planar silicon cmos*, pp. 11–38, 2009.
 - [87] P. Zheng and L. M. Ni, “Spotlight: the rise of the smart phone,” *IEEE Distributed Systems Online*, vol. 7, no. 3, pp. 3–3, 2006.
 - [88] M. Sarwar and T. R. Soomro, “Impact of smartphone’s on society,” *European journal of scientific research*, vol. 98, no. 2, pp. 216–226, 2013.
 - [89] T. N. Theis and H.-S. P. Wong, “The end of moore’s law: A new beginning for information technology,” *Computing in Science Engineering*, vol. 19, no. 2, pp. 41–50, 2017.
 - [90] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
 - [91] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
 - [92] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
 - [93] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.
 - [94] J. L. Elman, “Finding structure in time,” *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.
 - [95] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

- [96] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.
- [97] S. Ghosh, O. Vinyals, B. Strope, S. Roy, T. Dean, and L. Heck, “Contextual lstm (clstm) models for large scale nlp tasks,” *arXiv preprint arXiv:1602.06291*, 2016.
- [98] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [99] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
- [100] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [101] M. Torrisi, G. Pollastri, and Q. Le, “Deep learning methods in protein structure prediction,” *Computational and Structural Biotechnology Journal*, vol. 18, pp. 1301–1310, 2020.
- [102] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” 2014.
- [103] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.
- [104] M. K. Higgins, “Can we alphafold our way out of the next pandemic?” *Journal of Molecular Biology*, p. 167093, 2021.
- [105] D. T. Jones and J. M. Thornton, “The impact of alphafold2 one year on,” *Nature methods*, vol. 19, no. 1, pp. 15–20, 2022.
- [106] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackermann *et al.*, “A deep learning approach to antibiotic discovery,” *Cell*, vol. 180, no. 4, pp. 688–702, 2020.
- [107] A. Lohn and M. Musser, “Ai and compute: How much longer can computing power drive artificial intelligence progress,” *Center for Security and Emerging Technology*, 2022.
- [108] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, “Improving language understanding by generative pre-training,” 2018.
- [109] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

- [110] D. Amodei, D. Hernandez, G. Sastry, J. Clark, G. Brockman, and I. Sutskever, “Ai and compute,” 2018.
- [111] D. Berleant, “Trends in the cost of computing,” Jun 2022. [Online]. Available: <https://aiimpacts.org/trends-in-the-cost-of-computing/>
- [112] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2023.
- [113] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth *et al.*, “Gemini: a family of highly capable multimodal models,” *arXiv preprint arXiv:2312.11805*, 2023.
- [114] W. Knight, “Openai’s ceo says the age of giant ai models is already over,” *Wired, April*, vol. 17, p. 2023, 2023.
- [115] H. M. L. C. C. Power and D. A. I. Progress, “Ai and compute.”
- [116] J. Betker, G. Goh, L. Jing, T. Brooks, J. Wang, L. Li, L. Ouyang, J. Zhuang, J. Lee, Y. Guo *et al.*, “Improving image generation with better captions,” *Computer Science*. <https://cdn.openai.com/papers/dall-e-3.pdf>, vol. 2, no. 3, p. 8, 2023.
- [117] B. Yu, P. Lu, R. Wang, W. Hu, X. Tan, W. Ye, S. Zhang, T. Qin, and T.-Y. Liu, “Museformer: Transformer with fine-and coarse-grained attention for music generation,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 1376–1388, 2022.
- [118] H. L. Roitblat, *Algorithms are not enough: Creating general artificial intelligence*. Mit Press, 2020.
- [119] A. Roli, J. Jaeger, and S. A. Kauffman, “How organisms come to know the world: Fundamental limits on artificial general intelligence,” *Frontiers in Ecology and Evolution*, vol. 9, p. 806283, 2022.
- [120] K. Valmeekam, A. Olmo, S. Sreedharan, and S. Kambhampati, “Large language models still can’t plan (a benchmark for llms on planning and reasoning about change),” *arXiv preprint arXiv:2206.10498*, 2022.
- [121] Y. Li, “Deep reinforcement learning: An overview,” *arXiv preprint arXiv:1701.07274*, 2017.
- [122] A. R. Dooraki and D.-J. Lee, “An innovative bio-inspired flight controller for quad-rotor drones: Quad-rotor drone learning to fly using reinforcement learning,” *Robotics and Autonomous Systems*, vol. 135, p. 103671, 2021.
- [123] A. R. Fayjie, S. Hossain, D. Oualid, and D.-J. Lee, “Driverless car: Autonomous driving using deep reinforcement learning in urban environment,” in *2018 15th international conference on ubiquitous robots (ur)*. IEEE, 2018, pp. 896–901.
- [124] A. Murray, Feb 2014. [Online]. Available: <https://www.flickr.com/photos/andybadger/12622731023>

- [125] S. A. Kauffman, *Investigations*. Oxford University Press, 2000.
- [126] W. Byers, “How mathematicians think: Using ambiguity, contradiction, and paradox to create mathematics,” 2010.
- [127] J. J. Gibson, “The senses considered as perceptual systems.” 1966.
- [128] J. E. Niven and S. B. Laughlin, “Energy limitation as a selective pressure on the evolution of sensory systems,” *Journal of Experimental Biology*, vol. 211, no. 11, pp. 1792–1804, 2008.
- [129] D. Rolfe and G. C. Brown, “Cellular energy utilization and molecular origin of standard metabolic rate in mammals,” *Physiological reviews*, vol. 77, no. 3, pp. 731–758, 1997.
- [130] M. A. Holliday, “Body composition and energy needs during growth,” in *Post-natal growth neurobiology*. Springer, 1986, pp. 101–117.
- [131] S. A. Heldstab, K. Isler, S. M. Graber, C. Schuppli, and C. P. van Schaik, “The economics of brain size evolution in vertebrates,” *Current Biology*, vol. 32, no. 12, pp. R697–R708, 2022.
- [132] M. Tsuboi, A. Husby, A. Kotrschal, A. Hayward, S. D. Buechel, J. Zidar, H. Løvlie, and N. Kolm, “Comparative support for the expensive tissue hypothesis: big brains are correlated with smaller gut and greater parental investment in lake tanganyika cichlids,” *Evolution*, vol. 69, no. 1, pp. 190–200, 2015.
- [133] C. Schuppli, K. Isler, and C. P. van Schaik, “How to explain the unusually late age at skill competence among humans,” *Journal of human evolution*, vol. 63, no. 6, pp. 843–850, 2012.
- [134] L. E. Powell, K. Isler, and R. A. Barton, “Re-evaluating the link between brain size and behavioural ecology in primates,” *Proceedings of the Royal Society B: Biological Sciences*, vol. 284, no. 1865, p. 20171765, 2017.
- [135] J. T. van Woerden, C. P. van Schaik, and K. Isler, “Brief communication: Seasonality of diet composition is related to brain size in new world monkeys,” *American journal of physical anthropology*, vol. 154, no. 4, pp. 628–632, 2014.
- [136] NVIDIA, “Nvidia a100 — tensor core gpu.” [Online]. Available: <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf>
- [137] L. Chittka and J. Niven, “Are bigger brains better?” *Current biology*, vol. 19, no. 21, pp. R995–R1008, 2009.
- [138] F. Sandin, A. I. Khan, A. G. Dyer, A. H. M. Amin, G. Indiveri, E. Chicca, and E. Osipov, “Concept learning in neuromorphic vision systems: What can we learn from insects?” *Journal of Software Engineering and Applications*, vol. 7, pp. 387–395, 2014.
- [139] K. Berggren, Q. Xia, K. K. Likharev, D. B. Strukov, H. Jiang, T. Mikolajick, D. Querlioz, M. Salinga, J. R. Erickson, S. Pi, F. Xiong, P. Lin, C. Li,

- Y. Chen, S. Xiong, B. D. Hoskins, M. W. Daniels, A. Madhavan, J. A. Liddle, J. J. McClelland, Y. Yang, J. Rupp, S. S. Nonnenmann, K.-T. Cheng, N. Gong, M. A. Lastras-Montano, A. A. Talin, A. Salleo, B. J. Shastri, T. F. de Lima, P. Prucnal, A. N. Tait, Y. Shen, H. Meng, C. Roques-Carnes, Z. Cheng, H. Bhaskaran, D. Jariwala, H. Wang, J. M. Shainline, K. Segall, J. J. Yang, K. Roy, S. Datta, and A. Raychowdhury, "Roadmap on emerging hardware and technology for machine learning," *Nanotechnology*, vol. 32, no. 1, p. 012002, oct 2020. [Online]. Available: <https://dx.doi.org/10.1088/1361-6528/aba70f>
- [140] C. Mead, "How we created neuromorphic engineering," *Nature Electronics*, vol. 3, no. 7, pp. 434–435, 2020.
- [141] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, 2022.
- [142] D. V. Christensen, R. Dittmann, B. Linares-Barranco, A. Sebastian, M. Le Gallo, A. Redaelli, S. Slesazeck, T. Mikolajick, S. Spiga, S. Menzel *et al.*, "2022 roadmap on neuromorphic computing and engineering," *Neuromorphic Computing and Engineering*, vol. 2, no. 2, p. 022501, 2022.
- [143] N. Brunel and M. C. Van Rossum, "Lapicque's 1907 paper: from frogs to integrate-and-fire," *Biological cybernetics*, vol. 97, no. 5, pp. 337–339, 2007.
- [144] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952.
- [145] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.
- [146] R. Brette and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity," *Journal of neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005.
- [147] N. Mizoguchi, Y. Nagamatsu, K. Aihara, and T. Kohno, "A two-variable silicon neuron circuit based on the izevich model," *Artificial Life and Robotics*, vol. 16, pp. 383–388, 2011.
- [148] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. v. Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud *et al.*, "Neuromorphic silicon neuron circuits," *Frontiers in neuroscience*, vol. 5, p. 73, 2011.
- [149] C. Mayr, S. Höppner, and S. B. Furber, "Spinnaker 2: A 10 million core processor system for brain simulation and machine learning," *CoRR*, vol. abs/1911.02385, 2019. [Online]. Available: <http://arxiv.org/abs/1911.02385>
- [150] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

- [151] C. Pehle, S. Billaudelle, B. Cramer, J. Kaiser, K. Schreiber, Y. Stradmann, J. Weis, A. Leibfried, E. Müller, and J. Schemmel, “The brainscales-2 accelerated neuromorphic system with hybrid plasticity,” *Frontiers in Neuroscience*, vol. 16, p. 795876, 2022.
- [152] O. Richter, C. Wu, A. M. Whatley, G. Köstinger, C. Nielsen, N. Qiao, and G. Indiveri, “Dynap-se2: a scalable multi-core dynamic neuromorphic asynchronous spiking neural network processor,” *Neuromorphic Computing and Engineering*, vol. 4, no. 014003, 2024.
- [153] Q. Xia and J. J. Yang, “Memristive crossbar arrays for brain-inspired computing,” *Nature materials*, vol. 18, no. 4, pp. 309–323, 2019.
- [154] T. Dalgaty, F. Moro, Y. Demirağ, A. De Pra, G. Indiveri, E. Vianello, and M. Payvand, “Mosaic: in-memory computing and routing for small-world spike-based neuromorphic systems,” *Nature Communications*, vol. 15, no. 1, p. 142, 2024.
- [155] C.-K. Lin, A. Wild, G. N. China, T.-H. Lin, M. Davies, and H. Wang, “Mapping spiking neural networks onto a manycore neuromorphic architecture,” in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 78–89. [Online]. Available: <https://doi.org/10.1145/3192366.3192371>
- [156] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, “Advancing neuromorphic computing with loihi: A survey of results and outlook,” *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.
- [157] A. Patiño-Saucedo, H. Rostro-Gonzalez, T. Serrano-Gotarredona, and B. Linares-Barranco, “Event-driven implementation of deep spiking convolutional neural networks for supervised classification using the spinaker neuromorphic platform,” *Neural Networks*, vol. 121, pp. 319–328, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0893608019302692>
- [158] P. A. Bogdan, B. Marcinnò, C. Casellato, S. Casali, A. G. Rowley, M. Hopkins, F. Leporati, E. D’Angelo, and O. Rhodes, “Towards a bio-inspired real-time neuromorphic cerebellum,” *Frontiers in Cellular Neuroscience*, vol. 15, 2021, cited by: 8; All Open Access, Gold Open Access, Green Open Access. [Online]. Available: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85107877293&doi=10.3389%2ffncel.2021.622870&partnerID=40&md5=c763db9b2f40b514f637dc445571848d>
- [159] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, “A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps),” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 106–122, 2018.
- [160] S. Kumar, X. Wang, J. P. Strachan, Y. Yang, and W. D. Lu, “Dynamical memristors for higher-complexity neuromorphic computing,” *Nature Reviews Materials*, vol. 7, no. 7, pp. 575–591, 2022.

- [161] S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown, "Overview of the spinnaker system architecture," *IEEE transactions on computers*, vol. 62, no. 12, pp. 2454–2467, 2012.
- [162] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.
- [163] S. Schmitt, J. Klähn, G. Bellec, A. Grübl, M. Guettler, A. Hartel, S. Hartmann, D. Husmann, K. Husmann, S. Jeltsch *et al.*, "Neuromorphic hardware in the loop: Training a deep spiking network on the brainscales wafer-scale system," in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2227–2234.
- [164] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 106–122, 2018.
- [165] A. Neckar, S. Fok, B. V. Benjamin, T. C. Stewart, N. N. Oza, A. R. Voelker, C. Eliasmith, R. Manohar, and K. Boahen, "Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 144–164, 2018.
- [166] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm² 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos," *IEEE transactions on biomedical circuits and systems*, vol. 13, no. 1, pp. 145–158, 2018.
- [167] L. Deng, G. Wang, G. Li, S. Li, L. Liang, M. Zhu, Y. Wu, Z. Yang, Z. Zou, J. Pei *et al.*, "Tianjic: A unified and scalable chip bridging spike-based and continuous neural computation," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 8, pp. 2228–2246, 2020.
- [168] Y. Kuang, X. Cui, Y. Zhong, K. Liu, C. Zou, Z. Dai, Y. Wang, D. Yu, and R. Huang, "A 64k-neuron 64m-1b-synapse 2.64 pj/sop neuromorphic chip with all memory on chip for spike-based models in 65nm cmos," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 7, pp. 2655–2659, 2021.
- [169] J. Pu, W. L. Goh, V. P. Nambiar, M. M. Wong, and A. T. Do, "A 5.28-mm² 4.5-pj/sop energy-efficient spiking neural network hardware with reconfigurable high processing speed neuron core and congestion-aware router," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 12, pp. 5081–5094, 2021.
- [170] C. Frenkel and G. 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. IEEE, 2022, pp. 1–3.
- [171] J. Park, S. Ha, T. Yu, E. Neftci, and G. Cauwenberghs, "A 22-pj/spike 73-mspikes/s 130k-compartment neural array transceiver with conductance-based

- synaptic and membrane dynamics,” *Frontiers in Neuroscience*, vol. 17, p. 1198306, 2023.
- [172] M. Horowitz, “Energy table for 45nm process,” in *Stanford VLSI wiki*, 2014.
 - [173] L. Minati, “Across neurons and silicon: Some experiments regarding the pervasiveness of nonlinear phenomena,” *Acta Phys. Pol. B*, vol. 49, no. 12, pp. 2029–2094, 2018.
 - [174] J. Göltz, L. Kriener, A. Baumbach, S. Billaudelle, O. Breitwieser, B. Cramer, D. Dold, A. F. Kungl, W. Senn, J. Schemmel *et al.*, “Fast and energy-efficient neuromorphic deep learning with first-spike times,” *Nature machine intelligence*, vol. 3, no. 9, pp. 823–835, 2021.
 - [175] S. R. Kheradpisheh and T. Masquelier, “Temporal backpropagation for spiking neural networks with one spike per neuron,” *International Journal of Neural Systems*, vol. 30, no. 06, p. 2050027, 2020.
 - [176] P. J. Gerber, H. Steinfeld, B. Henderson, A. Mottet, C. Opio, J. Dijkman, A. Falcucci, G. Tempio *et al.*, *Tackling climate change through livestock: a global assessment of emissions and mitigation opportunities*. Food and Agriculture Organization of the United Nations (FAO), 2013.
 - [177] G. Myhre, D. Shindell, and J. Pongratz, “Anthropogenic and natural radiative forcing,” 2014.
 - [178] J. Martinez, P. Dabert, S. Barrington, and C. Burton, “Livestock waste treatment systems for environmental quality, food safety, and sustainability,” *Bioresource Technology*, vol. 100, no. 22, pp. 5527–5536, 2009. [Online]. Available: <http://dx.doi.org/10.1016/j.biortech.2009.02.038>
 - [179] N. N. Rabalais, R. E. Turner, and W. J. Wiseman Jr, “Gulf of mexico hypoxia, aka “the dead zone”,” *Annual Review of ecology and Systematics*, vol. 33, no. 1, pp. 235–263, 2002.
 - [180] A. H. Altieri and K. B. Gedan, “Climate change and dead zones,” *Global change biology*, vol. 21, no. 4, pp. 1395–1406, 2015.
 - [181] P. Agreement, “Unfccc, adoption of the paris agreement. cop,” *25th session Paris*, vol. 30, 2015.
 - [182] “The climate action tracker thermometer,” <https://climateactiontracker.org/global/cat-thermometer/>, accessed: 2020-06-21.
 - [183] “Ipcc special report on global warming of 1.5 °c,” <https://www.ipcc.ch/sr15/>, accessed: 2020-06-21.
 - [184] M. A. Bender, T. R. Knutson, R. E. Tuleya, J. J. Sirutis, G. A. Vecchi, S. T. Garner, and I. M. Held, “Modeled impact of anthropogenic warming on the frequency of intense atlantic hurricanes,” *Science*, vol. 327, no. 5964, pp. 454–458, 2010.
 - [185] C. Richards, R. Lupton, and J. Allwood, “Re-framing the threat of global warming: an empirical causal loop diagram of climate change, food insecurity and societal collapse,” *Climatic Change*, vol. 164, no. 3, pp. 1–19, 2021.

- [186] J. Zheng, L. Xiao, X. Fang, Z. Hao, Q. Ge, and B. Li, “How climate change impacted the collapse of the ming dynasty,” *Climatic change*, vol. 127, no. 2, pp. 169–182, 2014.
- [187] P. H. Gleick, “Water, drought, climate change, and conflict in syria,” *Weather, Climate, and Society*, vol. 6, no. 3, pp. 331–340, 2014.
- [188] “Coral reef near marine observatory 2 - okinawa institute of science and technology.” <https://www.flickr.com/photos/oistedu/33929405442>, accessed: 2020-06-22.
- [189] “Food supply - livestock and fish primary equivalent - faostat.” <http://www.fao.org/faostat/en/#data/CL/visualize>, accessed: 2020-06-22.
- [190] A. B. Milford, C. Le Mouël, B. L. Bodirsky, and S. Rolinski, “Drivers of meat consumption,” *Appetite*, vol. 141, p. 104313, 2019.
- [191] H. Westhoek, J. P. Lesschen, T. Rood, S. Wagner, A. De Marco, D. Murphy-Bokern, A. Leip, H. van Grinsven, M. A. Sutton, and O. Oenema, “Food choices, health and environment: Effects of cutting europe’s meat and dairy intake,” *Global Environmental Change*, vol. 26, pp. 196–205, 2014.
- [192] Y. Akhtar and M. Isman, “Insects as an alternative protein source,” in *Proteins in food processing*. Elsevier, 2018, pp. 263–288.
- [193] S. Smetana, M. Palanisamy, A. Mathys, and V. Heinz, “Sustainability of insect use for feed and food: Life cycle assessment perspective,” *Journal of Cleaner Production*, vol. 137, pp. 741–751, 2016.
- [194] A. Stamer, “Insect proteins—a new source for animal feed: The use of insect larvae to recycle food waste in high-quality protein for livestock and aquaculture feeds is held back largely owing to regulatory hurdles,” *EMBO reports*, vol. 16, no. 6, pp. 676–680, 2015.
- [195] G. Grossi, P. Goglio, A. Vitali, and A. G. Williams, “Livestock and climate change: impact of livestock on climate and mitigation strategies,” *Animal Frontiers*, vol. 9, no. 1, pp. 69–76, 2019.
- [196] M. M. Rojas-Downing, A. P. Nejadhashemi, T. Harrigan, and S. A. Woznicki, “Climate change and livestock: Impacts, adaptation, and mitigation,” *Climate Risk Management*, vol. 16, pp. 145–163, 2017.
- [197] D. Berckmans, “General introduction to precision livestock farming,” *Animal Frontiers*, vol. 7, no. 1, pp. 6–11, 2017.
- [198] T. M. Banhazi, H. Lehr, J. Black, H. Crabtree, P. Schofield, M. Tschärke, and D. Berckmans, “Precision livestock farming: an international review of scientific and commercial aspects,” *International Journal of Agricultural and Biological Engineering*, vol. 5, no. 3, pp. 1–9, 2012.
- [199] E. Tullo, A. Finzi, and M. Guarino, “Environmental impact of livestock farming and precision livestock farming as a mitigation strategy,” *Science of the total environment*, vol. 650, pp. 2751–2760, 2019.

- [200] L. Fadul-Pacheco, H. Delgado, and V. E. Cabrera, “Exploring machine learning algorithms for early prediction of clinical mastitis,” *International Dairy Journal*, vol. 119, p. 105051, 2021.
- [201] L. Keeling, H. Tunón, G. Olmos Antillón, C. Berg, M. Jones, L. Stuardo, J. Swanson, A. Wallenbeck, C. Winckler, and H. Blokhuis, “Animal welfare and the united nations sustainable development goals,” *Frontiers in veterinary science*, vol. 6, p. 336, 2019.
- [202] A. Dobermann, “Looking forward to 2030: nitrogen and the sustainable development goals,” in *Proceedings of the International Nitrogen Initiative Conference, Melbourne ()*, 2016.
- [203] D. Lovarelli, J. Bacenetti, and M. Guarino, “A review on dairy cattle farming: Is precision livestock farming the compromise for an environmental, economic and social sustainable production?” *Journal of Cleaner Production*, vol. 262, p. 121409, 2020.
- [204] “Coastal cattle near carmel head.” <https://www.geograph.org.uk/photo/5469672>, accessed: 2020-06-22.
- [205] N. Wagner, V. Antoine, M.-M. Mialon, R. Lardy, M. Silberberg, J. Koko, and I. Veissier, “Machine learning to detect behavioural anomalies in dairy cows under subacute ruminal acidosis,” *Computers and Electronics in Agriculture*, vol. 170, p. 105233, 2020.
- [206] M. Ebrahimi, M. Mohammadi-Dehcheshmeh, E. Ebrahimie, and K. R. Petrovski, “Comprehensive analysis of machine learning models for prediction of sub-clinical mastitis: Deep learning and gradient-boosted trees outperform other models,” *Computers in biology and medicine*, vol. 114, p. 103456, 2019.
- [207] N. O’Leary, D. Byrne, A. O’Connor, and L. Shalloo, “Invited review: Cattle lameness detection with accelerometers,” *Journal of dairy science*, vol. 103, no. 5, pp. 3895–3911, 2020.
- [208] J. Wang, M. Bell, X. Liu, and G. Liu, “Machine-learning techniques can enhance dairy cow estrus detection using location and acceleration data,” *Animals*, vol. 10, no. 7, p. 1160, 2020.
- [209] M. T. Gorczyca and K. G. Gebremedhin, “Ranking of environmental heat stressors for dairy cows using machine learning algorithms,” *Computers and Electronics in Agriculture*, vol. 168, p. 105124, 2020.
- [210] V. M. Suresh, R. Sidhu, P. Karkare, A. Patil, Z. Lei, and A. Basu, “Powering the iot through embedded machine learning and lora,” in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*. IEEE, 2018, pp. 349–354.
- [211] G. F. Marchioro, C. Cornou, A. R. Kristensen, and J. Madsen, “Sows’ activity classification device using acceleration data—a resource constrained approach,” *Computers and electronics in agriculture*, vol. 77, no. 1, pp. 110–117, 2011.
- [212] D. Gutierrez-Galan, J. P. Dominguez-Morales, E. Cerezuela-Escudero, A. Rios-Navarro, R. Tapiador-Morales, M. Rivas-Perez, M. Dominguez-Morales, A. Jimenez-Fernandez, and A. Linares-Barranco, “Embedded neural network

- for real-time animal behavior classification,” *Neurocomputing*, vol. 272, pp. 17–26, 2018.
- [213] S. Nissen *et al.*, “Implementation of a fast artificial neural network library (fann),” *Report, Department of Computer Science University of Copenhagen (DIKU)*, vol. 31, p. 29, 2003.
- [214] J. P. Dominguez-Morales, L. Duran-Lopez, D. Gutierrez-Galan, A. Rios-Navarro, A. Linares-Barranco, and A. Jimenez-Fernandez, “Wildlife monitoring on the edge: A performance evaluation of embedded neural networks on microcontrollers for animal behavior classification,” *Sensors*, vol. 21, no. 9, p. 2975, 2021.
- [215] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: <http://tensorflow.org/>
- [216] S. P. Le Roux, R. Wolluter, and T. Niesler, “Energy-Aware Feature and Model Selection for Onboard Behavior Classification in Low-Power Animal Borne Sensor Applications,” *IEEE Sensors Journal*, vol. 19, no. 7, pp. 2722–2734, 2019.
- [217] R. Arablouei, L. Currie, B. Kusy, A. Ingham, P. L. Greenwood, and G. Bishop-Hurley, “In-situ classification of cattle behavior using accelerometry data,” *Computers and Electronics in Agriculture*, vol. 183, p. 106045, 2021.
- [218] Q. Zhu, “On the performance of matthews correlation coefficient (mcc) for imbalanced dataset,” *Pattern Recognition Letters*, vol. 136, pp. 71–80, 2020.
- [219] “ice40 ultraplus family data sheet - lattice,” https://www.latticesemi.com/view_document?document_id=51968, accessed: 2020-06-18.
- [220] “Kx126-1063, tri-axis, user selectable $\pm 2g$, $4g$, $8g$, digital (i^2c /spi) output, integrated pedometer,” <https://www.kionix.com/product/KX126-1063>, accessed: 2020-06-17.
- [221] “Digital 4k video camera-recorder, actioncam - sony fdr-x3000/x3000r,” <https://www.sony.jp/actioncam/products/FDR-X3000/>, accessed: 2020-06-26.
- [222] A. Singh and A. Purohit, “A survey on methods for solving data imbalance problem for classification,” *International Journal of Computer Applications*, vol. 127, no. 15, pp. 37–41, 2015.
- [223] L. Chao, K. K. Tokgoz, J. Bartels, T. Ohashi, H. Ito, A. Okumura, K.-i. Takeda, K. Toda, and M. Hiroaki, “Data augmentation for cow behavior estimation systems based on neural network technology,” in *2020 International Workshop on Smart Info-Media System in Asia*. IEICE, 2020, pp. 95–100.

- [224] A. Page and T. Mohsenin, “Fpga-based reduction techniques for efficient deep neural network deployment,” in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2016, pp. 200–200.
- [225] M. N. Marsono, M. W. El-Kharashi, and F. Gebali, “Binary lns-based naïve bayes inference engine for spam control: noise analysis and fpga implementation,” *IET Computers & Digital Techniques*, vol. 2, no. 1, pp. 56–62, 2008.
- [226] Y. Sun, P. Li, G. Gu, Y. Wen, Y. Liu, and D. Liu, “Accelerating hmmer on fpgas using systolic array based architecture,” in *2009 Ieee International Symposium on Parallel & Distributed Processing*. IEEE, 2009, pp. 1–8.
- [227] J. A. Vázquez Diosdado, Z. E. Barker, H. R. Hodges, J. R. Amory, D. P. Croft, N. J. Bell, and E. A. Codling, “Classification of behaviour in housed dairy cows using an accelerometer-based activity monitoring system,” *Animal Biotelemetry*, vol. 3, no. 1, pp. 1–14, 2015. [Online]. Available: <http://dx.doi.org/10.1186/s40317-015-0045-8>
- [228] J. R. Struharik, “Implementing decision trees in hardware,” *SISY 2011 - 9th International Symposium on Intelligent Systems and Informatics, Proceedings*, pp. 41–46, 2011.
- [229] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.
- [230] S. Benaissa, F. A. Tuytens, D. Plets, T. de Pessemier, J. Trogh, E. Tanghe, L. Martens, L. Vandaele, A. Van Nuffel, W. Joseph, and B. Sonck, “On the use of on-cow accelerometers for the classification of behaviours in dairy barns,” *Research in Veterinary Science*, vol. 125, no. August 2017, pp. 425–433, 2019.
- [231] T. Tamura, Y. Okubo, Y. Deguchi, S. Koshikawa, M. Takahashi, Y. Chida, and K. Okada, “Dairy cattle behavior classifications based on decision tree learning using 3-axis neck-mounted accelerometers,” *Animal Science Journal*, vol. 90, no. 4, pp. 589–596, 2019.
- [232] L. A. González, G. J. Bishop-Hurley, R. N. Handcock, and C. Crossman, “Behavioral classification of data from collars containing motion sensors in grazing cattle,” *Computers and Electronics in Agriculture*, vol. 110, pp. 91–102, 2015.
- [233] S. Golestan, M. Ramezani, J. M. Guerrero, F. D. Freijedo, and M. Monfared, “Moving average filter based phase-locked loops: Performance analysis and design guidelines,” *IEEE Transactions on Power Electronics*, vol. 29, no. 6, pp. 2750–2763, 2013.
- [234] E. Tullo, A. Finzi, and M. Guarino, “Review: Environmental impact of livestock farming and Precision Livestock Farming as a mitigation strategy,” *Science of the Total Environment*, vol. 650, pp. 2751–2760, 2019. [Online]. Available: <https://doi.org/10.1016/j.scitotenv.2018.10.018>
- [235] F. Chollet, “keras,” <https://github.com/fchollet/keras>, 2015.
- [236] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *International conference on machine learning*. PMLR, 2013, pp. 1310–1318.

- [237] H. H. Tan and K. H. Lim, “Vanishing gradient mitigation with deep learning neural network optimization,” in *2019 7th International Conference on Smart Computing Communications (ICSCC)*, 2019, pp. 1–4.
- [238] R. Jozefowicz, W. Zaremba, and I. Sutskever, “An empirical exploration of recurrent network architectures,” in *International conference on machine learning*. PMLR, 2015, pp. 2342–2350.
- [239] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, “Independently recurrent neural network (indrnn): Building a longer and deeper rnn,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5457–5466.
- [240] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.
- [241] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” 2015.
- [242] M. Van Keirsbilck, A. Keller, and X. Yang, “Rethinking full connectivity in recurrent neural networks,” *arXiv preprint arXiv:1905.12340*, 2019.
- [243] F. Silfa, G. Dot, J.-M. Arnau, and A. Gonzalez, “E-pur: An energy-efficient processing unit for recurrent neural networks,” in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, 2018, pp. 1–12.
- [244] H. W. Lin, M. Tegmark, and D. Rolnick, “Why does deep and cheap learning work so well?” *Journal of Statistical Physics*, vol. 168, no. 6, pp. 1223–1247, 2017.
- [245] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, “Searching for mobilenetv3,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.
- [246] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.
- [247] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” *arXiv preprint arXiv:1611.01578*, 2016.
- [248] L. Xie and A. Yuille, “Genetic cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1379–1388.
- [249] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable architecture search,” in *International Conference on Learning Representations*, 2019. [Online]. Available: <https://openreview.net/forum?id=S1eYHoC5FX>
- [250] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, “Evaluating the search phase of neural architecture search,” *arXiv preprint arXiv:1902.08142*, 2019.

- [251] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.
- [252] “ice40 ultralite family data sheet - lattice,” https://www.latticesemi.com/view_document?document_id=50945, accessed: 2020-06-18.
- [253] “Igloo nano low power flash fpgas datasheet - microsemi,” https://www.microsemi.com/document-portal/doc_download/130695-igloo-nano-low-power-flash-fpgas-datasheet, accessed: 2020-06-18.
- [254] “Access line ultra-low-power 32-bit mcu arm®-based cortex®-m0+, up to 192kb flash, 20kb sram, 6kb eeprom, adc datasheet - stmicroelectronics,” <https://www.st.com/resource/en/datasheet/stm32l071c8.pdf>, accessed: 2020-06-18.
- [255] S. Hosseininoorbin, S. Layeghy, B. Kusy, R. Jurdak, G. J. Bishop-Hurley, P. L. Greenwood, and M. Portmann, “Deep learning-based cattle behaviour classification using joint time-frequency data representation,” *Computers and Electronics in Agriculture*, vol. 187, p. 106241, 2021.
- [256] T.-T. Wong and N.-Y. Yang, “Dependency analysis of accuracy estimates in k-fold cross validation,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, pp. 2417–2427, 2017.
- [257] S. Boughorbel, F. Jarray, and M. El-Anbari, “Optimal classifier for imbalanced data using matthews correlation coefficient metric,” *PloS one*, vol. 12, no. 6, p. e0177678, 2017.
- [258] M. A. Lones, “How to avoid machine learning pitfalls: a guide for academic researchers,” 2021.
- [259] A. P. Bradley, “The use of the area under the roc curve in the evaluation of machine learning algorithms,” *Pattern recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.
- [260] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, “Pact: Parameterized clipping activation for quantized neural networks,” *arXiv preprint arXiv:1805.06085*, 2018.
- [261] E. Sari, V. Courville, and V. P. Nia, “irn: Integer-only recurrent neural network,” *arXiv preprint arXiv:2109.09828*, 2021.
- [262] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.
- [263] “Tensorflow for mobile & iot - model optimization,” https://www.tensorflow.org/lite/performance/model_optimization, accessed: 2020-06-26.
- [264] “Tsubame 3.0 computing services top page,” <https://www.t3.gsic.titech.ac.jp/index.php/en>, accessed: 2020-06-26.

- [265] Y. Blaise, A. Andriamandroso, B. Heinesch, Y. Beckers, E. Castro Muñoz, F. Lebeau, and J. Bindelle, “Influences of feeding behaviour and forage quality on diurnal methane emission dynamics of grazing cows,” *Precision Livestock Farming '17*, pp. 759–769, 2017.
- [266] J. Wilkinson and P. Garnsworthy, “Impact of diet and fertility on greenhouse gas emissions and nitrogen efficiency of milk production,” *Livestock*, vol. 22, no. 3, pp. 140–144, 2017.
- [267] J. Bartels, K. K. Tokgoz, S. A, M. Fukawa, S. Otsubo, C. Li, I. Rachi, K.-I. Takeda, L. Minati, and H. Ito, “Tynecownet: Memory- and power-minimized rnns implementable on tiny edge devices for lifelong cow behavior distribution estimation,” *IEEE Access*, vol. 10, pp. 32 706–32 727, 2022.
- [268] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.
- [269] A. Shawahna, S. M. Sait, and A. El-Maleh, “Fpga-based accelerators of deep learning networks for learning and classification: A review,” *ieee Access*, vol. 7, pp. 7823–7859, 2018.
- [270] H. Ito, K. ichi Takeda, K. K. Tokgoz, L. Minati, M. Fukawa, L. Chao, J. Bartels, I. Rachi, and S. A, “Japanese Black Beef Cow Behavior Classification Dataset,” Sep. 2021. [Online]. Available: <https://doi.org/10.5281/zenodo.5399259>
- [271] D. Neil, J. H. Lee, T. Delbruck, and S.-C. Liu, “Delta networks for optimized recurrent network computation,” in *International conference on machine learning*. PMLR, 2017, pp. 2584–2593.
- [272] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck, “Deltarnn: A power-efficient recurrent neural network accelerator,” in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 21–30.
- [273] C. Gao, T. Delbruck, and S.-C. Liu, “Spartus: A 9.4 top/s fpga-based lstm accelerator exploiting spatio-temporal sparsity,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022.
- [274] A. Pikovsky, M. Rosenblum, J. Kurths, and A. Synchronization, “A universal concept in nonlinear sciences,” *Self*, vol. 2, p. 3, 2001.
- [275] S. H. Strogatz, *Sync: How order emerges from chaos in the universe, nature, and daily life*. Hachette UK, 2012.
- [276] A. Arenas, A. Díaz-Guilera, J. Kurths, Y. Moreno, and C. Zhou, “Synchronization in complex networks,” *Physics reports*, vol. 469, no. 3, pp. 93–153, 2008.
- [277] Y. Tang, F. Qian, H. Gao, and J. Kurths, “Synchronization in complex networks and its application—a survey of recent advances and challenges,” *Annual Reviews in Control*, vol. 38, no. 2, pp. 184–198, 2014.

- [278] F. Dörfler and F. Bullo, “Synchronization in complex networks of phase oscillators: A survey,” *Automatica*, vol. 50, no. 6, pp. 1539–1564, 2014.
- [279] D. Ghosh, M. Frasca, A. Rizzo, S. Majhi, S. Rakshit, K. Alfaro-Bittner, and S. Boccaletti, “The synchronized dynamics of time-varying networks,” *Physics Reports*, vol. 949, pp. 1–63, 2022.
- [280] T. Kapitaniak, L. O. Chua, and G.-Q. Zhong, “Experimental synchronization of chaos using continuous control,” *International Journal of Bifurcation and Chaos*, vol. 4, no. 02, pp. 483–488, 1994.
- [281] R. Roy and K. S. Thornburg Jr, “Experimental synchronization of chaotic lasers,” *Physical Review Letters*, vol. 72, no. 13, p. 2009, 1994.
- [282] M. Bennett, M. F. Schatz, H. Rockwood, and K. Wiesenfeld, “Huygens’s clocks,” *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 458, no. 2019, pp. 563–579, 2002.
- [283] M. C. Chiang, H.-F. Chen, and J.-M. Liu, “Experimental synchronization of mutually coupled semiconductor lasers with optoelectronic feedback,” *IEEE journal of quantum electronics*, vol. 41, no. 11, pp. 1333–1340, 2005.
- [284] R. Meucci, F. Salvadori, M. V. Ivanchenko, K. Al Naimee, C. Zhou, F. T. Arecchi, S. Boccaletti, and J. Kurths, “Synchronization of spontaneous bursting in a co 2 laser,” *Physical review E*, vol. 74, no. 6, p. 066207, 2006.
- [285] A. Buscarino, L. Fortuna, and M. Frasca, “Experimental robust synchronization of hyperchaotic circuits,” *Physica D: Nonlinear Phenomena*, vol. 238, no. 18, pp. 1917–1922, 2009.
- [286] D. Rijlaarsdam, A. Y. Pogromsky, and H. Nijmeijer, “Synchronization between coupled oscillators: an experimental approach,” in *Dynamics And Control Of Hybrid Mechanical Systems*. World Scientific, 2010, pp. 153–165.
- [287] F. Hoozeboom, A. Pogromsky, and H. Nijmeijer, “Huygens’ synchronization: Experiments, modeling, and local stability analysis,” *IFAC-PapersOnLine*, vol. 48, no. 18, pp. 146–151, 2015.
- [288] J. Peña Ramirez, L. A. Olvera, H. Nijmeijer, and J. Alvarez, “The sympathy of two pendulum clocks: beyond huygens’ observations,” *Scientific reports*, vol. 6, no. 1, p. 23580, 2016.
- [289] J. Pantaleone, “Synchronization of metronomes,” *American Journal of Physics*, vol. 70, no. 10, pp. 992–1000, 2002.
- [290] E. A. Martens, S. Thutupalli, A. Fourrière, and O. Hallatschek, “Chimera states in mechanical oscillator networks,” *Proceedings of the National Academy of Sciences*, vol. 110, no. 26, pp. 10 563–10 567, 2013.
- [291] S. Chhabria, K. A. Blaha, F. Della Rossa, and F. Sorrentino, “Targeted synchronization in an externally driven population of mechanical oscillators,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 28, no. 11, 2018.

- [292] L. M. Pecora, F. Sorrentino, A. M. Hagerstrom, T. E. Murphy, and R. Roy, “Cluster synchronization and isolated desynchronization in complex networks with symmetries,” *Nature communications*, vol. 5, no. 1, p. 4079, 2014.
- [293] S. Mahler, A. A. Friesem, and N. Davidson, “Experimental demonstration of crowd synchrony and first-order transition with lasers,” *Physical Review Research*, vol. 2, no. 4, p. 043220, 2020.
- [294] M. De Magistris, M. Di Bernardo, E. Di Tucci, and S. Manfredi, “Synchronization of networks of non-identical chua’s circuits: Analysis and experiments,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 5, pp. 1029–1041, 2012.
- [295] I. Leyva, R. Sevilla-Escoboza, J. Buldú, I. Sendina-Nadal, J. Gómez-Gardenes, A. Arenas, Y. Moreno, S. Gomez, R. Jaimes-Reategui, and S. Boccaletti, “Explosive first-order transition to synchrony in networked chaotic oscillators,” *Physical review letters*, vol. 108, no. 16, p. 168702, 2012.
- [296] L. Minati, “Experimental dynamical characterization of five autonomous chaotic oscillators with tunable series resistance,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 24, no. 3, 2014.
- [297] —, “Remote synchronization of amplitudes across an experimental ring of non-linear oscillators,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 25, no. 12, 2015.
- [298] M. de Magistris, M. di Bernardo, S. Manfredi, C. Petrarca, and S. Yaghouti, “Modular experimental setup for real-time analysis of emergent behavior in networks of chua’s circuits,” *International journal of circuit theory and applications*, vol. 44, no. 8, pp. 1551–1571, 2016.
- [299] R. Sevilla-Escoboza, J. Buldú, S. Boccaletti, D. Papo, D.-U. Hwang, G. Huerta-Cuellar, and R. Gutierrez, “Experimental implementation of maximally synchronizable networks,” *Physica A: Statistical Mechanics and Its Applications*, vol. 448, pp. 113–121, 2016.
- [300] R. Sevilla-Escoboza, I. Sendina-Nadal, I. Leyva, R. Gutiérrez, J. Buldú, and S. Boccaletti, “Inter-layer synchronization in multiplex networks of identical layers,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 26, no. 6, 2016.
- [301] L. Gambuzza, L. Minati, and M. Frasca, “Experimental observations of chimera states in locally and non-locally coupled stuart-landau oscillator circuits,” *Chaos, Solitons & Fractals*, vol. 138, p. 109907, 2020.
- [302] Á. Tar, G. Gandhi, and G. Cserey, “Hardware implementation of cnn architecture-based test bed for studying synchronization phenomenon in oscillatory and chaotic networks,” *International Journal of Circuit Theory and Applications*, vol. 37, no. 4, pp. 529–542, 2009.
- [303] G. Indiveri, B. Linares-Barranco, R. Legenstein, G. Deligeorgis, and T. Pro-dromakis, “Integration of nanoscale memristor synapses in neuromorphic computing architectures,” *Nanotechnology*, vol. 24, no. 38, p. 384010, 2013.

- [304] O. Richter, C. Wu, A. M. Whatley, G. Köstinger, C. Nielsen, N. Qiao, and G. Indiveri, “DYNAP-SE2: a scalable multi-core dynamic neuromorphic asynchronous spiking neural network processor,” *Neuromorphic Computing and Engineering*, vol. 4, no. 1, p. 014003, Jan. 2024. [Online]. Available: <https://doi.org/10.1088/2634-4386/ad1cd7>
- [305] D. A. Burbano-L, S. Yaghouti, C. Petrarca, M. De Magistris, and M. Di Bernardo, “Synchronization in multiplex networks of chua’s circuits: Theory and experiments,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 3, pp. 927–938, 2019.
- [306] L. V. Gambuzza, A. Buscarino, S. Chessari, L. Fortuna, R. Meucci, and M. Frasca, “Experimental investigation of chimera states with quiescent and synchronous domains in coupled electronic oscillators,” *Physical Review E*, vol. 90, no. 3, p. 032905, 2014.
- [307] A. Buscarino, L. Fortuna, and M. Frasca, *Essentials of Nonlinear Circuit Dynamics with MATLAB® and Laboratory Experiments*, 04 2017.
- [308] A. Bergner, M. Frasca, G. Sciuto, A. Buscarino, E. J. Ngamga, L. Fortuna, and J. Kurths, “Remote synchronization in star networks,” *Phys. Rev. E*, vol. 85, p. 026208, Feb 2012. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevE.85.026208>
- [309] B. Karakaya, L. Minati, L. V. Gambuzza, and M. Frasca, “Fading of remote synchronization in tree networks of stuart-landau oscillators,” *Physical Review E*, vol. 99, no. 5, p. 052301, 2019.
- [310] L. Minati, L. Faes, M. Frasca, P. Oświcimka, and S. Drożdż, “Apparent remote synchronization of amplitudes: A demodulation and interference effect,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 28, no. 6, p. 063124, 06 2018. [Online]. Available: <https://doi.org/10.1063/1.5026980>
- [311] J. Lenhard, “Computer simulation: The cooperation between experimenting and modeling,” *Philosophy of Science*, vol. 74, no. 2, p. 176–194, 2007.
- [312] S. Boccaletti, J. Kurths, G. Osipov, D. Valladares, and C. Zhou, “The synchronization of chaotic systems,” *Physics Reports*, vol. 366, no. 1, pp. 1–101, 2002. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0370157302001370>
- [313] S. Guan, Y.-C. Lai, and C.-H. Lai, “Effect of noise on generalized chaotic synchronization,” *Phys. Rev. E*, vol. 73, p. 046210, Apr 2006. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevE.73.046210>
- [314] N. Tabareau, J.-J. Slotine, and Q.-C. Pham, “How synchronization protects from noise,” *PLOS Computational Biology*, vol. 6, no. 1, pp. 1–9, 01 2010. [Online]. Available: <https://doi.org/10.1371/journal.pcbi.1000637>
- [315] J. H. Meng and H. Riecke, “Synchronization by uncorrelated noise: interacting rhythms in interconnected oscillator networks,” *Scientific reports*, vol. 8, no. 1, p. 6949, 2018.

- [316] Z. Budrikis, “Forty years of stochastic resonance,” *Nature Reviews Physics*, vol. 3, no. 12, pp. 771–771, 2021.
- [317] P. E. Kloeden and E. Platen, *Stochastic Differential Equations*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 103–160. [Online]. Available: https://doi.org/10.1007/978-3-662-12616-5_4
- [318] A. Buscarino, L. Fortuna, M. Frasca, M. Iachello, and V.-T. Pham, “Robustness to noise in synchronization of network motifs: Experimental results,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 4, p. 043106, 10 2012. [Online]. Available: <https://doi.org/10.1063/1.4761962>
- [319] A. Buscarino, L. V. Gambuzza, M. Porfiri, L. Fortuna, and M. Frasca, “Robustness to noise in synchronization of complex networks,” *Scientific reports*, vol. 3, no. 1, p. 2026, 2013.
- [320] L. V. Gambuzza, A. Buscarino, L. Fortuna, M. Porfiri, and M. Frasca, “Analysis of dynamical robustness to noise in power grids,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 7, no. 3, pp. 413–421, 2017.
- [321] M. Porfiri and M. Frasca, “Robustness of synchronization to additive noise: How vulnerability depends on dynamics,” *IEEE Transactions on Control of Network Systems*, vol. 6, no. 1, pp. 375–387, 2019.
- [322] L. Minati, “Experimental synchronization of chaos in a large ring of mutually coupled single-transistor oscillators: Phase, amplitude, and clustering effects,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 24, no. 4, p. 043108, 10 2014. [Online]. Available: <https://doi.org/10.1063/1.4896815>
- [323] L. Minati, M. Frasca, P. Oświcimka, L. Faes, and S. Drożdż, “Atypical transistor-based chaotic oscillators: Design, realization, and diversity,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 7, p. 073113, 07 2017. [Online]. Available: <https://doi.org/10.1063/1.4994815>
- [324] L. Minati, G. Innocenti, G. Mijatovic, H. Ito, and M. Frasca, “Mechanisms of chaos generation in an atypical single-transistor oscillator,” *Chaos, Solitons Fractals*, vol. 157, p. 111878, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0960077922000893>
- [325] D. J. Gauthier, E. Bollt, A. Griffith, and W. A. Barbosa, “Next generation reservoir computing,” *Nature communications*, vol. 12, no. 1, pp. 1–8, 2021.
- [326] W. Maass, T. Natschläger, and H. Markram, “Real-time computing without stable states: A new framework for neural computation based on perturbations,” *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.
- [327] M. Lukoševičius and H. Jaeger, “Reservoir computing approaches to recurrent neural network training,” *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1574013709000173>

- [328] P. Vlachas, J. Pathak, B. Hunt, T. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos, “Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics,” *Neural Networks*, vol. 126, pp. 191–217, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0893608020300708>
- [329] E. Bollt, “On explaining the surprising success of reservoir computing forecaster of chaos? The universal machine learning dynamical system with contrast to VAR and DMD,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 31, no. 1, p. 013108, 01 2021. [Online]. Available: <https://doi.org/10.1063/5.0024890>
- [330] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose, “Recent advances in physical reservoir computing: A review,” *Neural Networks*, vol. 115, pp. 100–123, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0893608019300784>
- [331] C. Fernando and S. Sojakka, “Pattern recognition in a bucket,” in *Advances in Artificial Life*, W. Banzhaf, J. Ziegler, T. Christaller, P. Dittrich, and J. T. Kim, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 588–597.
- [332] B. Jones, D. Stekel, J. Rowe, and C. Fernando, “Is there a liquid state machine in the bacterium escherichia coli?” in *2007 IEEE Symposium on Artificial Life*, 2007, pp. 187–191.
- [333] *Reservoir computing with a chaotic circuit*, ser. ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference, vol. ECAL 2017, the Fourteenth European Conference on Artificial Life, 09 2017. [Online]. Available: https://doi.org/10.1162/isal_a_039
- [334] *Introduction*, ch. Chapter 1, pp. 1–39. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/9789811240003_0001
- [335] A. Strehl and J. Ghosh, “Cluster ensembles—a knowledge reuse framework for combining multiple partitions,” *Journal of machine learning research*, vol. 3, no. Dec, pp. 583–617, 2002.
- [336] MINATI, BARTELS, LI, FRASCA, and ITO, “Synchronization phenomena in dual-transistor spiking oscillators realized experimentally towards physical reservoirs,” Aug. 2022. [Online]. Available: <https://doi.org/10.5281/zenodo.6800553>
- [337] B. J. Grzyb, E. Chinellato, G. M. Wojcik, and W. A. Kaminski, “Which model to use for the liquid state machine?” in *2009 International Joint Conference on Neural Networks*, 2009, pp. 1018–1024.
- [338] N. Bertschinger and T. Natschläger, “Real-time computation at the edge of chaos in recurrent neural networks,” *Neural Computation*, vol. 16, no. 7, pp. 1413–1436, 2004.
- [339] C. G. Langton, “Computation at the edge of chaos: Phase transitions and emergent computation,” *Physica D: Nonlinear Phenomena*, vol. 42, no. 1,

- pp. 12–37, 1990. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/016727899090064V>
- [340] J. M. Beggs, “The criticality hypothesis: how local cortical networks might optimize information processing,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1864, pp. 329–343, 2008. [Online]. Available: <https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2007.2092>
 - [341] M. A. Muñoz, “Colloquium: Criticality and dynamical scaling in living systems,” *Rev. Mod. Phys.*, vol. 90, p. 031001, Jul 2018. [Online]. Available: <https://link.aps.org/doi/10.1103/RevModPhys.90.031001>
 - [342] T. Mora and W. Bialek, “Are biological systems poised at criticality?” *Journal of Statistical Physics*, vol. 144, pp. 268–302, 2011.
 - [343] V. Ivanov and K. Michmizos, “Increasing liquid state machine performance with edge-of-chaos dynamics organized by astrocyte-modulated plasticity,” in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 25 703–25 719. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/d79c8788088c2193f0244d8f1f36d2db-Paper.pdf
 - [344] R. Legenstein and W. Maass, “Edge of chaos and prediction of computational performance for neural circuit models,” *Neural Networks*, vol. 20, no. 3, pp. 323–334, 2007, echo State Networks and Liquid State Machines. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0893608007000433>
 - [345] T. L. Carroll, “Do reservoir computers work best at the edge of chaos?” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 30, no. 12, p. 121109, 12 2020. [Online]. Available: <https://doi.org/10.1063/5.0038163>
 - [346] E. Izhikevich, “Which model to use for cortical spiking neurons?” *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063–1070, 2004.
 - [347] Y. Hirata and K. Aihara, “Typical chaotic dynamics in squid giant axons,” *Procedia IUTAM*, vol. 5, pp. 54–58, 2012, iUTAM Symposium on 50 Years of Chaos: Applied and Theoretical. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2210983812000685>
 - [348] J. G. Freire, R. Meucci, F. T. Arecchi, and J. A. C. Gallas, “Self-organization of pulsing and bursting in a CO₂ laser with opto-electronic feedback,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 25, no. 9, p. 097607, 04 2015. [Online]. Available: <https://doi.org/10.1063/1.4916923>
 - [349] L. Minati, M. Mancinelli, M. Frasca, P. Bettotti, and L. Pavesi, “An analog electronic emulator of non-linear dynamics in optical microring resonators,” *Chaos, Solitons Fractals*, vol. 153, p. 111410, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0960077921007645>
 - [350] R. C. Hilborn, *Chaos and nonlinear dynamics: an introduction for scientists and engineers*. Oxford university press, 2000.

- [351] E. Ott, *Chaos in dynamical systems*. Cambridge university press, 2002.
- [352] R. C. Elson, A. I. Selverston, R. Huerta, N. F. Rulkov, M. I. Rabinovich, and H. D. I. Abarbanel, “Synchronous behavior of two coupled biological neurons,” *Phys. Rev. Lett.*, vol. 81, pp. 5692–5695, Dec 1998. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevLett.81.5692>
- [353] G. Indiveri, B. Linares-Barranco, T. J. Hamilton, A. van Schaik, R. Etienne-Cummings, T. Delbruck, S.-C. Liu, P. Dudek, P. Häfliger, S. Renaud, J. Schemmel, G. Cauwenberghs, J. Arthur, K. Hynna, F. Folowosele, S. SAÏGHI, T. Serrano-Gotarredona, J. Wijekoon, Y. Wang, and K. Boahen, “Neuromorphic silicon neuron circuits,” *Frontiers in Neuroscience*, vol. 5, 2011. [Online]. Available: <https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2011.00073>
- [354] R. J. Jones, P. Rees, P. S. Spencer, and K. A. Shore, “Chaos and synchronization of self-pulsating laser diodes,” *J. Opt. Soc. Am. B*, vol. 18, no. 2, pp. 166–172, Feb 2001. [Online]. Available: <https://opg.optica.org/josab/abstract.cfm?URI=josab-18-2-166>
- [355] M. E. Hasselmo, “Neuronal rebound spiking, resonance frequency and theta cycle skipping may contribute to grid cell firing in medial entorhinal cortex,” *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 369, no. 1635, p. 20120523, 2014. [Online]. Available: <https://royalsocietypublishing.org/doi/abs/10.1098/rstb.2012.0523>
- [356] F. Darki and J. Rankin, “Methods to assess binocular rivalry with periodic stimuli,” *The Journal of Mathematical Neuroscience*, vol. 10, pp. 1–21, 2020.
- [357] S. Ostojic, “Interspike interval distributions of spiking neurons driven by fluctuating inputs,” *Journal of Neurophysiology*, vol. 106, no. 1, pp. 361–373, 2011, pMID: 21525364. [Online]. Available: <https://doi.org/10.1152/jn.00830.2010>
- [358] E. Williams, A. Payeur, A. Gidon, and R. Naud, “Neural burst codes disguised as rate codes,” *Scientific Reports*, vol. 11, no. 1, p. 15910, 2021.
- [359] A. A. Velichko, D. V. Ryabokon, S. D. Khanin, A. V. Sidorenko, and A. G. Rikkiev, “Reservoir computing using high order synchronization of coupled oscillators,” *IOP Conference Series: Materials Science and Engineering*, vol. 862, no. 5, p. 052062, may 2020. [Online]. Available: <https://dx.doi.org/10.1088/1757-899X/862/5/052062>
- [360] *Reservoir computing with self-organizing neural oscillators*, ser. ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference, vol. ALIFE 2021: The 2021 Conference on Artificial Life, 07 2021. [Online]. Available: https://doi.org/10.1162/isal_a_00409
- [361] T. Yamane, Y. Katayama, R. Nakane, G. Tanaka, and D. Nakano, “Wave-based reservoir computing by synchronization of coupled oscillators,” in *Neural Information Processing*, S. Arik, T. Huang, W. K. Lai, and Q. Liu, Eds. Cham: Springer International Publishing, 2015, pp. 198–205.

- [362] T. Sauer, “Interspike interval embedding of chaotic signals,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 5, no. 1, pp. 127–132, 03 1995. [Online]. Available: <https://doi.org/10.1063/1.166094>
- [363] H. Kantz and T. Schreiber, *Nonlinear time series analysis*. Cambridge university press, 2004, vol. 7.
- [364] Y. K. Mohammad, O. N. Pavlova, and A. N. Pavlov, “Estimating the predictability time of noisy chaotic dynamics from point sequences,” *Technical Physics Letters*, vol. 43, pp. 107–109, 2017.
- [365] M. Kramer, “Autoassociative neural networks,” *Computers Chemical Engineering*, vol. 16, no. 4, pp. 313–328, 1992, neural network applications in chemical engineering. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/009813549280051A>
- [366] C. Bandt and B. Pompe, “Permutation entropy: A natural complexity measure for time series,” *Phys. Rev. Lett.*, vol. 88, p. 174102, Apr 2002. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevLett.88.174102>
- [367] M. Riedl, A. Müller, and N. Wessel, “Practical considerations of permutation entropy: A tutorial review,” *The European Physical Journal Special Topics*, vol. 222, no. 2, pp. 249–262, 2013.
- [368] U. Sychou, “A single-node classifier implementation on chua oscillator within a physical reservoir computing framework,” *International Journal of Bifurcation and Chaos*, vol. 31, no. 11, p. 2150161, 2021. [Online]. Available: <https://doi.org/10.1142/S0218127421501613>
- [369] V. Siderskiy and V. Kapila, “Parameter matching using adaptive synchronization of two chua’s oscillators,” *International Journal of Bifurcation and Chaos*, vol. 24, no. 11, p. 1430032, 2014. [Online]. Available: <https://doi.org/10.1142/S0218127414300328>
- [370] M. A. Aoun and M. Boukadoum, “Chaotic liquid state machine,” *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)*, vol. 9, no. 4, pp. 1–20, 2015.
- [371] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, oct 2008. [Online]. Available: <https://dx.doi.org/10.1088/1742-5468/2008/10/P10008>
- [372] G. V. Osipov, A. S. Pikovsky, M. G. Rosenblum, and J. Kurths, “Phase synchronization effects in a lattice of nonidentical rössler oscillators,” *Phys. Rev. E*, vol. 55, pp. 2353–2361, Mar 1997. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevE.55.2353>
- [373] I. Belykh, V. Belykh, K. Nevidin, and M. Hasler, “Persistent clusters in lattices of coupled nonidentical chaotic systems,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 13, no. 1, pp. 165–178, 03 2003. [Online]. Available: <https://doi.org/10.1063/1.1514202>

- [374] R. ROGISTER, F; ROY, “Spatiotemporal chaos synchronization in a large array of locally coupled nonidentical lasers,” 2005.
- [375] L. V. Gambuzza, A. Cardillo, A. Fiasconaro, L. Fortuna, J. Gómez-Gardeñes, and M. Frasca, “Analysis of remote synchronization in complex networks,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 23, no. 4, p. 043103, 10 2013. [Online]. Available: <https://doi.org/10.1063/1.4824312>
- [376] C. Gallicchio, A. Micheli, and L. Pedrelli, “Deep reservoir computing: A critical experimental analysis,” *Neurocomputing*, vol. 268, pp. 87–99, 2017, advances in artificial neural networks, machine learning and computational intelligence. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0925231217307567>
- [377] D.-Z. Yue, Z.-M. Wu, Y.-S. Hou, C.-X. Hu, Z.-F. Jiang, and G.-Q. Xia, “Reservoir computing based on two parallel reservoirs under identical electrical message injection,” *IEEE Photonics Journal*, vol. 13, no. 1, pp. 1–11, 2021.
- [378] P. Wijesinghe, G. Srinivasan, P. Panda, and K. Roy, “Analysis of liquid ensembles for enhancing the performance and accuracy of liquid state machines,” *Frontiers in Neuroscience*, vol. 13, 2019. [Online]. Available: <https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2019.00504>
- [379] M. Takao and O. Hitoshi, “Effect of a shortcut link supplemented to ring networks,” in *IEEE International Conference on Communications, 2003. ICC '03.*, vol. 2, 2003, pp. 1330–1334 vol.2.
- [380] L. Tang, J.-a. Lu, and G. Chen, “Synchronizability of small-world networks generated from ring networks with equal-distance edge additions,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 2, p. 023121, 05 2012. [Online]. Available: <https://doi.org/10.1063/1.4711008>
- [381] C. Gallicchio and A. Micheli, “Ring reservoir neural networks for graphs,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7.
- [382] L. Minati, P. Chiesa, D. Tabarelli, L. D’Incerti, and J. Jovicich, “Synchronization, non-linear dynamics and low-frequency fluctuations: Analogy between spontaneous brain activity and networked single-transistor chaotic oscillators,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 25, no. 3, p. 033107, 03 2015. [Online]. Available: <https://doi.org/10.1063/1.4914938>
- [383] T. C. Potjans and M. Diesmann, “The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model,” *Cerebral Cortex*, vol. 24, no. 3, pp. 785–806, 12 2012. [Online]. Available: <https://doi.org/10.1093/cercor/bhs358>
- [384] J. J. J. Hjorth, A. Kozlov, I. Carannante, J. F. Nylén, R. Lindroos, Y. Johansson, A. Tokarska, M. C. Dorst, S. M. Suryanarayana, G. Silberberg, J. H. Kotaleski, and S. Grillner, “The microcircuits of striatum in silico,” *Proceedings of the National Academy of Sciences*, vol. 117, no. 17, pp. 9554–9565, 2020. [Online]. Available: <https://www.pnas.org/doi/abs/10.1073/pnas.2000671117>

- [385] K. Vandoorne, J. Dambre, D. Verstraeten, B. Schrauwen, and P. Bienstman, "Parallel reservoir computing using optical amplifiers," *IEEE Transactions on Neural Networks*, vol. 22, no. 9, pp. 1469–1481, 2011.
- [386] J. A. White, C. C. Chow, J. Rit, C. Soto-Treviño, and N. Kopell, "Synchronization and oscillatory dynamics in heterogeneous, mutually inhibited neurons," *Journal of computational neuroscience*, vol. 5, pp. 5–16, 1998.
- [387] S. Venkadesh, A. O. Komendantov, D. W. Wheeler, D. J. Hamilton, and G. A. Ascoli, "Simple models of quantitative firing phenotypes in hippocampal neurons: Comprehensive coverage of intrinsic diversity," *PLOS Computational Biology*, vol. 15, no. 10, pp. 1–27, 10 2019. [Online]. Available: <https://doi.org/10.1371/journal.pcbi.1007462>
- [388] E. Panteley, A. Loría, and L. Contevelle, "On practical synchronization of heterogeneous networks of nonlinear systems: application to chaotic systems," in *2015 American Control Conference (ACC)*, 2015, pp. 5359–5364.
- [389] D. Wang, W. Chen, and L. Qiu, "Synchronization of heterogeneous dynamical networks via phase analysis**this work was supported in parts by the research grants council of hong kong special administrative region, china, under the theme-based research scheme t23-701/14-n, and schneider electric, lenovo group (china) limited and hong kong innovation and technology fund (its/066/17fp) under the hkust-mit research alliance consortium." *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 3013–3018, 2020, 21st IFAC World Congress. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2405896320313495>
- [390] A. Bergner, M. Frasca, G. Sciuto, A. Buscarino, E. J. Ngamga, L. Fortuna, and J. Kurths, "Remote synchronization in star networks," *Phys. Rev. E*, vol. 85, p. 026208, Feb 2012. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevE.85.026208>
- [391] G. Tanaka, R. Nakane, T. Yamane, D. Nakano, S. Takeda, S. Nakagawa, and A. Hirose, "Exploiting heterogeneous units for reservoir computing with simple architecture," in *Neural Information Processing*, A. Hirose, S. Ozawa, K. Doya, K. Ikeda, M. Lee, and D. Liu, Eds. Cham: Springer International Publishing, 2016, pp. 187–194.
- [392] D. Snyder, A. Goudarzi, and C. Teuscher, "Computational capabilities of random automata networks for reservoir computing," *Phys. Rev. E*, vol. 87, p. 042808, Apr 2013. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevE.87.042808>
- [393] N. J. Von, "Non-linear capacitance or inductance switching, amplifying, and memory organs," Dec. 3 1957, uS Patent 2,815,488.
- [394] E. Goto, "The parametron, a digital computing element which utilizes parametric oscillation," *Proceedings of the IRE*, vol. 47, no. 8, pp. 1304–1316, 1959.
- [395] S. Muroga and K. Takashima, "The parametron digital computer musasino-1," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 308–316, 1959.

- [396] A. Raychowdhury, A. Parihar, G. H. Smith, V. Narayanan, G. Csaba, M. Jerry, W. Porod, and S. Datta, "Computing with networks of oscillatory dynamical systems," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 73–89, 2019.
- [397] D. E. Nikonov, I. A. Young, and G. I. Bourianoff, "Convolutional networks for image processing by coupled oscillator arrays," 2014.
- [398] G. Csaba, T. Ytterdal, and W. Porod, "Oscillatory neural network from ring oscillators," in *CNNA 2016; 15th International Workshop on Cellular Nanoscale Networks and their Applications*, 2016, pp. 1–2.
- [399] P. Kaluza and T. Cioacă, "Phase oscillator neural network as artificial central pattern generator for robots," *Neurocomputing*, vol. 97, pp. 115–124, 2012. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S092523121200478X>
- [400] G. Csaba and W. Porod, "Coupled oscillators for computing: A review and perspective," *Applied Physics Reviews*, vol. 7, no. 1, p. 011302, 01 2020. [Online]. Available: <https://doi.org/10.1063/1.5120412>
- [401] N. Shukla, W.-Y. Tsai, M. Jerry, M. Barth, V. Narayanan, and S. Datta, "Ultra low power coupled oscillator arrays for computer vision applications," in *2016 IEEE Symposium on VLSI Technology*, 2016, pp. 1–2.
- [402] D. E. Nikonov, G. Csaba, W. Porod, T. Shibata, D. Voils, D. Hammerstrom, I. A. Young, and G. I. Bourianoff, "Coupled-oscillator associative memory array operation for pattern recognition," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 1, pp. 85–93, 2015.
- [403] C. Delacour, S. Carapezzi, M. Abernot, G. Boschetto, N. Azemard, J. Salles, T. Gil, and A. Todri-Sanial, "Oscillatory neural networks for edge ai computing," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2021, pp. 326–331.
- [404] X. He, T. Liu, F. Hadaeghi, and H. Jaeger, "Reservoir transfer on analog neuromorphic hardware," in *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*, 2019, pp. 1234–1238.
- [405] G. Dion, S. Mejaouri, and J. Sylvestre, "Reservoir computing with a single delay-coupled non-linear mechanical oscillator," *Journal of Applied Physics*, vol. 124, no. 15, p. 152132, 10 2018. [Online]. Available: <https://doi.org/10.1063/1.5038038>
- [406] W. J. Goh and N. T. Crook, "Pattern recognition using chaotic transients." in *ESANN*, 2007, pp. 7–12.
- [407] M. Breakspear, "Dynamic models of large-scale brain activity," *Nature neuroscience*, vol. 20, no. 3, pp. 340–352, 2017.
- [408] K. Lehnertz, S. Bialonski, M.-T. Horstmann, D. Krug, A. Rothkegel, M. Staniek, and T. Wagner, "Synchronization phenomena in human epileptic brain networks," *Journal of neuroscience methods*, vol. 183, no. 1, pp. 42–48, 2009.

- [409] V. Jirsa, H. Wang, P. Triebkorn, M. Hashemi, J. Jha, J. Gonzalez-Martinez, M. Guye, J. Makhalova, and F. Bartolomei, “Personalised virtual brain models in epilepsy,” *The Lancet Neurology*, vol. 22, no. 5, pp. 443–454, 2023.
- [410] WHO, *Epilepsy: a public health imperative*. World Health Organization, 2019.
- [411] B. O. Olusanya, S. M. Wright, M. Nair, N.-Y. Boo, R. Halpern, H. Kuper, A. A. Abubakar, N. A. Almasri, J. Arabloo, N. K. Arora, S. Backhaus, B. D. Berman, C. Breinbauer, G. Carr, P. J. de Vries, C. del Castillo-Hegy, A. Eftekhari, M. J. Gladstone, R. A. Hoekstra, V. Kancherla, M. C. Mulaudzi, A. Kakooza-Mwesige, F. A. Ogbo, H. E. Olsen, J. O. Olusanya, A. Pandey, M. E. Samms-Vaughan, C. Servili, A. Shaheen, T. Smythe, D. Wertlieb, A. N. Williams, C. R. Newton, A. C. Davis, N. J. Kassebaum, and on behalf of the Global Research on Developmental Disabilities Collaborators (GRDDC), “Global Burden of Childhood Epilepsy, Intellectual Disability, and Sensory Impairments,” *Pediatrics*, vol. 146, no. 1, p. e20192623, 07 2020. [Online]. Available: <https://doi.org/10.1542/peds.2019-2623>
- [412] A. A. Asadi-Pooya, F. Brigo, S. Lattanzi, and I. Blumcke, “Adult epilepsy,” *The Lancet*, vol. 402, no. 10399, pp. 412–424, 2023.
- [413] M. Rahimiazghadi, C. Lammie, J. K. Eshraghian, M. Payvand, E. Donati, B. Linares-Barranco, and G. Indiveri, “Hardware implementation of deep network accelerators towards healthcare and biomedical applications,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 6, pp. 1138–1159, 2020.
- [414] M. K. Siddiqui, R. Morales-Menendez, X. Huang, and N. Hussain, “A review of epileptic seizure detection using machine learning classifiers,” *Brain informatics*, vol. 7, no. 1, pp. 1–18, 2020.
- [415] J. Koren, S. Hafner, M. Feigl, and C. Baumgartner, “Systematic analysis and comparison of commercial seizure-detection software,” *Epilepsia*, vol. 62, no. 2, pp. 426–438, 2021.
- [416] J. Tang, R. El Atrache, S. Yu, U. Asif, M. Jackson, S. Roy, M. Mirmomeni, S. Cantley, T. Sheehan, S. Schubach *et al.*, “Seizure detection using wearable sensors and machine learning: Setting a benchmark,” *Epilepsia*, vol. 62, no. 8, pp. 1807–1819, 2021.
- [417] C. Lainscsek, N. Rungratsameetaweemana, S. S. Cash, and T. J. Sejnowski, “Cortical chimera states predict epileptic seizures,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, no. 12, p. 121106, 2019.
- [418] R. G. Andrzejak, C. Rummel, F. Mormann, and K. Schindler, “All together now: Analogies between chimera state collapses and epileptic seizures,” *Scientific reports*, vol. 6, no. 1, p. 23000, 2016.
- [419] T. Hannon, K. M. Fernandes, V. Wong, E. S. Nurse, and M. J. Cook, “Over- and underreporting of seizures: How big is the problem?” *Epilepsia*, 2024.
- [420] J. Zelano, S. Beniczky, P. Ryvlin, R. Surges, T. Tomson, and I. S. T. Force, “Report of the ILAE SUDEP task force on national recommendations and

- practices around the world regarding the use of wearable seizure detection devices: A global survey,” *Epilepsia open*, vol. 8, no. 4, pp. 1271–1278, 2023.
- [421] D. J. DiLorenzo, “Neurovista: Concept to first-in-man: The war story behind launching a venture to treat epilepsy,” *Surgical Neurology International*, vol. 10, 2019.
 - [422] B. Razavi, V. R. Rao, C. Lin, K. A. Bujarski, S. E. Patra, D. E. Burdette, E. B. Geller, M.-G. M. Brown, E. A. Johnson, C. Drees *et al.*, “Real-world experience with direct brain-responsive neurostimulation for focal onset seizures,” *Epilepsia*, vol. 61, no. 8, pp. 1749–1757, 2020.
 - [423] C. Mead, “How we created neuromorphic engineering,” *Nature Electronics*, vol. 3, no. 7, pp. 434–435, 2020.
 - [424] E. Chicca, F. Stefanini, C. Bartolozzi, and G. Indiveri, “Neuromorphic electronic circuits for building autonomous cognitive systems,” *Proceedings of the IEEE*, vol. 102, no. 9, pp. 1367–1388, Sep. 2014.
 - [425] E. Donati and G. Indiveri, “Neuromorphic bioelectronic medicine for nervous system interfaces: from neural computational primitives to medical applications,” *Progress in Biomedical Engineering*, vol. 5, no. 1, p. 013002, Jan. 2023. [Online]. Available: <http://dx.doi.org/10.1088/2516-1091/acb51c>
 - [426] D. V. Christensen, R. Dittmann, B. Linares-Barranco, A. Sebastian, M. Le Gallo, A. Redaelli, S. Slesazek, T. Mikolajick, S. Spiga, S. Menzel, I. Valov, G. Milano, C. Ricciardi, S.-J. Liang, F. Miao, M. Lanza, T. J. Quill, S. T. Keene, A. Salleo, J. Grollier, D. Markovic, A. Mizrahi, P. Yao, J. J. Yang, G. Indiveri, J. P. Strachan, S. Datta, E. Vianello, A. Valentian, J. Feldmann, X. Li, W. H. Pernice, H. Bhaskaran, S. Furber, E. Neftci, F. Scherr, W. Maass, S. Ramaswamy, J. Tapon, P. Panda, Y. Kim, G. Tanaka, S. Thorpe, C. Bartolozzi, T. A. Cleland, C. Posch, S.-C. Liu, G. Panuccio, M. Mahmud, A. N. Mazumder, M. Hosseini, T. Mohsenin, E. Donati, S. Tolu, R. Galeazzi, M. E. Christensen, S. Holm, D. Ielmini, and N. Pryds, “2022 roadmap on neuromorphic computing and engineering,” *Neuromorphic Computing and Engineering*, 2022. [Online]. Available: <http://iopscience.iop.org/article/10.1088/2634-4386/ac4a83>
 - [427] G. Indiveri and Y. Sandamirskaya, “The importance of space and time for signal processing in neuromorphic agents,” *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 16–28, 2019.
 - [428] C. Mayr, S. Hoeppe, and S. Furber, “Spinnaker 2: A 10 million core processor system for brain simulation and machine learning,” *arXiv preprint arXiv:1911.02385*, 2019.
 - [429] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

- [430] C. Pehle, S. Billaudelle, B. Cramer, J. Kaiser, K. Schreiber, Y. Stradmann, J. Weis, A. Leibfried, E. Müller, and J. Schemmel, “The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity,” *Frontiers in Neuroscience*, vol. 16, 2022. [Online]. Available: <https://www.frontiersin.org/articles/10.3389/fnins.2022.795876>
- [431] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, “A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs),” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 106–122, Feb. 2018.
- [432] F. Corradi and G. Indiveri, “A neuromorphic event-based neural recording system for smart brain-machine-interfaces,” *Biomedical Circuits and Systems, IEEE Transactions on*, vol. 9, no. 5, pp. 699–709, 2015.
- [433] Sleep-Wake-Epilepsy-Center, “SWEC-ETHZ iEEG database,” inselspital Bern and ISL, ETH Zurich. [Online]. Available: <http://ieeg-swez.ethz.ch/>
- [434] J. Bartels, O. Gallou, H. Ito, M. Cook, J. Sarnthein, G. Indiveri, and S. Ghosh, “Spiking seizure classification dataset,” 2024. [Online]. Available: <https://doi.org/10.5281/zenodo.10800794>
- [435] D. E. Sava, Rachel and G. Indiveri, “Feed-forward and recurrent inhibition for compressing and classifying high dynamic range biosignals in spiking neural network architectures,” *arXiv preprint arXiv:2309.16425*, 2023.
- [436] I. M. Park, S. Seth, A. R. Paiva, L. Li, and J. C. Principe, “Kernel methods on spike train space for neuroscience: a tutorial,” *IEEE Signal Processing Magazine*, vol. 30, no. 4, pp. 149–160, 2013.
- [437] M. Sharifshazileh, K. Burelo, J. Sarnthein, and G. Indiveri, “An electronic neuromorphic system for real-time detection of high frequency oscillations (HFOs) in intracranial EEG,” *Nature Communications*, vol. 12, no. 1, pp. 1–14, 2021.
- [438] F. Costa, E. Schaft, G. Huiskamp, E. Aarnoutse, M. van’t Klooster, N. Krayenbühl, G. Ramantani, M. Zijlmans, G. Indiveri, and J. Sarnthein, “Robust compression and detection of epileptiform patterns in ecog using a real-time spiking neural network hardware framework,” *Research Square Preprint*, 2023.
- [439] O. C. Akgun and Y. Leblebici, “Energy efficiency comparison of asynchronous and synchronous circuits operating in the sub-threshold regime,” *Journal of Low Power Electronics*, vol. 4, no. 3, pp. 320–336, 2008.
- [440] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, “A million spiking-neuron integrated circuit with a scalable communication network and interface,” *Science*, vol. 345, no. 6197, pp. 668–673, Aug. 2014.
- [441] D. Zendrikov, S. Solinas, and G. Indiveri, “Brain-inspired methods for achieving robust computation in heterogeneous mixed-signal neuromorphic

- processing systems,” *Neuromorphic Computing and Engineering*, vol. 3, no. 3, p. 034002, Jul. 2023. [Online]. Available: <http://dx.doi.org/10.1088/2634-4386/ace64c>
- [442] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” in *International Conference on Learning Representations*, 2018.
 - [443] J. Diffenderfer and B. Kailkhura, “Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network,” in *International Conference on Learning Representations*, 2020.
 - [444] H. Inose, T. Aoki, and K. Watanabe, “Asynchronous delta-modulation system,” *Electronics Letters*, vol. 3, no. 2, pp. 95–96, 1966.
 - [445] T. Hawkes and P. Simonpieri, “Signal coding using asynchronous delta modulation,” *IEEE Transactions on Communications*, vol. 22, no. 5, pp. 729–731, 1974.
 - [446] M. C. van Rossum, “A novel spike distance,” *Neural computation*, vol. 13, no. 4, pp. 751–763, 2001.
 - [447] A. Burrello, K. Schindler, L. Benini, and A. Rahimi, “One-shot Learning for iEEG Seizure Detection Using End-to-end Binary Operations: Local Binary Patterns with Hyperdimensional Computing,” in *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. Cleveland, OH: IEEE, Oct. 2018, pp. 1–4. [Online]. Available: <https://ieeexplore.ieee.org/document/8584751/>
 - [448] —, “Hyperdimensional computing with local binary patterns: One-shot learning of seizure onset and identification of ictogenic brain regions using short-time ieeg recordings,” *IEEE Transactions on Biomedical Engineering*, vol. 67, no. 2, pp. 601–613, 2019.
 - [449] S. R. Kheradpisheh, M. Mirsadeghi, and T. Masquelier, “Bs4nm: binarized spiking neural networks with temporal coding and learning,” *Neural Processing Letters*, vol. 54, no. 2, pp. 1255–1273, 2022.
 - [450] M. Mirsadeghi, M. Shalchian, S. R. Kheradpisheh, and T. Masquelier, “Stidi-bp: Spike time displacement based error backpropagation in multilayer spiking neural networks,” *Neurocomputing*, vol. 427, pp. 131–140, 2021.
 - [451] F. Liu, W. Zhao, Y. Chen, Z. Wang, T. Yang, and L. Jiang, “Sstdp: Supervised spike timing dependent plasticity for efficient spiking neural network training,” *Frontiers in Neuroscience*, vol. 15, 2021.
 - [452] D. Khurana, A. Koli, K. Khatter, and S. Singh, “Natural language processing: State of the art, current trends and challenges,” *Multimedia tools and applications*, vol. 82, no. 3, pp. 3713–3744, 2023.
 - [453] S. A. Alowais, S. S. Alghamdi, N. Alsuhebany, T. Alqahtani, A. I. Alshaya, S. N. Almohareb, A. Aldaire, M. Alrashed, K. Bin Saleh, H. A. Badreldin, M. S. Al Yami, S. Al Harbi, and A. M. Albekairy, “Revolutionizing healthcare: the role of artificial intelligence in clinical practice,” *BMC Medical Education*, vol. 23, no. 689, 2023.

- [454] Z. Zhao, R. Cao, K.-F. Un, W.-H. Yu, P.-I. Mak, and R. P. Martins, "An fpga-based transformer accelerator using output block stationary dataflow for object recognition applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 1, pp. 281–285, 2022.
- [455] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [456] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in neuroscience*, vol. 12, p. 409662, 2018.
- [457] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural networks*, vol. 111, pp. 47–63, 2019.
- [458] L. Kang, X. Yang, C. Zhang, S. Yu, R. Dou, W. Li, C. Shi, J. Liu, N. Wu, and L. Liu, "A 24.3 μ j/image snn accelerator for dvs-gesture with ws-los dataflow and sparse methods," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.
- [459] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.
- [460] J. Bartels, I. Heidtmann, A. Hagihara, L. Minati, and H. Ito, "NimbleSNN: A 1.34 J/Image 4639 Logic Element Single-Spike Accelerator on FPGA Using Synchronization," May 2024. [Online]. Available: <https://doi.org/10.5281/zenodo.11096897>
- [461] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.
- [462] K. S. Gansel, "Neural synchrony in cortical networks: mechanisms and implications for neural information processing and coding," *Frontiers in Integrative Neuroscience*, vol. 16, p. 900715, 2022.
- [463] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.
- [464] Y. Chen, W. Ye, Y. Liu, and H. Zhou, "Sibrain: A sparse spatio-temporal parallel neuromorphic architecture for accelerating spiking convolution neural networks with low latency," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.
- [465] A. Carpegna, A. Savino, and S. Di Carlo, "Spiker+: a framework for the generation of efficient spiking neural networks fpga accelerators for inference at the edge," *arXiv preprint arXiv:2401.01141*, 2024.
- [466] H. Liu, Y. Chen, Z. Zeng, M. Zhang, and H. Qu, "A low power and low latency fpga-based spiking neural network accelerator," in *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2023, pp. 1–8.

- [467] Z. Wang, Y. Zhong, X. Cui, Y. Kuang, and Y. Wang, "A spiking neural network accelerator based on ping-pong architecture with sparse spike and weight," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5.
- [468] Y. Liu, Y. Chen, W. Ye, and Y. Gui, "Fpga-nhap: A general fpga-based neuromorphic hardware acceleration platform with high speed and low power," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2553–2566, 2022.
- [469] Q. Chen, C. Gao, and Y. Fu, "Cerebron: A reconfigurable architecture for spatiotemporal sparse spiking neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 10, pp. 1425–1437, 2022.
- [470] M. Liang, J. Zhang, and H. Chen, "A 1.13 μj /classification spiking neural network accelerator with a single-spike neuron model and sparse weights," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.
- [471] T. Wang, M. Tian, Z. Zhong, H. Wang, J. He, F. Tang, X. Zhou, S. Yu, N. Wu, L. Liu *et al.*, "Morphbungee: A 65nm 7.2 mm² 27 μj /image digital edge neuromorphic chip with on-chip 802 frame/s multi-layer spiking neural network learning," in *2023 IEEE Asian Solid-State Circuits Conference (A-SSCC)*. IEEE, 2023, pp. 1–3.
- [472] C. Sun, H. Sun, J. Xu, J. Han, X. Wang, X. Wang, Q. Chen, Y. Fu, and L. Li, "An energy efficient stdp-based snn architecture with on-chip learning," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 12, pp. 5147–5158, 2022.
- [473] Q. Chen, G. He, X. Wang, J. Xu, S. Shen, H. Chen, Y. Fu, and L. Li, "A 67.5 μj /prediction accelerator for spiking neural networks in image segmentation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 2, pp. 574–578, 2021.
- [474] J. Park, J. Lee, and D. Jeon, "7.6 a 65nm 236.5 nj/classification neuromorphic processor with 7.5% energy overhead on-chip learning using direct spike-only feedback," in *2019 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2019, pp. 140–142.
- [475] G. K. Chen, R. Kumar, H. E. Sumbul, P. C. Knag, and R. K. Krishnamurthy, "A 4096-neuron 1m-synapse 3.8-pj/sop spiking neural network with on-chip stdp learning and sparse weights in 10-nm finfet cmos," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 4, pp. 992–1002, 2018.
- [476] S. Li, Z. Zhang, R. Mao, J. Xiao, L. Chang, and J. Zhou, "A fast and energy-efficient snn processor with adaptive clock/event-driven computation scheme and online learning," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 4, pp. 1543–1552, 2021.
- [477] Y. Wu, L. Deng, G. Li, J. Zhu, and L. Shi, "Spatio-temporal backpropagation for training high-performance spiking neural networks," *Frontiers in neuroscience*, vol. 12, p. 331, 2018.

- [478] H. Zheng, Y. Wu, L. Deng, Y. Hu, and G. Li, “Going deeper with directly-trained larger spiking neural networks,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 12, 2021, pp. 11 062–11 070.
- [479] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, “Conversion of continuous-valued deep networks to efficient event-driven networks for image classification,” *Frontiers in neuroscience*, vol. 11, p. 682, 2017.
- [480] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1,” *arXiv preprint arXiv:1602.02830*, 2016.
- [481] R. Sarfati, K. Joshi, O. Martin, J. C. Hayes, S. Iyer-Biswas, and O. Peleg, “Emergent periodicity in the collective synchronous flashing of fireflies,” *Elife*, vol. 12, p. e78908, 2023.
- [482] D. J. Watts and S. H. Strogatz, “Collective dynamics of ‘small-world’ networks,” *nature*, vol. 393, no. 6684, pp. 440–442, 1998.
- [483] F. M. Quintana, F. Perez-Peña, P. L. Galindo, E. O. Neftci, E. Chicca, and L. Khacef, “Etlp: Event-based three-factor local plasticity for online learning with neuromorphic hardware,” *arXiv preprint arXiv:2301.08281*, 2023.
- [484] T. Ortner, L. Pes, J. Gentinetta, C. Frenkel, and A. Pantazi, “Online spatio-temporal learning with target projection,” in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2023, pp. 1–5.
- [485] B. Yin, F. Corradi, and S. M. Bohté, “Accurate online training of dynamical spiking neural networks through forward propagation through time,” *Nature Machine Intelligence*, vol. 5, no. 5, pp. 518–527, 2023.
- [486] M. I. Tiwana, S. J. Redmond, and N. H. Lovell, “A review of tactile sensing technologies with applications in biomedical engineering,” *Sensors and Actuators A: physical*, vol. 179, pp. 17–31, 2012.
- [487] V. Dharmaraj and C. Vijayanand, “Artificial intelligence (ai) in agriculture,” *International Journal of Current Microbiology and Applied Sciences*, vol. 7, no. 12, pp. 2122–2128, 2018.
- [488] S.-C. Liu, J. Kramer, G. Indiveri, T. Delbruck, and R. Douglas, *Analog VLSI: Circuits and Principles*. MIT Press, 2002.
- [489] G. Indiveri, F. Stefanini, and E. Chicca, “Spike-based learning with a generalized integrate and fire silicon neuron,” in *International Symposium on Circuits and Systems, (ISCAS)*. Paris, France: IEEE, 2010, pp. 1951–1954.
- [490] C. Bartolozzi and G. Indiveri, “Synaptic dynamics in analog VLSI,” *Neural Computation*, vol. 19, no. 10, pp. 2581–2603, 2007.
- [491] K. Burelo, M. Sharifshazileh, G. Indiveri, and J. Sarnthein, “Automatic detection of high-frequency oscillations with neuromorphic spiking neural networks,” *Frontiers in Neuroscience*, vol. 16, Jun. 2022. [Online]. Available: <http://dx.doi.org/10.3389/fnins.2022.861480>

- [492] M. Sharifshazileh and G. Indiveri, “An adaptive event-based data converter for always-on biomedical applications at the edge,” *International Symposium on Circuits and Systems, (ISCAS)*, May 2023. [Online]. Available: <http://dx.doi.org/10.1109/ISCAS46773.2023.10181549>
- [493] C. Bartolozzi, S. Mitra, and G. Indiveri, “An ultra low power current-mode filter for neuromorphic systems and biomedical signal processing,” in *Biomedical Circuits and Systems Conference, (BioCAS)*. IEEE, 2006, pp. 130–133.