Politecnico di Torino

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

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (35^{th}cycle)

Study and implementation of new computational paradigms exploiting neuromorphic hardware architectures

By

Evelina Forno

******

Supervisors:
Prof. Gianvito Urgese
Prof. Enrico Macii

Doctoral Examination Committee:
Prof. Steve Furber, Referee, University of Manchester
Prof. Alejandro Linares Barranco, Referee, Universidad de Sevilla
Prof. Andrea Calimera, Politecnico di Torino
Prof. Egidio Falotico, Scuola Superiore Sant'Anna
Dr. James Knight, University of Sussex

Politecnico di Torino
2023

Declaration

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

Evelina Forno
2023

* This dissertation is presented in partial fulfillment of the requirements for a **Ph.D. degree** at the Graduate School of Politecnico di Torino (ScuDo).

Abstract

The increased use of machine learning (ML) techniques for predictive maintenance and real-time anomaly detection in embedded devices, IoT, and edge computing has led to challenges with limited power and memory. Neuromorphic systems offer low power consumption and effective data interchange, and spiking neural networks (SNNs) offer potential for offline and online learning. SNN-based solutions have shown comparable accuracy to ANNs while outperforming them in energy consumption.

This dissertation presents a general approach to generating neuromorphic models for IoT applicatios that can be used directly by users to facilitate the design process of solutions by exploring new computational paradigms. The structure of the thesis follows the flow of data in a neuromorphic pipeline, from raw sensory data to a completed application.

Embedded neuromorphic applications for edge deployment require interaction with sensors, including digital and event-based sensors. Event-based sensors offer extreme power efficiency and are likely to see widespread adoption for specialized tasks. However, digital sensors remain a more accessible and low-cost alternative for now, and their use is likely necessary for systems exploiting neuromorphic platforms to be deployed in the near future.

The author discusses encoding techniques for event-based and digital sensors, finding temporal contrast encoding to be a suitable first choice for most applications handling input data with an important time-varying component. The efficiency of coding techniques correlates with input frequency, so a technique must produce a sufficiently high spike count to stimulate all layers of the classification network. The work presents experiments involving a variety of SNN architectures, finding that for time-varying data in the spiking domain, recurrent neural networks like Lagrange Memory Unit and RSNN are more suitable due to

their memory trace of past events. Model compression by edge pruning is also evaluated, obtaining significant reductions in synaptic activity without impacting performance.

Hyperparameter optimization is important for finding the optimal configuration of a network design. Results are reported for two HPO case studies and an exploration of the flexibility of the SpiNNaker system software to expand capabilities by implementing new placement and routing strategies. To evaluate the potential of neuromorphic hardware, the author performs a profiling of the parallel processing ability of the SpiNNaker board using a Message Passing Interface (MPI) implementation and the PageRank application as a benchmark. The performance of spiking neural networks on the neuromorphic Loihi chip is also compared to equivalent traditional and spiking classifiers on GPU, focusing on energy consumption per sample inference as a metric for assessing the advantages of a neuromorphic paradigm in this kind of application.

Finally, the author provides a summary of the research work by illustrating partial and complete neuromorphic pipelines, starting with a description of an interface for integrating the neuromorphic chip ODIN with a RISC-V coprocessor on a single SoC prototype. The processes used for neuromorphic classification of IoT time-varying signals are then compared through three case studies, showing the evolution of the pipeline from one project to the next, eventually encompassing the entire neuromorphic stack. The resulting method demonstrates a viable strategy for using neuromorphic tools in creating novel designs and implementations that offer outputs with limited error compared to those generated by applications implemented for general purpose architectures, while reducing execution time and power consumption, resulting in a net gain in adaptability for IoT and edge computing applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As a result of recent technological advancements, embedded devices are now more effective and ubiquitous than ever [7]. The rise of embedded devices, the Internet of Things (IoT), and edge computing has transformed the way we interact with technology. The adoption of machine learning (ML) techniques in IoT and edge computing applications employing such devices has significantly increased as a result [8], as they can help to optimize performance, reduce latency, and improve accuracy. For example, in a smart factory [9], machine learning algorithms can be used to analyze data from sensors on manufacturing equipment to identify patterns that indicate when maintenance is needed, helping to prevent equipment failure and reduce downtime while improving the overall efficiency. This process is known as predictive maintenance [10]. In addition, machine learning can be used to analyze data from IoT devices in real-time, which can help to detect anomalies [11, 12] and potential security threats; for instance, monitoring security cameras to identify suspicious behavior and alert security personnel [13].

There are several advantages to deploying ML models directly on the remote endpoint rather than distributing them as a service via the cloud. For starters, edge devices can perform real-time inference, lowering system latency in time-critical operations like autonomous driving. Second, the retention of all data on the device increases the reliability of the application, avoiding the consequences of network outages; at the same time, user privacy is enhanced as no data is exchanged between the machine and the internet. Finally, the removal of the cloud dependency lowers the cost of the infrastructure as a whole as well as the power consumption associated with communication operations.

While producing ML code for embedded devices has become much easier over the years, there are still a few challenges. Because edge devices continue to be limited in terms of power, memory, and computational capacity [14], candidate models must be carefully selected based not only on the type of input information, but also on the hardware requirements. After collecting a massive quantity of sample data from the exact embedded sensors that will be utilized in the end product, the models go through a training phase, which is often performed on a server computer because of the significant computing load. Several models can be evaluated, and the most promising ones are chosen for deployment on the system at hand. Specific end-to-end solutions for data gathering, labeling, model optimization, and deployment are available from major hardware manufacturers (for example, Qeexo AutoML [15]) as well as open source releases (for example, the Embedded Learning Library [16]). However, the efficacy of ML solutions is heavily informed by the size of the training dataset: large amounts of information are needed in order to produce meaningful results. The unprecedented volume of data generated by IoT devices is creating new issues for cloud-based solutions reliant on back-and-forth communication with end devices. Edge computing can provide solutions to these problems by bringing data processing closer to sensors by relocating computation from the cloud application layer down to the edge devices [17].

Artificial Neural Networks (ANN) are without a doubt the area of ML that has attracted the greatest interest from the scientific and corporate worlds over the last decade. While simpler machine learning models can be effective for simpler tasks, deep learning models are suitable for complex edge computing tasks that require a deep understanding of high-dimensional data; ANNs are capable of learning complex and non-linear relationships and are particularly effective in handling high-dimensional data — such as images, audio, and text — which are common in edge computing applications. The majority of ANN models require large quantities of memory and specialized, power-hungry technology like GPUs, making them too resource-intensive to execute on embedded systems.

This situation creates a bottleneck in the development of new effective solutions for automatic data handling, analysis and interpretation [18, 19]. In order to circumvent the consequences of the end of Moore's law [20], this past decade has seen the advent of novel architectures such as heterogeneous platforms, which integrate multicore von Neumann machines with specialized hardware accelerators;

however, these developments must face growing economic and environmental costs in the operation and maintenance of said hardware, as the computational power increase brought by platforms such as GPUs come with exponentially higher power consumption figures.

Various alternative solutions have been proposed in order to overcome the limitations imposed by the currently available hardware roster. Among these are neuromorphic systems, which employ design innovations inspired by neuroscience and biology [21]. The development of computational models based on Spiking Neural Networks (SNNs), a biologically-inspired model built around the behavior of animal neurons and characterized by sparse internal activity [22], is the focus of neuromorphic systems. These event-driven architectures guarantee very low power consumption while enabling effective data interchange between several independent processing units [23], meeting edge computing's demands for low power consumption, localized memory, and real-time response. This variety of neural network can be accelerated on compact, low-power neuromorphic hardware such as SpiNNaker [24], Intel Loihi [25], and Dynap-SEL [26], with the potential to execute both offline and online learning [27], although much work remains to be done on the software support [28, 29] and compilers [30, 4, 31] in order to attain said goal.

The most effective data encoding types, network designs, training methods, and hardware platforms for exploiting the benefits of SNNs are still under investigation. Nonetheless, despite the fact that neuromorphic technology is still in its early stages, several neuromorphic applications are starting to emerge in the field of embedded systems [32]. It is already possible to achieve same-chip integration of neuromorphic hardware with a traditional embedded processor using industry standard tools, enabling SNN-based co-processing [6]. Image and video frame analysis [33], dataset clustering [34], detection of pedestrians [35, 36], self-driving robots [37], and robotic fine-touch sensing, including dynamic motor control and Braille [38] or texture recognition [39], are examples of implementation areas that have been studied in the literature. Neuromorphic computing platforms can also be coupled with natively event-based sensors for improved efficiency, such as DVS imaging devices for gesture identification [40] or robotic vision [41].

Neuromorphic applications are also gaining popularity as viable options for the deployment of human-related time series data analysis: while deep learning

techniques have been successful in classifying time-varying data, their execution on hardware with limited resources encounters difficulties, due to the required signal pre-processing and to the need to identify both long- and short-term dependencies in the data, which affect the effectiveness of the selected model [19]. Previous work by the authors [2] found that SNN-based solutions obtained accuracies comparable to ANN implementations, while outperforming them in terms of energy consumption.

Spiking Neural Networks (SNNs)

Spiking Neural Networks (SNNs) are a bio-inspired form of Artificial Neural Network (ANN) that encodes input data with a series of pulse spikes [42]. This distinguishing feature of SNNs over non-spiking ANNs can be traced back to the main characteristic differentiating biological neural networks from non-spiking ANNs: in the brain, neurons interact by spreading information in the form of spikes, or action potentials [43–45]. These network models are of major importance as a natural programming paradigm in neuromorphic computing due to their brain-inspired features.

Despite their origins as a tool for neuroscience researchers to investigate computational representations of the living brain, SNNs have exhibited a number of advantages that set them apart from other neural network models. First and foremost, because spike-based encoding produces inputs that are sparse in time as well as space, SNNs can offer computational capabilities comparable to equivalent ANNs while consuming less power [46, 47]. Second, because SNNs' biologically-inspired learning algorithms support online learning, synaptic weights allow for real-time reconfiguration in response to the cause-and-effect relationships that emerge in the network's neuronal firing events. Finally, the ability to choose from several encoding techniques in the spike domain is yet another benefit [48].

Since SNNs rely on event-based activity, they are intrinsically suited for processing temporal information, allowing them to regard time as an additional dimension of the input signals [49, 27]. Additionally, because of their remarkable energy efficiency, SNNs are ideal choices for embedded applications. However, due to the substantial amounts of data transfer between the computational and memory units needed for their implementation, von Neumann-based architectures are not a suitable platform; cutting-edge neuromorphic processors, on the

other hand, employ colocation of memory and computation to address this issue. These include Digital ASIC (Loihi, ODIN, TrueNorth) and Mixed-signal (Braindrop, BrainScaleS, DYNAP-SE) solutions, as well as early devices featuring memristive synapses [50].

Benchmarks and IoT applications of neuromorphic solutions

In the last few years, the prospects for brain-inspired and neuromorphic technologies promising energy efficiency increases have been increasingly explored. Blouw et al. [51] benchmarked various hardware platforms executing keyword identification tasks, demonstrating the lowest energy consumption when utilizing Intel Loihi. Davies et al. [52] summarized researchers' successes with Loihi, while Yan et al. [53] compared the platform's performance with that of a SpiNNaker 2 prototype.

The effectiveness of transforming an ANN to an SNN tackling the heartbeat identification task and then deploying it on Loihi is reviewed in Buettner and George [54]. Azghadi et al. [55] then presented an expanded evaluation of neuromorphic hardware by testing biomedical applications on multiple platforms.

The advantages of a neuromorphic method are underlined in Blouw and Eliasmith [56], comparing the computational cost decrease given by SNNs produced in Nengo to architecturally equivalent deep neural networks (DNNs).

We examined the advantages of employing the SpiNNaker neuromorphic architecture [24] for performing massively-parallel general-purpose programs such as PageRank and DNA sequence matching developed with the MPI paradigm in Forno et al. [5] and Urgese et al. [57].

The Internet of Things (IoT) field is expected to profit the most from the growth of neuromorphic modeling and technology. Kim et al. [58] compiled a survey of IoT platforms supporting artificial intelligence (AI) applications, while An et al. [59] explored the impact of neuromorphic systems on Industry 4.0. Chang et al. [60] instead analyzed the importance of edge computing in the field of artificial intelligence of things (AIoT), healthcare, and other smart environments. Stuijt et al. [61] presented the neuromorphic IC μBrain, demonstrating promising results for event-driven edge AI applications in the IoT sector.

1.1 The building blocks of a neuromorphic pipeline

While the entirety of this thesis's contents constitutes original research conducted in part or fully by the candidate, parts of the contents presented have been previously published in academic literature. For the sake of transparency, Table 1.1 features a reference map matching each chapter in the thesis to its source papers.

Table 1.1 Table of contents

	References				
	[2]	[1]	[3]	[5]	[6]
Chapter 2: Sensors	■	■	■		
Chapter 3: Encoding and pre-processing		■	■		
Chapter 4: Neural models	■	■	■		
Chapter 5: Software frameworks	■		■		
Chapter 6: Hardware platforms			■	■	
Chapter 7: Neuromorphic pipeline	■	■	■		■

[2] Vittorio Fra, Evelina Forno, Riccardo Pignari, Terrence C. Stewart, Enrico Macii, and Gianvito Urgese. Human activity recognition: suitability of a neuromorphic approach for on-edge AIoT applications. *Neuromorphic Computing and Engineering*, 2(1):014006, 2022. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/4.0/`.

[1] Evelina Forno, Vittorio Fra, Riccardo Pignari, Enrico Macii, and Gianvito Urgese. Spike encoding techniques for IoT time-varying signals benchmarked on a neuromorphic classification task. *Frontiers in Neuroscience*, 16, 2022. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/4.0/`.

[3] Simon F. Müller-Cleve, Vittorio Fra, Lyes Khacef, Alejandro Pequeño-Zurro, Daniel Klepatsch, Evelina Forno, Diego G. Ivanovich, Shavika Rastogi, Gianvito Urgese, Friedemann Zenke, and Chiara Bartolozzi. Braille letter reading: A benchmark for spatio-temporal pattern recognition on neuromorphic hardware. *Frontiers in Neuroscience*, 16, 2022. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/4.0/`.

[5] Evelina Forno, Alessandro Salvato, Enrico Macii, and Gianvito Urgese. Pagerank implemented with the MPI paradigm running on a many-core neuromorphic platform. *Journal of Low Power Electronics and Applications*, 11(2):25, 2021. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit `http://creativecommons.org/licenses/by/4.0/`.

[6] Evelina Forno, Andrea Spitale, Enrico Macii, and Gianvito Urgese. Configuring an embedded neuromorphic coprocessor using a RISC-V chip for enabling edge computing applications. In *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 328–332. IEEE, 2021. © 2021 IEEE.

The goal of this dissertation is to present a general approach to the generation of neuromorphic models that can be exploited directly by users for describing their algorithms, so as to further facilitate the modeling process and the exploration of new neuromorphic computational paradigms which can be practically applied in data analysis and integration performed in the industrial field. To that end, the author endeavored to acquire experience with all elements of the neuromorphic pipeline and build a catalog of neuromorphic-based computational blocks, tools,

and frameworks, and identify favorable pipelines and strategies to implement new algorithms matching various use cases in the IoT and industrial fields.

The structure of the thesis follows the flow of data in a neuromorphic pipeline, starting from the raw input caught by sensors and ending with a completed application realizing a given task. The hardware and software building blocks identified during the PhD work are displayed in the diagram in Figure 1.1.

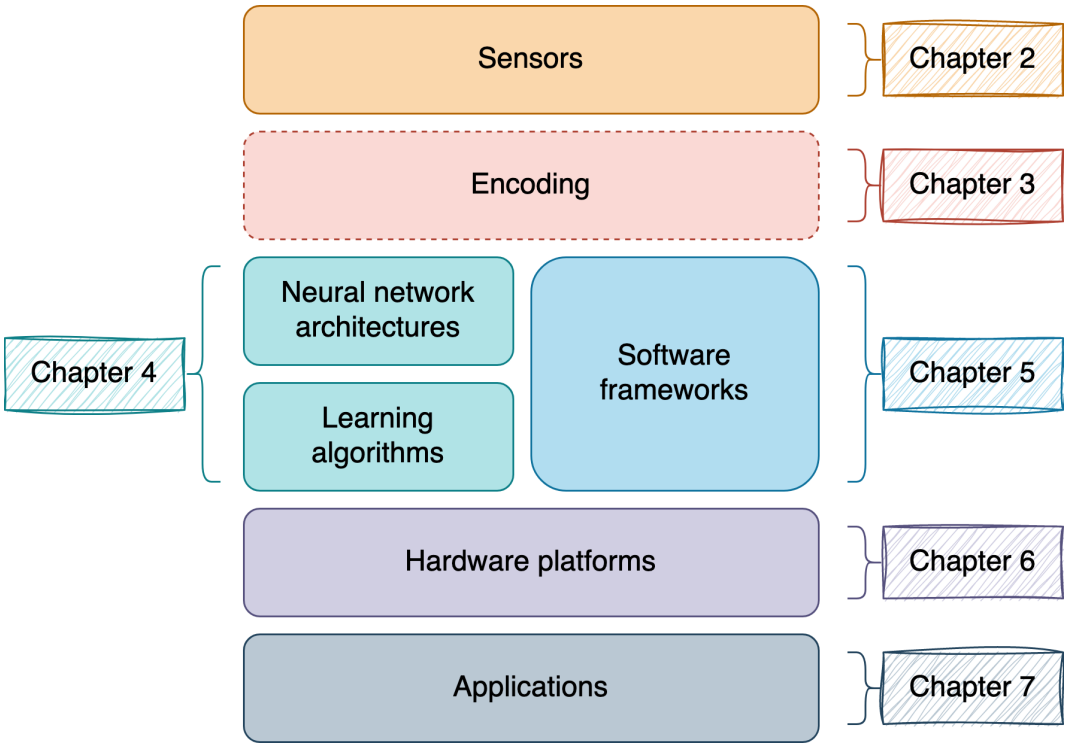

Fig. 1.1 Block diagram of a complete neuromorphic pipeline for data processing.

Each of the following chapters will present a brief overview of the available tools, models and techniques for each layer of the stack, followed by in depth reports of the analyses and benchmarks performed on selected tools during the PhD research work. This structure intends to guide the reader through the gradual building of a complete neuromorphic pipeline, focusing on classification of time-varying signals as a main task, but also exploring other applications of neuromorphic devices.

We will begin in Chapter 2 by tackling the issue of gathering input for the analysis stack. Suitable sensors are identified both among off-the-shelf digital solutions and in the novel, brain-inspired domain of event-based sensors. For

event-based sensors, we explore the gathering of a new event-based dataset using tactile sensors to acquire the Braille dataset; for digital sensors, an encoding step is needed to enable efficient processing by a SNN.

Chapter 3 is an examination of multiple encoding and preprocessing techniques, benchmarked on time-varying data from audio and inertial datasets acquired from digital sensors. After data is collected and properly encoded, it must be fed to the downstream SNN. In order to design and deploy a neural network application, choices must be made in two major areas: the neural modeling architecture and the software framework to implement it. Neural models comprise a variety of building blocks: from the model of the single neuron, to the interconnecting architecture, to the learning method used to train the network. These are explored in Chapter 4, along with an architecture optimization method using synapse reduction.

On the software side, Chapter 5 will offer a few examples of SNN specification software, which allows users to describe and deploy standard and novel neural models either on PC simulators or neuromorphic hardware accelerators. A special section is dedicated to hyperparameter optimization, that is, the automated search for the optimal configuration of a network design. Results are reported here for two different case studies. Finally, the chapter features an exploration of the SpiNNaker system software and its flexibility to expand on the capabilities of the underlying system by implementing dedicated placement and routing methods for a new task.

Neuromorphic hardware proper is investigated in Chapter 6. Continuing research conducted at the Politecnico di Torino in collaboration with the University of Manchester, a thorough profiling of the parallel processing ability of the SpiNNaker board is carried out with the development and refinement of a Message Passing Interface (MPI) implementation for this neuromorphic hardware and using the PageRank application as a benchmark. In the second part of the chapter, the performance of spiking neural networks running on the neuromorphic Loihi chip is compared to that of equivalent traditional and spiking classifiers running on GPU, using the classification of the Braille tactile dataset as reference. Particular attention is paid to the comparison in energy consumption per sample inference, which is the most helpful metric for assessing the advantages brought by adopting a neuromorphic paradigm in this kind of application.

Chapter 7 represents the culmination of the research work, providing a bird's eye view of the full neuromorphic pipelines realized during the PhD research work. First, we describe an interface for integration of the neuromorphic chip ODIN with a RISC-V coprocessor on a single SoC prototype. The embedded system is evaluated in RTL simulation and synthesized for FPGA. Then, we compare the processes used throughout this research for neuromorphic classification of IoT time-varying signals. Three case studies are presented, showing how the data processing pipeline evolved from one project to the next, eventually encompassing all 6 layers displayed in Figure 1.1.

The resulting method highlights a viable strategy for the employment of the many neuromorphic tools at the disposal of developers and researchers in creating novel designs and implementations; the applications generated by the neuromorphic pipeline are shown to provide outputs with a limited error with respect to experimental data generated by applications implemented for general purpose architectures, while reducing execution time and the power consumption, resulting in a net gain in adaptability for IoT and edge computing applications.

Chapter 2

Event-driven and standard sensors

In order to deploy low-power solutions in an edge-computing environment, input data acquisition must be considered as a required aspect of the pipeline. For instance, the IoT paradigm relies on networks of millions of sensors across a vast region. These sensors are the foundation of IoT, and they generate the vast bulk of measurement data in networks. These sensors can supply many forms of data to assist the IoT in environmental awareness. Furthermore, the majority of the resource requirements are supplied by the end devices of consumers: end users can utilize the devices as human-computer interfaces to recognize user needs and send them to the IoT. All of these sensors and end devices will be linked together so that they can exchange data and deliver new services [62].

The integration of sensing devices opens up a world of possibilities for data collection, system management, and resource optimization. Smart grid planning, transportation, and healthcare are examples of systems that can benefit from the employment of smart devices. Smart grid interoperability is achieved through the deployment of smart measurement equipment such as smart meters, sensors, and phasor measurement units (PMU) [63–65]; distributed sensors that respond to changes in energy needs, supplies, and costs allow energy providers to assess the state of the power grid and ensure the efficient operation of the power grid [66], furthermore providing readily available data for establishing relevant key performance indicators [67]. In the realm of transportation, information collected from inertial and visual sensors in a distributed network can provide input for real-time route guidance [68, 69], traffic signal control [70], and infrastructural monitoring [71]. Finally, recent advancements in technology have produced non-

invasive wearable sensors supporting a variety of healthcare applications, such as neonatal health monitoring [72], nutritional management of ketosis and diabetic ketoacidosis [73], and smart solutions for continual quality-of-life improvement for elderly and disabled people [73, 74]. A variety of wearable sensor applications [75–78] have become possible thanks to device miniaturization and the reduction of Body Sensor Networks to compact packages. Edge computing can provide the lowest-latency processing and storage options, while increasing the resilience of privacy- and safety-sensitive data environments. Despite this, the size and portability of the devices leads to additional constraints [79]: further growth of edge computing for wearable devices necessitates a paradigm shift to produce a reduction in processing efforts [80]. In order to overcome these obstacles, the environment must be set up for efficient real-time data processing, which will increase the variety of tailored services that can be effectively and extensively used on smart edge devices [81–85].

There are a few opportunities for neuromorphic technology to penetrate the domain of sensor networks at the edge:

- Embedded sensors provide better reactivity to the front-end environment of the edge computing system. However, digital sensors often come embedded into devices with limited computing capacity due to the power restrictions on the edge. Typically, data elaboration is then offloaded to powerful servers on the cloud, creating a communication bottleneck and a possible point of failure for data security. With neuromorphic platforms implemented close to the sensors, neural network-based tasks could be performed locally, limiting the need for cloud computing while respecting power constraints.

- A typical IoT sensor network produces massive amounts of data, which cannot be sent directly to cloud servers without compression or preprocessing: it would require considerable network capacity, causing a variety of challenges such as transmission delays and packet loss. As a result, IoT gateways often undertake data pre-processing and even aggregate before delivering it to remote cloud servers. There are a few ways neuromorphic paradigms could help here, as the event-based encoding required by SNNs significantly reduces the size of the relevant data: said encoding can either be performed within the sensor, by a dedicated external ASIC solution, or in software by an embedded digital coprocessor acting as a middleman between a digital

sensor and a neuromorphic computer. In this and later chapters, we will see examples of all these solutions.

- User data storage is also generally outsourced to various third-party vendors, whose storage devices are installed at the network's edge and situated at a variety of physical locations. The fragmentation of data thus makes it difficult to maintain data integrity and may expose the information to malicious attacks. As stated in the previous point, spike-encoded data can drastically reduce storage needs; at the same time, using SNN accelerators to execute computational tasks close to the sensor could completely eliminate the need to store data, if not for historical reasons.

A neuromorphic pipeline can interface with two sorts of sensors: regular digital sensors, and event-based neuromorphic sensors. At each time instant, digital sensors generate floating-point or integer samples, and their signals must be encoded into spikes before they can be processed by a Spiking Neural Network. Event-based sensors, on the other hand, produce output in the form of spike trains, and remain quiet when no input is present; the spike encoding happens entirely within the sensor, and its output may be immediately fed into an SNN.

Sensors and datasets also come into play as neuromorphic researchers face the problem of fairly assessing the performance of SNN designs. As things currently stand, while many natively event-based sensors have been proposed, very few data acquisition campaigns have been organized exploiting them. On the other hand, there is a vast availability of datasets recorded by digital sensors, which nonetheless require an encoding stage before they can be handled by an SNN.

In this chapter, we will examine both types of sensors, investigate different techniques for event-based encoding, and study the production of various datasets useful for benchmarking the performance of SNNs on time-varying sensory data.

Section 2.1 gives an overview of digital sensors and their interaction with downstream neuromorphic processes. Entering more detail, we select two areas of applications for digital sensors in the IoT domain: human activity recognition (Section 2.1.1) and speech classification (Section 2.1.2). For each of these applications, representing time-varying signals occupying different part of the energy spectrum, we select representative datasets (WISDM and FSD) that will be used in

subsequent chapters to benchmark several characteristics of the neuromorphic pipeline.

In Section 2.2, we will move on to consider neuromorphic sensors. This type of sensor outputs data that is already in the spike domain and can be delivered to a downstream SNN without the need for an encoding step. First, we will consider the novel case of event-based tactile sensing (Section 2.2.1), describing the background of this task and the method for creating the Braille dataset, a repository of time-varying spiking data produced by sliding a robotic finger over 3-dimensional braille characters. Then, we will examine two well-known neuromorphic sensors: the Neuromorphic Auditory Sensor (NAS) (Section 2.2.2) and the Dynamic Vision Sensor (DVS) (Section 2.2.3). Finally, in Section 2.2.4, we will describe LIPSFUS, a sensor fusion dataset that seeks to combine NAS and DVS data to improve speech recognition tasks in the neuromorphic domain.

2.1 Digital sensors and datasets

Spiking Neural Networks (SNNs), with their low energy consumption and computational cost, can provide major benefits to the field of embedded machine learning for edge applications. Input from normal digital sensors, however, needs to be encoded into spike trains before it can be processed by neuromorphic computing methods. The author and colleagues examined the characteristics of data acquired from digital sensors in Fra et al. [2] and Forno et al. [1], focusing on the creation of neuromorphic pipelines for the classification of time-varying signals.

We sampled time-varying signals from two distinct datasets: the Free Spoken Digit (FSD) Dataset [86], which provides audio signals, and the WISDM Smartphone and Smartwatch Activity and Biometrics Dataset [87, 88], which is commonly used for human activity recognition (HAR) [2]. Their distinction can be drawn from two different angles: first, the nature of the activity involved, and second, the signal frequency. Of these, only the signal frequency has important repercussions on the brain-inspired pre-processing procedures being carried out.

Using the range of human audible frequencies as a discriminant, we can distinguish between signals at *very low frequency* (below 20 Hz), *low frequency* (20–500 Hz), *middle frequency* (500 Hz–2 kHz), and *high frequency* (2–20 kHz). The samples from the FSD dataset and the WISDM dataset can be assigned to the

middle frequency and *very low frequency* ranges, respectively, based on these definitions and the on Nyquist-Shannon sampling theorem.

The spectral densities for both datasets are depicted in Figure 2.1, which demonstrates how the two forms of data occupy distinct, non-overlapping regions of the frequency spectrum.

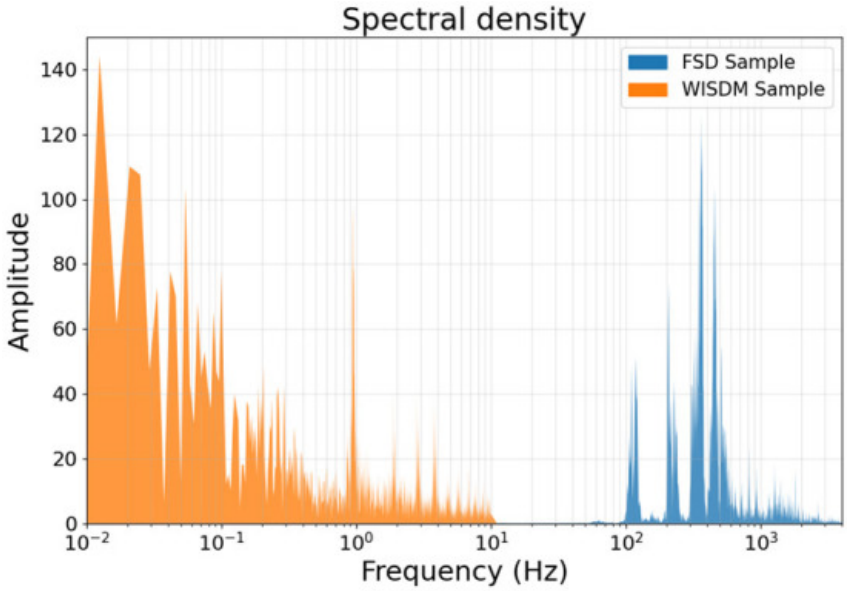

Fig. 2.1 Spectral densities of two samples, one from the WISDM dataset and the other from the FSD. The WISDM data from the accelerometer is in the very-low frequency region, whereas the FSD sample is in the middle range of human-audible signals. [1]

2.1.1 Inertial sensors for human activity recognition

Body sensor networks have undergone significant change in the last few years: the work of complete BSNs once consisting of a number of wearable, interconnected sensor nodes can nowadays be performed by a single consumer device with sufficient precision [89–91]. As a result, smartphone sensors, most often inertial measurement units (IMUs), are frequently used for collecting datasets describing human activity: the most well-known are are WISDM [92] and UCI-HAR [93] (also known as SBHAR [94]), but a more recent version of the WISDM dataset [87] is also gaining popularity because of its more evenly distributed classes and the inclusion of smartwatch signals. Other datasets, like MHEALTH [95, 96], OPPORTUNITY [97,

98], PAMAP2 [99, 100], PUC-Rio [101], USC-HAD [102], UTD-MHAD [103], and WHARF [104] take into account additional wearables and sensors.

Mekruksavanich et al. [105] provided a thorough benchmarking for various network architectures on diverse datasets. Said work also highlighted how data segmentation affects classification accuracy. In fact, there are two good reasons to carefully select the sample window size for time-varying signals like those handled by HAR: it can improve accuracy, resulting in more dependable classifiers, and it determines the classification time, which can be critical in determining applicability for real-time applications. Peppas et al. [106] gave an overview of representative window sizes used in the HAR task, demonstrating that typical options fall between 1 and 10 seconds. Exceptions are research by Ordóñez et al. [107], Wan et al. [108], and Xia et al. [109], which use temporal frames as small as 0.25 s for distinct datasets. Ihianle et al. [110], Mekruksavanich et al. [111, 112] and Oluwalade et al. [113] instead used a 10-second signal segmentation.

The WISDM dataset

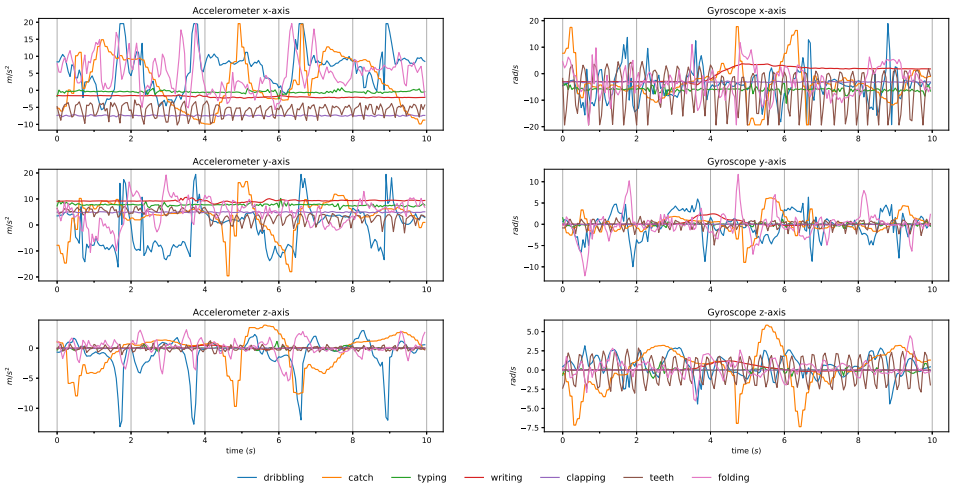

Fig. 2.2 A representative comparison of 10-second samples collected by the smartwatch on the 6 IMU sensors for the 7 classes in the WISDM dataset's *general hand-oriented* subset. [2]

The Wireless Sensor Data Mining Lab released the WISDM Smartphone and Smartwatch Activity and Biometrics Dataset in 2019. Using the accelerometers

and gyroscopes on consumer-grade smartphones and smartwatches, it gathers 3D data signals linked to 18 different activities carried out by 51 people, with an acquisition rate of 20 Hz and a total time of 3 min for each activity. With a percentage contribution of each activity ranging from 5.3 to 5.8 % of the 15 630 426 total samples, it also assures stronger class balancing and a larger selection of activities with respect to the previous version of the WISDM dataset [92]. The dataset can be divided into three subsets based on the different types of activities: *on-hand-oriented*, *general hand-oriented*, and *eating hand-oriented*. Figure 2.3 depicts the kernel density estimate of 3D smartwatch data from accelerometer and gyroscope in the *general hand-oriented* subset of the WISDM dataset. An overlap between the raw signal values can be seen here.

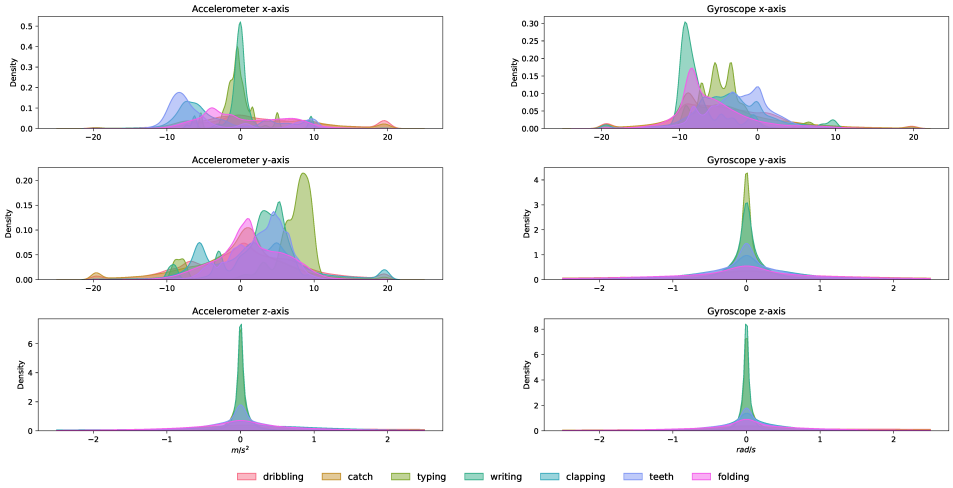

Fig. 2.3 Kernel density estimation of smartwatch values recorded on the 6 IMU sensor axes for the 7 classes in the WISDM dataset's *general hand-oriented* subset. [2]

2.1.2 Audio sensors for speech recognition

One of the most popular and effective uses of ANNs has been audio classification. This method is crucial to consumer-grade AI technology, which includes virtual assistants, speech recognition algorithms, and text-to-speech apps [114]. Precise information about the sensors employed in the gathering of speech datasets is rarely available; in fact, this type of dataset can easily be crowd-sourced by having speakers submit recordings taken with off-the-shelf devices. For instance, the LibriSpeech dataset [115] has been assembled using audiobooks from the Lib-

riVox project, which is driven by volunteer readers. Their quality can therefore vary from studio-recorded, high-quality samples to amateur home recordings. This variety is desirable for many applications, especially mobile tasks where real-time input is usually noisy. In the neuromorphic field, research into auditory sensors has received a great deal of attention, starting with the pioneering work of Carver Mead [116] and finding a groundbreaking first implementation in Liu et al. [117]. Datasets of note in neuromorphic research include the Spoken MNIST dataset [118], the Free Spoken Digit (FSD) dataset [86], and the Google Speech Commands (GSC) [119] dataset, which are generally used in a spoken word classification challenge. The Google Speech commands has spurred the creation of derivative datasets: the written and spoken digits database [120] is a subset of the Google Speech Commands dataset restricted to utterances of the ten digits, with added preprocessing and feature extraction using the Mel Frequency Cepstral Coefficients (MFCC) method, coupled to written digits from the MNIST project, while the Heidelberg Spiking dataset [121] was created by applying a software method based on the inner ear model to encode the audio files into spike trains. Cramer et al. [121] then benchmarked a variety of spiking and non-spiking classifiers using the Heidelberg Spiking dataset, finding that the best performing models were those having explicit recurrence, such as LSTMs and RSNNs. Rostami et al. [122] instead chose to process the GSC with MFCC extraction, FFT and spectrogram generation in order to test a recurrent SNN on the SpiNNaker 2 neuromorphic hardware prototype. The FSD is also present in several research works in the field: Peterson et al. [123] recently used it as a baseline for the performance of SNNs trained with spike-timing-dependent plasticity (STDP) on the speech classification task, and it has also been employed to test cutting-edge neuromorphic hardware based on memristors [124] and Atomic Switch Networks [125].

The FSD dataset

The Free Spoken Digit Dataset is a crowd-sourced open database of spoken digits, first released on GitHub by Zohar Jackson in 2017. It contains homemade recordings contributed by volunteers. In its latest release (v1.0.10, published in 2020), it counts 3000 recordings by 6 individuals speaking English with diverse accents, with each speaker pronouncing each of the 10 digits 50 times. All recordings are `.wav` files encoded at 8 kHz; since they are cut in such a way to guarantee minimal

silence at the start and end, each sample has variable length. While the dataset has mostly seen use in word classification tasks, the dataset also provides metadata for speaker identification. Furthermore, a simple API for data access and manipulation is included, with dedicated trimming and spectrogram plotting scripts. The FSD dataset is included in the Tensorflow and Accord.NET frameworks and the official train-test set provided has a 90 : 10 split. This dataset has been utilized in previous research on spike encoding in the neuromorphic domain [126].

2.2 Neuromorphic sensors

Neuromorphic event-driven sensors use circuits that mimic the behavior of biological sensors, like the retina and cochlea, to generate dynamic binary events in response to constantly varying stimuli. Locally sensing a signal (light, sound, touch, etc.), biological sensing systems transform it to an asynchronous spike format and convey it to the central nervous system. The electronic equivalent of biological spikes, known as events, are also asynchronous in time and therefore carry information about the location and time at which the input was perceived [127].

Event-based sensing is a great fit for resource-constrained computing and robotics because of its built-in data encoding features and sparse communication [128]. In fact, this type of sensors reduce processing and communication efforts by only sending signals when a change in their sensory region is recognized [129]; the event-driven domain [130] avoids the ongoing polling of sensor readouts by using binary, time-discrete events. Additionally, since the exchange of information is initiated by sensor atoms (pixels or taxels activated by an above-threshold input), events are relayed with minimal latency, offering these sensors an improved latency/power tradeoff [127].

2.2.1 Event-based tactile sensing

Tactile perception is by nature sparse in terms of time and space, with only local correlation where multiple nearby receptors or sensors are activated by the same event. Therefore, the peculiar advantages of event-based sensing are particularly desirable in the touch domain. Tactile events are perceived for a short duration, as long as the stimulus is applied, and in a confined portion (*patch*) of the sensor.

When there is no stimulus, the tactile system is said to be at rest. While other event-driven neuromorphic sensors, such as the Dynamic Vision Sensor (DVS) [131] and the silicon cochlea [132], have piqued researchers' interest, giving rise to specialized data pipelines and standardized benchmarks like the TIDigits [133] and DVS gesture recognition dataset [134], there have been relatively few developments in the field of touch. One of a handful of classifiers utilizing tactile neuromorphic sensors [135, 39] is the Spiking Tactile MNIST (ST-MNIST) dataset of handwritten digits, created by writing on a neuromorphic tactile sensor array [136]: because of the type of input, the information in the ST-MNIST spike patterns is primarily spatial, and a feedforward Convolutional Neural Network (CNN) has the most accuracy when adding all the spikes together to produce a "tactile image".

Bologna et al. suggested a neuroengineering system for robotic applications that included spatiotemporal event coding, probabilistic decoding, and closed-loop motion policy adaptation implementing active touch. This method was tested on Braille text, demonstrating that fingertip kinematics could be modulated effectively based on the complexity of the letters encountered. A subset of 7 Braille symbols were used to classify the signals captured by a fingertip sensor attached on a robotic arm, yielding an $(89 \pm 5.3)\%$ identification rate [137, 38].

The Braille dataset

During the 2021 Telluride Neuromorphic Cognition Engineering Workshop, the author joined a group of international researchers focusing on the Tactile Perception topic area in order to produce a novel dataset of haptic information based on the Braille dataset. The team's work resulted in a journal paper published the following year [3]; the result in this and subsequent sections draw from said article.

The Braille dataset was created with the intention of investigating the possibility of an end-to-end neuromorphic system for tactile perception that would incorporate event-based communication (sensor level), asynchronous processing (hardware level), and spike-based computing (algorithmic level). However, to the best of the authors' knowledge, no event-based tactile sensor was yet available; therefore, the output of such a device was emulated by utilizing a digital capacitive sensor and converting its output to spikes.

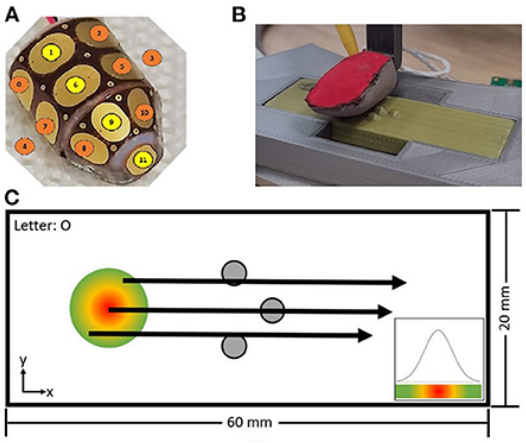

Fig. 2.4 Setup for the recording of the Braille dataset. **(A)** Diagram of the sensorized fingertip. **(B)** Experiment configuration with a braille sample. **(C)** Random distribution for the starting location of the fingertip's sliding motion. [3]

The dataset was recorded at IIT using an Omega.3 robot [138], sliding a sensorized fingertip [139] with 12 capacitive sensors over 3D printed Braille letters from "A" to "Z" as well as "space" at a regulated speed and position. The robotic fingertip and the experiment setup are shown in Figures 2.4: the inner region of the fingertip is made up of 12 capacitance plates (A), which are wrapped in a three-layer fabric and moved over the braille letters with a constant sliding distance and velocity (B). To introduce noise, the starting location was randomly changed from sample to sample according to a Gaussian distribution (C).

The size of the Braille letter was chosen to match the geographical dispersion of the fingertip's taxels, which allows the entire letter to be identified with a single sliding movement. The sliding distance (15.5 mm), sliding velocity (20 $\mathrm{mm\,s^{-1}}$), and distance to the plate's flat surface all remained constant. 200 instances of each letter were recorded at a sample rate of 40 Hz, encoding the capacitance value of the sensor as a sequence of 8-bit positive integers.

2.2.2 A silicon cochlea: the Neuromorphic Auditory Sensor (NAS)

Event-based audio sensors typically take inspiration from the biological cochlea, a structure in the inner ear that is sensible to vibrations in the audible spectrum. The hair cells in the membrane stimulate spiral ganglion cells, which encode the auditory sensation to spikes [140]. Since the first proposal for a silicon cochlea [141],

many analog and digital implementation of said design have been realized; one of these is the Neuromorphic Auditory Sensor (NAS) [142], developed in 2016 at the University of Seville, Spain. It reproduces the frequency-decomposing ability of the cochlea using two cascaded banks of specially-designed spike-based low pass filters (SLPFs) [143], which convert digitalized stereo streams to spike trains and then decompose them into N frequency bands. The sensor outputs the spike-encoded data on $2N$ channels (to account for the ON and OFF spikes in each frequency band) through an Address Event Representation (AER) interface. The NAS was originally designed as an FPGA implementation, and it was released in 2021 as a modular open-source HDL netlist, OpenNAS [144]. Since its inception, the NAS has been put to the test on various types of audio data. In Domínguez-Morales et al. [145], the NAS was used to preprocess and encode heart sound recordings from the PhysioNet/CinC Challenge database [146], and in later work by the same group [147], the NAS served as the entry point for a neuromorphic speech recognition pipeline powered by the SpiNNaker hardware; the input dataset was the GSC discussed here in Section 2.1.2. Recently, the OpenNAS has served as one of the input devices involved in the LIPSFUS sensory fusion dataset; we will discuss it in section 2.2.4.

2.2.3 A silicon retina: the Dynamic Vision Sensor (DVS)

Neuromorphic vision was also among the first applications of neuromorphic sensing [116], with a first realization of a silicon retina with adaptive photoreceptors by Misha Mahowald in 1991 [148]. Over the past two decades, a variety of solutions have been developed which try to emulate different features of the biological eye [149]; among these, the most well-known and tested is the Dynamic Vision Sensor (DVS), originally realized by Lichtsteiner et al. in 2006 [130]. The DVS is composed of a collection of pixels which asynchronously reacts to the temporal contrast in the scene dynamics, outputting an array of ON/OFF events which capture the outline of any moving object in the observed frame. The highly efficient integration of the DVS circuit gives it exceptionally low power consumption and excellent temporal resolution irrespective of lighting conditions [149]; for instance, a later implementation of the DVS by Serrano-Gotarredona and Linares-Barranco [150] reached a power requirement of 4 mW with an output of 100 000 frames per second, 30 µs latency, and a contrast sensitivity of 1.5 %. These char-

acteristics make it an ideal candidate for motion detection and object tracking tasks [149].

Hu et al. [151] presented a comprehensive dataset realized by exposing a DVS vision sensor to existing benchmark videos on a monitor, including object-tracking, action recognition, and object recognition sequences. The 37 410-sample library remains one of the largest neuromorphic datasets in the field of machine vision. Other derivative datasets include the CIFAR10-DVS [152], Poker-DVS and MNIST-DVS [153], created from existing frame-based image datasets by animating the images in order to make them detectable by the DVS. Examples of datasets created ad hoc for the DVS are still few and contain a limited number of samples. Miao et al. [154] presented three event-based datasets for pedestrian detection, action recognition, and fall detection, while IBM's DVS128 dataset [134] contains recordings of hand gestures under different lighting conditions.

2.2.4 Neuromorphic sensor fusion: the LIPSFUS dataset

Sensor fusion is the merging of synchronized sensor data received from multiple devices observing the same phenomenon, so that the fused information is less uncertain than when these sources are used separately. Developments in audio-visual sensory fusion with neuromorphic engineering offer up new opportunities for latency and power-critical applications in robotics, IoT and edge computing. In Rios-Navarro et al. [155], the novel LIPSFUS dataset was presented, which integrates the NAS with the DVS to record a speech command dataset where each event-based audio sample is coupled to the corresponding lip movement for each word. The dataset was recorded in two environments with different noise levels, and involved 22 speakers of diverse nationality, gender and age. The NAS received input from two microphones mounted on a binaural dummy head, allowing for stereo recording; the samples were recorded by rotating the dummy at different angles with respect to the speaker (0º, -45º, -90º, 45º and 90º). The dataset is intended to serve as a baseline for the development of integrated audio-visual sensory systems capable of improving speech recognition at the edge by integrating environmental and contextual signals other than sound (in this case, lip reading).

2.3 Chapter summary

Sensors are a vital part of any application deployed in an edge environment. At the moment, the expansion of the low-cost sensor market thanks to the development of MEMS devices has created a large field of applications for low-cost edge computing platforms, which constitutes an appealing entry point for the use of neuromorphic technology. In order to deploy these SNN-based solutions effectively, the sensing input must be in the form of spikes. This can be achieved either by using novel event-based sensors or by retrofitting digital sensors with an additional encoding step.

Event-based sensors, while promising, are still under active development and research. This type of device has shown excellent results in low-power operation and represents the most natural pairing for neuromorphic applications. However, given the market predominance of digital sensors and their low cost, they must not be discounted as a solution in the short term.

We face, then, the problem of interfacing a neuromorphic computational pipeline with digital inputs. The encoding technique to bridge the digital floating-point signal to the event-based spiking signal must be chosen carefully, as the transition into the spike domain can heavily affect the information content of the data and, thus, the efficacy of the downstream neural network. In Chapter 3, we will examine in detail the problem of translating digital signals into spikes and the implications of each encoding technique.

Chapter 3

Input encoding and pre-processing

In Chapter 2, we established that incoming analog and digital data must be transformed into a stream of spiking signals before they can be processed by an SNN. Biological research shows that, even in nature, sensory information can be rendered into spikes in a variety of ways; likewise, several kinds of encoding schemes have been derived from the animal neuron [156]. One such method are *rate-based encodings*, which have been in use since the early days of SNN research [157] and have proven to be an effective method for converting trained ANNs into SNNs [158–160] for classification applications. The event-based neuromorphic cochlea [161] also converts audible signal amplitudes into neural firing rates, and rate codes are generally prominent in the field of robotic control [162]. In contrast, interest in *temporal coding* methods has grown in recent years: these approaches have been applied to reduce the power consumption of converted networks [163–165], but a variety of applications that use time-based encodings natively have also emerged. These include bio-inspired olfactory sensors [166] and cameras [167], hybrid ANN/SNN [168] and entirely spiking [169–171] networks for image classification, speech recognition [172, 173] and speaker authentication [174] systems, time series forecasting [175] and anomaly detection [176, 177], and many other applications.

This chapter examines the effects of various spike encoding methods on the performance of a spiking convolutional neural network (sCNN) that was trained with the process of *transfer learning*. The generated networks have been evaluated using time-varying input signals from the WISDM and FSD datasets (see Section 2.1) and encoded into spikes before being supplied to the classifiers. This

approach allows us to exploit the abundance of digital-output sensors on the market compared to the scarcity of neuromorphic sensors; it also ensures fairness in the comparison of various encodings because they are all produced from the same input data. The research offered here, previously published in Forno et al. [1], attempts to trace a roadmap for the future advancement of encoding/decoding approaches for integrated System of Systems with interoperability of various modules and neuromorphic sensing solutions.

3.1 Preprocessing

We drew inspiration from biology and the animal kingdom to build various pre-processing approaches for time-varying signals. Specifically, we implemented a procedure that mimicked the working principle of the cochlea using a methodology inspired by the human auditory system. The cochlea is the final section of the auditory system, consisting of a spiral structure whose nerve cells, due to their location along the so-called basilar membrane, allow for filter-bank-like activity. As a result, the incoming stimulus is frequency-decomposed, and each of the resulting components, identified by the excitation of specific regions of the basilar membrane when exposed to their matching characteristic frequency, is translated into pulses, producing the electrical signal to be processed by the brain [178–182]. According to the literature, gammatone and Butterworth filters are suitable methods for simulating such a mechanism [183–187]; this type of filter bank may also successfully extract characteristics from other time-varying signals, such as vibrations acquired by an accelerometer [188], by carefully altering the expected frequency range. We used these two filter types to execute a pre-processing step on the data, resulting in the division of time-varying signals into separate frequency channels.

We also looked at how the cochlea-inspired frequency filter affected a set of spiking and non-spiking Recurrent Neural Networks (RNNs). The frequency filter was applied to the input, decomposing the original signals into five channels via the application of a Butterworth filter bank. All the spiking RNNs under consideration used the rectified integrate and fire neuron model.

3.1.1 Frequency decomposition

Decomposing the input signal into frequency channels can improve encoding performance by increasing the number of extractable features and resulting a more information-rich input. We did extensive testing to see how input frequency filtering affected the accuracy of a sCNN doing classification on the FSD and WISDM datasets. The network under examination consists of 1 convolutional layer with I feature maps, 1 average pooling, 1 convolutional layer with J feature maps, 1 average pooling, and K fully-connected layers; we identify each variation on this structure with the acronym CI-CJ-FK. Figure 3.1 depicts a sample design with the combination C6-C12-F2.

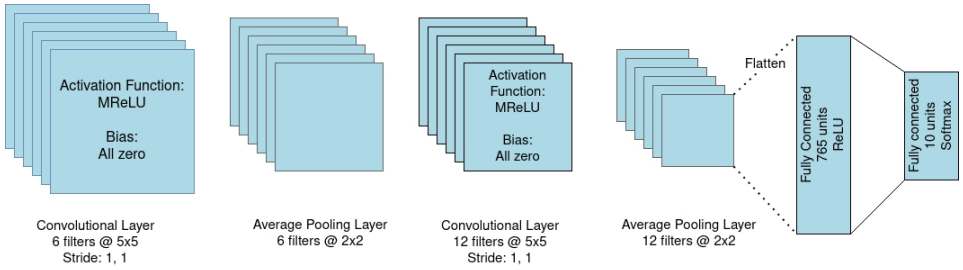

Fig. 3.1 Architecture of the C6-C12-F2 convolutional neural network. [1]

In all experiments, the gammatone filter performed better than the Butterworth filter. When classifying input data from the FSD, we tested decomposition with 32 and 64 channels, achieving a median accuracy of 77.50 % for the Butterworth filter and 84.00 % for the gammatone across all channel configurations. In the case of the WISDM dataset, due to the lower sample frequency, only the 4-, 8-, and 16-channel separation setups could be examined. Compared to the FSD, lower average test accuracy rates were observed for all encoding options: 66.67 % with the Butterworth filter and 46.67 % with the gammatone, both paired with a C12-C24-F2 network architecture.

3.1.2 Feature extraction

The feature extraction stage is required to employ the transfer learning approach with a non-spiking CNN model, and it consists of creating the *sonogram*, which is a binned representation of the input appropriate for elaboration by convolutional

layers. We adopt this definition of sonogram from [147]; while it was initially used to describe the binned representation of an audio signal, hence the name, we utilize the same definition for the WISDM dataset's analogous representation as well as the FSD. The number of bins, or intervals into which the spike-coded signal is divided, is the parameter that defines the sonogram's resolution and the quality of feature extraction. We experimented with several interval lengths to get the best bin separation. The tested binning intervals for the FSD dataset are 50 and 250 for the 32-channel filter bank and 50 and 125 for the 64-channel filter bank. We have 24, 18, and 18 bins for the separation of the WISDM dataset into 4, 8, and 16 channels; we chose only one binning type for each channel separation because other values exhibited inadequate accuracy results. Figure 3.2 shows the comparison results. As seen in section 3.1.1, the WISDM dataset had overall lower accuracies. Finally, regardless of the number of channels used in the pre-processing step, greater bin counts result in overall lower performance. In fact, excessively big or small values for this parameter result in a quasi-uniform pattern with less information, because the difference in intensity between the sonogram's pixels becomes too small; an example is illustrated in Fig. 3.3.

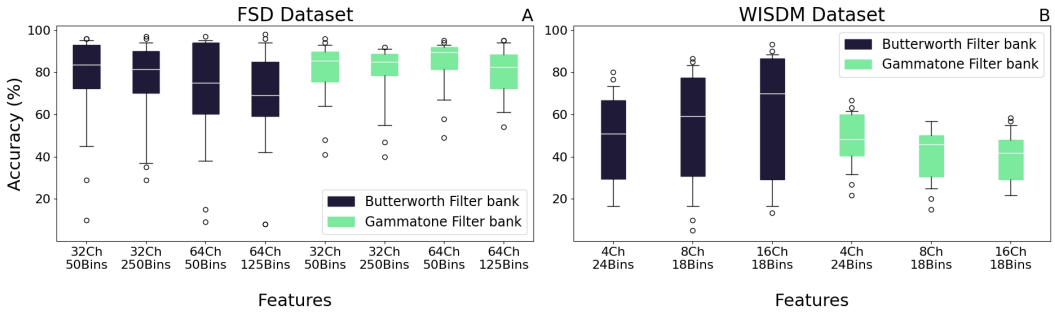

Fig. 3.2 Median accuracy values for each feature extraction class for various network design and encoding technique combinations. [1]

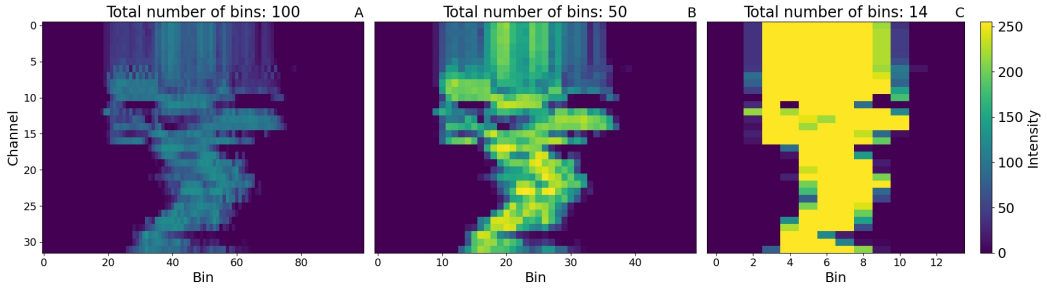

Fig. 3.3 A 32-channel sonogram rendered in 100, 50, and 14 time bins. The 50-bin subdivision provides the finest compromise between resolution and information density. [1]

3.2 Encoding techniques

Although the availability of neuromorphic, event-based sensors is increasing, as evidenced by Sony [189] and Prophesee commercializing silicon retina cameras [190], spiking neural networks are typically used for the analysis of continuous data from conventional sensors. As a result, spike encoding of these signals is required to provide sparse, event-based input data. There are two approaches to spike generation: in the first, the neural response to a continuous signal is produced using specific neuron models and characteristics, similar to what is defined as the Representation Principle of the Neural Engineering Framework (NEF) [191]; in the second, a continuous signal is transformed into discrete spikes using a variety of possible algorithms. In this section, we focused on the second method, which assures that neuro-inspired tactics can be fully used in the context of IoT applications even in the absence of dedicated neuromorphic hardware.

Rate Coding and Temporal Coding are the two main categories of spike encoding algorithms, with significant differences in the number of degrees of freedom allowed in the encoding: in Rate Coding, a signal is encoded by the number of spikes per time unit, whereas Temporal Coding includes a variety of approaches. Figure 3.4 shows an example of spike train production. In the following, we will go over the encoding approaches employed in our study, each of which falls into one of the two aforementioned coding categories.

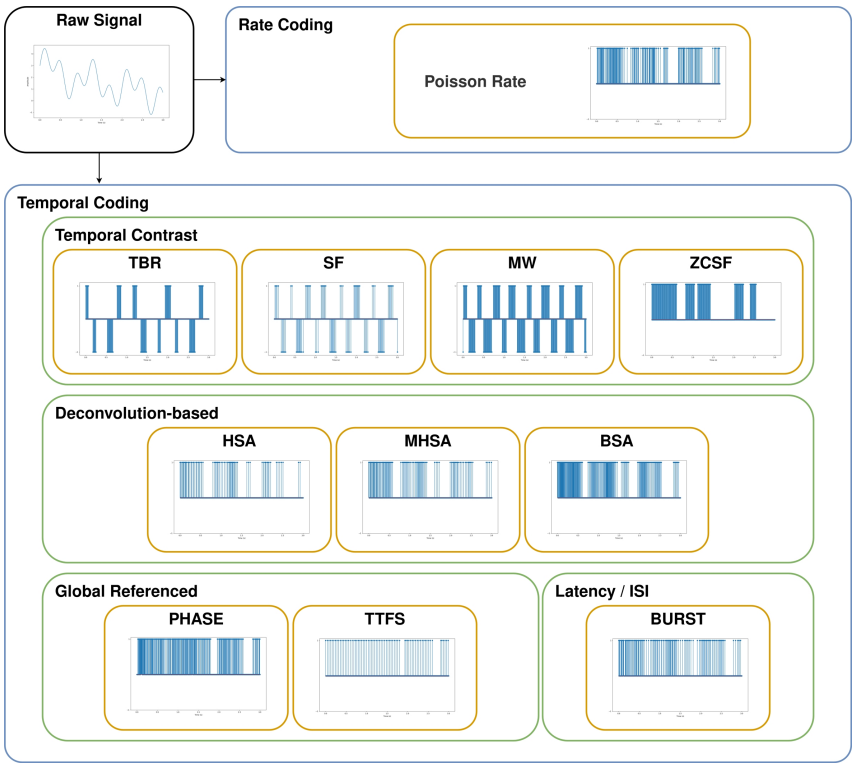

Fig. 3.4 Example of the spike trains generated by each of the examined encoding techniques from an arbitrary input signal. [1]

3.2.1 Rate coding

Rate coding, which is widely used for ANNs due to its simplicity and robustness, provides a mechanism for information representation based on the number of spikes per unit time [48].

Among the many different rate coding algorithms [156], we focused on the Poisson rate method, based on the Poisson distribution. With this approach, the likelihood of having $n \in \mathbb{N}$ spikes in a time interval Δt via this process is:

$$P_n(\Delta t) = \frac{(r\Delta t)^n}{n!} e^{-r\Delta t} \tag{3.1}$$

where $r \in \mathbb{R}$ is the value to be encoded, corresponding to the *spike rate*.

From an operational standpoint, the algorithm can be implemented as follows [192]:

1. Define the time interval Δt during which the spike train will be generated.
2. Generate a random number sequence $x \in [0,1] \subset \mathbb{R}$;
3. Define spike timings t_i from $t = 0$ as:

$$t_i = t_{i-1} + ISI_i \qquad \text{for } i \geq 1 \tag{3.2}$$

where

$$ISI_i = \frac{-\log(1 - x_i)}{r} \tag{3.3}$$

is the ith inter-spike interval, defined as the ith time interval in which the probability of having $n = 0$ spikes is equal to x_i;

4. Generate a spike at every time step t_i until $t_i > \Delta t$.

3.2.2 Temporal coding

The encoding processes in the temporal coding family encompass diverse information representation strategies. Aside from the quantity of spikes per unit time, one distinctive aspect of temporal coding is the ability to account for accurate spike timing to transport information [193]. Furthermore, factors such as relative spike timing and temporal distance between spikes can be exploited. Temporal Contrast, Deconvolution-based, Global Referenced, Latency/ISI, and Correlation and Synchrony are the five kinds of temporal coding algorithms that may be identified based on which of these qualities is taken into account [156].

Temporal contrast

This category's algorithms are primarily concerned with signal fluctuations in time, and they are used to generate spikes with either a positive or negative sign. Because the fundamental feature encoded by this class is time-based variation, it is not well suited for solely spatial data, such as still photographs. Audio signals [117], electromiography data [194], voice recognition [195], failure prediction based on machine vibrations [188], and robotic Braille reading [3] are examples of applications.

Threshold-based representation The Threshold-Based Representation (TBR) algorithm is a constituent element of the Temporal Contrast category [196]. It encodes data by creating spikes based on the absolute signal fluctuation in relation to a specified threshold. The key actions in this approach are as follows:

1. Given an n-channel signal, fluctuations along each channel are analyzed between consecutive timesteps.

2. A certain threshold is specified for each channel as:

$$\mathit{Threshold} = \text{mean}(\mathit{Variation}) + \gamma \cdot \text{std}(\mathit{Variation}) \tag{3.4}$$

   where γ is a configurable parameter that affects the *Variation* values between $-\mathit{Threshold}$ and $+\mathit{Threshold}$ by directly reflecting on the amplitude of the noise-reduction band: e.g., greater values of γ lead to a wider threshold band, and a smaller number of spikes. Different ranges of values for γ can be identified, depending on the noise level to be filtered out:

   - $\gamma = 0$: all signal fluctuations are retained, and the threshold is defined as the mean of their values;
   - $0 < \gamma \leq 1$: noise in the signal is not a serious issue, but fine signal changes are not required to preserve the information content;
   - $\gamma > 1$: there is meaningful noise present, and its impact must be reduced to generate meaningful spikes.

3. The spike train timesteps are defined by dividing the Δt interval in which the spikes are generated by the length (L) of the input signal;

4. If *Variation* exceeds *Threshold* in absolute value, a spike is generated with polarity given by the sign of both *Variation* and *Threshold*.

Moving window Moving Window (MW) employs the same underlying notion of employing a threshold value as the TBR algorithm. However, unlike the previous encoding approach, this threshold is used in conjunction with a number known as *Base*, which is defined as the mean of the signal along each channel inside a

sliding window of specified length:

$$Threshold = \text{mean}(Variation) \tag{3.5}$$

$$Base = \text{mean}(Signal[1 : Window]) \tag{3.6}$$

Furthermore, unlike TBR, the requirement for spike emission depends on the signal's value rather than its variation in time. A positive spike is formed when the signal exceeds the value $Base + Threshold$, while a negative spike is produced when the signal is less than $Base - Threshold$. With the introduction of a sliding window along the signal, this spike generating process appears to be more resilient to noise than TBR [197].

Step-forward The Step-Forward (SF) technique, proposed by Kasabov et al. [197] as an enhancement over the encoding used for the artificial silicon retina in Delbruck and Lichtsteiner [196], likewise depends on the concept of an iteratively updated baseline value. Similarly to MW, *Base* and *Threshold* are used to compute said baseline, and their definitions for each signal channel are as follows:

$$Threshold = \frac{\text{mean}(Jump)}{\gamma} \tag{3.7}$$

$$Base = Signal[1] \tag{3.8}$$

where *Jump* is an array of maximum-to-minimum differences for each channel and γ is an adjustable parameter. As is the case for TBR and MW, SF produces both positive and negative spikes: the former occur when the signal exceeds the value $Base + Threshold$, whereas the latter occur when the signal is less than $Base - Threshold$.

Zero-crossing step-forward (ZCSF) By taking zero-crossings into account, we can derive an alternate implementation of SF [198, 199]. The Zero-Crossing Step-Forward (ZCSF) method inherits the *Threshold* specification from SF but does not include the *Base* value, which is replaced by a half-wave rectifying behavior induced by the condition $Signal > 0$. With ZCSF, spike emission happens for all positive signal levels greater than *Threshold*, resulting in exclusively positive spikes, as opposed to the preceding encoding techniques.

Deconvolution-based

This class of encoding techniques, which includes the Hough Spiker Algorithm (HSA) [200] and the subsequent Modified HSA and Ben's Spiker Algorithm (BSA) [201], stems from the inverse problem of reconstructing an analog signal from a spike train using a finite impulse response filter (FIR). The algorithms in this class, in fact, allow analog-to-spike conversion by reversing said operation and applying the convolution function in a subtractive approach [200]. They produce unipolar spikes, like in the case of ZCSF.

Hough Spiker Algorithm The HSA performs progressive subtraction by comparing the value of the analog signal to the result of a specified convolution operation. If the level of the signal to be encoded exceeds this value, the convolution value is subtracted. As a result, for each signal channel, the main iterative step in the Hough Spiker Algorithm is:

$$Signal[i+j-1] = Signal[i+j-1] - filter[j] \tag{3.9}$$

where i represents the signal's time steps, *filter* the convolution result, and j its value indices. As the convolution function in our investigation, we used a rectangle window.

Modified Hough Spiker Algorithm The Modified HSA maintains the HSA's core idea of a subtractive, deconvolution-based procedure, but differs by the introduction of a *Threshold* value. The operation in Eq. (3.9) is executed at any time step where $error \leq Threshold$. This *error* is the result of an accumulation that occurs at each time step where the input signal is less than the convolution function. For each signal channel, the accumulation is defined by the equation:

$$error = error + (filter[j] - Signal[i+j-1]) \tag{3.10}$$

Ben's Spiker Algorithm (BSA) With respect to the Modified HSA, the Ben's Spiker Algorithm adds two cumulative error metrics for each signal channel alongside

the *Threshold* value:

$$error_1 = error_1 + \text{abs}(Signal[i+j-1] - filter[j]) \tag{3.11}$$

$$error_2 = error_2 + \text{abs}(Signal[i+j-1]) \tag{3.12}$$

The criterion to be tested before applying Equation (3.9) in the original study discussing BSA [201] is that $error_1$ does not exceed the value $error_2 - Threshold$. However, in this work, we refer to the implementation proposed by Petro et al.[202], where the condition is adjusted as follows:

$$error_1 \leq error_2 \cdot Threshold \tag{3.13}$$

Global referenced

This third class of temporal coding algorithms includes approaches whose spike generation process is based on some global temporal characteristic of the input signal. The time difference with respect to an oscillatory reference is the relevant feature in the case of Phase Encoding [203], whereas Time-to-First-Spike (TTFS) measures the time since the stimulus began [204, 205].

Phase encoding Montemurro et al. [206] showed the possibility of effectively establishing an encoding system based on a phase evaluation with regard to an oscillating reference. In our work, we use the approach described by Kim et al.[163], in which the binary representation of the input by β fractional bits is used as the oscillatory reference after rectifying and normalizing the signal into the range $[0,1]$ for each channel.

Time-to-first-spike Different ways for applying Time-to-First-Spike encoding depending on the membrane potential threshold specification have been examined by Rueckauer and Liu [164]. In our study, we use an exponentially decaying function to build a dynamic threshold, similar to Park et al. [207]:

$$P_{th}(t) = \vartheta_0 e^{-t/\tau_{th}} \tag{3.14}$$

where ϑ_0 is a constant and τ_{th} indicates the membrane potential decay time. For the experiment described here, we utilized $\vartheta_0 = 1$ and $\tau_{th} = 0.1$. In comparison to

other solutions, we used a bitwise approach similar to Phase Encoding, eventually producing a bin-based binary-like representation of the input signal values. Although this additional phase increases the overall number of spikes, it can result in more robust encoding typical of spike bursts [208].

Latency/ISI

It is well known that neural communication via bursts of spikes, i.e. increasing the number of spikes delivered to carry information about a given event from 1 to N, improves dependability. However, the latency between these N spikes, known as the inter-spike interval (ISI), can also be used to effectively encode information [209]. As a result, the Latency/ISI encoding algorithm class is established, with Burst Encoding as a representative example.

Burst encoding Burst Encoding, as stated by Guo et al. [48], is a technique that takes advantage of two different time-based properties of a single spike train. In fact, the algorithm is based on both the number of spikes and the ISI, and it employs three variables: N_{max} is the greatest number of spikes in each burst, t_{min} is the minimum ISI, and t_{max} is the maximum ISI. The number of spikes and their relative spacing are defined as follows, using these variables and the additional parameter *rate*, which is obtained by normalizing each signal channel:

$$SpikeNumber = \lceil rate \cdot N_{max} \rceil \tag{3.15}$$

$$ISI = \begin{cases} \lceil t_{max} - rate(t_{max} - t_{min}) \rceil & \text{if } SpikeNumber > 1 \\ t_{max} & \text{otherwise} \end{cases} \tag{3.16}$$

Burst Encoding, like the last two algorithm classes, produces spike trains with a single polarity.

3.2.3 Comparing different classes of encoding algorithms

While the encoding stage is crucial and required for using digital input data with an SNN, selecting the best encoding approach for the signal to be analyzed can increase accuracy. Figure 3.5 compares the median accuracy attained by various families of encoding algorithms when paired with various channel separations,

feature extraction methods, and network architectures. The Temporal Contrast class had the highest accuracy for the FSD, with a median of around 91.00 % (Fig. 3.5 A). The Global Referenced class, on the other hand, has the lowest median result (about 53 %), with a considerable variance. This is because the two algorithms in the Global Referenced family perform extremely differently: while Phase Encoding produces respectable results (median 77.5 %, with a maximum of 93 %), TTFS yields very low accuracy (median 35 %, with a minimum of 8 %). This is most likely due to the lower number of spikes produced by TTFS, resulting in insufficient network stimulation: we will go over this notion in greater detail in Section 3.2.3.

While the different algorithms produce quite disparate results on the WISDM dataset, the median accuracy aggregated by algorithm class remains around 48 % for all classes except Rate Coding, which produces the worst median results at 21.67 % and the overall minimum at 5 %. Burst Encoding achieves the best median result with 55 % accuracy, while the ZCSF algorithm combined with a 16-channel Butterworth filter and a C6-C12-F2 network architecture achieves the best single result with 93 % accuracy.

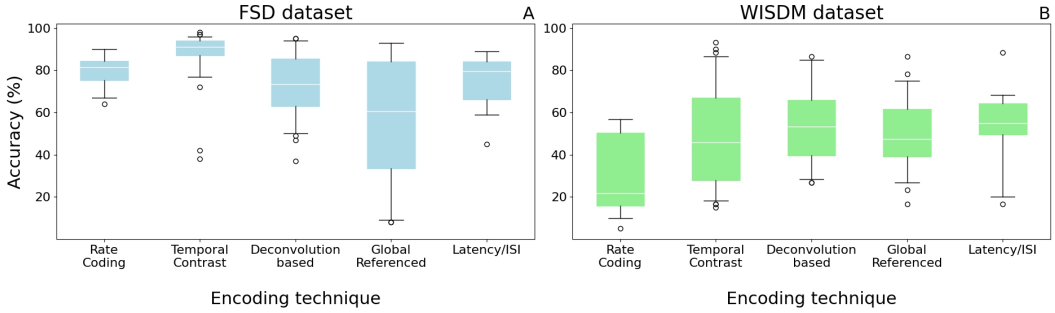

Fig. 3.5 Median accuracy values of each encoding class for various network architecture, filter type, number of channels, and feature extraction bins combinations. [1]

Spike density

Spike density is the number of spikes produced per unit time. When developing a neuromorphic system, this parameter should be carefully examined: a lower spike density leads to energy savings due to less communication between network layers, but a too low number of spikes can prove insufficient to successfully encode information without loss. Our observations reveal that the encoding strategy has

a relevant influence on this quantity: given identical input data, the differing implementation logic of each approach results in distinct spike densities. Figures 3.6 and 3.7 depict the distribution of spikes created by each coding approach following channel separation by a Butterworth (left column) or gammatone (right column) filter bank. The Deconvolution-based family of encoding methods (HSA, MHSA, BSA) generates the maximum spike count in all scenarios studied.

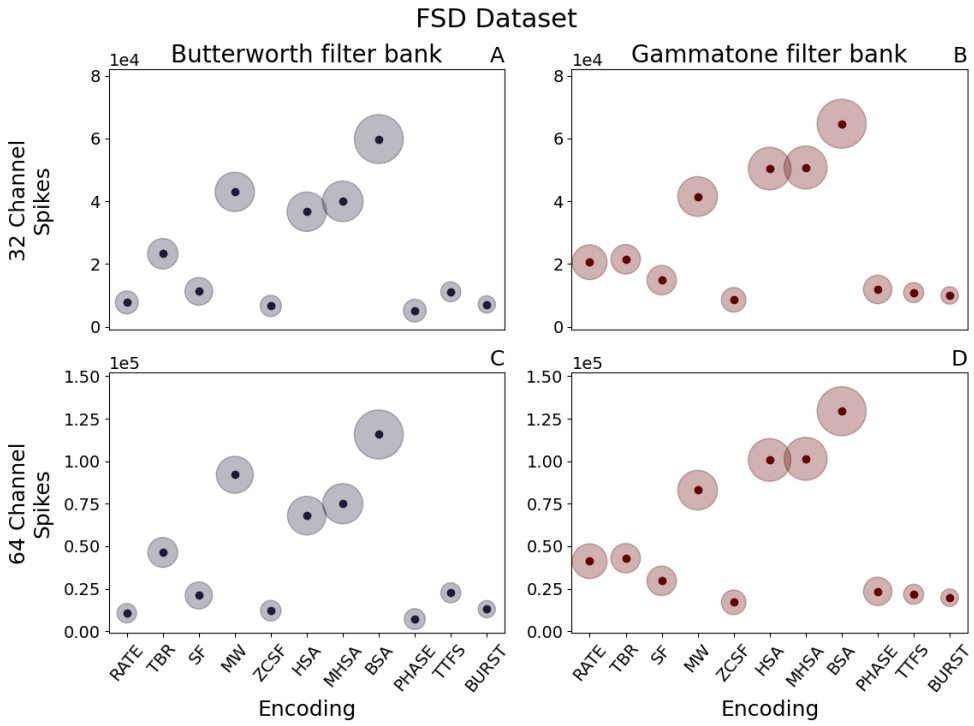

Fig. 3.6 Median spike counts per sample generated by various combinations of encoding technique, channel count, and filter type for the FSD dataset. [1]

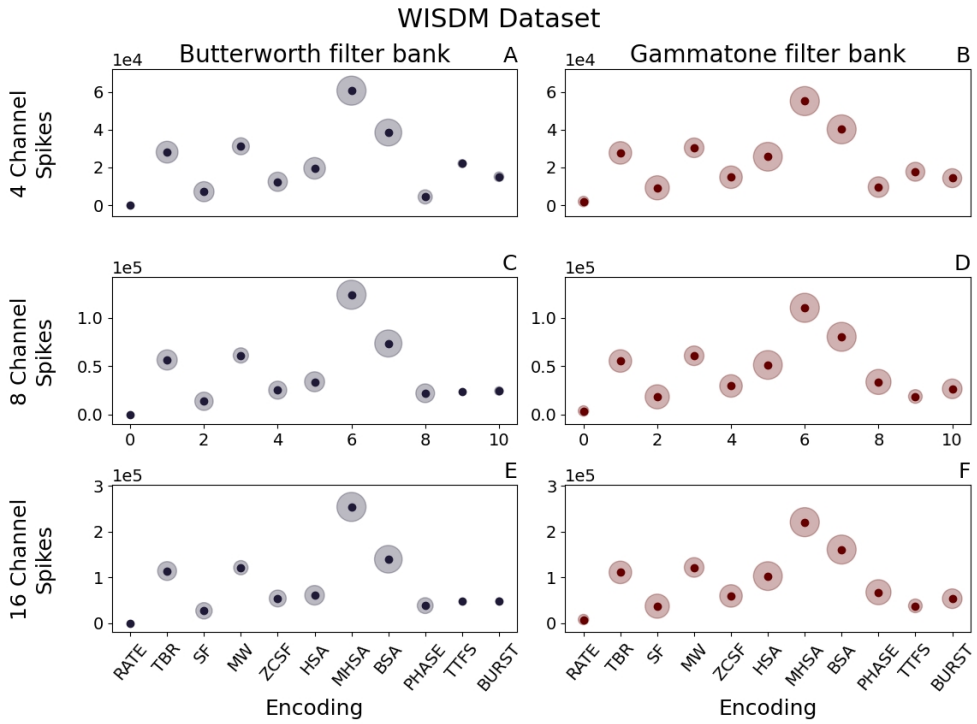

Fig. 3.7 Median spike counts per sample generated by various combinations of encoding technique, channel count, and filter type for the WISDM dataset.

The number of spikes produced is also affected by including the refractory period in the encoding model. We conducted early experiments with different values for the refractory period τ_{ref}: 3 ms, 2 ms and 1 ms. In every instance, using this parameter in the encoding step results in an extreme reduction in the spike count, preventing the SNN layers from being appropriately stimulated. This drastically reduces classification performance: the median test accuracy for the FSD across all architectures, channel decomposition, and encoding schemes is 22.00 %. In the case of WISDM, the value of τ_{ref} is constrained by the low sampling frequency $f_s = 20$ Hz of the dataset signals, resulting in a lower bound of 50 ms. Because of the performance degradation we saw even with modest values for τ_{ref}, all findings provided are obtained with $\tau_{ref} = 0$.

Discussion

The performance of the encoding techniques under consideration is affected by the frequency of the input data. With a broader bandwidth for the middle-

frequency FSD dataset, more features may be retrieved from the signal, and it is easier to identify the encoding classes that enable more accurate classification. For very-low frequency data, such as the WISDM dataset, there is no clear advantage for one algorithm class over another; however, several configurations featuring Temporal Coding, such as ZCSF encoding, vastly outperformed Rate-based Coding, demonstrating that, while the algorithm for the encoding must be carefully chosen, Temporal Coding has a higher ability to extract features suitable for analysis in the neuromorphic domain. We saw especially strong classification accuracy with Phase Encoding, with Butterworth filters reporting 83.00 % and gammatone reaching 93.00 %. This is because redundant components in the frequency response result in a higher number of spikes for this algorithm class, allowing it to encode more information. We also discovered that the spike count produced by each coding must be sufficiently high to appropriately excite all layers of the downstream SNN; thus, spike count reduction aiming at power savings must be carefully balanced with information preservation. Figure 3.8 depicts a quantitative and comparative summary of all the evaluated encoding strategies. Each of them is defined by five metrics: Shannon entropy $\mathscr{S}$ [210], mutual information of the encoded signal with the original input [211] normalized with respect to entropy $\mathscr{MI}_{\mathscr{S}}$, sparsity $\mathscr{HS}$ [212], spiking efficiency $\mathscr{E}$ [193], and computational complexity $\mathscr{O}(f)$.

Table 3.1 A summary of encoding strategies, with emphasis on their performance in relation to the type of input data. A ✓ indicates that the technique is highly suitable for the purpose, whereas – shows that the technique has some downsides and an ✗ indicates that it is not suitable for the purpose. [1]

Encoding class and technique			Temporal data		Spatial data[1]
			Very low frequency	**Middle frequency**	
Rate coding	**Poisson Rate**		✗	✓	✓
Temporal Coding	**Temporal Contrast**	TBR	✓	✓	✗
		SF	✓	✓	✗
		MW	✓	✓	✗
		ZCSF	✓	✓	✗
	Deconvolution-based	HSA	–	–	✗
		MHSA	–	–	✗
		BSA	✓	–	✗
	Global Referenced	PHASE	✗	✓	✓
		TTFS	✗	✓	✓
	Latency/ISI	BURST	✗	✓	✓

[1] Guo et al. (2021); Auge et al. (2021)

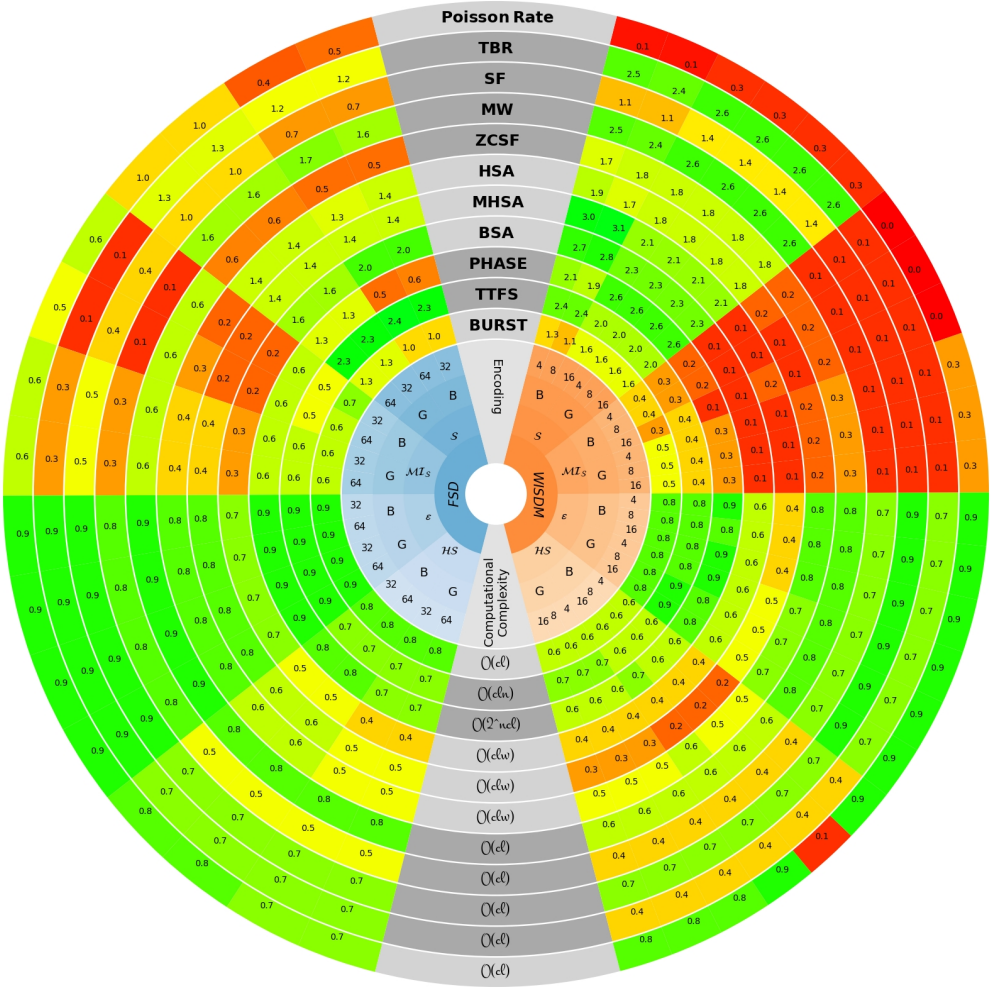

Fig. 3.8 Each encoding approach is characterized along a specific ring of the circle-shaped graph. The computational complexity is reported in the bottom center section by the quantities l (signal length), c (number of channels), n (length of the bitwise representation), and w (width of the convolution function). The results produced using FSD data are on the left, while those obtained with the WISDM dataset are on the right. The four signal-related metrics, $\mathcal{S}$, $\mathcal{MIS}$, $\mathcal{HS}$ and $\mathcal{E}$, are shown in a mirrored configuration with respect to the circle's vertical symmetry axis. The results of the two filter types, Butterworth (B) and Gammatone (G), are reported for each of them based on the number of channels utilized to separate the original signal. [1]

A summary of suggestions matching each encoding approach to the input frequency of the time-variant input data is presented in Table 3.1. This analysis serves as a first step toward a more thorough evaluation of the tools available for signal encoding in the neuromorphic domain. The need for directions for the solution of engineering challenges in the realms of IoT and Industry 4.0 will increase as commercial interest in this field of study grows.

3.3 Additional analysis: Event-based encoding of tactile sensor data

For the acquisition of the Braille dataset (Section 2.2.1), the authors used a sigma-delta modulator [213] to encode the frame-based digital input into temporally sparse streams of spikes. An offline preprocessing phase transforms each original stream of frames from a 12-taxel time sequence to 24 binary event-based channels, simulating an event-based touch sensor. Figure 3.9A shows how threshold (ϑ) crossings result in ON or OFF digital events for an increase or decrease in pressure, respectively: the upper section of the graph depicts 600 ms of sensor measurements from a single taxel while sliding, while the bottom section displays the generated events for the ON (green) and OFF (red) channels, with rising threshold values resulting in a decrease in the number of events. Figure 3.9B compares a sequence reconstructed from event-based data to the original; the same frame-based sequence is reconstructed using a different threshold in each subgraph. This analysis highlights how increasing the threshold improves compression, but it also increases the reconstruction error. The highest implemented precision for lossless conversion is the one with the threshold value $\vartheta = 1$; larger values of ϑ lead to greater sparsity, a lower data rate, and improved efficiency, at the expense of information loss and decreased accuracy.

The authors compared the reconstructed temporal sequences to the original frame-based signal to characterize the event-based datasets. The same analysis was again conducted after performing the temporal binning process, required to prepare the data for clock-driven computation (see Section 4.2.2). Table 3.2 summarizes the results by reporting the mean number of events, compression ratio γ with respect to encoded data at $\vartheta = 1$, and reconstruction MSE values ϵ both before and after the binning step for each threshold.

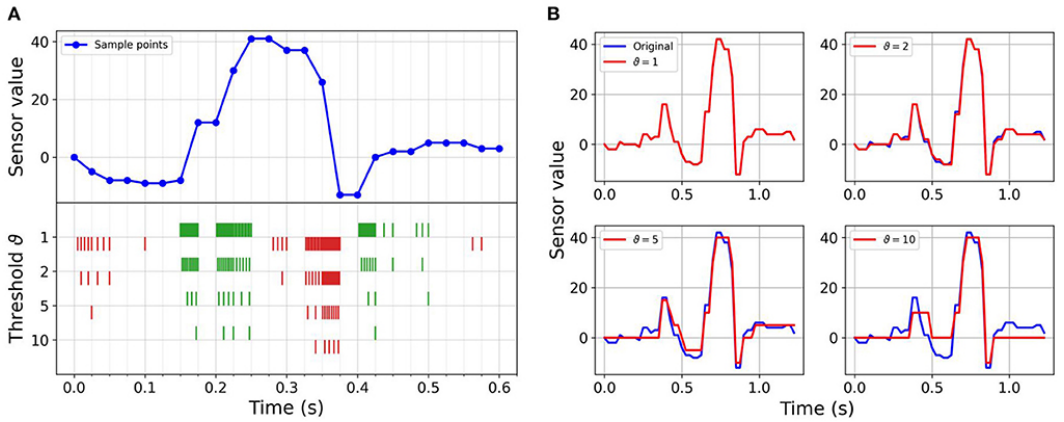

Fig. 3.9 Event-based sample encoding and reconstruction: **(A)** Sensor reading sequence for a sample letter, with the respective sigma-delta modulated spikes. **(B)** Reconstructed sequences from event-based data compared to the original sequence for a full letter reading. [3]

Table 3.2 Event-based encoding characterization for each of the generated datasets at different threshold settings. [3]

	Before time binning			After time binning			
Threshold (θ)	**Events**	**Comp. ratio** (γ)	**MSE** (ε)	**Bin size** (ms)	**Events**	**Comp. ratio** (γ)	**MSE** (ε)
1	87.6	1	0	5	58.1	1.5	39.5
2	38.0	2.3	0.35	3	35.5	2.5	12.5
5	10.5	8.3	3.70	3	10.5	8.3	4.1
10	3.4	25.7	12.29	5	3.4	25.7	12.3

Compression ratio γ is defined as the number of events at perfect encoding ($\theta = 1$) divided by the number of events at each higher threshold value. Values of the reconstruction error ϵ are calculated per reconstructed frame by MSE. Mean events are calculated per sample.

Signal reconstruction before time binning

Starting from zero, the signal was reconstructed from the event stream by increasing (in the case of an ON event) or decreasing (OFF event) the output by an amount equal to the threshold used in the encoding. Figure 3.9B depicts the reconstruction values for a single sample and taxel at various threshold settings. The compression ratio γ is expressed as the quantity of events at $\vartheta = 1$ divided by the quantity of events at each threshold level. For each event-based dataset, the reconstruction error ϵ is the mean square error between the original series and its reconstructed frame-based series.

The study of the reconstructed frame-based signal shows that increasing the threshold reduces the number of events considerably, worsening the reconstruction error. However, the compression ratio γ grows faster than the reconstruction error ϵ, indicating that the event-based dataset gains sparsity at the expense of information content.

Signal reconstruction after time binning

The segmentation of data into time bins is necessary for certain classification network implementations. Therefore, we tallied the total number of events after time binning to measure the impact of time binning on the different encoded datasets. Regardless of time binning, a lower encoding threshold always corresponds to a higher total number of events, as illustrated in the top panel of Figure 3.10A. For this reason, higher encoding thresholds result in less signal degradation in the temporal binning phase; while increasing the encoding threshold considerably reduces the number of events, lower-threshold encoding loses many events with increasing *time_bin_size*, as shown in the bottom panel of Figure 3.10A.

The reconstruction of the frame-based signal from the event stream was conducted for each feasible *time_bin_size* of the event stream and for each threshold value. The reconstruction error ϵ is determined by the encoding threshold on the one hand, and the implemented time binning on the other. Figure 3.10B depicts the results, with markers at the optimal *time_bin_size* for each encoding threshold. With increasing *time_bin_size*, there is a significant rise in reconstruction error ϵ. This can be explained by the loss of events when many events fall within a single time bin, which increases the reconstruction error by introducing an accumulating offset, as illustrated in Figure 3.10C. Furthermore, the discriminative strength of amplitudes is lost, resulting in identical amplitudes for minor and major changes following reconstruction. As seen in Figure 3.10D, as the threshold increases, ISIs generally become longer, due to the growing sparsity of spikes. The effect of time binning then diminishes, while the inaccuracy produced by higher encoding thresholds becomes more significant. Overall, higher thresholds are more resistant to the effect of *time_bin_size*, but they are also less capable of reflecting temporal dynamics.

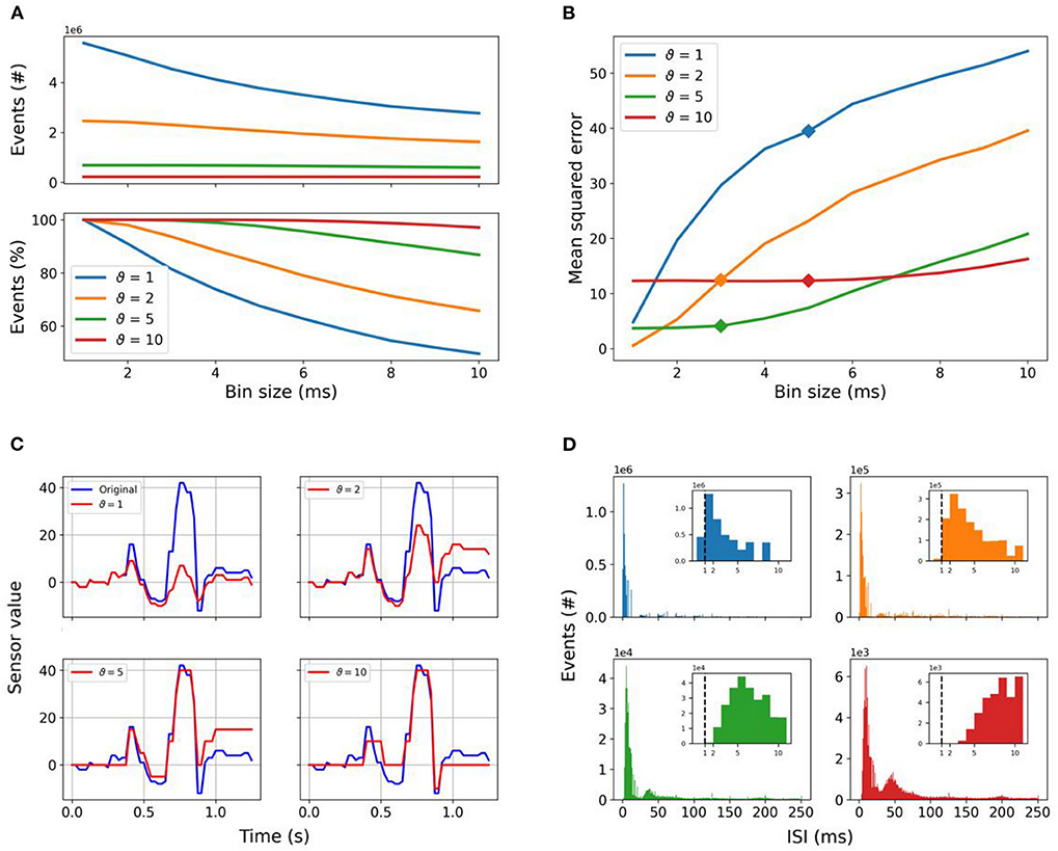

Fig. 3.10 Spike encoding: **(A)** Top panel: total number of events counted in the entire dataset with respect to the threshold and *time_bin_size*. Bottom panel: relative number of events detected in the dataset with *time_bin_size* = 1. **(B)** MSE of the reconstructed time-binned signal as a function of the *time_bin_size* for each encoding threshold. The markers emphasize the final *time_bin_size* selected to preprocess the event stream. **(C)** Frame-based signal reconstructed from the event stream after time binning with a bin size of 5 ms for all given thresholds. **(D)** The same color coding as in **(A,B)** is used to represent the number of events as a function of the ISI, with fixed *time_bin_size* as reported in Table 3.2. The insets demonstrate the detail at ISI values equal to the *time_bin_size* employed, with the vertical dashed line showing the minimal temporal resolution of 1 ms. [3]

3.4 Chapter summary

Whether using event-based or digital sensors, the issue of selecting the proper encoding is of great importance within the neuromorphic pipeline. Performing a thorough comparison of the available encoding techniques, we found a dependency between the quality of the encoding and the shape of the input data: its frequency spectrum and bandwidth of the input especially make a difference in which encoding techniques perform better than others. Additionally, the spike count generated by an encoding technique influences the performance of the downstream SNN: too long intervals between spikes — i.e., a reduced number of spikes — lead to insufficient stimulation of the deep layers of the network and inaccurate performance. This results poses a lower bound to the potential of SNNs to limit power consumption by using sparse communication.

The encoding used for gathering the Braille dataset highlights a different aspect of the issue. Despite not having access to an event-based tactile sensor, the authors effectively created one by surrounding the digital sensor and a software sigma-delta encoder with a conceptual black box, and recording its output. This simulated event-based sensor effectively recreates the design process behind a proper event-based sensor: an efficient encoder must be carefully prototyped and tested, ensuring that the output format retains sufficient information about the sensor input. In summary, the problem of encoding remains relevant even when dealing with event-based sensors; meanwhile, the use of digital sensors coupled with software encoders is not only a more accessible solution, but it can also be useful for prototyping the qualities of different coding techniques and tuning their interactions with other elements in the neuromorphic pipeline.

Once the input data has been gathered and encoded, it can be applied to the inputs of a SNN. In Chapter 4, we will begin to examine the issue of building an efficient neuromorphic application, starting from its abstract building blocks: neuron models and network architectures.

Chapter 4

Neural models

Deep neural networks have seen tremendous success in a variety of disciplines during the last decade. This accomplishment has been enabled by the massive availability of annotated data, as well as the spread of high-performance computer equipment such as GPUs. On the other hand, DNNs are extremely resource-intensive when it comes to energy usage, quantity and quality of training data, and computing effort. Additionally, although ANNs have demonstrated remarkable success in a wide range of applications, they are still constrained in their ability to cope with temporal information. These factors have drawn researchers' attention to biological neural networks, which, in comparison to ANNs, use fundamentally different architectures, neural computations, and learning rules to attain extraordinary energy efficiency and online learning abilities. Within Spiking Neural Networks (SNNs), biologically plausible neuron models such as those explored in Section 4.1 communicate with one another in real time through discrete electrical signals or spikes. Although SNNs have not yet matched the performance of DNNs, there are specific tasks for which the gap is closing, and in nearly every case, SNNs use significantly less energy to operate [214].

This chapter delves into the algorithms that underpin SNNs. Neuromorphic computing has progressed thanks to significant advances in neuron modeling, learning methods, and network topologies. Because event-based computing introduces inputs with time domain information, it is necessary to rethink the learning techniques heretofore used with non-spiking ANNs.

4.1 Neuron models

Spiking Neural Networks (SNNs) are made up of spiking neurons, linked together by synapses. Biological neurons interact through the transmission of voltage spikes, which determine the activity of recipient neurons: when a neuron reaches a certain voltage polarization, it produces a spike and then reverts to its reset state, or resting potential [215]. This allows neurons to pass around data asynchronously. Since the energy consumed by a neuron depends on the amount of stimulation it receives over time, biological brains are extremely energy efficient. This appealing characteristic is what compels neuromorphic computing to emulate the mechanics of the animal neural network [216].

The field of neuron modeling has been stimulated by neuroscience researchers seeking to generate accurate biological simulations in order to gain a better understanding of the brain. The Izhikevich model [217] is an example of a neuron model that tries to bridge biological plausibility while maintaining computational efficiency; the popularity of this model in the neuroscience world has led to it being integrated into neuromorphic hardware accelerators such as ODIN [218]. On the other hand, neuromorphic engineers interested in brain-inspired computing for its power efficiency and unique time-encoding properties are willing to deviate from biological plausibility in order to obtain better SNN performance or higher compatibility with the hardware platforms that support them. For instance, most widespread optimization strategies such as transfer learning and surrogate gradient techniques perform better with simple models such as Integrate and Fire.

This variety of interests and applications in the neuromorphic computing field has led to the development of a large variety of neuron models. The majority are composed of a few common elements: an internal state, that keeps track of the stimulation received; a threshold value for said internal state, which triggers the firing output; and an optional refractory period τ_{ref}, in which the neuron does not fire and remains insensitive to stimuli [219].

4.1.1 Leaky Integrate and Fire (LIF)

The Leaky Integrate and Fire (LIF) model is by far the most popular and supported archetype along all neuromorphic platforms and applications. It is the simplest

and least computationally demanding representation, originally codified by neuroscientist Louis Lapicque in 1907 [220]. First, the internal state of the neuron is its *membrane voltage* V_{mem}, which evolves as described by the time derivative of the capacitance law with respect to a variable input current I_{syn}:

$$I_{syn}(t) = C_{mem} \cdot \frac{dV_{mem}(t)}{dt}. \tag{4.1}$$

The above equation describes the leak-free version of the model, simply referred to as Integrate and Fire (IF). When a current appears at the input, V grows with time until it hits a threshold voltage V_{th}. As soon as this happens, the neuron emits a spike, then resets its membrane voltage to a preset resting potential V_{rest}. Optionally, the model can remain at the resting potential for a *refractory period* τ_{ref}, reflecting more accurately the biological working of the neuron.

In the LIF model, a leak term $\frac{V_{mem}(t)}{R_{mem}}$ is introduced which allows the membrane potential to slowly discharge over time, reflecting the diffusion of ions that naturally occurs through the neuronal cell's membrane. This not only matches more closely the biological working of the neuron, but it also strengthens the temporal dimension of the model, making repeated and frequent inputs more relevant than inputs that are spaced apart in time. The LIF formula changes to

$$I_{syn}(t) - \frac{V_{mem}(t)}{R_{mem}} = C_{mem} \cdot \frac{dV_{mem}(t)}{dt}. \tag{4.2}$$

The LIF model has long been a staple of computational neuroscience [221], and it is currently a favorite method of building SNNs, not only for its computational and conceptual simplicity, but also because it allows compatibility with existing ANN training methods. In fact, when $\tau_{ref} = 0$, $\tau_{mem} = 1$, $R_{mem} = 1$, and $V_\vartheta = 1$, the firing rate of a LIF neuron is similar to that of the ReLU activation function in ANNs [214]: this allows to convert trained ReLU-based ANNs to an equivalent spiking version, side-stepping the difficulty of training networks directly in the spiking domain [222]. In general, the LIF model does not account for neural plasticity, and several modified models have been proposed in an attempt to maintain the LIF's computational simplicity while adding adaptive terms that would allow for more powerful cognitive capabilities, especially in time-dependent deep learning applications that require the neural network to withhold memory of a prolonged input. One such model is the Adaptive LIF.

Adaptive LIF (ALIF)

The Adaptive LIF (ALIF) model modifies the LIF neuron with an adjustable threshold that is increased after each emitted spike, then decays exponentially with a time constant τ_a. Bellec et al. [223] demonstrated that this adaptive spiking neuron model improves SNN performance. Additionally, the ALIF has attracted much attention because its use in recurrent SNNs enables a powerful new training method called *e-prop*, which aims to reproduce the performance of back-propagation through time (BPTT), the most well-known training approach for recurrent neural networks in traditional machine learning [224]. This advancement has shown enough promise that future next-generation neuromorphic platforms such as SpiNNaker 2 [122] and Loihi 2 [225] have announced support for some type of ALIF.

4.1.2 Multi-compartmental neurons

Multi-compartmental neuron models differ from LIF and other simple models such as Izhikevich because they attempt to replicate not only the pointwise behavior of a neuron but also its spatial configuration. The division of the neuron into compartments allows to model the behavior of the dendrites separately from that of the cell body (*soma*). The Urbanczik-Senn model [226], for instance, is composed of 2 neuron compartments (somatic and dendritic) and 2 synapse types (static and plastic); this implementation aims to emulate spike-timing-dependent plasticity to enable supervised, unsupervised, and reinforcement learning. Another example is the pyramidal model, which reproduces the topology of the biological pyramidal brain cell: with a pyramidal-shaped soma and two separate dendritic trees. This model can be described by 3 compartments (somatic, basal dendritic, and apical dendritic) and 3 synapse types (static apical, plastic apical, and plastic basal).

As things currently stand, the real-time (and faster-than-real-time) simulation of multi-compartmental neurons is an important target for neuroscientists. Learning and development occur over lengthy durations in real brains, making long-term investigation of these traits a substantial scientific problem; computer models capturing accurate neuron and synapse dynamics can enable the exploration of long-term phenomena such as lifelong learning or the exploration of

brain disorders. While many analog or mixed analog-digital neuromorphic hardware do not support multi-compartmental neurons, focusing instead on ASIC implementation of simpler and more popular point-neuron models, there is an effort to implement more complex models via software modifications on more flexible digital platforms such as SpiNNaker [227].

4.2 Spiking Neural Network architectures

SNNs incorporate the biological concept of communication sparsity, and they are very compatible with time-varying signals because of their functional similarities to biological neural networks. On the other hand, direct training of SNNs poses significant challenges, because of the complicated inter-neuron dynamics and the non-differentiability of spiking signals [214]. For this reason, the most widespread method of developing SNN architectures for practical applications is ANN-to-SNN conversion; we will explore an example of this type of network in Section 4.2.1. The development of ad hoc SNN structures is still a subject of lively debate and research, and we will see some examples of proposed solutions in section 4.2.2. The experiments detailed in this section will include architectures that are particularly suited for the treatment of time-varying signals, either because of proven functionality and good performance (as in the case of the spiking CNN), or because of promising innovations brought by a new paradigm (as in Recurrent Spiking Neural Networks).

4.2.1 ANN-to-SNN conversion: a spiking CNN

A convolutional layer, one of the most common ANN building blocks, is a type of fully-connected layer that enables optimal analysis of data with a grid-like layout, such as pictures, by making use of the spatial correlation of the input signal. The representational qualities of early layers in CNNs are comparable to the response properties of neurons in the primate's primary visual cortex (V1) [214]. The substantial improvement achieved by the CNN architecture in the ImageNet classification problem [228] remains one of the most prominent breakthroughs in the field of deep learning.

The analysis of time-varying signals can be accomplished using either convolutional or recurrent architectures. In previous work by the author and colleagues [2, 1], the convolutional neural network (CNN) was chosen as the basic architecture to undertake initial training and enable transfer learning of a spiking version of the same network. This decision was based on the state-of-the-art results obtained by this type of network in the classification of audio signals captured by a neuromorphic cochlea [147]; subsequent trials [2] validated the CNN's computational and energetic efficiency in the analysis of time-varying data.

Architectural parameters

Starting with the work published in Dominguez-Morales et al. [147] and performing structural hyperparameter optimization, we created multiple test configurations for the CNN structure. The tested networks all had the fundamental structure shown in Figure 3.1, but the number of filters in the two convolutional layers and the number of fully-connected layers differ. All configurations were trained and tested using the WISDM and FSD datasets (see Chapter 2).

C12-C24-F1, C6-C12-F2, and C12-C24-F2 were the best performing networks. The median accuracy recorded by the C12-C24-F1 configuration for FSD classification is 53.00 %, whereas the other two networks perform significantly better, getting 82.50 % for C6-C12-F2 and 84.00 % for C12-C24-F2. C6-C12-F2 and C12-C24-F2 are the best-performing architectures for the WISDM dataset, with median accuracies of 45.00 % and 52.50 %, respectively. The explanation for these low values is not fundamentally linked to the network structure, but rather to a lower efficiency of the evaluated encoding techniques with this type of low-frequency input.

4.2.2 Feed-forward and recurrent SNNs

Recurrent SNNs are composed of recurrently connected layers of spiking neurons. Yin et al. [229] showed that RSNNs could outperform traditional ANNs for tasks with an inherent temporal dimension, including speech recognition. Furthermore, their RSNNs brought a significant advantage in energy savings over non-spiking RNNs with comparable accuracy, thanks to their sparse spiking activity. In Müller-Cleve et al. [3], the author and colleagues deployed a RSNN to

achieve a quantitative comparison of the different possible strategies suitable for effectively dealing with time-based Braille reading signals. In Fra et al. [2], instead, a Legendre Memory Unit was considered for the classification of HAR signals. The characteristics of these two networks are detailed in the following paragraphs.

RSNN for event-based Braille data

The authors used a two-layer RSNN based on Cramer et al. [121] and Zenke and Vogels [230] to perform classification on the Braille dataset (see Chapter 2). The input was encoded as an event stream with four different thresholds.

The network employed a modified version of the LIF, the current-based (CUBA) LIF model:

$$\tau_{mem}\frac{dU_i^{(l)}}{dt} = -(U_i^{(l)} - U_{rest}) + RI_i^{(l)}, \tag{4.3}$$

with U_i representing the membrane potential of neuron i (hidden state) in layer l, U_{rest} representing the resting potential, τ_{mem} representing the membrane time constant, R representing the input resistance. The input current I_i is defined as:

$$\frac{dI_i}{dt} = \frac{I_i(t)}{\tau_{syn}} + \sum_j W_{ij}S_j^{(0)}(t) + \sum_j V_{ij}S_j^{(1)}(t), \tag{4.4}$$

with τ_{syn} representing the synaptic decay time constants, $S_j(l)$ representing the spike train of the jth neuron at the lth layer, W_{ij} representing the forward weights, and $V_i j$ representing the recurrent weights.

$$I_i^{(l)}(t) = \alpha I_i^{(l)}(t-1) + \sum_j W_{ij} \cdot S_j(t) \tag{4.5}$$

$$U_i^{(l)}(t) = (\beta U_i^{(l)}(t-1) + \alpha I_i^{(l)}(t)) \cdot (1.0 - U_{reset}), \tag{4.6}$$

with $\beta = \exp(\frac{-time\_bin\_size}{\tau_{mem}})$ being the voltage decay constant, $\alpha = \exp(\frac{-time\_bin\_size}{\tau_{mem}})$ the current decay constant, U_{reset} the reset potential after an event is elicited, and $I_i(l)$ the synaptic input current from neuron i in layer l multiplied by the input resistance R. $R = 1\,\Omega$ for convenience.

The learning technique adopted was Backpropagation Through Time (BPTT). In this algorithm, the error at the output must be transmitted backwards throughout the whole network, unrolled in time. The feedforward and recurrent weight matrices W_{ij} and V_{ij} change after a specified loss $\mathscr{L}$ to accomplish supervised learning:

$$W_{ij} \leftarrow W_{ij} - \eta \frac{\partial \mathscr{L}}{\partial W_{ij}} \quad \text{and} \quad V_{ij} \leftarrow V_{ij} - \eta \frac{\partial \mathscr{L}}{\partial V_{ij}}, \tag{4.7}$$

with η as the learning rate. The partial derivative of a fast sigmoid function $\sigma(x)$ is used as a surrogate gradient in the backward pass (training) to avoid vanishing issues when using a binary step function $\Theta(x)$ in the forward pass (inference), whose derivative is zero everywhere except at the zero crossing, where it is infinite.

$$\sigma(U_i^{(l)}) = \frac{U_i^{(l)}}{1 + \lambda |U_i^{(l)}|}, \tag{4.8}$$

While $\Theta(x)$ is insensitive to multiplicative re-scaling, $\sigma(x)$ requires the inclusion of the scale parameter λ as part of the hyperparameter optimization.

To compute the gradients, the authors used custom PyTorch code [231, 230] to substitute the derivative of spiking non-linearity with said differential function.

We apply the cross entropy to the active readout layer $l = L$ for the loss. It is formulated as follows for data with N_{batch} samples and N_{class} classes:

$$\mathscr{L} = -\frac{1}{N_{batch}} \sum_{s=1}^{N_{batch}} \mathbb{1}(i = y_2) \cdot \log \left\{ \frac{\exp\left(\sum_{n=1}^{T} S_i^{(l)}[n]\right)}{\sum_{i=1}^{N_{class}} \exp\left(\sum_{n=1}^{T} S_i^{(l)}[n]\right)} \right\}, \tag{4.9}$$

with n as the time step.

Finally, we must define the regularization loss functions $\mathscr{L}_1$ and $\mathscr{L}_2$:

$$\mathscr{L}_1 = \frac{s_l}{N_{batch} + N} \sum_{s=1}^{N_{batch}} \sum_{i=1}^{N} \left[\max\left\{0, \frac{1}{T} \sum_{n=1}^{T} S_i^{(l)}[n] - \theta_l \right\} \right]^2 \tag{4.10}$$

indicates the lower-threshold spike count regularization per neuron, with strength s_l and threshold ϑ_l, and

$$\mathscr{L}_2 = \frac{s_u}{N_{batch}} \sum_{s=1}^{N_{batch}} \left[\max\left\{0, \frac{1}{N} \sum_{i=1}^{N} \sum_{n=1}^{T} S_i^{(l)}[n] - \theta_u \right\} \right]^2 \tag{4.11}$$

is the upper-threshold mean population spike count regularization, with strength s_u and threshold ϑ_u. The overall loss is represented by

$$\mathscr{L}_{\text{tot}} = \mathscr{L} + \mu_1 \mathscr{L}_1 + \mu_2 \mathscr{L}_2, \tag{4.12}$$

using μ as a scaling factor, and is reduced using the Adamax optimizer [232].

It is necessary to account for a time binning step for the input event stream while implementing and simulating SNNs based on this model in PyTorch. Despite the goal of working with asynchronous and sparse event-based data, fixed frame lengths were required to correctly represent algorithmic time steps in the domain of traditional clock-driven hardware such as CPUs and GPUs. The time retrieved from the signal recordings (T_{rec}) was divided into T chunks, with T defined as $T = \int(T_{rec}/\,time\_bin\_size)$ and the quantity *time_bin_size* inserted as an additional hyperparameter of the HPO. The encoded signal was then iterated over with a stride equal to *time_bin_size*, and a value of 1 was assigned to any time bin where at least one spike was found, otherwise a 0. The neuron in the output layer with the greatest spike count at the end of a trial is the winner.

Legendre memory unit

The Legendre memory unit (LMU) is a novel recurrent architecture capable of approximating the behavior of time cells using ordinary differential equations (ODEs) integrated over time [233, 234] for a continuous-time delay [235]. The LMU network's fundamental attribute is its capacity to decode a delayed signal $u(t-\theta')$ contained within a sliding window of length θ via a high-dimensional projection of the input $u(t)$ orthogonalized using the shifted Legendre polynomials [236]. Equation 4.13 gives the ith shifted Legendre polynomial:

$$P_i(r) = (-1)^i \sum_{j=0}^{i} \binom{i}{j} \binom{i+j}{j} (-r)^j, \tag{4.13}$$

which is used to delay the input signal via Equation 4.14:

$$u(t-\theta^{'}) \approx \sum_{i=0}^{d-1} P_i\left(\frac{\theta^{'}}{\theta}\right) m_i(t), \tag{4.14}$$

where the highest-order $d-1$ in the series expansion is related to the dimension of the state vector $\mathbf{m}(t)$ — defined by the input $u(t)$ — as shown in equation 4.15:

$$\theta\dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t), \tag{4.15}$$

with **A** and **B** representing the ideal state-space matrices derived using the Padé approximants as follows in Equations 4.16 and 4.17:

$$\mathbf{A} = [a]_{ij} \in \mathbb{R}^{d\times d}, \quad a_{ij} = (2i+1)\begin{cases} -1 & i < j \\ (-1)^{i-j+1} & i \geq j \end{cases} \tag{4.16}$$

$$\mathbf{B} = [b]_i \in \mathbb{R}^{d\times 1}, \quad b_i = (2i+1)(-1)^i, \quad i,j \in [0, d-1]. \tag{4.17}$$

Despite the fact that there is limited literature on LMU applications, remarkable findings have already been reported, demonstrating state-of-the-art outcomes in terms of accuracy and an interestingly small number of parameters while performing keyword detection [237].

4.2.3 A comparison of convolutional and recurrent SNNs for Human Activity Recognition

In Fra et al. [2], the author and colleagues benchmarked spiking and non-spiking networks of both convolutional and recurrent types with the HAR task. As shown in figure 4.1(a), the CNN consisted of two convolutional layers followed by a max pooling layer, a flattening layer, and two dense layers. We used the same structure for both non-spiking and spiking convolutional neural networks (figure 4.1(b)), which are referred to as CNN and sCNN in the following. The spiking CNN was converted from its non-spiking counterpart using the Nengo neural simulator and the NengoDL converter. In contrast, we constructed a recurrent architecture with a structure consisting of two long short-term memory (LSTM) layers connected to a dropout layer, followed by a dense layer (figure 4.1(c)).

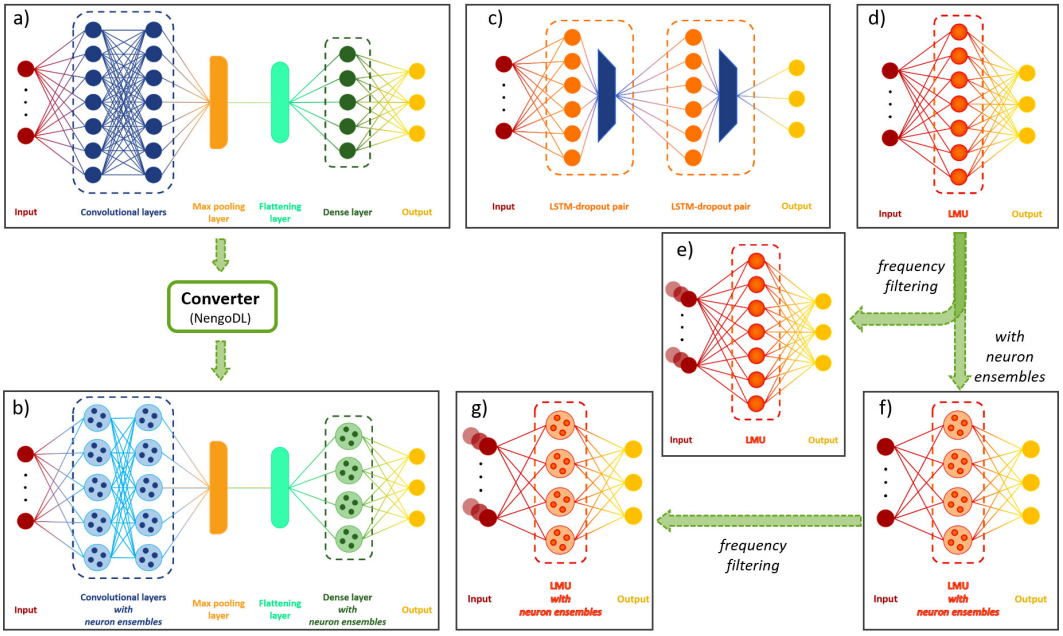

Fig. 4.1 A summary of the networks investigated. The Converter in NengoDL was used to transfer the convolutional architecture used in the non-spiking domain (a) into the spiking domain (b). Instead, the recurrent architectures have distinct structures in the two domains: the non-spiking implementation (c) used LSTM units followed by a dropout layer, whereas the recurrent SNN was obtained using a Legendre Memory Unit (f), which was also implemented in the non-spiking domain (d). An additional modification has been investigated for both the non-spiking (e) and spiking (g) LMU-based architectures by introducing frequency filtering on the input. [2]

Unlike convolutional designs, our spiking implementation of recurrent networks does not rely on the same architecture as in the non-spiking domain: Figure 4.1(f) summarizes how we employed an LMU instead of an LSTM, using a single LMU layer instead of a sequence of LSTM-dropout pairs. We used the LMU in a non-spiking network as well to extend the network comparison and benchmarking (Figure 4.1(e)). The spiking CNN, the LMU-based network, and its spiking version (sLMU) were built straight from their non-spiking counterparts using the Nengo neural simulator and the NengoDL converter.

Comparison criteria

We produced a trained classifier with optimal hyperparameters for each network architecture at the end of the proposed workflow, denoted as step (f) in figure 4.1.

We then set out to evaluate these classifiers in order to assess the benefits of neuro-inspired techniques.

We examined various metrics in addition to classification accuracy to make a full comparison across networks that rely not only on distinct designs, but also on different intrinsic functioning principles. For instance, the number of parameters and memory footprint have been taken into account for all networks. For Intel's Movidius Neural Compute Stick 2, we calculated the number of floating point operations (FLOPs) and the accompanying anticipated energy consumption for non-spiking networks. For Intel Loihi, we analyzed the number of neurons, the number of synaptic operations (SOPs), and the accompanying predicted energy consumption for spiking networks. Our energy assessments are based on the findings in Blouw and Eliasmith [56].

Results and discussion

In the neuromorphic realm, accuracy can not be treated as the only significant parameter in neural network benchmarking, and classifiers comparison in general. Other metrics, such as energy consumption and memory footprint, provide critical information for a more thorough evaluation and understanding of neuro-inspired solutions to classification challenges.

As a result, we accounted for several metrics. They were evaluated for each network by taking into account the ideal hyperparameter configuration provided by specially built NNI experiments, each of which performed 1000 trials; therefore, the development effort in optimizing the parameters for the various systems was equivalent. Table 4.1 summarizes the metrics studied, along with the associated values for each network.

We consider classification accuracy first: the LSTM-based network (table 4.1) achieves the best performance in this regard, scoring $(96.42 \pm 0.03)\%$. Remarkably, its spiking counterpart achieves the second-highest accuracy. The recurrent network based on the spiking implementation of the LMU with rectified integrate and fire neurons achieves a test accuracy of $(94.51 \pm 0.15)\%$, outperforming convolutional architectures in the non-spiking domain. Similarly, the spiking LMU supplemented with auditory-inspired frequency filtering surpasses both spiking and non-spiking CNNs, with a test accuracy of $(94.39 \pm 0.13)\%$.

Table 4.1 Summary of the metrics evaluated. The stated values were achieved using the best hyperparameter setup for each network. [2]

	LSTM	**CNN**	**sCNN**	**LMU**	**LMU (ff)**	**sLMU**	**sLMU (ff)**
Test accuracy (%)	96.42 ± 0.03	93.81 ± 0.10	92.47 ± 0.08	91.71 ± 0.13	88.16 ± 0.13	94.51 ± 0.15	94.39 ± 0.13
Number of parameters	2,125,222	144,899	167,973	76,130	89,014	91,200	132,540
Memory footprint (MB)	8.50	0.58	0.67	0.30	0.36	0.36	0.53
FLOPs ($x10^3$)	4,249.65	2,828.89	/	158.66	197.00	/	/
SOPs ($x10^3$)	/	/	10.82	/	/	99.91	127.95
Energy on Movidius (μJ)	3,199.99	2,130.15	/	119.47	148.34	/	/
Energy on Loihi (μJ)	/	/	5.49	/	/	50.66	64.87

The total number of parameters, which directly relates to the memory footprint, is the second statistic we analyzed further down the rows of table 4.1. From this point of view, the LSTM-based network's dominance as the ideal option is challenged: this design, with more than two million parameters, is by far the most demanding in terms of memory footprint, with a size of 8.50 MB. The network based on the non-spiking LMU is more than one order of magnitude smaller, with only 0.30 MB of memory footprint. Similar values are reported for LMU (ff) and sLMU, while the spiking LMU with frequency filtering slightly surpasses them, with a size comparable to convolutional architectures. The diameter of the circles in Figure 4.2 indicates the relative size of the various networks.

It is simple to find the best network in terms of accuracy and the one with the least memory footprint by combining the information received from the results given above. However, as these two networks do not coincide, we must add a third metric to our multi-objective assessment: energy consumption. This step quantifies the advantage of a neuromorphic approach for the considered task; the set of three fundamental quantities extracted from each network elevates the reported benchmarking above a simple comparison of values, transforming it into a useful tool for future applications of this neuro-inspired approach. The

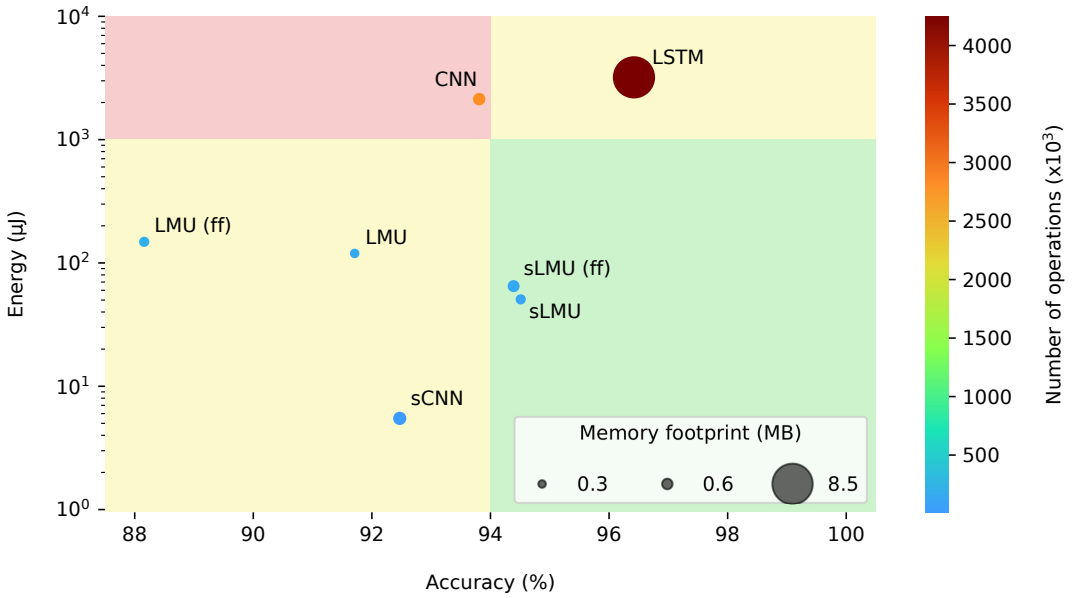

Fig. 4.2 An *energy vs. accuracy* graphic can be used to effectively depict how meaningful the gain in terms of energy reduction is with respect to a hypothetical decline in classification accuracy. The results provided here show that all of the examined SNNs and LMU-based networks consume at least one order of magnitude less energy than typical DNNs. In terms of memory footprint, the same conclusion can be obtained, with CNN and LSTM proving to be the largest networks. Spiking LMUs perform comparably to CNN and LSTM in terms of accuracy, even outperforming the former. [2]

bottom two rows of table 4.1 highlight that energy consumption is assessed using two different and specialized pieces of hardware: Intel Movidius Neural Compute Stick 2 for non-spiking networks and Intel Loihi for spiking networks. Quantitative analyses are conducted in both circumstances using the results of Blouw and Eliasmith [56], which provide the energy cost for a single operation. For each analyzed network, the number of operations and the necessary energy per inference are presented in table 4.1. The same results are shown in figure 4.2, where the energy is on the y-axis and the number of operations determines the color of the circles. As expected, all of the spiking networks are less computationally expensive, with the spiking CNN delivering the lowest value of 5.49 µJ. It is also worth noting that the assessed energy usage for all LMU-based networks is at least one order of magnitude lower than that of CNN and LSTM. LSTM reports the highest energy consumption as well as the largest memory footprint. With more than 3000 µJ, said architecture consumes over three orders of magnitude more energy than the sCNN. Within the range defined by these two opposites — the sCNN and the

LSTM — the sLMU is the one pairing the highest accuracy to the lowest energy cost: 50.66 µJ to achieve the second-best accuracy in this test, which makes it about two orders of magnitude less energy-hungry, but comparable in accuracy to the LSTM-based architecture.

The spiking LMU's trade-off between high classification accuracy and low energy consumption, together with its tiny memory footprint, makes this architecture a viable contender for on-edge applications of neuromorphic classifiers aiding real-time tasks. The reported results are also summarized in Figure 4.3, where a radar graphic is employed to emphasize the strengths and limitations of the researched networks, in order to evaluate the different designs' capability to address different tasks and applications.

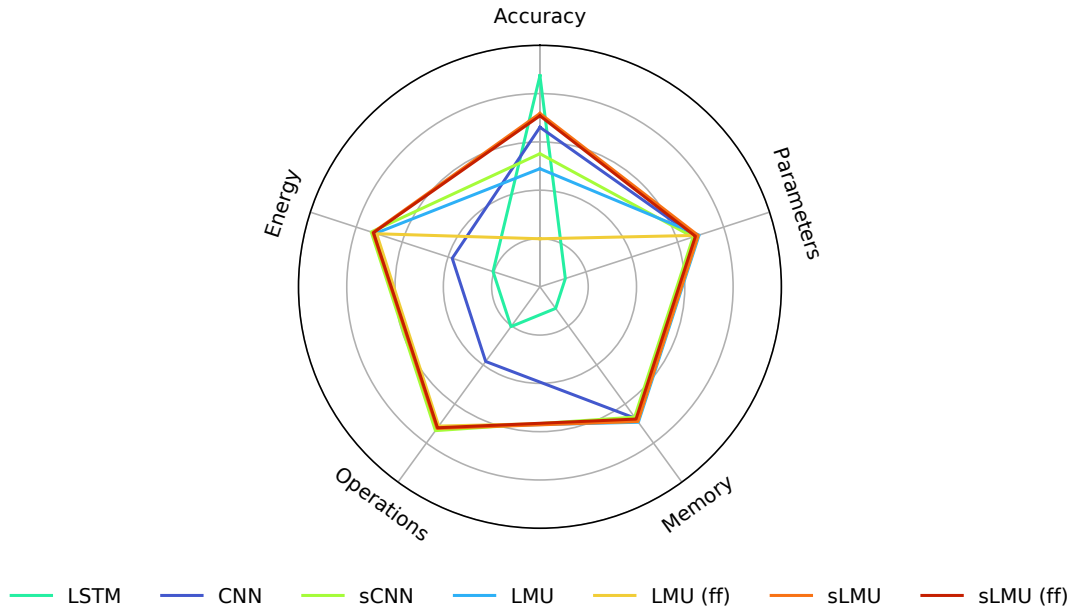

Fig. 4.3 A radar chart of the supplied results for quick comparison of the analyzed networks based on each of the evaluated variables. The classic DNN designs under consideration, namely the LSTM and the CNN, are surpassed by the alternative ones based on the LMU in all energy and memory parameters; the spiking CNN also outperforms the LSTM and non-spiking CNN in terms of both energy and memory. [2]

4.3 Learning methods

Learning in neural networks, also known as training, entails changing the connections between neurons. While ANNs have found successful training methods in

stochastic gradient descent and backpropagation, research in the field of SNNs has yet to uncover a reliable learning algorithm.

Several candidates for an effective native training method for SNNs have been proposed [214].

Such methods can be based on a global or local technique. Global learning approaches, like the classical approach used for ANN architectures, involve updating all of the network's hyperparameters at each training step; such algorithms include Spike-Time-Dependent Plasticity [169] and Back-Propagation Through Time [238]. Local techniques, on the other hand, modify only a subset of the hyperparameters at each step; examples are Hebbian learning [239] and E-prop [224].

Spike-time-dependent plasticity (STDP) [240] is an unsupervised learning method, incorporating a localized learning rule at the synapse level. Synaptic plasticity determines changes in synaptic strength based on patterns detected in the synapse's activity. Although the synapse-based learning rule is thought to be biologically more realistic, learning performance is typically lower than that of supervised learning.

Other training methods take inspiration from established methods in the world of ANNs, adapting supervised learning with surrogate gradient descent and spike backpropagation. SLAYER [241] is a backpropagation-based learning mechanism which establishes a time-based credit distribution policy to propagate the output error to preceding layers in the network. This method uses a stochastic spiking neuron approximation rather than a surrogate gradient method to calculate the error backpropagation. Bellec et al. [224] proposed instead the eligibility propagation (e-prop) algorithm, which leverages both synaptic plasticity and gradient descent methods to achieve optimal training for recurrent SNNs. The e-prop method was iterated upon by Frenkel and Indiveri [242] to improve the detection of second-long time dependencies, and constitutes one of the most promising novel learning methods in the neuromorphic field.

Finally, converting ANNs to SNNs [243] is an alternate method for indirectly training SNNs which consistently delivers state-of-the-art results in terms of accuracy. In the following section, we will delve deeper into this method, also known as transfer learning.

4.3.1 Transfer learning

The *Transfer learning* strategy is the most commonly used way to train an SNN in the field of neuromorphic state-of-the-art for categorization of time-variant signals [244, 245, 147, 246]. The reason for its popularity is that it allows network designers to use established ANN methods (such as backpropagation and gradient descent) to train SNNs, sidestepping the difficulty of training a network in the spiking domain. These difficulties mainly stem from the non-differentiability of the spike signals, and despite ongoing efforts [224, 247], a universally effective spike-based training method has not yet emerged; transfer learning, then, represents a useful shortcut to quickly optimize and deploy working SNN prototypes.

Implementing ANN-to-SNN conversion allows to maintain the ANN's information transmission and function, while lowering the effort associated with signal transmission and matrix operations, thanks to the sparse and binary nature of spiking signals [214]. The transfer learning strategy involves training an artificial neural network (ANN) and then transferring the resulting weights to a spiking network with the same topology [248]. These conversion methods allow the derived SNNs to achieve results comparable to the original network, and therefore represent a readily employable solution for the deployment of energy-efficient neural network applications.

In Forno et al [1], we used transfer learning to study the effect of input signal encoding on the training of a spiking convolutional neural network (sCNN). The layer structure and perceptor model for CNN training were carefully selected to ensure identical behavior in the SNN and CNN architectures after weight transfer. A pooling layer implements an average pooling operation, using the method proposed by Liu et al. [168] and used in Dominguez-Morales et al. [147] for the CNN architecture. The neuron model utilized in the convolutional and fully connected layers is the modified ReLU described by Liu et al. [168], and the softmax activation function is used in the output layer. Finally, all neurons have their bias parameter set to 0. The input data is fed to the non-spiking CNN in the form of sonograms, as seen in the bottom left of Figure 7.5; the network can then be trained using any of the traditional methods, such as the Back-propagation or Stochastic gradient descent algorithms. Following the completion of the CNN training process, a "twin" SNN network is constructed; this new network adopts the Leaky Integrate and Fire (LIF) neuron model, with the synaptic weights set

equal to those recovered from the CNN. To adapt the data to the SNN's input layer, the sonogram is subjected to a final Poisson Rate encoding step. Because all encoded data is processed in the same manner, this strategy allowed for a consistent and fair evaluation method for the various coding systems.

4.4 Optimizing the architecture: model compression

After the training process, numerous optimization strategies may be used to reduce the size of the network prior to deployment. One of these is model compression. We used two phases [1] to perform model compression on a spiking CNN trained by transfer learning: *synapse reduction*, which allows us to deliberately minimize the number of connections between neurons, and *fine-tuning*, which optimizes the network's remaining parameters. The qualitative and quantitative benefits of using model compression are numerous: obtaining a much smaller network, suitable for embedded systems due to a smaller memory footprint and a lower computational cost; making network simulation in non-neuromorphic hardware faster; and, in some cases, improving accuracy thanks to a reduction in the overall stimulus transmitted through the synapses, which can introduce noise during the computation.

The synapse reduction approach used here involves selectively eliminating connections between neurons in all layers except the pooling layer based on the weight associated with the synapse. This elimination is applied without affecting the network's structure in terms of the number of layers and the neurons contained in each, by determining the distribution of the values for synaptic weights and then gradually deleting the connections with the lowest weights. Since for excitatory connections a higher weight associated with a synapse equates to increased excitability of the neuron when stimulated, this technique allows for the elimination of synapses that contribute only marginally to the production of spikes. Inhibitory synapses exhibit the opposite behavior.

The network's classification accuracy often degrades after the synapse reduction phase. To restore or even improve on the original classification performance, a fine-tuning step is used: after restricting the removed weights to 0, a CNN with the remaining connections is retrained for 5 epochs, then the weights are transferred back to the SNN version.

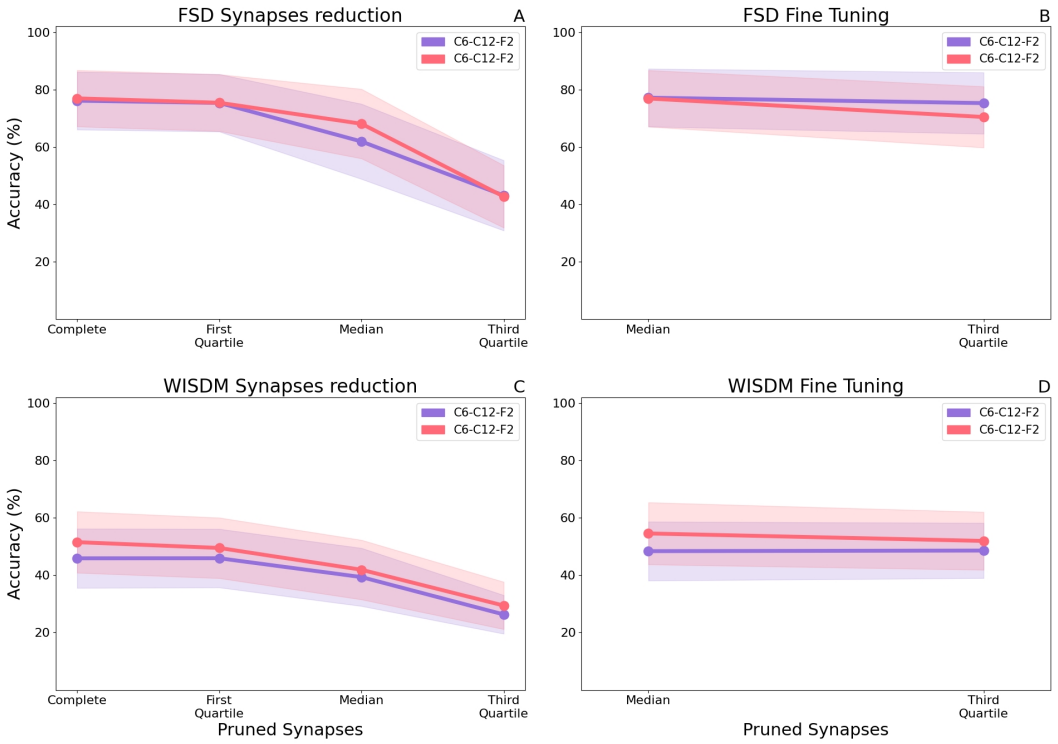

Fig. 4.4 Median test accuracy after synapse reduction (A, C) and fine-tuning (B, D) of all encoding class, filter type, number of channels, and feature extraction bins combinations for architectures C6-C12-F2 and C12-C24-F2 performing classification of the FSD and WISDM datasets. [1]

After determining the best-performing CNN structure (as explained in Section 4.2.1), we employed the model compression approach to reduce network connectivity, therefore lowering memory, compute, and energy needs. To put the aforementioned strategies to the test, we apply the synapse reduction process to the best-performing network topologies from prior experiments: C6-C12-F2 and C12-C24-F2. Based on the distribution of their absolute values, we gradually eliminate connections with increasing synapse weights: initially, we remove connections whose weight is less than or equal to the first quartile, then the median, and lastly the third quartile. Figure 4.4A depicts the effect of synapse reduction on FSD dataset classification: the more connections eliminated, the lower the classification performance. This trend is caused by a decrease in the number of spikes in the network, which makes it difficult to excite the neurons in the fully connected layers correctly. To optimize the model described by the residual synapses, the smallest network (C6-C12-F2) is fine-tuned by copying the connection settings

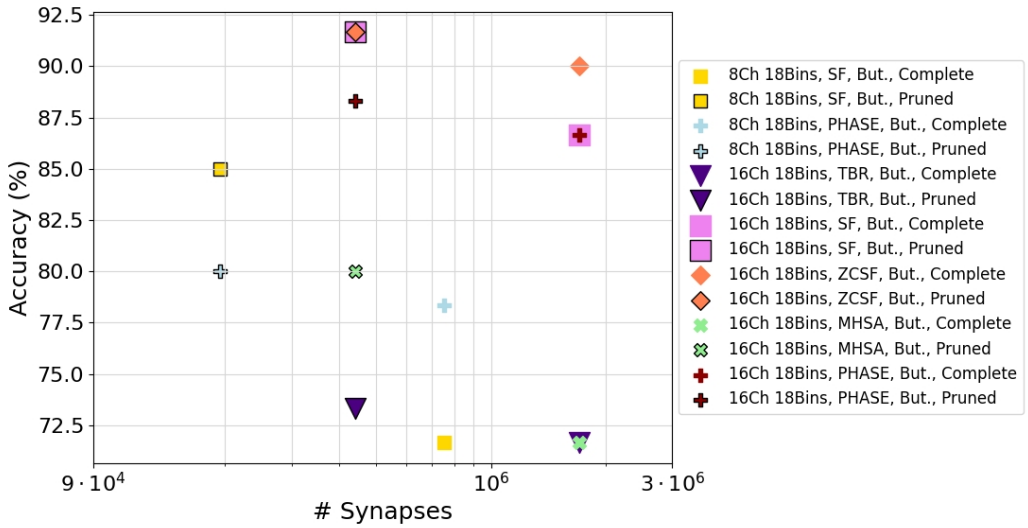

Fig. 4.5 Summary of network settings that enhanced performance on the WISDM dataset following model compression. [1]

back to the original CNN and retraining it for 10-20 epochs. When the retrained weights are applied to the final version of the sCNN, the accuracy is equivalent to the entire network, with a few configurations surpassing the original network by up to 1.75 %. By pruning up to the third quartile, the compressed networks achieve a median test accuracy of 81.00 % while keeping only 25 % of the original network size (Fig. 4.4B); this median value is attained across all filter bank, feature extraction, and encoding algorithm combinations for this particular design.

In the case of WISDM, synapse reduction reduces accuracy as well (Fig. 4.4C). However, after fine-tuning the network for the C12-C24-F2 design, an increase in the maximum possible accuracy can be recorded for some configurations, resulting in greater performance for the minimized network than the whole network. For instance, the ZCSF algorithm for 16 channels, 18 bins with 3rd-quartile synapse reduction achieves an increase of of 1.7 %, to 91.7 %, whereas the SF algorithm for 16 channels, 18 bins with median synapse reduction achieves an increase of 8.3 % to 95.0 % accuracy.

This enhancement can be attributed to the combined effect of synapse reduction and fine-tuning, which allows for a reduction in the number of connections in the network while maintaining a model suitable for data representation: this results in a reduction in the noise traveling through the network, which benefits

the classification process. Figure 4.5 depicts configurations that improve their performance after model compression.

4.5 Chapter summary

Choosing the right network architecture for a neuromorphic system is crucial for its successful operation. Neuromorphic technology allows for a very detailed modeling of the internal dynamics of a network layer, including the complex behavior of a single neuron. When it comes to designing neuronal models, there is a high degree of flexibility as they can be tailored to meet specific requirements, either by drawing inspiration from biology or by creating computational units that are less realistic but serve a particular purpose. Despite the numerous possibilities, the most commonly used neuron model for practical applications is the Leaky Integrate & Fire (LIF) model, which is preferred because of its low computational load. In particular cases, the drawbacks of the LIF model can be compensated by integrating populations of more complex neurons (such as Adaptive LIFs) to enable additional features in these computational units. Whatever the chosen neuron model, the interactions between layers and single neurons must be carefully evaluated in order to validate proper propagation of information throughout the network.

The choice of the classifier network is especially important because it determines the efficiency and accuracy of the neuromorphic system. The spiking Convolutional Neural Network (sCNN) is appealing as a classification architecture, due to its ease of implementation and the possibility of transfer learning using well-known and reliable Artificial Neural Network (ANN) methods. However, more complex networks such as the Legendre Memory Unit (LMU) and the Reservoir Spiking Neural Network (RSNN) are capable of natively creating a memory trace of past events in their recurrently connected reservoirs; this capability enables the classifier to correlate time-varying events on a longer scale. In summary, selecting the right classifier network, and incorporating the most advantageous neuron models, can greatly enhance the functionality and performance of a neuromorphic system.

While it is crucial to carefully choose the best classifier network for the task at hand and to consider the strengths and limitations of different neuron models,

the modelling and architectural choices for a neuromorphic realization of a SNN are also necessarily informed by the software and hardware available to implement them. In Chapter 5, we will examine some of the available software for the design and deployment of neuromorphic applications, delving into two use cases for hyperparameter optimization and an example of system hardware handling placement and routing tasks for the SpiNNaker platform.

Chapter 5

Software frameworks

While research in neuromorphic hardware has been intense, creating a plethora of alternatives focusing on many different aspects of SNN applicability, the effort on the software side has only recently begun to see significant results. Yet, while neuromorphic hardware shows tremendous promise for time and resource-constrained application, its potential cannot hope to be fully realized in practice without a solid and accessible software stack enabling widespread access to these platforms for as many developers as possible.

At the present moment, there is not a single all-encompassing framework that has been embraced by the community to design the entire stack of a neuromorphic application. In this chapter, we will focus on software designed to aim two main goals: first, the design, optimization and simulation of a SNN, and second, the mapping of such a network to a specific hardware platform and its interfacing with the external world.

5.1 SNN specification software

In the wider world of deep learning, there are several end-to-end network specification platforms, such as Tensorflow/Keras and PyTorch. Building on decades of experience with Von Neumann-based computing, these frameworks are designed to deliver an all-encompassing development environment that allows to design, compile, test and deploy a deep learning model on CPU or GPU using the same tool library. Furthermore, their availability as free/libre open source software has

made knowledge sharing and experimentation easy and accessible to developers of all experience levels, enabling continuous innovation both in academic research and in the commercial field.

The comparative complexity of Spiking Neural Networks and of the novel hardware platforms designed to accelerate their execution has made it difficult to realize software frameworks as powerful as those currently available for deep learning. However, especially in recent years, an increasing number of solutions have appeared, proposing to close this gap.

5.1.1 PyNN

PyNN [249] is one of the most well-known development frameworks in the neuromorphic field. When this Python library was proposed in 2009, the neuromorphic community was seeing the advent of specialized SNN simulators, such as NEST and Brian, which exploited efficient multi-threading parallelism to run complex event-based simulations on multi-core machines and computer clusters. PyNN provided a unified front-end to most unified simulators, allowing developers to describe neuron models and network architectures at various levels of abstraction and to run them on a variety of software and hardware simulators. With time, PyNN was extended with the ability to compile and map SNN models on neuromorphic hardware, such as BrainScaleS [250] and SpiNNaker [28, 251]. Thanks to its widespread adoption across different research institutions, PyNN has become one of the most useful tools to quickly build portable SNN applications, encouraging code sharing and reuse and serving as a basic framework to build more analysis, visualization, and data-management applications.

5.1.2 Nengo

Nengo [252] was developed as a simulator capable of providing sophisticated networks with cognitive abilities starting from single neuron models, using the Neural Engineering Framework [191] as a guiding principle to build neural models accounting for functional objectives as well as anatomical constraints.

Nengo translates the three NEF principles, namely *representation*, *transformation*, and *dynamics*, into the fundamental units for network development, defining

three core objects: *ensemble*, *node*, and *connection*. Their combinations yield two more items, *network* and *model*, while a *probe* allows data collection during simulations. This set of six front-end objects is the toolbox for creating the neural model that will be handed to the Nengo simulator, which contains the back-end functionality for network execution.

The versatility of Nengo's simulator is a major feature, as seen by the ability to adapt it to specific, and possibly specialized, hardware [253]. NengoLoihi, for example, is a dedicated backend to execute Nengo models on Intel Loihi. Furthermore, because of this flexibility, models from various frameworks may be easily integrated using NengoDL's converter, which adapts deep learning models to event-based models by substituting standard activation functions and layers with Nengo's spiking neuron populations.

5.1.3 EONS

Because of the complex interactions between spiking neurons, reusing knowledge from the DL field to build SNNs may not yield the best results. An alternative approach is EONS (Evolutionary Optimization for Neuromorphic Systems), a platform that exploits the principles of evolutionary optimization to easily build novel SNN applications and prototype them on neuromorphic systems. Starting from randomly generated populations of neural architectures, EONS evaluates them and assigns a fitness score to each solution, then employs evolutionary operands such as *crossover* or *mutation* to "evolve" a new generation of networks from the first one. This system allows to automatically build network "blobs" with sometimes unintuitive connections, such as recurrently connected reservoirs, which nonetheless perform well on classification and control tasks. EONS can also be supplied with neuromorphic hardware constraints in order to adapt the network mapping to a target machine.

5.2 SNN optimization

ANNs can be described from two complementary viewpoints. On the one hand, there is the *architecture*, which specifies the number and kind of layers used, as well as how they are interconnected; on the other, there are the *hyperparameters*,

which specifically identify each network and determine its fundamental behavior. As a result, hyperparameter optimization (HPO) must be considered while investigating and comparing different network topologies, especially when needless complexity must be avoided [254].

The neural network intelligence (NNI) toolbox [255] is a software kit for running automated ML experiments. Users supply the program with one or more target parameters and define the desired search space; then, NNI uses various tuning algorithms to automatically generate trial jobs and determine the parameter configurations that optimize the target result. In the following, we will explore in detail two use cases for NNI with different applications.

5.2.1 Case study 1: HPO for HAR

In our study of spiking and non-spiking networks for HAR classification [2], we performed NNI optimization for each of the networks introduced in Section 4.2.3 and described in Figure 4.1. Each optimization experiment consisted of 1000 trials, using the built-in *annealing* algorithm as the tuner. To help reduce the problem of local minima affecting annealing techniques [256], each trial set included 4 regularly spaced random tuner reinitializations. At the end of each trial, made up of 100 training epochs, the sets of weights delivering the best training accuracy were retrieved to evaluate the test accuracy, which was the experiment's optimization target. All of the analyzed networks were trained using the Adam optimizer with constant learning rate, including learning rate optimization during the experiment trials. Table 5.1 provides an overview of the hyperparameters found by the HPO.

Table 5.1 Summary and description of the optimized hyperparameters. All of the hyperparameters reported for the non-spiking implementations are also used for the corresponding spiking networks. [2]

Network	Hyperparameter	Description
LSTM	units_1	Number of units in the LSTM layers
	units_2	
	dropout_1	Dropout rate between the LSTM layers
	dropout_2	
	l2_2	L2 regularization applied to the recurrent weight matrix in the second LSTM layer
CNN	filters_1	Number of filters in the convolutional layers
	filters_2	
	kernel_size_1	Dimension of the kernel in the convolutional layers
	kernel_size_2	
	dense_1	Number of units in the first Dense layer
Spiking CNN	target_rate_1	Target value for neurons firing rates regularization in the convolutional layers
	target_rate_2	
	reg_conv_1	L2-like regularization applied to the neurons firing rates in the convolutional layers
	reg_conv_2	
	scale_firing_rates	Scale factor for the neurons firing rates
	synapse	Time constant of the synaptic low-pass filter on the output of all the neurons
	n_steps	How long (in simulation time steps *) the input is presented to the network
LMU	units	Size of the LMU kernels
	order	Number of Legendre polynomials
	theta	Length of the sliding window
	synapse_in	Time constant of the synaptic low-pass filter on the input connection of the LMU
	synapse_out	Time constant of the synaptic low-pass filter on the output connection of the LMU
	tau	Time constant of the discretized synaptic low-pass filter on the internal connections to memory
Spiking LMU **	n_neurons	In place of `units`, size of the neuron ensembles (whose number is defined by `order`)
	synapse_all	Time constant of the synaptic low-pass filter on the connections between neuron ensembles
	max_rate	Firing rate for neuron input equal to 1
All	batch size	Number of training examples in each learning iteration
	learning rate	Step size for weights update in each learning iteration

* The default value in Nengo of 1 ms is used
** All the hyperparameters for the non-spiking LMU are specifically re-optimized for the spiking implementation

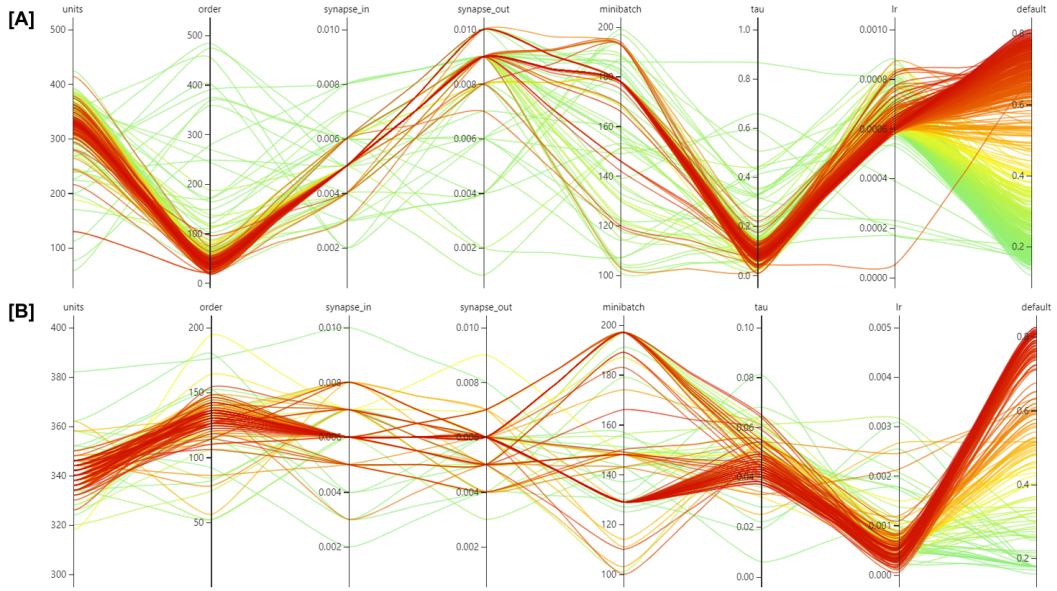

Fig. 5.1 Two iterations of NNI search for the LMU performing HAR classification. **(A)** A first iteration of the search finds a range of optimal values for the network parameters. **(B)** To refine the search, the parameter optimization is repeated, restricting the search space to the best performing ranges (the red areas in **(A)**). [2]

5.2.2 Case study 2: HPO for Braille reading

In order to validate the novel Braille dataset [3], recurrent (RSNN) and feedforward (FFSNN) models were built and tuned with the same parameter optimization approach presented in Case Study 1. A separate optimization was performed for each of the event streams derived from the original frame-based signal by varying the encoding threshold value ϑ. Each HPO took 600 trials using the Anneal algorithm with the values provided in Table 5.2. To limit the impact of local minima, each experiment included two evenly spaced random reinitializations of the tuner. All trials were made of 300 training epochs, with intermediate outcomes for both training and test at the conclusion of each epoch and an 80/20 train-test split. In order to account for possible overfitting, the test accuracy was selected as the optimization objective of the HPO study, and the greatest value was extracted at the end of each trial.

Following the annealing-based procedure, the authors explored a portion of the initial search space using a grid search on the two most relevant hyperparame-

Table 5.2 Description of the hyperparameters contained in the HPO problem search space. [3]

Hyperparameter	Description
scale, λ	Steepness of surrogate gradient
time_bin_size	Time binning of the encoded input
nb_input_copies	Copies of the encoded signals provided to the input layer
tau_mem, τ_{mem}	Decay time constant of the membrane
tau_ratio	Ratio between the decay time constants of the membrane and the synapse (τ_{syn})
fwd_weight_scale	Scaling factor for weight initialization of the forward connections (W_{ij})
weight_scale_factor	Scaling factor for weight initialization of the recurrent connections (V_{ij})
reg_neurons, $\mathscr{L}_1$	Regularization on the number of spikes per neuron
reg_spikes, $\mathscr{L}_2$	Regularization on the total number of spikes

ters in terms of energy consumption, *time_bin_size* and *nb_input_copies*, because they determine the number of operations that must be computed per inference.

This two-step HPO approach yielded an optimized network for each threshold value in the sigma-delta encoding. All of these RSNNs have a recurrent, fully connected hidden layer with 450 LIF neurons and an output layer with 28 LIF neurons. The number of output neurons includes the 27 classes dictated by the Braille alphabet plus 1 extra class designed to identify edge scenarios, such as faulty contact between the fingertip and the letters, an eventuality that would need to be addressed in the event of future online implementations. The size of the input layer was instead optimized: the number of input neurons is defined as ($2 \cdot$ *n_taxels*$\cdot$ *nb_input_copies*), with the factor 2 representing the event polarity, *n_taxels* = 12 the sensors in the robotic fingertip, and *nb_input_copies* the variable to optimize. All networks used a batch size of 128 and a learning rate of $\eta = 0.0015$.

The primary objective for the optimization using the two-step HPO technique was classification accuracy, but we also monitored the Time-to-Classify (TTC) and power consumption, with the latter separately reported in Chapter 6. The minimal temporal length of the input required for successful classification is an instructive figure of merit to be considered in the context of an online implementation of the suggested RSNN. Therefore, TTC was defined as the ratio of the signal length required for successful classification over the whole acquisition time of the Braille letter, which was fixed to 1.35 s due to the fixed sliding speed.

Table 5.3 Optimized hyperparameter values for each encoding scheme after grid search. [3]

	Threshold (θ)			
	1	**2**	**5**	**10**
scale	5	15	10	10
time_bin_size (ms)	5	3	3	5
nb_input_copies	2	8	4	2
tau_mem (ms)	60	50	70	70
tau_ratio	10	10	10	10
fwd_weight_scale	1	1	1.5	4
weight_scale_factor	1e-2	2e-2	3.5e-2	1.5e-2
reg_spikes	4e-3	1.5e-3	1e-3	1.5e-3
reg_neurons	1e-6	0	0	0

For each threshold, the parameter space following NNI optimization and grid search showed no notable trends: the complicated interaction of many parameters results in a variety of local optima with comparable test accuracy. Singling out the trials with the highest classification accuracy for each encoding threshold, as in Table 5.3, yields similar results: only the forward weight scale (*fwd_weight_scale*) appears to increase as the threshold rises. The membrane potential time constant τ_{mem} varies just slightly, with no discernible trend, and the *tau_ratio*, which describes the relationship between τ_{mem} and τ_{syn}, is constant; this shows that the membrane time constants are determined by the spatial-temporal features of the data, regardless of the encoding threshold. This unvarying membrane-to-synapse time constant ratio appears to strike an ideal balance between neuron and synapse dynamics for the Braille reading task.

Figure 5.2 depicts a summary of the RSNN classification accuracy after grid search optimization. The highest threshold ($\vartheta = 10$) has the worst overall performance. Out of all the explored combinations of *time_bin_size*, *nb_input_copies*, and encoding thresholds, the best result in terms of accuracy was reported by the network with encoding threshold $\vartheta = 5$, a *time_bin_size* of 3 ms, and 4 input copies. However, the standard deviation of its accuracy was larger than for other combinations, as shown in Figure 5.2B. The best configuration found for an encoding threshold $\vartheta = 2$ resulted in a similar mean accuracy, but with a significantly lower standard deviation: $(80.9 \pm 0.3)\%$ test accuracy compared to $(80.9 \pm 1.9)\%$. Taking into account this information, the optimum configuration has encoding threshold $\vartheta = 2$ with a *time_bin_size* of 3 ms and a number of copies equal to 8.

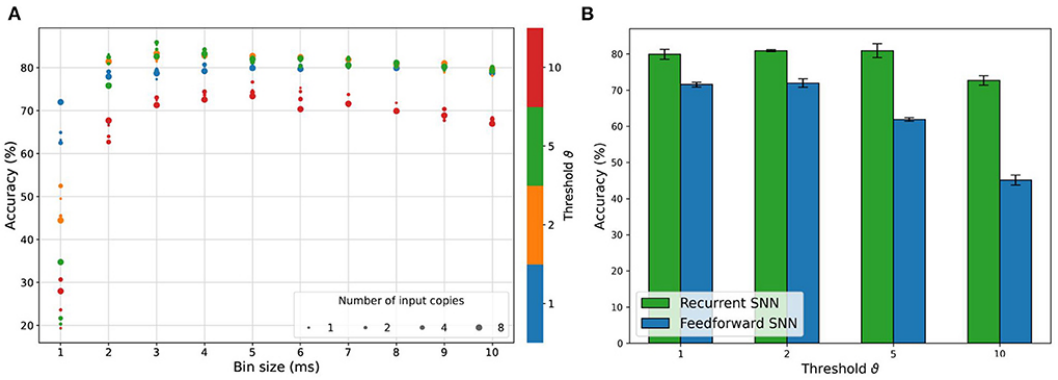

Fig. 5.2 The accuracy performances of RSNN and FFSNN as a result of grid search exploration in the two-step HPO technique are summarized. **(A)** Best test accuracy results produced by the RSNN for all combinations of *time_bin_size* and *nb_input_copies*. **(B)** Mean and standard deviation of the FFSNN and RSNN accuracy results, with the best parameters for each encoding threshold. [3]

The accuracy results analysis provided an additional insight about the impact of the encoding threshold and *time_bin_size*, as shown in Figures 5.2A-B. A variation in either of these two parameters causes a similar pattern in test accuracy: after an initial increase that leads to a maximum, a further increase causes a decline in classification performance. In contrast to the findings about the preservation of events for different thresholds discussed in 3.3, the accuracy for higher encoding thresholds reduces the most for greater *time_bin_size*. When comparing the network performance of the RSNN with the FFSNN, as shown in Figure 5.2B, we see a steady decrease for the FFSNN with rising thresholds, but practically constant performance for the RSNN up to $\vartheta = 10$, where it begins to drop off.

When planning an online hardware implementation, one must consider not just classification performance, but also energy efficiency. In this regard, the encoding threshold of 1 is the most promising candidate, since it requires fewer input copies and a larger *time_bin_size*, resulting in a much lower energy footprint at comparable performance. Regardless of the hyperparameters, the TTC is constant across all situations, and the entire time series is required for the best classification performance. The authors also compared these findings to a 27-class implementation, in order to confirm the utility of the extra class. This investigation indicated that similar results are obtained in both scenarios, indicating that the addition of a 28th class has no negative impact on classification performance.

5.3 System software: the SpiNNaker example

Any novel neuromorphic platform necessitates the development of system software, containing the platform-specific compiler and run-time managers that handle the execution of SNN applications on the hardware. The SpiNNaker software stack provides a particularly complete example of an end-to-end toolchain for the design and execution of SNN applications on neuromorphic software. As application software, SpiNNaker adopts the PyNN libraries, allowing developers to abstract the SNN design from the underlying software.

The sPyNNaker middleware [28] is composed of two parts: a set of Python libraries that preprocess the PyNN models and translates them into SpiNNaker applications, and an event-driven operating system running on the hardware platform, which interfaces the user application with the underlying SARK management software. The SpiNNaker system, being composed of fully-programmable ARM cores, provides researchers with the flexibility to upgrade and expand the system's capabilities in tandem with advances in neuroscience and computing. In fact, while sPyNNaker includes implementations of the standard cell models defined in PyNN (such as LIF and Izhikevich neurons), it also allows users to implement their own neuron models.

Another important step in the sPyNNaker toolchain is the optimization of the application for execution on the physical hardware. The software must map the SNN to an application graph, partition that into a machine graph, and generate the routing information among machine vertices. Efficient mapping and routing on SNN applications is a problem that has been previously tackled by researchers at Politecnico di Torino. Urgese et al. [4] extensively profiled the SpiNNaker board's communication bottlenecks and proposed the SNN-PP methodology for SNN partitioning and placement focused on reducing inter-core spike communication. Barchi et al. [257, 258] re-evaluated the neuron-to-core mapping issue and realized a task-based placement pipeline based on the minimization of the distance traveled by packets exchanged by cores. These works especially highlighted the importance of communication directionality between network vertices: chips intercepting the flow of packets in opposing directions create traffic "hot spots" due to the increased burden placed on the local router. Figure 5.3 summarizes this finding.

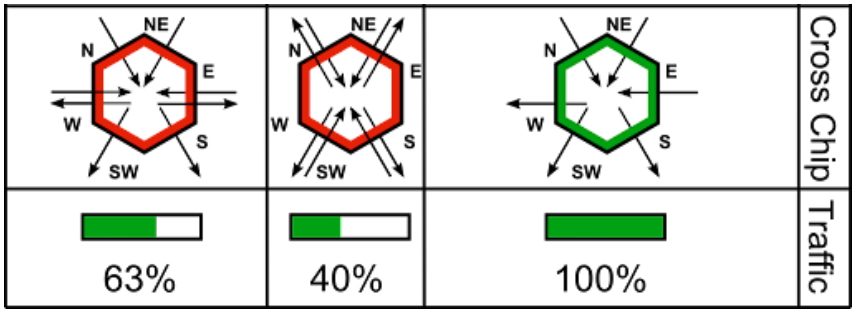

Fig. 5.3 A visual representation of "hot spots" on a SpiNNaker chip, courtesy of Urgese et al. [4]. The top row shows the flow of packets traversing the chip, which the local router needs to deliver to neighboring chip. The bottom row shows the percentage of successfully delivered packets with the given configuration in tests featuring heavy packet traffic. © 2016 IEEE.

Building on the above work, the author joined researchers at the University of Manchester to assist in the implementation of new neural models exploiting the flexibility of the SpiNNaker hardware. The goal of this research was to create functional multi-compartmental neuron models based on the Urbanczik-Senn and pyramidal designs (see Chapter 4). Due to hardware limitations, in order to make real-time simulation feasible on the machine, the choice was made to use rate-based neuron models and to split the execution of the synapse and neuron compartments on separate hardware cores. While the researchers at Manchester oversaw the development of the neuronal model, the author undertook the task of exploring placement strategies to enable the execution of a multi-compartmental network on the SpiNNaker supercomputer. This led to a collection of results presented for the first time in the following section.

5.3.1 Placement and routing exploration on SpiNNaker

As a test application for the multi-compartmental model, we selected a 2-layer MNIST classifier using said neurons, the structure of which is portrayed in Figure 5.4. The goal was to reach real-time execution at 1 ms timestep, and in order to avoid overwhelming the communication infrastructure, rate-based coding was an obligatory choice for interneuron data exchange. However, SpiNNaker is designed for spike-based coding; that is, the router was built with the assumption that each communication packet contains one spike worth of information. Therefore, the router is designed to drop packets when its buffer is full, because

the loss of a single spike does not significantly degrade communication quality in event-based encoding. This poses a significant problem with the adoption of rate-based coding: with this paradigm, a packet now encapsulates a whole timestep worth of information. Packet loss, then, leads to significant information loss, which can snowball throughout the simulation, leading to classification failure. While SpiNNaker's system software includes a reinjector system which intercepts dropped packets and attempts to re-circulate them, the complexity of the multi-compartmental network is such that this measure is not sufficient to guarantee correct performance. Then, it becomes necessary to respect strict timing constraints in order to reduce packet loss as much as possible. This issue can be tackled not only by reducing the processing time, but also by finding the most efficient hardware placement that minimizes the distance traveled by the packets.

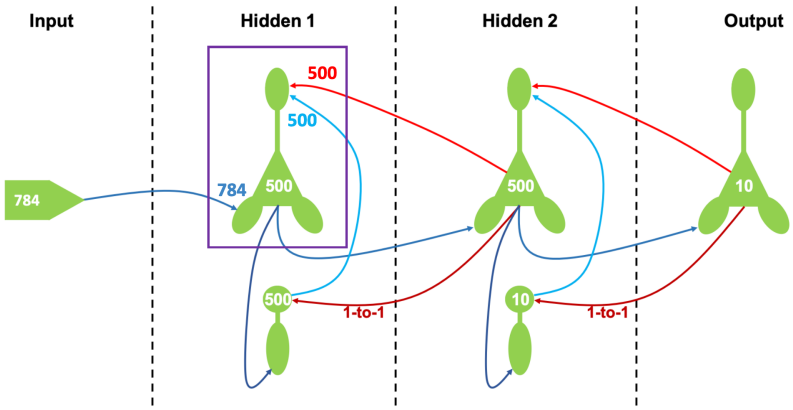

Fig. 5.4 Diagram of the 2-layer multi-compartmental MNIST classifier. Highlighted is the pyramidal neuron, showing the fan-in on the different synaptic compartments.

Communication test with a single neuron

In our first test, we profiled the performance of a single pyramidal neuron, based on the MNIST network design. The source-to-synaptic input part of the network was isolated for a feasibility study, without taking into consideration any delay introduced by the later processing of the input by the neuron model. In order to reproduce the total fan-in of 1784 for the first-layer pyramidal neurons, we used random Poisson spike generators to excite the target synapses with the same rate. Then, we extensively tested a number of configurations by varying the coordinate

of the target chip, the number and placement of the spike source populations connected to each synaptic core, and the number of neurons in each source and target population.

Two example configurations are showed in Figure 5.5. In this particular test, 16 source populations were connected to each of the 14 synapse populations in the target core. The yellow square in the figure highlights the coordinate of the target chip; 42 chips were used to generate the random input, and the chips with a gray hash pattern were not used in the simulation. The most notable result is the rightmost column, showing the heatmap of dumped packets for each chip on the board. In all simulations, the communication tests showed that packet loss has a clear dependence on the relative placement of the sources and targets. Additionally, the middle column in Figure 5.5, showing the overall traffic of multicast packets, demonstrates that the packet traffic is heaviest in the central diagonal set of chips, regardless of target placement. For this reason, in subsequent experiments, we try to avoid placing traffic-heavy nodes in this region.

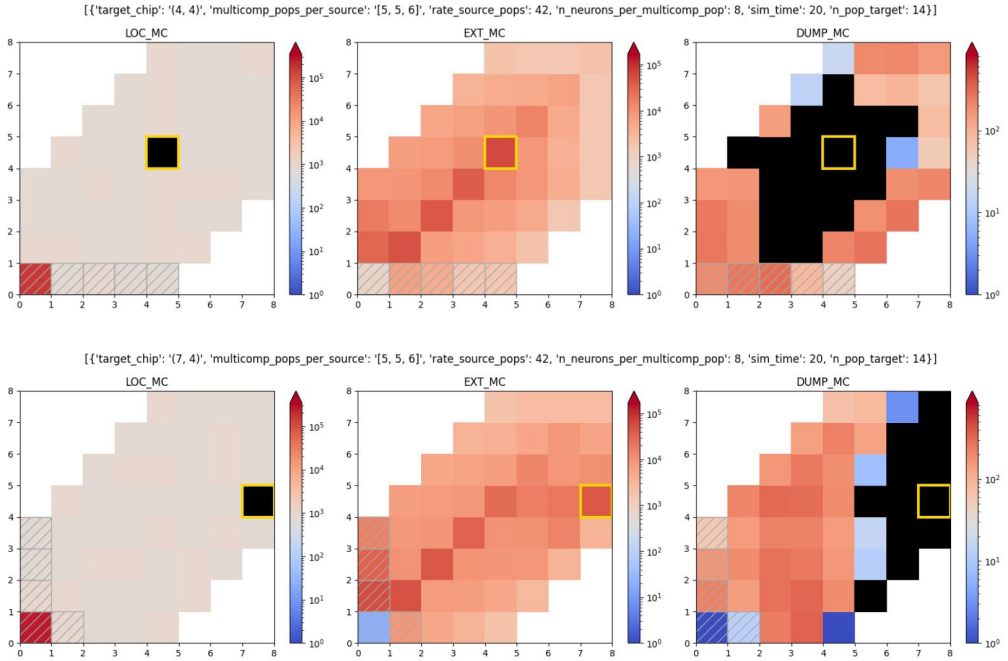

Fig. 5.5 Results of the single-neuron communication test with 16 input populations connected to each target synapse core. In the top figure, the target is placed on chip (4,4), while in the bottom figure, the target is chip (7,4). Left column: number of local multicast packets. Middle column: number of external multicast packets. Right column: number of multicast packets dumped by the chip router.

Custom placement, routing, and data visualization

In our second test, we attempted to find an optimal placement for a single-board network. The target was a reduced single-layer network featuring as many pyramidal neurons as could fit in a single 48-chip board.

In order to accomplish this research, the author developed several pieces of software interacting with the sPyNNaker framework. First, a custom placement script intervenes at the partitioning and placement stage of the sPyNNaker stack, in order to inject a predefined placement that overrides the default setting. Second, a visualizer script was developed in order to quickly analyze the results of the placement. The visualizer shows the position of all neuronal populations and automatically calculates the size and direction of packet flow. It also visualizes a heatmap of lost multicast packets in the hexagonal mesh layout.

In Figure 5.6 we can see the visualizer output when the sPyNNaker default algorithm attempts to place the populations on the Spin5 hardware. This default placement results in failure, with 1105 packets lost for a 6 s simulation.

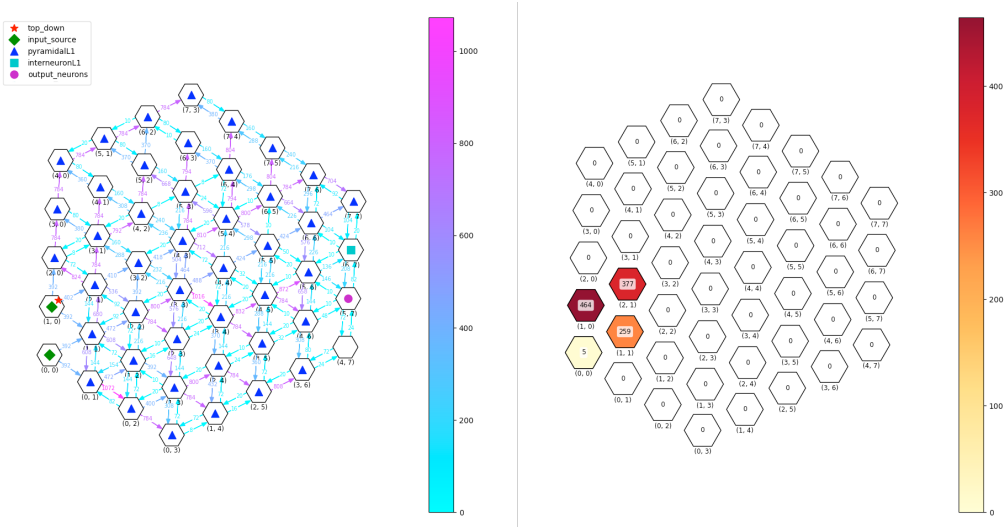

Fig. 5.6 Visualization of the default placement for the reduced pyramidal-based network.

Figure 5.7 shows the best placement, empirically found by moving the source populations off the diagonal and placing the strongly-interconnected top-down and output neuron populations on the same chip. With this custom placement, simulation is successful and no packets are lost.

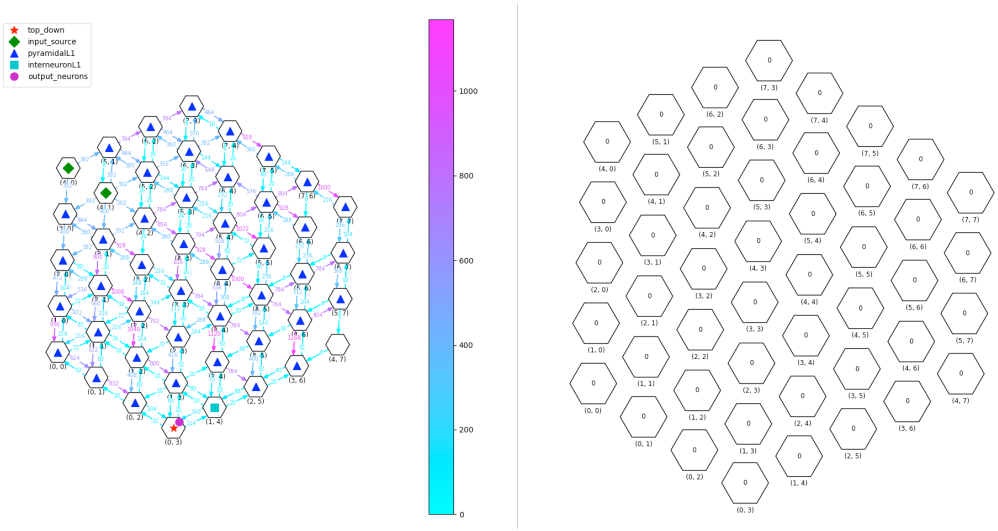

Fig. 5.7 Visualization of the custom placement for the reduced pyramidal-based network.

The customized routing script was later expanded to include customized routing. By default, SpiNNaker uses a technique called Neighbour Exploring Routing (NER) [259]. While this algorithm is adequate for normal SNN applications, it can create unwanted excessive load on the diagonal path, as seen in Section 5.3.1, as well as conflicting directions for the traffic paths, as in Figure 5.3. As a proof of concept, the author created an ad hoc traffic-aware routing algorithm based on the original NER. In this algorithm, the source neurons are identified and given priority for routing, avoiding conflicting directions and minimizing the number of packets using the diagonal. The generated routing lists were successfully injected and executed within the sPyNNaker framework. The visualizer was also expanded with GUI options allowing to select a neuron population in order to visualize the routing of its packets. Figure 5.8A shows the custom routing executed for the input source at chip $(4,0)$. The arrows show the path followed by packets originating from this population.

The custom placement and routing options have also been tested on a multi-board environment. At the time of writing, the only multi-board environment available to the author was the remote SpiNNaker supercomputer accessible through the Spalloc client. Due to limitations in the way machines are allocated to individual users, however, the placement and routing could not yet be tuned in an optimal way. For instance, in Figure 5.8B, a slightly larger network was placed on

3 adjacent boards. However, the allocated cluster had a dead chip at $(4,6)$, which could not be controlled or foreseen at the moment of deployment.

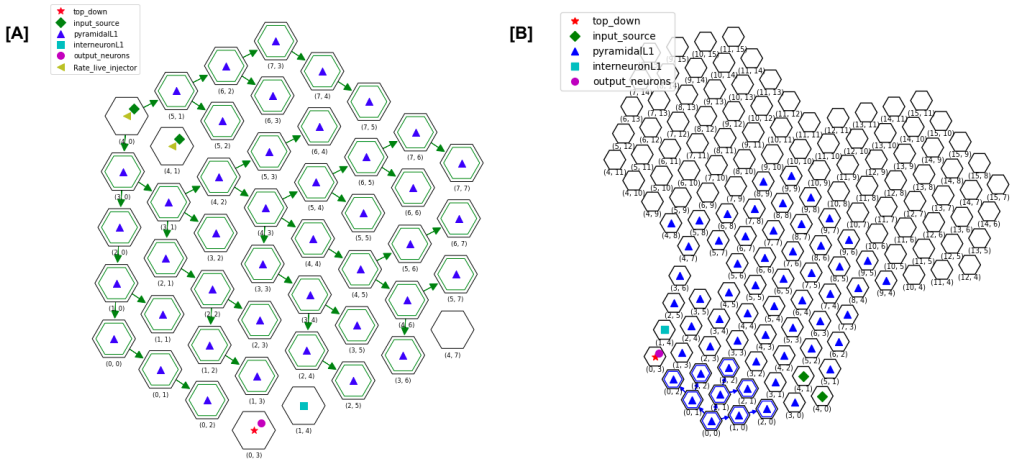

Fig. 5.8 **(A)** Custom routing and visualization feature. Highlighted is a customized route for the input source at chip $(4,0)$. **(B)** Multi-board placement and routing feature. Highlighted is the route for packets originating from the pyramidal population at core $(0,0)$.

The lack of accessibility to multi-board testing has therefore slowed down the research into multi-board performance, which is nonetheless extremely important for the feasibility study of the multi-compartmental network, as the board-to-board connections inevitably introduce delays and non-idealities in the data exchange. It remains to be seen, then, whether innovations in placement and routing will be sufficient to guarantee efficient simulation of the multi-compartmental neurons on SpiNNaker. However, this study demonstrates the exceptional flexibility and adaptability of this neuromorphic platform, and confirms the importance of fully-programmable hardware simulators in this field of research, as they allow for continued software-based innovation and experimentation that could not have been foreseen at the time of their development.

5.4 Chapter summary

As we have seen in this chapter, the proper functioning of a neuromorphic application requires the interaction of many complex pieces of software. On the one hand, SNN specification software empowers developers to create simple, abstract and portable descriptions of the desired models, also handling their compilation

and deployment on simulated or physical hardware backends. On the other, techniques that had been previously developed for ML and ANN applications remain relevant for the specific needs of SNNs. Methods such as hyperparameter optimization are an important tool to refine the details of a network and significantly improve the results.

Finally, the importance of flexible system software serving as a middle layer between the specification and deployment software and the hardware's low-level operation is not to be discounted. The author's investigation found that SpiNNaker's system software stack could effectively interact with the PyNN libraries to enable the implementation of new functions, even compensating for restrictions that could not have been foreseen at the moment of designing the hardware; for example, exploring new placement and routing strategies to accommodate novel and more computationally demanding neural models. In Chapter 6, we will follow this lead into an exploration of the computational abilities of neuromorphic hardware.

Chapter 6

Hardware platforms

Neuromorphic computing focuses on the replication of neural processes in novel computer architectures [260, 52], directing efforts toward the development of specialized neuromorphic hardware [24, 261, 25, 26, 262–264, 61].

The goal of a neuromorphic platform is to imitate animal brain behavior using an event-driven mesh of biologically realistic neuron models called a Spiking Neural Network (SNN), wherein each computational unit is a set of neurons. Among the neuromorphic platforms in active development are:

- BrainScaleS [265]: the project's purpose is to produce a hardware platform capable of simulating biological neurons at speeds faster than real-time. It is realized using transistors that operate over their threshold, fitted with high-end FPGAs to externally configure synapses and neurons, and also offers analogue synapses integrated at wafer scale. This architecture's goal is to simulate SNNs in accelerated time, so that a simulation that would ordinarily take months or years can be completed in minutes or hours.

- Dynap-SEL [26]: an acronym for Dynamic Asynchronous Processor Scalable and Learning, it is a VLSI microprocessor. It has five neuromorphic cores, and the neurons are linked together by a multi-router hierarchical organization based on a mesh schema. There are two versions with different grids: 16x16 and 4x4. This architecture was designed with edge computing applications in the IoT and Industry 4.0 areas in mind.

- Loihi [25]: created by Intel in 2017, it is a self-learning neuromorphic research chip with 128 cores and around 130 000 neurons. The entire architecture is digital, with an asynchronous design to reduce power usage. The design has potential uses as an SNN-based coprocessor in heterogeneous SoCs.

- SpiNNaker [266]: a platform that targets real-time SNN simulations using an event-driven computing method similar to that observed in the human brain [24]. The fundamental distinction between SpiNNaker and other architectures is that it does not rely on VLSI customization at the wafer or transistor level: the platform's nodes are made up of hundreds of ARM-based general-purpose processors. As a result, SpiNNaker can natively support any C program written for the ARM platform.

In this chapter, we will see two case studies delving into the details of neuromorphic hardware characteristics and performance metrics. The main platforms used within this research are SpiNNaker and Loihi. In Section 6.1, we will thoroughly benchmark the SpiNNaker board's efficient communication infrastructure using the MPI paradigm for parallel computing. Section 6.2 is a detailed comparison of the performance of the Jetson GPU and the Loihi neuromorphic platform on the Braille dataset classification task [3].

6.1 Exploring the SpiNNaker communication infrastructure with MPI

SpiNNaker is an entirely digital neuromorphic hardware platform that integrates high-throughput multicore and distributed-memory devices with a dense network of interconnections, forming a homogeneous lattice that connects all of its computing units. The system was created specifically for the simulation of Spiking Neural Networks (SNNs): to accomplish this, the platform includes massively parallel processing and a powerful communication architecture based on small packet transfer. Previous research [267, 268] demonstrated that the SpiNNaker topology outperforms traditional multicore architectures when tackling massively parallel computation, ensuring superior scalability as input sizes grow. While most neuromorphic designs are focused on a particular implementation for a spiking

neuron model, the ARM-based SpiNNaker platform supports the computation of a wide range of neuron models as well as general-purpose applications [28, 269, 270]. To enhance the accessibility and efficiency of general-purpose programs, Barchi et al [271] extended the SpiNNaker application stack by integrating the SpinMPI library, which takes full advantage of the platform's brain-inspired connectivity mesh for effective inter-chip communication. In the research presented in this section, the author demonstrates through on-board tests that MPI provides for simple and efficient general-purpose code implementations, and compares its results to those produced using the SNN framework [5]. The targeted application is PageRank (PR), an algorithm that had been evaluated in previous work by Blin et al. [267] through the development of a bespoke SNN implementation on SpiNNaker. The author implemented PageRank with the MPI programming model and transferred it to the SpiNNaker platform, in order to compare the scalability of the MPI application to that of the corresponding SNN implementation and use the PageRank algorithm's characteristics to test the behavior of the MPI implementation on SpiNNaker when faced with large communication requirements.

6.1.1 The SpiNNaker hardware

The SpiNNaker platform is available in two flavors. The Spin5 board contains 48 chips, whereas the Spin3 board contains four chips. A SpiNNaker chip contains 18 ARM processors, a proprietary 6-link router, a System NoC, and 128 MB of SDRAM [24]. The chip is regarded as the fundamental building component of the design; its architecture is depicted in Figure 6.1, which describes the four chips of a Spin3. By default, internal clocks run at 200 MHz to ensure minimal power consumption. Furthermore, each CPU has two tightly coupled memories (DTCM and ITCM) for data and instructions. Within the low-level programming elements included in SpiNNTools [272], cores are organized as follows: 1 serves as a Monitor Processor (MP), 16 as Application Processors (AP), and the final as a backup in case of hardware failure. The Monitor Processor is in charge of running the low-level SARK operating system, while the Application Processors execute the user program.

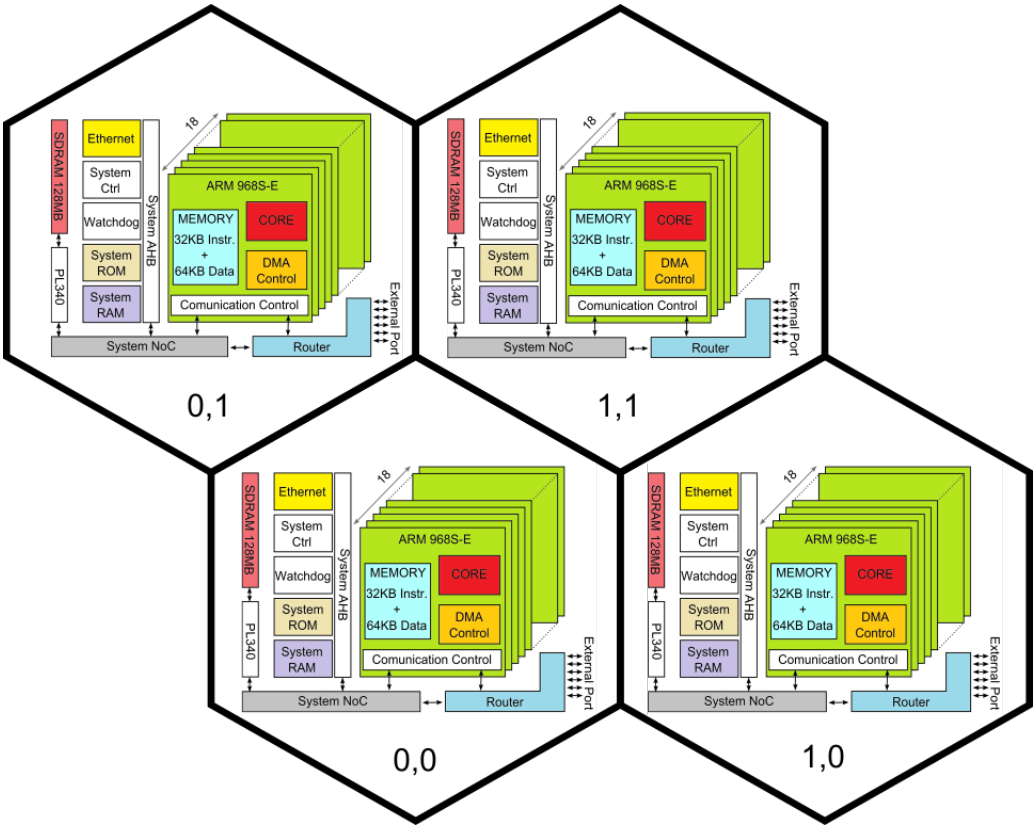

Fig. 6.1 The Spin3 design. [5]

A critical component of the design is the physical setup of the communication channels. These channels could connect two cores on the same chip or two cores on different processors. The former is the simplest scenario, where "close" communication occurs in a synchronous manner, taking advantage of a 128 MB Synchronous DRAM. In the latter situation, communication is asynchronous, drawing on mechanisms found in the biological brain. A 1024-line integrated router, one on each SpiNNaker chip, drives packets. Because embedded routers are small CAMs, they have a low latency (~0.1 µs per hop) [270]. Despite constraints on synchronous packet transmission [30, 4], the router's unique architecture enables the transmission of two operative packet types: Multicast (MC) and Point to Point (P2P). Routers facilitate the transfer (and re-transmission) of 72-bit packets; for longer communications, the platform's low-level APIs must be used to encapsulate them into datagrams. Said APIs also provide mechanisms for packet reconstruction: the SpiNNaker Datagram Protocol (SDP) layer manages

large-packet communication up to 256 bytes [266]. The Monitor Processor is in charge of running SDP facilities.

SpiNNaker comes with three basic communication profiles, regardless of the type of data:

- P2P (*Point-to-Point*): a core communicates data to another core on the same or a different chip via the monitor processor.
- *Multicast*: a single core communicates to a subset of cores on the connected chips at the same time.
- *Broadcast*: a single core communicates to every other core on the board at the same time.

Because the base network is structured as a mesh, any chip can initiate a communication channel (broadcast, multicast, or P2P) or act as a forwarding spot to assure connectivity between two chips that are not physically connected.

Applications and experiments on SpiNNaker are sustained by two environments: the Host PC, which runs Python modules [28], and SpiNNaker itself, which runs a software stack entirely written in C and Assembly language [272]. On top of this core is the SpinMPI library [271], a SpiNNaker translation of the MPI paradigm that seeks to provide a high-level interface for the user to easily control communication among physical cores: the Application Command Framework (ACF), which uses lengthy datagrams to encode application commands as a Remote Procedure Call (RPC) [273]. The Multicast Communication Middleware on the hardware side determines the format of each packet based on the communication profile (unicast or broadcast).

6.1.2 SpinMPI

SpinMPI [271] is a package that provides support for MPI (*Message-Passing Interface*) communication and synchronization primitives on SpiNNaker. The MPI paradigm includes a synchronization barrier function as well as message-passing primitives that can be used to handle synchronous communication on a platform with distributed-memory computing units such as SpiNNaker. SpinMPI also includes standards for the implementation of communicators. A communicator is

an interface that collects methods for managing synchronous communication among variously structured groups of units; many MPI communicators can exist on the same physical network; SpinMPI's current release includes point-to-point and broadcast communicators. Each process in the MPI paradigm is assigned a unique identification, or *rank*. A communicator's processes are identified by their rank: for example, a point-to-point communicator involves two processes, one triggering an *MPI Send* and the other enabling an *MPI Receive*; on the other hand, a broadcast communicator includes every process in the system. To synchronize all processes, the *MPI Barrier* is invoked. At the end of the execution, the findings and metadata are saved into dedicated reports, which are downloadable straight from the memory of SpiNNaker cores.

SpinMPI includes synchronization methods to ensure that multiprocess computations are valid. Although synchronization is not needed in point-to-point communication, both multicast and broadcast require it to ensure that messages are received by all processes in the communicator. The board's chips are separated into logic subregions within the SpinMPI communication logic: a hierarchical strategy assigns chips to concentric ring-shaped layers based on the distance of a node from the axis origin, which corresponds to the chip $(0,0)$. Ring 0 includes only the chip $(0,0)$, which manages Ethernet communications with the host and other Spin5 boards. Ring 1 is formed by its neighboring chips — $(1,0)$, $(1,1)$, and $(0,1)$; the other rings are constructed as shown in Figure 6.2.

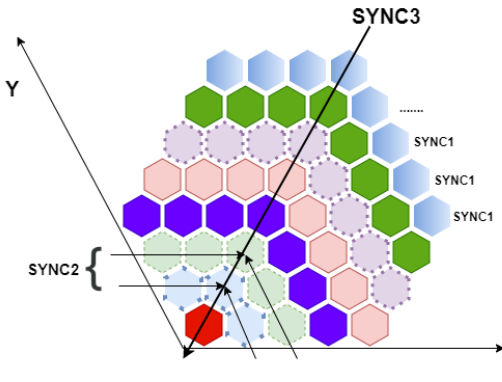

Fig. 6.2 The synchronization rings of the 48-chip Spin5 board. [5]

A three-tiered hierarchy is used for broadcast synchronization. Each level is overseen by one or more *managers*, who are responsible for gathering all synchro-

nization messages from that level. The manager then sends a synchronization packet to the upper level (Figure 6.2). The tiers of hierarchy are as follows:

1. *Chip level*: *SYNC1* packets are sent to all cores of a SpiNNaker processor. After collecting all synchronization packets, a *SYNC2* packet is produced and transmitted to the upper level.

2. *Ring level*: a ring is made up of chips that are the same distance from $(0,0)$. The chips labeled $(x,y)|x = y$ are the *SYNC2* managers, who are in charge of collecting the group's synchronization packets. Each ring master knows how many *SYNC1* packets should be sent; once all of the predicted packets have been received, they send a *SYNC3*.

3. *Board level*: all level 2 managers send *SYNC3* signals to the level 3 manager, i.e., chip $(0,0)$. When all level 2 managers have submitted their packets, the level 3 management sends a *SYNCunlock*, also known as an ACK packet, through MPI Broadcast. The synchronization phase is now complete.

6.1.3 PageRank

As a benchmark application for the developed SpinMPI module, we selected the PageRank algorithm. The PageRank method [274] appears to be a logical fit for architecture designs such as multiprocessing neuromorphic hardware, as it involves iterative computing on a highly interconnected graph with high communication requirements. The graph topology is complex enough that scalability concerns easily arise, both in terms of processing time and in the number of packets traveling throughout the network. In fact, at every iteration, each node must share its data with all of its neighbors; the creation of an expansive, densely connected graph where each node acts independently necessitates an architecture that supports a large number of concurrent operations and efficient inter-process communication: SpiNNaker meets all of these characteristics.

Blin et al. [267] set out to prove that, for massively parallel queries that also require the exchange of numerous short messages among workers, neuromorphic technology outperforms standard architectures in terms of scalability. We propose to validate this assertion by presenting MPI libraries specifically created for SpiNNaker, a valuable tool to be employed alongside Spiking Neural Networks on

neuromorphic hardware. A secondary result of the presented research are the clear potential benefits that a mesh of computing elements duplicating the SpiNNaker interconnection design might bring for running PageRank-like algorithms.

PageRank algorithm

PageRank is a well-known graph-based method that provides a quantitative classification of a set of items. Its original purpose was to categorize web pages based on the number of links a single page has in comparison to all others [274]. When PageRank was introduced in 1998, the algorithms used by internet crawlers for indexing web pages were becoming an increasingly important topic as the World Wide Web expanded exponentially in size. The PageRank model was designed to provide effective scalability for the WWW ecosystem by creating high-quality categorization based on hypertextual and keyword-based search rather than statically indexed directories. The classification is based on providing a rating, or rank, to each website that represents its "popularity" across the web, depending on the cardinality of incoming links to the page. The formula in Equation 6.1 is used to compute a page's rank [275]:

$$\mathrm{PR}(A) = \frac{1-d}{N} + d \cdot \left(\sum_{k=1}^{n} \frac{\mathrm{PR}(P_k)}{\mathrm{C}(P_k)} \right), \tag{6.1}$$

where:

- PR(A) is the PageRank score for the target page A
- N is the total number of pages in the domain
- n is the number of pages on which a link involving both P_k and A exists
- C(P_k) denotes the total number of incoming links for P_k
- d is the damping factor, which is used for calibration; a typical value is 0.85.

PR() and d are both probabilities. PR(P_k) indicates the likelihood that a user would randomly query a particular page P_k; d is the possibility that a user will stop following links and begin a completely new and unrelated search. In contrast, $1-d$ represents the complimentary probability that the hypothetical visitor will

move on to a connected page. To calculate the PageRank score for a generic item *A* in a graph, each of the vertices must know the PageRank score for every other vertex.

PageRank Implementation on SpiNNaker Using *sPyNNaker*

Blin et al. [267] suggested a PageRank solution for SpiNNaker that uses the SNN framework (*SNN-PR*). The implementation is built on the *sPyNNaker* software libraries [28], which provide high-level services for developers to interact with SpiNNaker. *sPyNNaker* is primarily used to map "web page" objects onto spiking neurons in *SNN-PR*. For that purpose, the rank of a page is modeled as a neuron's membrane potential. This modeling choice is based on the real-world low-level activity of a spiking neural network (SNN), allowing it to take advantage of the SpiNNaker board's existing simulation infrastructure. The actual neuron implementation is a variation of the classic Leaky Integrate-and-Fire (LIF) model, which provides both the PageRank application's transient function and a message parsing algorithm. While *SNN-PR* strives to design a synchronous method, the underlying SpiNNaker modules do not support synchronization across cores, since it isn't generally required in SNN simulations. *SNN-PR* uses a semaphore to synchronize the computation of vertices belonging to the same core. Because cores iterate asynchronously to each other, a buffer mechanism is used to ensure that incoming messages are handled at the right time step.

SNN-PR exhibits good scaling due to its effective SNN-based operation, which fully uses the SpiNNaker platform to its advantage. However, the degree of complexity of such a unique implementation is fairly significant because a new neuron model, as well as auxiliary utilities to manage message forwarding and asynchronous buffering, must be developed. Furthermore, due to inherent constraints in the SNN libraries, each core can handle a maximum of 255 vertices, each of which corresponds to a spiking neuron. Finally, due to the huge quantity of data circulating in the network, *SNN-PR* necessitates an increase in the machine time step from 1 to 25 ms; this gives the routers and cores sufficient time to handle incoming messages and avoids simulation errors due to packet loss.

6.1.4 Implementation of PageRank with MPI

The computation of PageRank for a particular blob of sites is not difficult in a single iteration; but, due to the huge size of the average web network, a parallelization framework is frequently necessary. Because the rank of any node is determined by the rank of its neighbors, each worker's computation is not independent: at the end of each iteration, a synchronized data exchange step is required. As detailed in Forno et al. [5], our MPI-PageRank (*MPI-PR*) implementation takes as input a binary file holding the list of edges in the graph. Each edge is represented by an integer tuple (*Source, Destination*), each of which represents the ID of a node.

The software can be separated into two parts, as shown in Figure 6.3: setup (A) and PageRank loop (B). The MPI worker with rank 0 is designated as the MPI Master during the configuration step (A). This core is responsible for collecting the edges list from the file system and sending it to the other MPI workers; each worker core receives a copy of the entire graph description. This decision was made with the intention of porting to SpiNNaker, a distributed-memory platform. The problem data is transmitted via a single MPI_Bcast call, and the graph data's segmentation into packets is entirely managed by the underlying SpinMPI library. The core that was previously operating as the MPI Master resumes work as a regular MPI worker at the end of this transmission.

After receiving the edges list, each MPI worker executes a graph preprocessing step. Each worker is given a subgraph of size $k = n_{vertices}/n_{workers}$; for each node in the subgraph, the worker scans the edge list, creates the list of incoming connections (represented by the source vertex's ID), and counts the node's outgoing links. As a result, each worker prepares the bare minimum of data structures required to calculate PageRank for each node, and the preprocessing effort is distributed evenly among the MPI workers.

During the PageRank loop (B), each worker updates the rank for all the vertices to which it is allocated, as shown in Equation 6.1. Then, each worker sends its new rankings to each other worker in turn, which is performed with a single MPI_Bcast instruction. To reduce computational efforts, each worker delivers the ranks of its nodes divided by the number of their own outbound links [276] [23]. Loop (B) is repeated until convergence or a defined number of steps is reached. At the end of the computation, each worker returns a list of PageRank values for the vertices to which it was allocated.

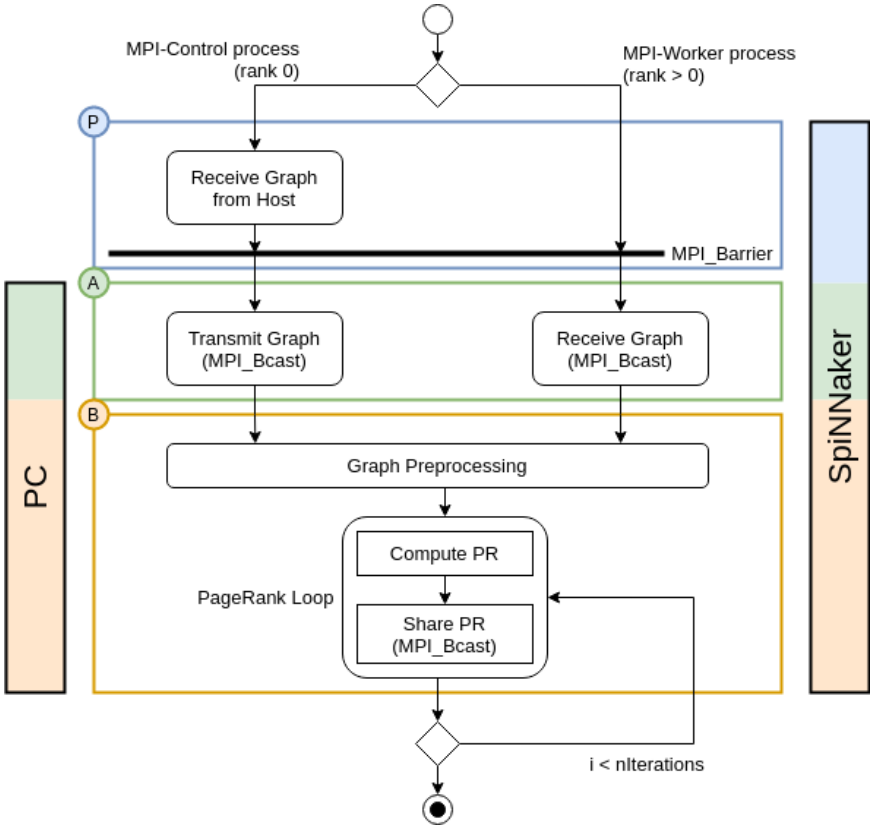

Fig. 6.3 Flowchart of *MPI-PR* implementation on a general-purpose architecture and on SpiNNaker. Step A is for configuration, Step B is for PageRank calculation, and Step P is for transferring the problem data to the SpiNNaker board. [5]

Adaptation of PageRank with MPI for SpiNNaker

The MPI program for PageRank outlined in Section 6.1.4 is compatible with any MPI-enabled device, including ordinary PCs. To run the same software on SpiNNaker, an additional stage (P) is introduced to the process: data are sent from a host machine to the SpiNNaker board during this phase.

The SpinMPI Python library enables the host to configure the size of the MPI Context, which is defined as the number of chips and cores to be employed during computation: the capacity of the MPI Context equates to the number of available MPI workers. Urgese et al. [57] provides detailed information regarding the host-board connectivity and setup process. Finally, the MPI Runtime installs the application binaries onto the board and starts the program.

After starting the application, the host writes the problem data straight to the memory of the MPI worker of rank 0, which corresponds to processor $(0,0,1)$. To make the most of the available memory on SpiNNaker, we choose to treat each node ID as a 16-bit unsigned integer, allowing for a maximum of $2^{16} = 65\,536$ nodes in a graph.

Until this procedure is completed, all MPI employees must wait on an MPI Barrier. Phase (A) can begin once MPI worker 0 receives the problem data; calculation continues as specified in Section 6.1.4.

6.1.5 Comparison of SNN-PR and MPI-PR implementations

In this part, we compare the efficiency of our MPI-based PageRank implementation (*MPI-PR*) on SpiNNaker to that of Blin et al. [267] (*SNN-PR*). We set up our tests in the same manner as the cited study describes: a runtime option indicates how many nodes should be assigned to a single worker, and every experiment runs the PageRank loop 25 times.

We compare the execution times of *MPI-PR* with *SNN-PR* on a fixed-size graph ($|V| = 255, |E| = 2550$) with different number of cores in Figure 6.4. It should be noted that the size of 255 was chosen in the original work because it reflected the highest number of vertices per core permitted by the PyNN framework; here, because the MPI framework places no such constraints, the total number of vertices that can be handled is limited only by available memory. A single worker is sufficient for a graph of said size, and the time cost of MPI transmission is never incurred. Even with MPI engaged and several workers involved, we get faster computation speeds than *SNN-PR*, up to 12 cores. We can also see that MPI improves computation time only up to 8 workers: after that, the cost of MPI communication outweighs the time saved by multithreading the computation, and employing many cores becomes less advantageous than in the *SNN-PR* version. The dashed line in the graph represents the time spent by workers in the broadcast phase: we can see that the cost of communication grows with the number of cores faster than the cost of the PR compute step shrinks. Additionally, in this experiment, assigning MPI workers to the same chip or dispersing them across four chips has minimal effect on execution time.

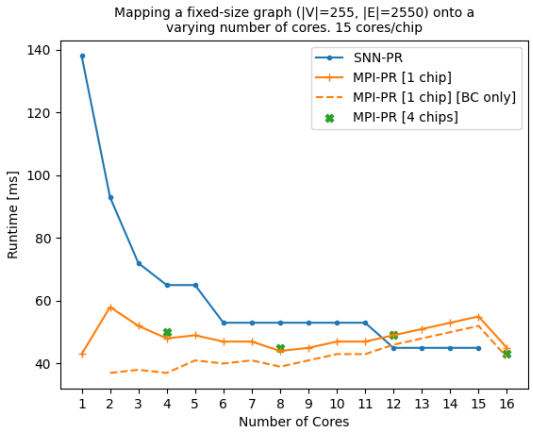

Fig. 6.4 *SNN-PR* and *MPI-PR* execution times on a fixed-size graph utilizing only one SpiNNaker chip. [5]

The behavior of *SNN-PR* and *MPI-PR* on a bigger graph, distributed among cores employing up to four SpiNNaker chips, is depicted in Figure 6.5. We see a repeat of the prior behavior: MPI performs best at 10 cores, when the subgraphs are sufficiently small that all of the problem data fits in the cores' DTCM. This is also the point at which the calculation/communication cost offers the optimal tradeoff. However, because of the higher communication costs associated with MPI, *MPI-PR* fails to scale as well as *SNN-PR* on this graph.

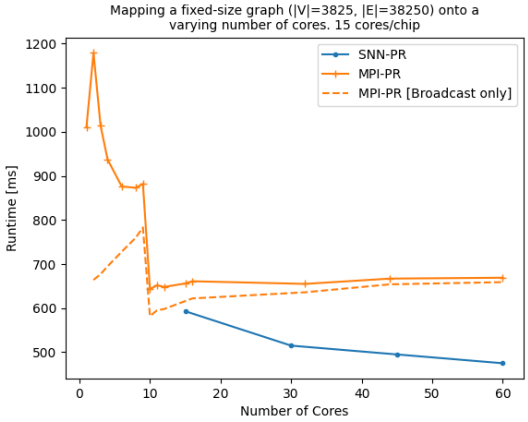

Fig. 6.5 *SNN-PR* and *MPI-PR* execution times on a fixed-size graph utilizing up to four SpiNNaker chips. [5]

In Figure 6.6, we evaluate the scalability of *MPI-PR* compared to *SNN-PR* and the standard multicore implementations studied in Blin et al. [267]. The results are shown as the normalized execution time relative to the single-core execution

time, corresponding to the 255-node graph. Both SpiNNaker implementations exhibit smoother and more favorable scaling, demonstrating the efficiency of the SpiNNaker many-core architecture's custom toroidal-shaped, triangular-mesh communication network. Overall, however, *MPI-PR* scales better than the PyNN implementation; when employing 15 cores, *MPI-PR* scales around 1.75× faster than *SNN-PR*.

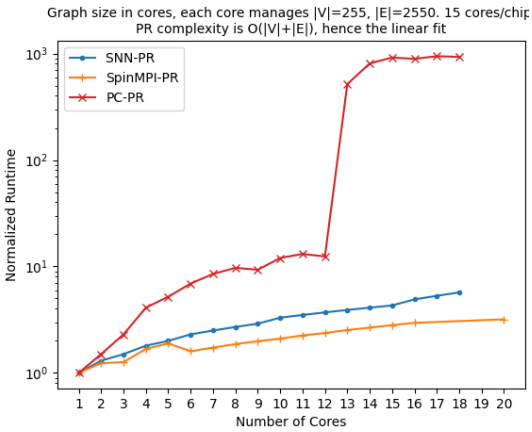

Fig. 6.6 Scalability of three different PageRank implementations: *SNN-PR* and *MPI-PR* on SpiNNaker, and PC-PR on a typical multicore architecture. [5]

6.1.6 SpinMPI Performance Analysis on PageRank

Let us now evaluate *MPI-PR*'s performance on larger graphs. Figure 6.7 depicts the PR computation time as a function of the number of workers (i.e., cores) involved in the computation for a fixed-size graph of $|V| = 768, |E| = 7680$. Depending on the number of cores and rings engaged in the context, multiple configurations are feasible for a given number of workers; for instance, a 48-worker configuration can be obtained by selecting (7 rings, 1 core per chip), (4 rings, 2 cores per chip), or (3 rings, 3 cores per chip). The graph's black line depicts the mean runtime for all of the equivalent settings.

The SpinMPI design does not scale indefinitely for the PageRank task; in fact, the computation time trend is rather erratic and tends to grow when the number of cores exceeds a specific threshold. There are a few points to be made: first, the trend in PR execution time is predominantly determined by the duration of the broadcast communication step. It is common for communication times

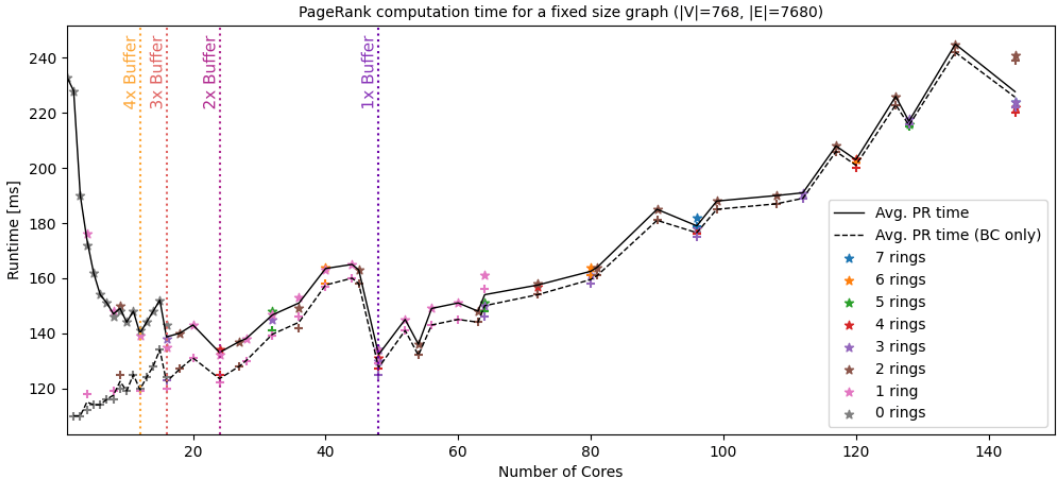

Fig. 6.7 *MPI-PR* computation and communication timings on SpiNNaker with a medium-sized graph: the plot shows how the communication buffer's consumption rate affects the broadcast time. [5]

to increase as the MPI context expands, especially when employing broadcast communications that require board-wide synchronization; as a result, the best PR execution time is at a relatively small context of 48 cores.

Several variables contribute to the uneven trend of the broadcast time. As shown in Figures 6.4 and 6.5, one such element is the memory location of the data to be sent and received. For this experiment, we changed the software so that the PR array would always be stored to DTCM, hence this factor is no longer relevant. The dimension of the MPI transmission buffer, on the other hand, does play a role.

The data to be transferred is written to a fixed-size communication buffer in the SpinMPI framework. This buffer's size is specified at compile time, and the default value is 64 B. The more cores participating in the context, the smaller the subgraphs allotted to each core: because the PR rankings are stored as 4-Byte fixed-point integers, the size of the send/receive buffer for this arrangement is exactly 64 Bytes at 48 cores, when each core handles 16 vertices. Therefore, 48 is the minimum number of workers required to fill the communication buffer once; by minimizing write/read access to this buffer, the broadcast time is decreased as well. The graph's vertical lines illustrate places where the broadcast time decreases due to variations in the number of required buffer accesses.

Finally, in Figure 6.8, we examine *MPI-PR*'s behavior when dealing with a very large graph. Due to the 16-bit integer format of the vertex ID, the 65 536-vertex

graph is the largest that the algorithm can assess. The figure depicts the trend in execution time for various ring arrangements as the number of cores per chip rises. Again, we can see how each ring layout has an optimal number of cores per chip after which communication costs become prohibitive in comparison to the time saved in PageRank computation. Most notably, we can see how execution times increase when the same number of workers are concentrated in fewer chips (fewer rings), but decrease when the workers are spread out over numerous chips (more rings). This is due to the fact that for a huge graph like this, none of the problem data can be saved to DTCM; instead, the vertex information and PR arrays are stored in RAM. All cores on a single chip must fight for access to the same memory bank, since RAM is chip-local. We can see how RAM access time is another crucial component in SpinMPI performance, as having fewer workers competing for the same chip's RAM produces better results.

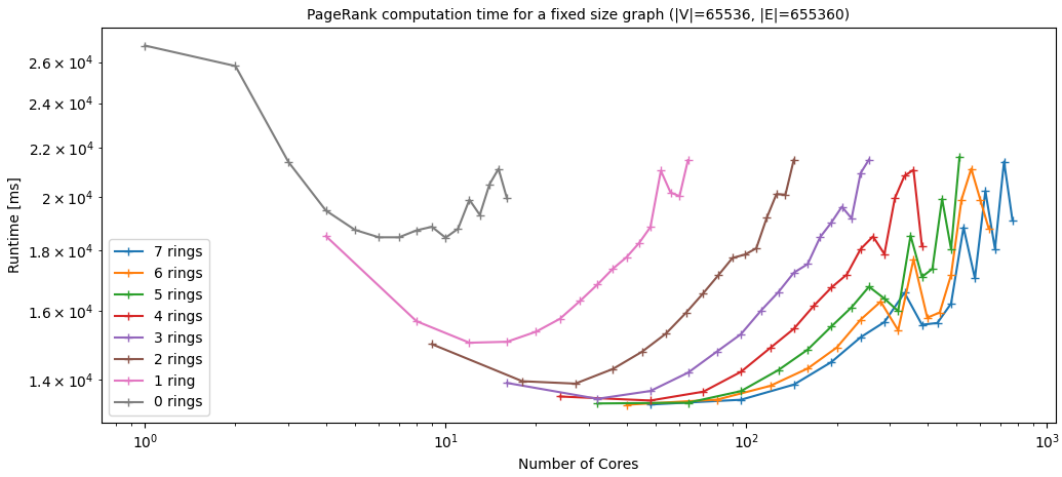

Fig. 6.8 *MPI-PR* computation and communication timings on SpiNNaker with a large graph: the diagram shows how different placements of the same number of cores affect execution time. [5]

Overall, the study detailed here showed that for communication-heavy applications like PageRank, the parameters influencing MPI execution time on SpiNNaker are numerous, complex, and difficult to determine. To summarize our findings, such factors include (but are not necessarily limited to):

1. The tradeoff between the calculation time saved by parallelization and the cost of MPI Broadcast communication, which increases with worker count (Figures 6.4 and 6.5).

2. The memory location of the data to be sent and received over MPI (DTCM or SDRAM) (Figures 6.4 and 6.5).

3. The size compatibility between the data to be delivered and the MPI communication buffer (Figure 6.7).

4. The worker density on each chip, which effects SDRAM access time (Figure 6.8).

6.1.7 Conclusions

An MPI-based version of PageRank was developed to test the scalability of the SpiNNaker multicore architecture when running a parallel algorithm with high communication requirements and a low computational effort per node. We contrasted the simple MPI paradigm to Blin et al.'s method (*SNN-PR*), which uses a customized spiking neuron model to accomplish the rank update while leveraging the conventional SNN communication infrastructure. In comparison to *SNN-PR*, the MPI solution supported larger graphs, as well as worker synchronization and a lower computational cost per core. Finally, we confirmed that the SpinMPI library, which provides MPI support for SpiNNaker, enables users to easily adapt any MPI method built for normal computers to the SpiNNaker neuromorphic platform, creating an interface between any MPI-compliant C program and the native SpiNNaker communication framework without the need for modification of the original code.

The SpiNNaker platform's efficient connectivity architecture, in addition to being well-suited for SNN applications, shows promise for low-power parallel execution of tasks in the edge computing domain. On the other hand, as a collection of massively parallel computation elements immersed in a distributed-memory environment with linearly scaling intercore communication, said architecture may be the ideal silicon implementation for the MPI paradigm, to the point where it may be worth consideration even for the realization of systems on a larger scale.

6.2 Braille classification on Loihi vs. GPU

With the perspective of implementing classification of time-varying signals on real-time embedded devices, it was necessary to measure critical performance indicators relevant to real-world deployment by deploying the networks on several hardware platforms. The authors examined power utilization, energy consumption, and computing latency in order to gain some insight about deployment viability in real-world circumstances. The NVIDIA Jetson Xavier NX, a commercially available computing platform with a System-on-Chip (SoC) that integrates a CPU and GPU, and Intel Loihi, a neuromorphic processor dedicated to accelerating SNNs, were the two platforms selected for comparison. This choice was motivated by platform-related factors, such as high integration and availability, as well as our ultimate goal of deploying the algorithms in a real-world environment on robots.

6.2.1 NVIDIA Jetson Xavier NX

The Xavier NX is the most powerful model in the NVIDIA Jetson product line, a collection of small, embedded computing platforms with a focus on edge AI. Despite being a general-purpose platform, its architecture and software are similar to those of full-fledged ML workstations. Using this off-the-shelf hardware, the authors compared several standard time-series classifiers and evaluated the differences between conventional and event-based algorithms. The obtained inference metrics provide insight into the performance that may be expected using the same hardware for deployment.

The module's integrated power monitor was used to measure power usage. While the CPU/GPU and a SoC power rail are also measured by the power monitor, due to a lack of public information on what components these rails supply, and because a productive application requires the complete system, the measurement of the main power rail was nevertheless considered a suitably representative figure for comparison.

Standard time-series classifiers, known to work with time-variant datasets, were executed on this platform: Fully Convolutional Network (FCN) [277], Residual Neural Network (ResNet) [277, 278], Encoder [279], Time-CNN (TCNN) [280],

and Inception [281], available as implementation on GitHub [281, 282]. Given the time-dependent nature of the dataset, a recurrent neural network was also selected. The design tested consists of a single-layer Long-Short Term Memory (LSTM) with 228 hidden nodes, followed by a regular fully-connected layer of 228×27 output neurons that conduct the classification, for a total of 225 975 trainable parameters. The authors used one of these LSTM with frame-based input and one with event-based input, denoted as eLSTM.

The conventional time-series classifiers were then compared with a Jetson implementation of the spike-based FFSNN and RSNN described in Section 5.2.2. For each inference, the following metrics were assessed:

- Inference time (i.e., computational delay) per sample, which is calculated by dividing the total inference time by the number of samples processed.
- Power utilization across overall inference time as well as per sample (minimum, maximum, and average).
- Energy consumption overall and per sample. The total energy is computed by multiplying and adding each power measurement over a 50-ms polling interval. We then divide the total energy by the number of samples processed to get the energy per sample.

Conventional time-series classifiers

Energy consumption may be considered the most important parameter, as it combines both power consumption and inference time per sample. Average power utilization provides insight into the power budget required to accomplish specific inference times. While energy and power consumption are critical requirements for applications running on battery power, inference time is essential in real-time tasks.

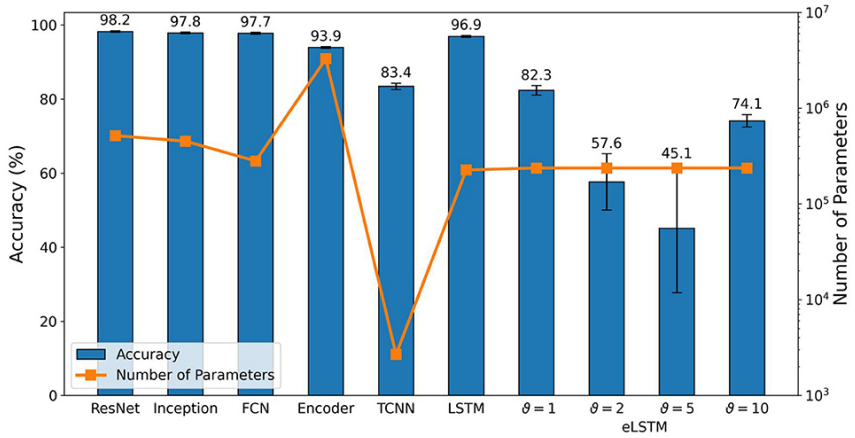

Fig. 6.9 Test accuracy and number of trainable parameters of conventional classifiers after 300 epochs of training and average across three runs. eLSTM is an abbreviation for LSTM with event-based input. [3]

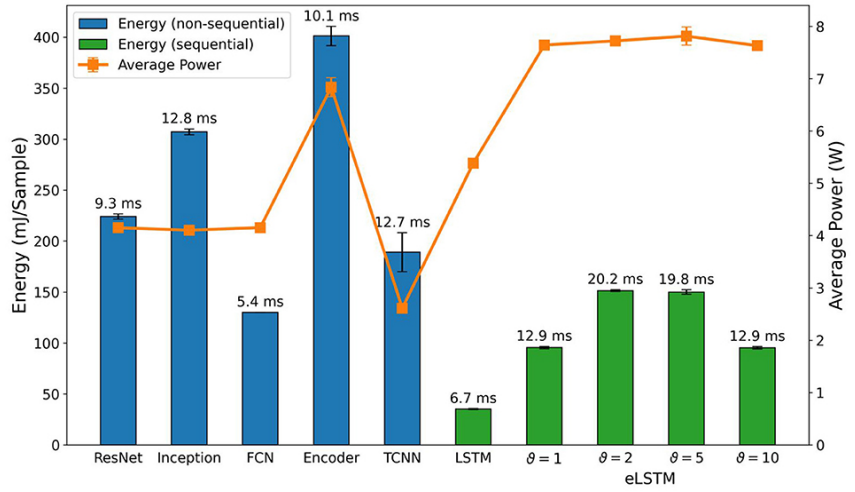

Fig. 6.10 Comparison of inference metrics from common classifiers for frame-based data in terms of energy consumption and average power utilization as measured on an NVIDIA Jetson Xavier NX. eLSTM is an abbreviation for LSTM with event-based input. Each bar's label indicates the inference time per sample on the relevant network. [3]

Figure 6.9 depicts each network's test accuracy as well as their total number of trainable parameters. Figure 6.10 shows the energy consumption, average power utilization, and inference time of standard classifiers running on the NVIDIA Jetson. Some similarities can be seen when comparing these results to the parameter counts in Figure 6.9. ResNet, Inception, and FCN all showed similar average power consumption during inference, and also had equivalent accuracy and parameter

count: this indicates that their energy use is closely related to the amount of time they spend on the inference phase. The energy consumption of Inception and its parameter count surpassed both ResNet and FCN; while the number of parameters does not necessarily indicate higher computational complexity, an explanation for this result could be that Inception's processes are less optimized or do not use GPU acceleration. The similarities continue with Encoder, where average power usage and parameter count both increased by an order of magnitude compared to the previous three networks. Despite that, the difference in energy consumption is not as significant, especially when compared to Inception. Given the higher average power utilization, this implementation probably exploited the GPU more efficiently and hence benefited from higher overall acceleration. This assumption is supported by the fact that inference time was lower on the Encoder than on Inception.

The results for the LSTM with frame-based input are the most notable among these classifiers. Despite being a sequential network, which are usually slower than traditional classifiers due to its iterative and recurrent nature, it obtained the shortest inference time and consumed the least energy; however, this came at the expense of having the second-highest power usage. This, as for the Encoder, implies that the network could benefit from increased GPU use or general acceleration of internal activities. For the LSTM with event-based input (eLSTM), the energy per sample and inference time were proportional to the number of time steps processed, with a ratio of 5 : 3, which corresponds to the time binning performed for each encoding threshold (see Section 3.3): similar *time_bin_sizes* have been used for $\vartheta = 1$ and $\vartheta = 10$ with 5 ms, resulting in 270 time steps, and for $\vartheta = 2$ and $\vartheta = 5$ with 3 ms, resulting in 450 time steps. This ratio, however, does not hold for the frame-based input with 54 time steps. The nature of the data, with float numbers for frame-based data and integer numbers for event stream, could explain this discrepancy. Finally, all eLSTMs consumed about the same amount of power.

When compared to the other networks, the TCNN performed poorly. Despite having by far the fewest parameters among the presented networks, it had a similar energy consumption but a long inference time. In general, it appears that this particular design was not well-suited to addressing the issue at hand.

Spiking neural networks

Measurements on the NVIDIA Jetson are displayed in Figure 6.11. Results for SNNs and RSNNs were proportional to *time_bin_size* and *nb_input_copies*, as well as whether the feedforward or recurrent architecture was involved; these are the main factors determining the number of operations to be computed during inference. In contrast, the threshold had little to no effect on performance, because the implementation on general-purpose computers does not take advantage of the data's temporal sparsity.

The average power utilization of the feedforward and recurrent designs was quite close, varying from their respective norms by less than 0.7 % for the former and less than 0.2 % for the latter. This implies that computational resources were constantly in use, and that energy consumption is directly proportional to inference time for any architecture. Thresholds $\vartheta = 1$ and $\vartheta = 10$ required about the same amount of energy per inference as thresholds $\vartheta = 2$ and $\vartheta = 5$. As seen in Table 5.3, the results for both threshold pairs appear to have a correlation with *time_bin_size* and *nb_input_copies*. However, the *nb_input_copies* for $\vartheta = 2$ and $\vartheta = 5$, which are 8 and 4 respectively, do not follow this trend. In conclusion, *nb_input_copies* doesn't have a significant effect on energy usage and inference time in real-world circumstances, but *time_bin_size* and architecture type are the primary drivers for computational burden.

A comparison of SNNs and eLSTMs reveals a 20× increase in inference time, despite the fact that both execute the same number of time steps, reflecting a lack of algorithmic optimization within the SNN simulations. The energy per sample is 10× higher, while the average power use is 1.5×.

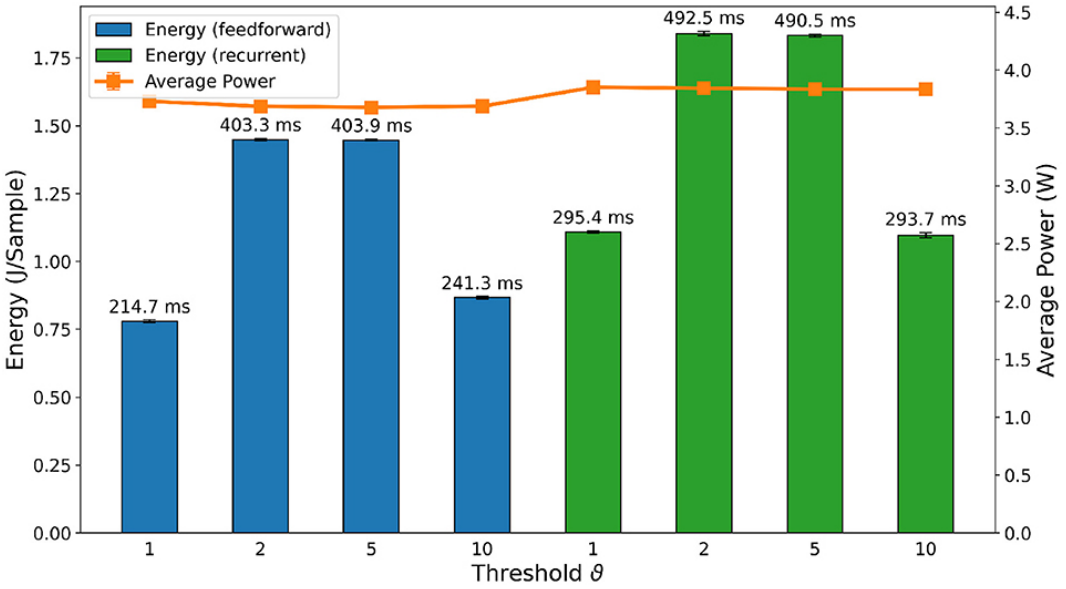

Fig. 6.11 Inference metrics for all spiking neural networks compared in terms of energy usage and average power consumption on an NVIDIA Jetson Xavier NX. Each bar's label indicates the inference time per sample on the relevant network. [3]

When comparing the absolute numbers in Figure 6.11 to the standard classifiers in Figure 6.10, our FFSNN and RSNN implementations have a clear disadvantage in terms of energy consumption and inference time when run on a GPU accelerated device. The most efficient SNN used approximately ~ 88% more energy than the least efficient traditional classifier, and the fastest spiking network needed 16.8× more time to make a single inference than the slowest traditional classifier. These figures demonstrate the critical necessity for dedicated neuromorphic hardware to fully express the potential of event-based algorithms.

6.2.2 Intel Loihi

Loihi [25] is a completely digital neuromorphic processor from Intel. Each Loihi chip contains 128 neuron cores, with each neural core capable of running up to 1024 CUBA LIF neurons through Time-division Multiplexing (TDM). The equations for the current and voltage compartments of the Loihi neuron are shown below.

$$I_i(t) = I_i(t-1) \cdot (2^{12} - \delta_i^I) \cdot 2^{-12} + 2^6 \sum_j w_{ij} \cdot s_j(t) \tag{6.2}$$

and

$$U_i(t) = U_i(t-1) \cdot (2^{12} - \delta_i^U) \cdot 2^{-12} + I_i(t), \tag{6.3}$$

where t is the algorithm's time step, $I_i(t)$ and $U_i(t)$ are the current and voltage of neuron i, δ_i^I and δ_i^U are the current and voltage decay constants, w_{ij} is the synaptic weight from neuron j to i, and $s_j(t)$ is the spike state (0 or 1) of neuron j.

As long as the capacity of the in-core memories for storing axons and synapses is not exceeded, a Loihi neuron core may support any connection topology. These neuron cores are spread in parallel and use local on-chip SRAMs to store network state and configurations; they are totally asynchronous, executing synaptic accumulation only when an input event occurs, which lets the system take advantage of the spatio-temporal sparsity of event-based sensors and encoding. The Loihi system's algorithmic time step is maintained through a distributed handshaking approach known as barrier synchronization. Furthermore, each Loihi chip contains three synchronous integrated x86 cores that participate in barrier synchronization. The x86 cores run C code and are used for monitoring and communicating with the SNN running on the neuron cores, as well as to handle data IO between the on-chip asynchronous neuron cores and off-chip devices and, optionally, to synchronize the algorithmic time steps duration (in physical time) between the chip and an external sensor.

The original networks were described in PyTorch, a well-known Python library for the implementation of ML programs. In order to run the networks on Loihi, the already-trained models must be exported to the HDF5 format, adapting the neuron parameters and learned synaptic weights to match the Loihi hardware specs and restrictions: the Loihi decay constants δ^I and δ^U were calculated from the PyTorch time constants τ_{syn} and τ_{mem}, and since Loihi only supports up to 8-bit fixed point weights for synaptic weights and neuron thresholds, the weights from PyTorch training were quantized into 256 states.

After deploying the network on Loihi and executing the inference on the event-based tactile data, the authors measured the classification accuracy, the energy consumption and the computation time. The test inference was performed by presenting the input events of all test samples in a continuous flow, with samples interspersed with idle windows of 100 time steps during which the neurons' cur-

rents and voltages decayed to zero. The resultant spikes were collected during the inference process, and the classification score was computed offline.

The voltage regulators and power telemetry on Loihi system boards can be used to measure the total power usage of the Loihi chip while a network is running. To estimate energy usage, the power measurements can be paired with timing information captured by the on-chip x86 cores during model execution. When a workload is underway, the dedicated Loihi software stack (NxSDK) presents a high-level user interface to monitor power, energy, and timing: this interface was used to test the performance of the SNN models on Loihi.

Results

The overall accuracy trend in Loihi for SNNs, shown in Figure 6.12, mirrors the accuracy trend in software simulations, shown in Figure 5.2B of the previous chapter. Nonetheless, there is a small reduction in accuracy (e.g., ~ 1.58% for the RSNN with encoding threshold $\vartheta = 1$), which is caused by the PyTorch training approach not accounting for the Loihi hardware constraints, specifically the 8-bit fixed point weights. The loss varies based on the distribution of PyTorch weights.

The hardware performance of the recurrent and feedforward SNNs was then compared in terms of latency (i.e., execution time), power, and energy consumption. Before delving into the results, it's crucial to note that the neural cores mapping has no effect on accuracy, but does have an effect on hardware efficiency. Loihi provides freedom in how network neurons are mapped into neural cores, limited only by the number of cores in a chip and the number of input axons, synapses, neurons, and output axons in a neural core. The goal is to find a trade-off between parallelism (using more neural cores with fewer neurons per core) and time-multiplexing (using fewer neural cores with more neurons per core), in order to balance a neural core's power, mesh routing power, and algorithmic time step duration to achieve an optimal configuration for the application requirements. This dilemma is not dissimilar to the issues encountered in the process of mapping and routing the multi-compartmental neuron models on SpiNNaker in Section 5.3.1. In Loihi's case, spreading a network over more cores increased the power and energy consumption without any noticeable benefit for the computation time. As a result, we used the smallest number of cores feasible for satisfying all hardware constraints, which is 8 cores for all trained networks.

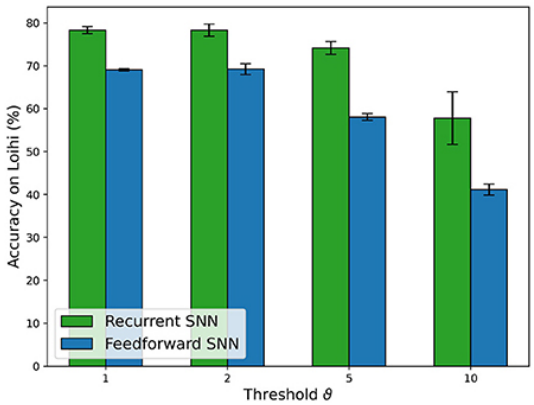

Fig. 6.12 Comparison of the FFSNN and RSNN accuracy results on Loihi, with the best parameters discovered by the two-stage HPO for each encoding threshold. [3]

Figure 6.13 depicts the latency, power, and energy consumption of the deployed SNNs on Loihi. Energy consumption was again chosen as the primary statistic. For the same thresholds, FFSNNs consume less energy than RSNNs, owing to the recurrent synaptic connections' memory and processing overhead. For varying thresholds, FFSNNs and RSNNs follow a similar trend: networks with $\vartheta = 2$ and $\vartheta = 5$ consume more energy, because they have smaller bin sizes and therefore more time bins per sample (450) compared to networks with thresholds $\vartheta = 1$ and $\vartheta = 10$ (270); as seen in Table 5.3, they also require more input copies. Networks with $\vartheta = 2$ consume more than networks with $\vartheta = 5$, owing to the fact that they have more input copies (8 vs. 5). Nonetheless, the FFSNN with $\vartheta = 1$ consumes more than the FFSNN with $\vartheta = 10$, while the RSNN with $\vartheta = 1$ consumes less than the RSNN with $\vartheta = 10$. Even though the networks with $\vartheta = 1$ have more events in the input and fewer events in the hidden layer than the networks with $\vartheta = 10$, the hidden layer events have a different impact, because every event in the FFSNN hidden layers is transmitted to the 28 output neurons, whereas every event in the RSNN hidden layers is transmitted to both the 28 output neurons and the 450 hidden neurons. As a result, the advantage found in the input layer for the RSNN with threshold $\vartheta = 10$ vanishes due to the recurrent architecture increasing the number of synaptic operations. Finally, although the Jetson GPU was largely affected by the number of bins, as shown in Figure 6.11, Loihi is also affected by the number of input copies as well as the spatio-temporal sparsity of the network's spikes and synaptic operations.

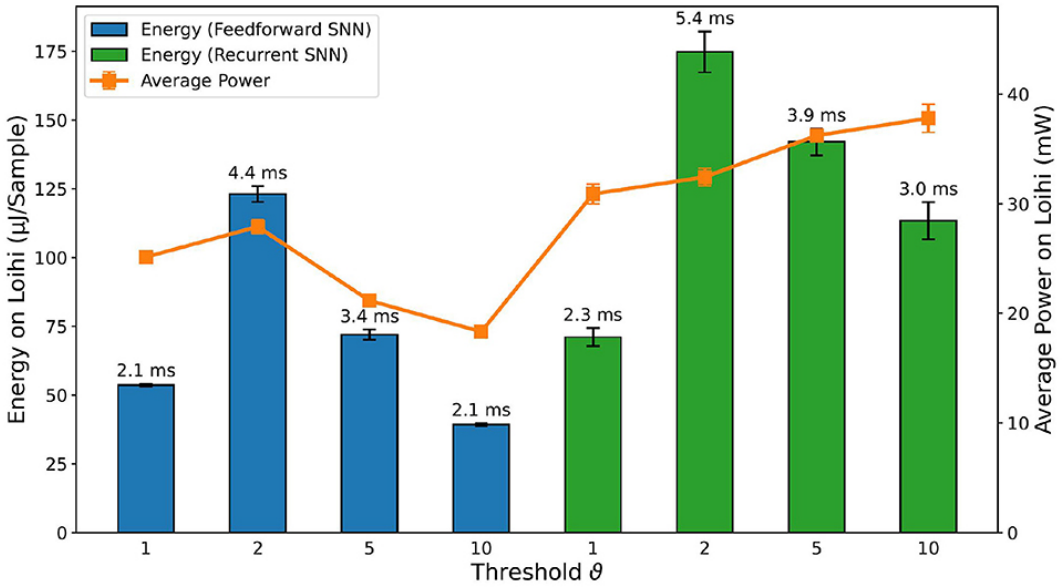

Fig. 6.13 A comparison of inference metrics for all trained spiking neural networks measured on Loihi in terms of energy consumption and average power utilization. Each bar's label indicates the inference time per sample on the relevant network. [3]

After comparing the deployed SNNs on Loihi, the RSNN with encoding threshold $\vartheta = 1$ was the best option, taking into account accuracy, power, energy, and time. For the remainder of this section, we shall refer to it as the RSNN.

Table 6.1 compares the RSNN in Loihi against the RSNN on Jetson, as well as the LSTM and eLSTM on Jetson. The RSNN on Loihi loses 1.58 % accuracy as compared to the RSNN on Jetson, owing to the quantization performed after training. It also underperforms by 17 % when compared to the LSTM on Jetson, but just by 3 % when compared to the eLSTM. An LSTM architecture with the same number of parameters as the RSNN was employed and trained for 300 epochs. However, the RSNN on Loihi achieves hardware efficiency increases of several orders of magnitude. First, it is 124× more power-efficient and 172× quicker than the RSNN on Jetson, making it four orders of magnitude (15615×) more energy-efficient. It is evident that SNNs are unsustainable when implemented on conventional GPU hardware. It should be noted, however, that the RSNN on Jetson still respects the real-time limitation given by the sensor, which has a sampling frequency of 40 Hz (i.e., a maximum algorithmic time step duration of 25 ms). Despite the fact that the average computation time of the RSNN on Jetson is quite long (295.38 ms), it is still less than the overall duration of each sample

Table 6.1 RSNN on Loihi and RSNN, eLSTM, and LSTM on Jetson: accuracy, total power, energy per sample, latency, and energy-delay product summary. [3]

	Results summary				**Comparison with RSNN on Loihi**		
Network	**RSNN**	**RSNN**	**eLSTM**	**eLSTM**	**RSNN**	**eLSTM**	**eLSTM**
Hardware	**Loihi**	**Jetson**	**Jetson**	**Jetson**	**Jetson**	**Jetson**	**Jetson**
Input	**Events**	**Events**	**Events**	**Events**	**Events**	**Events**	**Frames**
Accuracy (%)	78.32	79.90	82.31	96.92	+1.58	+3.99	+18.60
Total power (mW)	31	3851	7642	5385	124×	247×	174×
Total energy per sample (µJ)	71	1 108 695	96 000	35 212	15 615×	1.352×	496×
Delay per sample (ms)	2.3	295.3	12.9	6.7	172×	5.6×	2.9×
Energy-delay product (µJ × s)	0.16	327 398	1238	236	2 046 237×	7738×	1475×

The number of trainable parameter (i.e., synaptic weights) are similar between the RSNN (236 700), the LSTM (225 975), and the eLSTM (236 919). Event-based inputs are encoded with threshold $\theta = 1$. Comparisons with respect to RSNN on Loihi are evaluated as differences for the accuracy and as ratios for all the other quantities.

(1350 ms). It should be noted that in real-life situations, this delay can rise when off-chip communication with the robot is added. Second, as compared to the LSTM on Jetson, the RSNN on Loihi is more than 170× more power-efficient and has a 2.9× shorter average execution time, resulting in a three-order-of-magnitude (1475×) improvement in energy efficiency.

Finally, when compared to the conventional eLSTM classifier on the Jetson GPU with identical event data, the neuromorphic approach using the Loihi chip and RSNNs is approximately 4 % less accurate but two orders of magnitude (247×) more power-efficient, reducing total power from 7642 mW to approximately 31 mW. Furthermore, because the execution time is lowered from 12.9 ms to 2.3 ms, the neuromorphic pipeline obtains a gain in energy efficiency of 1352× and a gain in energy-delay product of 7738×. This is consistent with recent results comparing SNNs on Loihi to standard methods and hardware, where the highest performing workloads on Loihi use highly recurrent networks (Davies et al., 2021). In addition, because the RSNN uses the spatio-temporal sparsity of the event-driven encoding, we should expect an even bigger improvement in energy efficiency when applying it on Loihi in a real-world environment. As a result, if the robot does not move its finger, no event is conveyed to the Loihi chip, significantly reducing the dynamic power, which is approximately 20 mW out of the total 31 mW. On the contrary, the Jetson GPU would continually process redundant data from the quiescient sensor. This research demonstrates how to combine event-driven encoding, neuromorphic hardware, and SNNs to increase

the overall efficiency of tactile pattern recognition, stressing the importance of a neuromorphic approach for embedded systems with a constant input stream.

6.3 Chapter summary

Within this chapter, we have taken a close look at the characteristics of two important representatives of the neuromorphic hardware class: SpiNNaker and Loihi. While both of these platforms can be considered part of the first generation of neuromorphic hardware, there are differences in the design philosophy behind the two designs. SpiNNaker is a fully programmable digital platform, and it was designed with on intense focus on flexibility and scalability. In fact, while a single Spin5 houses 768 processors, the board's connectivity and routing was designed for multi-board operation involving up to a million cores. In Section 6.1, we put the hardware's flexibility and scalability to the test by executing a massively parallel non-spiking algorithm designed to measure the board's performance in applications using a large communication context with the SpinMPI library. While the MPI implementation of PageRank scales better than CPU or SNN-based solutions, we discovered a few bottlenecks affecting SpiNNaker performance on a large communication context: the size of the cores' tightly-coupled data memory (DTCM), RAM access times, and the usage ratio of the MPI buffer size, which is determined by task placement across different SpiNNaker chips. In summary, despite the high quality of the board's communication infrastructure, there are numerous algorithmic difficulties that affect the scalability of applications on a large scale. Most of these issues may be mitigated or resolved with the release of SpiNNaker 2, which is planned to have faster computation and larger storage. Given the interoperability of the communication architecture between the old and new platforms, running these MPI tests on SpiNNaker 2 once it is released is an intriguing avenue for future research.

Loihi is not a fully programmable platform, as its artificial neurons are hard-coded; while this implementation delivers better computation speeds and power consumption with respect to SpiNNaker, it requires that the SNN be designed with the CUBA neuron in mind. In the case of the Braille classification, a few mathematical transformations were necessary to translate the neuron model from the PyTorch LIF to the CUBA representation. When compared to other event-

based classifiers implemented on GPU, an RSNN operating on Loihi provided comparable results, but underperformed by 17% when compared to the best-performing non-spiking classifier (LSTM on GPU). This is not an uncommon result for neuromorphic solutions. However, the focus of this type of technology must be on giving the best possible accuracy in its spectrum of low-power solutions, rather than beating the best accuracy available. While the LSTM offers nearly unparalleled performance for classification of time-varying signals, it also features a very high number of parameters and is computationally expensive in training and inference. By contrast, when considering both the inference time and the power consumption, the RSNN on Loihi offers three-orders-of-magnitude gains in energy efficiency.

The most important result of this comparison is that, while the development of bespoke architectural models that fully exploit the possibilities of spiking computation is not yet complete, neuromorphic solutions should not be overlooked due to the accuracy gap from state-of-the-art ML models; on the contrary, the advantages in energy consumption they offer can already make the difference in applications where an approximate solution is acceptable and low power consumption is required.

Chapter 7

Bringing it all together: towards a complete neuromorphic pipeline

Edge computing is one of the most promising potential applications for neuromorphic technology in the near future. Neuromorphic processors' low power consumption, high parallelism, and real-time computation capabilities would provide efficient elaboration of large amounts of data directly on the edge, eliminating the need to transfer data to power-hungry cloud servers [283, 18]; the adoption of biology-inspired Spiking Neural Networks could additionally provide useful new computing capabilities.

Although neuromorphic hardware has already been accessible for several years, efficient techniques for edge applications have only recently become available: there have been proposals for Constraint Satisfaction Problem solving [284] in addition to real-time data analysis applications like burst event detection via Neuromorphic Auditory Sensors [285], ECG-based heartbeat identification for cardiac defect identification [286], analysis and classification of various biometric signals captured via wearable devices [80], pattern matching [57], hand-gesture detection [78] and learning [287], robotic controllers [288, 289], concurrent mapping and localization [290, 291], adaption of radio-frequency waveforms for noisy situations [292], and on-edge facial recognition [293].

Despite advances in algorithm study, integrating neuromorphic devices in the context of edge computing remains challenging; a neuromorphic system by itself cannot be stationed as an edge device; these kinds of systems have to depend on a

separate host that uploads the network setup and input data before computations can start [273]. This configuration stage is often carried out on a desktop PC or a cloud server. However, certain commercial solutions, such as NeuroEdge [293], which incorporates an NM500 neuromorphic processor into a Raspberry Pi, are beginning to emerge, which may allow developers to avoid the need for additional configuration. Attempts to integrate von Neumann-based computation are also visible in the architecture of neuromorphic designs like that of Loihi [25], featuring microcontroller-class x86 chips at the mesh's periphery, primarily used for data format conversion between the standard computing and neuromorphic encodings. Frameworks for easing the creation and implementation of software on neuromorphic hardware have likewise started to emerge: NeuroXplorer [294] is a tool to support simulation and design exploration for SNN use cases, allowing users to explore performance indicators for an assortment of neural network models and hardware combinations, whereas the Nengo libraries [288] supplement popular Keras and Python utilities to facilitate the construction of SNNs, their compilation on different neuromorphic hardware, and their deployment.

In this chapter, we will see a few examples of partial and complete neuromorphic pipelines realized during the author's research activity. These case studies exemplify the important issues in interfacing neuromorphic sensors, encoders, models, software tools and hardware with each other and with traditional computing-based frameworks. Section 7.1 illustrates on-chip integration between a novel neuromorphic hardware platform and a RISC-V coprocessor, including the development of a simple command and data exchanging interface and a simulation study of the synthesized system on FPGA. This work was previously published as a conference paper [6]. Then, we move on to Section 7.2, where we explore the complete pipeline processes for some neuromorphic classifiers of IoT time-varying signals. In this section, concepts from chapters 2 through 6 are drawn together to create a unified approach for the implementation and benchmarking of this type of application. Subsection 7.2.1 presents a general approach for the design and optimization of a HAR classifier, from the dataset selection to the implementation of a spiking or non-spiking neural architecture. Subsection 7.2.2 drills down on this approach, extending the input selection to multiple dataset types with different characteristics and implementing spike encoding, feature extraction, neural network training through Transfer learning, and model compression. Finally, in Subsection 7.2.3, the workflow further grows to

include implementation of the neuromorphic classifiers on traditional (GPU) and neuromorphic (Loihi) hardware.

7.1 Configuring an embedded neuromorphic coprocessor with RISC-V

Neuromorphic hardware platforms generally cannot be deployed as edge devices on their own, since they require an external host for setup and data input management. This study [6] describes a chip-level integrated system that performs on-edge reconfiguration of a neuromorphic architecture. The proposed method incorporated two open-source platforms: the low-power RISC-V microprocessor Rocket Chip and the digital SNN microprocessor ODIN. The resulting design enables the RISC-V processor to configure a Spiking Neural Network operating on the coupled neuromorphic device in real time over the standard SPI interface. Using the Chipyard framework, we combined the two systems into a single SoC and connected them by building an interface for communication with ODIN's SPI and AER input/outputs, then validated the setup by running an RTL simulation of a *synfire chain* on ODIN, wherein Rocket Chip would configure the network, trigger the first spike, and gather the simulation data. These results represent a proof of concept for endorsing greater integration of neuromorphic systems into the data flow of edge computing.

ODIN: a Spiking Neural Network coprocessor

ODIN (Online-learning DIgital spiking Neuromorphic processor) [218] can be considered a representative example of emergent neuromorphic architectures. This platform, which is freely available as an open source netlist, is a neurosynaptic core that can support up to 256 neurons with all-to-all synaptic linkages. It has I/O interfaces that implement the Address Event Representation (AER) protocol and supports two neuron models: Leaky Integrate & Fire (LIF) and Izhikevich.

The ODIN registers can be configured using the Serial Peripheral Interface (SPI) unit, which also supports write and read operations on neurons and synapses. The system is administered by a controller, which also handles AER requests from outside, and a scheduler, which handles spiking and bursting events from ODIN

neurons or other devices via the AER interface; events are processed in accordance with the First In, First Out policy.

Manually setting up ODIN can be difficult; it is a complicated piece of hardware, and configuration via the standard Serial Peripheral Interface (SPI) protocol can result in a lengthy and error-prone setup phase. To make this operation simpler, we created a C program that allows to easily set ODIN's SPI internal registers and load the neurons' and synapses' SRAM contents. The following functions are available in the current version of the program:

1. *Set SPI Configuration Registers.* Network parameters can be tuned by writing appropriate values into the registers: for example, by setting the synapses of a certain neuron as inhibitory or excitatory.

2. *Add Synapse.* The presynaptic and postsynaptic neuron numbers are obligatory parameters: these are the neurons that will be connected via the synapse that will be created. The user can then configure the mapping table bit and apply a weight to the previously formed synapse.

3. *Add Neuron.* Allows to fine-tune the settings of the new neuron. After providing the address of the neuron to be modified, all its LIF-specific characteristics can be written.

The above functions are utilized on a host computer to generate a binary configuration file. We direct the reader to [295] for more information on the system's implementation.

The RISC-V Instruction Set Architecture

Because it is free and open source, architecture-agnostic, and easily expandable, RISC-V is one of the most extensively used Instruction Set Architectures (ISA) in both academic and industry settings. RISC-V is made up of a fundamental integer ISA that is guaranteed to be stable, as well as a variety of optional modular extensions. The RISC methodology provides higher performance and reduced program sizes, while the vast number of integer registers and Program-Counter relative addressing make complex programs easier to design. Finally, the ISA's modularity enables deep integration of domain-specific coprocessors, allowing for additional specialized instructions within the unused opcode space.

There are several free and open source implementations of the RISC-V ISA, with varying degrees of processing performance, size, and energy efficiency. Choosing one of these solutions relies on the target technology (such as FPGA or ASIC) and the required parameter trade-offs; the Rocket core, which we use in this work, scored well across the board for numerous criteria [296].

7.1.1 ODIN integration with Chipyard

The architecture of this simple system consists of ODIN and Rocket Chip coupled via SPI. The underlying RISC-V system-on-chip is based on Chipyard, a simple-to-use and open source framework with a great degree of flexibility. The Chipyard framework [297] is publicly available on GitHub, as are the ODIN files and documentation [298].

Chipyard supports the integration of external designs written in Verilog, SystemVerilog, or Chisel. The Verilog top-level entity for ODIN can be included in the design as is; a Chisel-specified ODIN black box component wraps over the core Verilog model. Each Chisel black box exposes the following items:

- an I/O field, containing all the ODIN top level entity's input and output signals (the SPI signals, the AER connection signals, Clock, and Reset);
- the constructor arguments (the number of neurons N, the number of bits indicating the neuron handle M, and the *address* that will be associated with ODIN in the SoC's memory map);
- a list of all Verilog resources required to build the design.

ODIN is incorporated as a Memory-Mapped I/O (MMIO) peripheral: AER-specific signals are defined as registers, and their locations are specified in the SoC memory map as offsets with respect to the ODIN module's base address. A TileLink connector is used to communicate with the memory-mapped registers. A *ODINTL* class needed to be created to configure ODIN for use with the TileLink interconnection: this class extends the *TLRegisterRouter* class in Chipyard, passing as arguments ODIN's address in the global memory map, the signals available in TileLink, and the constructor to connect ODIN to the TileLink bus. Because this is a memory-mapped peripheral, connecting the ODIN TileLink-specific node to the MMIO crossbar was sufficient.

Finally, we built a *WithODIN* configuration class to instantiate ODIN with configurable *M* and *N* parameters. The *DigitalTop* class in Chipyard was also changed to make ODIN accessible to the Chipyard framework. The components are then placed in a new RocketChip configuration, and the entire design is assembled.

We put the system to the test by running a basic SNN model: the *synfire chain*, a feed-forward system composed of few neurons in which all synapses are excitatory, so that the membrane potential of the postsynaptic neuron increases whenever any of the presynaptic cells fire. After the first neuron emits a spike, all succeeding neurons are stimulated and fire, resulting in a series of spikes traveling synchronously from one neuron to the following. This network's simple and regular behavior makes it a good reference point to verify the proposed design. Figure 7.2 shows an example of a synfire chain with 8 neurons.

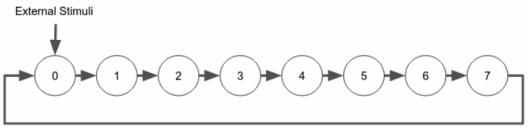

Fig. 7.2 Synfire chain network with 8 neurons. This is the setup used to validate the architecture integrating ODIN and Rocket Chip. © 2021 IEEE.

The correct stimulation and operation of the synfire chain was tested via a Register Transfer Level (RTL) simulation of the system. To set up the SPIFlash controller device, initialize ODIN, and gather results through the output AER interface, we used the custom C program described in Section 7.1.

7.1.2 RTL simulation and synthesis

The synfire chain RTL simulation consists of 8 neurons with addresses ranging from 0 to 7, beginning with a zeroed membrane potential and a threshold voltage equal to 1. The leakage mechanisms were not used. The simulation's configuration and execution consists of a few steps: first, ODIN's SPI input registers are configured. The synaptic interconnections and neuron parameters are then set up by using the newly-established SPI connection to write directly to the synapse and neuron SRAMs. Once all inputs are ready, the host system (in this case, the RISC-V processor) signals the beginning of the simulation, and begins the communication of input and output data for the network through ODIN's AER interface.

Fig. 7.3 Synfire chain with 8 neurons: neuron 0 is stimulated by a virtual synapse event (signals 1-3), then every neuron of the synfire chain fires in sequence (signals 4-7). © 2021 IEEE.

Figure 7.3 depicts the execution of the synfire chain execution as seen by monitoring the AER interface signals. The ODIN controller is triggered through the AERIN_REQ (1) port, and AERIN_ADDR (2) indicates the type of AER event being requested: in this case, a VIRTUAL SYNAPSE EVENT targeting neuron 0, an ODIN event which does not update synapses but simply triggers the start of the simulation. Once a neuron fires, the event is sent over the AER output interface, asserting the AEROUT_REQ (4) and AEROUT_CTRL_BUSY (5) signals. AEROUT_ADDR (6) is filled with the address of the neuron that has just fired and generated the event. AEROUT_ACK (7) is a software-programmed acknowledge which informs ODIN that the latest firing event has been correctly received by the RISC-V CPU. Finally, each neuron fires, and the CPU reads and stores the relevant address.

The entire architecture was synthesised to FPGA using a Xilinx PYNQ Z2 as a feasibility study. To minimize area and reduce the number of I/O pins, unused modules such as the UART connector, Inclusive Cache, auxiliary DRAM, and TileLink probes were removed; the SPI Flash controller was modified as read-only, and the standard core was replaced with the smallest RISC-V RocketChip core available. The results of the synthesis are shown in in Tables 7.1 and 7.2.

The PYNQ Z2 board uses 15.99 % of its LUT slices (14.9 % Logic, 1.09 % Memory) and 11.07 % of its Block RAMs, with the latter implementing ODIN's neuron and synapse states, as well as RocketChip's data and instruction caches. The top module has 8 I/O pins: *clock* (primary clock source), *reset* (global synchronous reset), *SCK* (clock source for the SPIFlash controller and ODIN's SPI port), *CS* (the SPIFlash controller's Chip Select), and 4 Quad-SPI transmission pins. This is the bare minimum of pins required for a workable design; because the PYNQ Z2 has a total of 125 programmable I/O pins, that leaves 117 spare pins for integrating other peripherals or systems.

Table 7.1 ODIN + ROCKETCORE SYNTHESIS - SLICES. [6] © 2021 IEEE.

Site Type	Scope	Type	Used	Available	Utilization %
Slice LUTs			8506	53200	15.99
	Logic		7928	53200	14.90
	Memory		578	53200	1.09
		Distributed RAM	578		
		Shift Registers	0		
Slice Registers			4317	106400	4.06
		Flip-Flop	4317		4.06
		Latch	0		
F7 Muxes			179	26600	0.67
F8 Muxes			34	13300	0.26

Table 7.2 ODIN + ROCKETCORE SYNTHESIS - RAM. [6] © 2021 IEEE.

Site Type	Scope	Type	Used	Available	Utilization %
BRAM Tile			15.5	4140	11.07
	RAMB36/FIFO		15	140	10.71
		RAMB36E1	15		
	RAMB18		1	280	0.36
		RAMB18E1	1		

The proposed method for incorporating a RISC-V Computer and a neuromorphic coprocessor on a single chip makes use of free and open source assets, and allows for SNN application customization on the fly. This work represents a first step toward seamless combination of neuromorphic technologies with cutting-edge processors, creating fully self-contained systems that can handle operations appropriate for conventional computers while also enabling computations which only deep learning algorithms can handle; at the same time, these systems match the low-power constraints required for deployment in the edge computing environment. The addition of a RISC-V CPU capable of configuring the co-embedded neuromorphic device offers users an easier-to-use interface in addition to a well-known, open source, and adaptable ISA. All in all, the combination of these two platforms opens up new possibilities for developing IoT and industrial applications.

7.2 From sensor to neuron: processes for neuromorphic classification of IoT time-varying signals

Spatiotemporal pattern identification is a fundamental brain capacity that is essential for many real-world activities. Recent deep learning algorithms have achieved exceptional accuracy in such tasks, but their execution on traditional embedded systems is still computationally costly and energy-intensive. For example, tactile sensing in robotic applications is a task in which instantaneous processing and energy conservation are essential. In the following, we provide several examples of complete pipelines for the classification of IoT time-varying signals, discussing the problems and prospects of event-based coding, neuromorphic technology, and spike-based computation for spatiotemporal pattern identification at the edge.

7.2.1 A neuromorphic approach for on-edge HAR applications

Because of their event-based asynchronous processes, spiking neural networks (SNNs) [42] can be a strong contender for energy-efficient solutions [299] in the world of on-edge classification of time-variant signals captured by IoT devices. In order to give a comparative examination of various models and architectures for such challenges, in Fra et al. [2] we used the Wireless Sensor Data Mining (WISDM) smartphone and wristwatch activity and biometrics dataset [87, 88] to evaluate a raw data-only classification strategy. Using Nengo as a basic framework demonstrated the benefits of a neuromorphic paradigm as an alternative to traditional deep learning solutions, presenting a first evaluation of bio-inspired models for HAR directly from raw data.

We compared several neural networks, both recurrent and convolutional, spiking and non-spiking, to study this issue; we also applied neuro-inspired techniques to the HAR challenge via novel solutions such as the LMU, highlighting the contrasts between typical DNNs and SNNs from two perspectives: classification performance, and computational effort and memory consumption. This comparison among different solutions is carried out at the last stage of the optimization pipeline illustrated in Figure 7.4. Vertical arrows indicate preliminary steps, specifically dataset selection (a) and optimization experiment design (c)

and (d), while horizontal arrows show subsequent phases in the main timeline of the study: neural network architecture selection (b), hyperparameter optimization (e), and final implementation of the HAR classifiers (f).

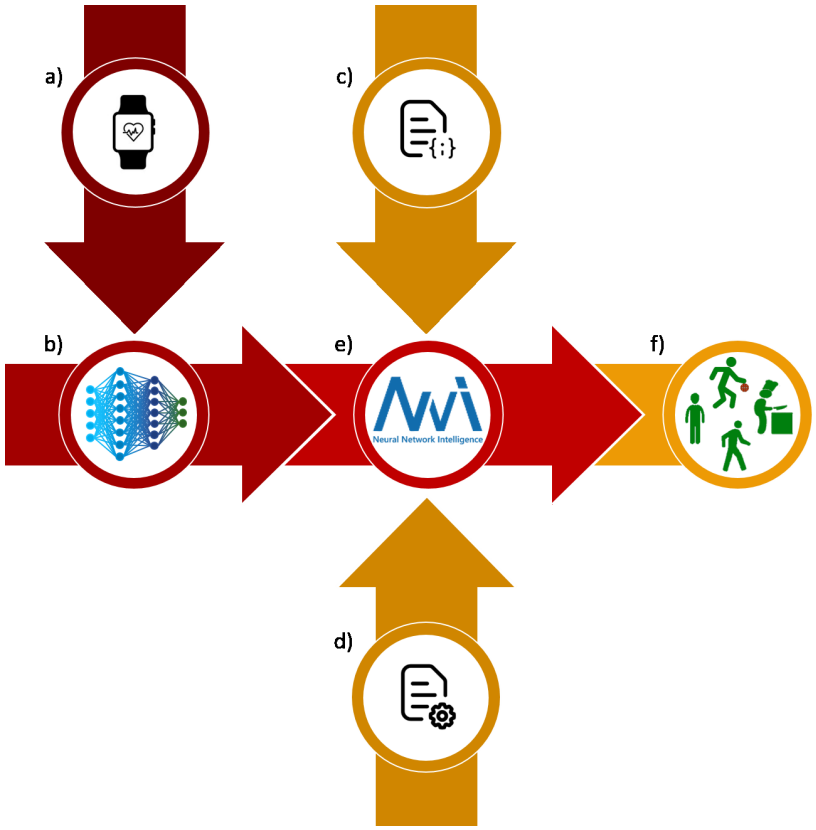

Fig. 7.4 Preliminary processes are shown by the vertical arrows: dataset selection in (a), hyperparameter search space specification in (c), and optimization experiment configuration in (d). The pipeline's main structure, instead, is represented by the horizontal arrows: neural network architecture selection in (b), hyperparameter optimization in (e), and classifier evaluation in (f). [2]

7.2.2 A time-varying signal benchmark for spike encoding techniques

In Forno et al [1], we expanded on the above pipeline in order to benchmark the effect of different encoding techniques at the input. In addition to the WISDM, we applied the same pipeline to audio data from the FSD dataset, in order to evaluate the performance of the entire pipeline with two different types of time-varying signals.

Figure 7.5 depicts the methods undertaken to train the network and classify the data. Starting with raw data, a *filter bank* decomposes the time-varying signal into multiple frequency channels, using a battery of either Butterworth [300] or gammatone [301, 184, 188] parallel band-pass filters [301] to mimic the ability of cochlear hair cells in the human ear to deconstruct audio signals. The original signal is then translated into the spike domain by encoding each individual frequency channel using one of the methods described in Chapter 3. To continue with the training and classification process, a *feature extraction* step is needed, resulting in the *sonogram*, which is a reprocessing of the encoded spike-domain signal in the form of an image. We use the *Time Binning* approach to convert spike signals into frame-based features by counting events over non-overlapping, fixed-length time periods to create the sonogram, as described in Anumula et al. [245].

The goal is to explore the effect of input signal encoding on the training of a spiking convolutional neural network (sCNN) using transfer learning (see Section 4.3.1). For time-varying signals, the use of a convolutional network architecture is common practice [147, 187, 302] because it avoids the use of recurrent neural networks, which are more elaborate and computationally intensive [2]. The sonogram is used as input for the transfer learning approach, which allows us to indirectly train an SNN network using ANN learning techniques. Finally, the sonogram is converted into the spike domain again to validate the accuracy performance. Various CNN/SNN setups are tested to get the highest classification accuracy. The SNN undergoes a *model compression* stage to minimize its dimensions by gradually eliminating synaptic links between neurons depending on their weight (see Section 4.4).

We thoroughly tested several combinations of frequency decomposition filter type, encoding algorithm, feature extraction settings, and network design, using transfer learning and a spiking CNN. The goal of this effort is to provide neuromorphic engineers with useful information on the relative effectiveness of various encoding strategies. Our experiment demonstrated the relevance of customizing the encoding method to the input data by using the same pipeline on two distinct datasets.

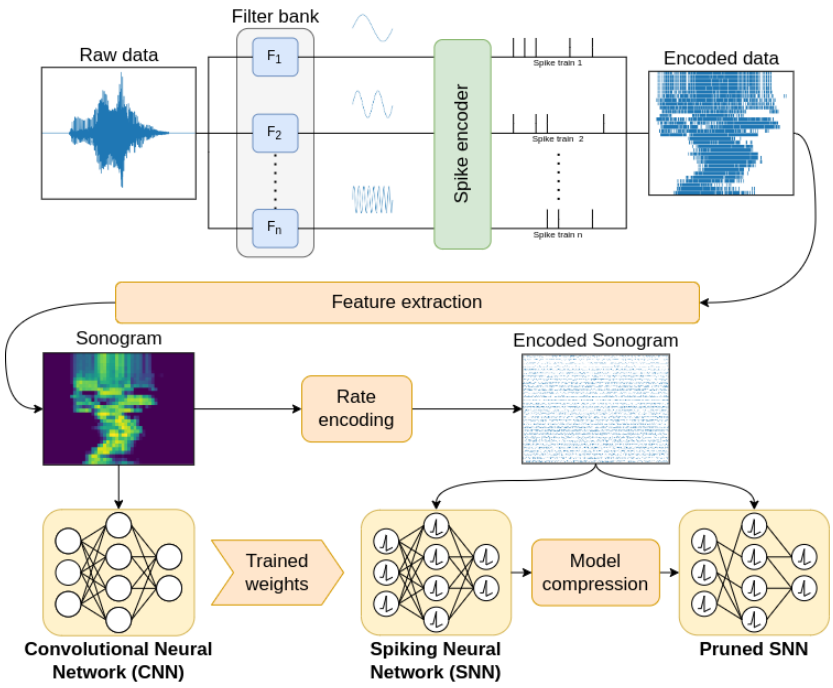

Fig. 7.5 The proposed encoding benchmark pipeline includes a frequency decomposition stage via a filter bank, a spike encoding phase, feature extraction by sonogram creation, transfer learning via a non-spiking network, and model compression. [1]

7.2.3 Braille letter reading benchmark on neuromorphic hardware

For the Braille letter reading task, the authors further extended the pipeline from the previous two sections with proper implementations on neuromorphic hardware.

Figures 7.6.A-D depict the development path for neuromorphic challenges in the tactile space that we proposed in Müller-Cleve et al. [3]. The suggested method was developed and tested on tactile output from capacitive sensors, but it can be applied to a wide range of time-dependent data, including audio streams, motion sensor outputs, and temperature or voltage tracking. The Braille reading task was chosen as a benchmark due to the inherently time-dependent character of its information content, for which we devised a complete event-based neuromorphic classification technique. After collecting an appropriate dataset (see Section 2.2.1), the authors implemented a widely used and well-known encoding technique based on sigma-delta modulation (see Section 3.3), and then performed

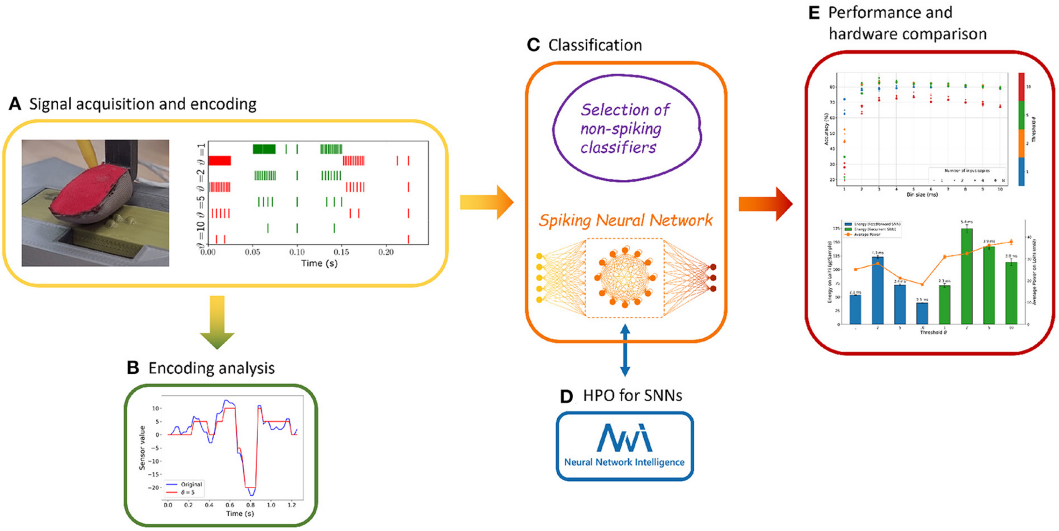

Fig. 7.6 The workflow is broken down into five parts. **(A)** Data acquisition and encoding. **(B)** Information content and reconstruction loss analysis. **(C)** Various non-spiking classifiers are used to generate a benchmark for the proposed RSNN. **(D)** The RSNN is subjected to hyperparameter tuning. **(E)** Performance is analyzed, taking into consideration multiple metrics and hardware solutions. [3]

classification using a neuro-inspired approach employing Feedforward Spiking Neural Network (FFSNN) and RSNN models (see Section 4.2.2), which were implemented both in software (see Section 5.2.2) and hardware (see Section 6.2). NVIDIA Jetson (a modular embedded GPU system) and Intel Loihi (a specialized neuromorphic chip) were the targets of the hardware implementation, entering a comparison with standard classification algorithms with respect to quality of classification, average power consumption, energy use, and computation delay during inference.

Results showed that Braille reading can be carried out in a highly energy-efficient manner by using event-based data and using Spiking Neural Networks (SNNs) on specialized neuromorphic hardware. This assertion is supported by various factors such as the information retention in the encoding, the reliability of the classifiers, and the amount of power needed on different hardware running different models.

Initial examination of the frame-based data via a linear classifier [3] demonstrated that distinguishing information in this type of dataset is stored in both the spatial and temporal domains: in fact, when no time dimension was provided in frame- or event-based data, the accuracy of linear classifiers decreased, demon-

strating the need for architectures capable of learning spatio-temporal patterns from the data. Although linear classifiers provide excellent accuracy using all time bins as predictors, they cannot be used to analyze spatio-temporal patterns when data is being detected in real-time rather than being already accessible; this contrasts with the progressive learning techniques and spike-based algorithms enabled by neuromorphic technology.

Performing a data encoding study illustrated the trade-off between the quantity of information from the initial frame-based data and event stream sparsity. The original frame-based data is intrinsically redundant because the information content drops slower than the event compression ratio, as exposed by comparing the datasets at various threshold levels.

In addition to the effect of the encoding threshold, the investigation revealed time binning to be a major influencing element. The minimum *time_bin_size* represents the smallest boundary within which the temporal evolution of the event stream can be accurately represented in the sparse vector format used on a GPU; when ISIs in the data are shorter than the *time_bin_size*, information content and time dynamics degrade and information is lost. The implementation outcomes show that the network likely failed to capture the sparsity in the source event stream at each layer: although the dataset supplied to the input stage had a compression ratio greater than 1 in terms of events, the optimized network demonstrated an increase in the number of events and the overall energy consumption of the architecture. The quantity of time bins has a significant impact on power consumption, but when comparing thresholds $\vartheta = 1$ with $\vartheta = 10$ and $\vartheta = 2$ with $\vartheta = 5$ for the RSNN, which have the same *time_bin_size* of 5 and 3 ms, respectively, the higher threshold always has a higher power consumption, as shown in Figure 6.13. This can only be explained by an increase in the number of spikes broadcast in the network. As a result, encoding strategies that reduce the number of events in input data can still lead to higher overall energy usage in the system.

The spiking neuron voltage and current time constants, after being independently optimized for each encoding threshold, were all of comparable magnitude: in fact, they appear to be linked to the underlying temporal fluctuations in the input event stream and not to the encoding threshold or binning time window. The HPO did not find a global best for any single encoding threshold, highlighting the

complicated interplay of the included parameters, which resulted in numerous locally optimal solutions. The findings of the optimization grid search demonstrated that any *time_bin_size* bigger than 2 ms led to a similar trend, with a minor decline in classification accuracy as *time_bin_size* increased. The choice of a larger *time_bin_size* for future robotic implementation is preferred, given the moderate drop in accuracy and the improvement in energy and power savings.

The implementation of SNNs on the NVIDIA Jetson indicated that the platform is capable of meeting the requirements of real-time functionality (i.e., any network's inference time being less than the duration of one sample). However, the difference in inference time between non-spiking and spiking networks is significant. It varies between $\sim 16\times$ (Inception vs. FFSNN with threshold $\vartheta = 1$) and $\sim 91\times$ (FCN vs. RSNN with threshold $\vartheta = 2$). As a result, energy consumption increases significantly, ranging from $\sim 2\times$ (Encoder vs. FFSNN with threshold $\vartheta = 1$) to $\sim 40\times$ (LSTM vs. RSNN with threshold $\vartheta = 2$). These figures demonstrate the clear disadvantage of the application on standard hardware: an effective deployment would require either a more optimized design that is better accelerated by GPUs, or dedicated hardware that can make use of spiking domain properties such as temporal sparsity.

The authors were able to demonstrate the feasibility of doing time series classification on neuromorphic hardware utilizing only event-based coding and asynchronous event-driven computation. Using only 450 recurrently interconnected hidden units and utilizing a total of 31 mW on the Intel Loihi neuromorphic processor, the deployed RSNN could discern from 27 classes of Braille letters at 78.32 % accuracy. Of course, this is still insufficient to report competitive classification performance when compared to conventional classifiers or other SNN results on diverse tasks: the choice of sigma-delta encoding in particular led to excessive information loss, preventing the network from further improvement. Nevertheless, compared to an LSTM running on the NVIDIA Jetson embedded GPU, the RSNN with threshold $\vartheta = 2$ on Loihi yielded a power-efficiency boost of $250\times$. These results highlight the challenges of spike-based computing in terms of accuracy when compared to standard algorithms; at the same time, they demonstrate the potential of the neuromorphic approach paired with event-based transmission and asynchronous computation in terms of power/energy efficiency and delay, particularly for mobile robotics or strongly energy-constrained application fields. In these scenarios, efficient event-based encoding is a necessary companion to

neuromorphic computing for executing the task. Event-based devices can be considered quiescent when no substantial input change happens, and their power consumption is exceptionally low during this period. Despite this, thanks to its asynchronous nature, the system will be able to respond to changes immediately.

7.3 Chapter summary

As the ideal endpoint of this thesis, this chapter has reported a few examples of partial and complete neuromorphic pipelines.

First, the author demonstrated interoperability between a neuromorphic processor and a RISC-V CPU on the same chip, enabling seamless data interchange and simulation control for edge computing. Then, we examined how three subsequent research projects have gradually built a neuromorphic pipeline for classification of time-varying signals.

An experiment using raw, low-frequency inertial sensor data from human activity monitors highlighted the energy/accuracy tradeoff in favor of spiking LMU networks, indicating that recurrent networks could provide better results for time-varying data while maintaining the energy advantage of spiking computation.

In a second experiment, we expanded our analysis to middle-frequency audio data, using a variety of preprocessing and encoding techniques. We also introduced model compression, resulting in favorable classification accuracy and energy savings.

Our latest effort involved collaboration with international researchers to accelerate the neural model on neuromorphic hardware. While we found a small gap in accuracy between the RSNN on the Loihi neuromorphic hardware and the Jetson GPU, the energy efficiency of the Loihi version was 1475× that of the best-performing network on Jetson.

While the designs proposed in this chapter constitute workable prototypes for benchmarking the behavior of neuromorphic system designs, they do not yet exemplify complete end-to-end neuromorphic pipelines ready for deployment in IoT and industrial applications. Limitations such as the scarce availability of event-based sensors and lack of resources for native SNN training have informed the growth of our neuromorphic pipeline, with the necessity of gradual adoption

of new tools, techniques and models selected via careful benchmarking and examination of the interaction with the other elements in the process. At the same time, as seen throughout Section 7.2, this gradual and modular method has proven stable and reliable, creating a valuable platform to build upon in support of future work in this area of research.

Chapter 8

Conclusions

The goal of this research was to create a catalog of computational blocks, tools, and frameworks based on neuromorphic technology, as well as to define advantageous pipelines and methodologies for implementing new algorithms that match diverse test cases in the industrial sphere. This thesis focused on developing a working neuromorphic pipeline with special attention to applications involving the categorization of IoT time-varying data in order to provide relevant didactic examples of practical use cases for neuromorphic technology. This type of application represents an ideal use case for the SNNs at the core of neuromorphic computing, because this type of deep learning architecture inherently features a more accurate internal representation of spatio-temporal dynamics compared to other ML solutions. After identifying the building blocks of the neuromorphic system, we have examined each in detail and observed their interactions with one another.

For edge deployment, an embedded neuromorphic application must necessarily interact with its environment via sensors. This purpose can be served by both typical digital sensors (accelerometer, gyroscope) and novel event-based sensors (silicon retina, silicon cochlea). The two types of solution offer different tradeoffs and advantages: event-based sensors represent a promising avenue, and they are likely to see widespread adoption in the near future for specialized tasks because of their extreme power efficiency. On the other hand, as of now, digital sensors are a far more accessible alternative that can be used at a low cost; therefore, it is likely that anyone trying to create a system exploiting neuromorphic platforms to be deployed in the very near future will need to interface with this kind of input.

Whether using digital or event-based sensing, the neuromorphic designer must address the issue of spike encoding and its repercussions on downstream elements, especially as it affects the information content of the obtained signal. While an event-based sensor generally encapsulates encoding hardware that cannot be changed a posteriori, input from a digital sensor must be transformed into a spike train before it can be elaborated by a SNN. As seen in Chapter 3, our thorough investigation of spike-based encoding solutions found a notable correlation between the efficiency of the coding technique and the frequency of the input data: therefore, an important requirement for any spike coding technique is that it produce a sufficiently high spike count to properly stimulate all the cascading layers of the classification network. Because spike sparsity is a determinant of the low power attributes of neuromorphic systems, selecting an encoding strategy requires careful consideration of the tradeoff between information preservation and energy reduction.

As far as time-varying signals are concerned, rate-based coding does not properly represent the fine temporal dynamics of an input signal. Temporal coding exploits the spatio-temporal representation ability of SNNs more effectively, leading to more accurate results. Within the temporal coding class, *deconvolution-based* and *temporal contrast* techniques deliver the best accuracies, with comparable performance. However, deconvolution-based methods require more insight into the frequency characteristics of the input data in order to properly construct the underlying filters, and are therefore generally harder to design. On the contrary, temporal contrast techniques are simpler and more effective at algorithmically capturing the time-varying dynamics of the data with little input from the designer. As a result, we can say that temporal contrast encoding is a suitable first choice for most applications handling input data that has an important time-varying component.

The selection of the operative classifier network for the neuromorphic system is also of paramount importance. Neuromorphic technology allows for very fine modeling of the internal dynamics of a network layer, down to the complex behavior of a single neuron. While neuronal models can be endlessly customized — either taking inspiration from biology or creating less realistic computational units that serve a specific purpose — the most widespread neuron model for practical applications remains the Leaky Integrate & Fire (LIF), because of its very light computational load. The use of the LIF model can be extended by integrating

populations of Adaptive LIF neurons to enable effective learning in recurrent SNNs.

Many of the experiments illustrated in this thesis have made use of the spiking CNN as a classification architecture, because of its ease of implementation and the possibility of transfer learning using well-known and reliable ANN methods. However, as supported by the results found in the comparative classification of the Braille dataset (see Section 5.2.2), the most suitable architecture for time-varying data in the spiking domain is the recurrent neural network. Networks such as the LMU and RSNN natively create a memory trace of past events in their recurrently connected reservoirs, which allow the classifier to correlate time-varying events on a longer scale. As a result, in future work, the author proposes to integrate cutting-edge recurrent networks — such as the LSNN, a spiking variant of the LSTM architecture that enables novel learning techniques — into the classification pipeline. In addition to pure and simple architectural design, refining techniques such as automated hyperparameter optimization and model compression remain as important for neuromorphic engineering as they are in the general ML field.

Research into neuromorphic hardware platforms has continued to flourish in the past few years. Nowadays, the field is on the verge of a minor revolution, as the first generation experimental platforms put forward by the research community (such as SpiNNaker and BrainScaleS, later followed by Loihi and DYNAP-SE) prepares to give way to a second generation that builds on the successes of the previous platforms (presently: SpiNNaker 2, Loihi 2, ReckOn), bringing in technological innovations that will empower more complex computation. For instance, while SpiNNaker 2 is expected to provide a 50× increase in computational ability over its predecessor, mostly due to a higher number of CPUs, a higher clock rate, and the integration of hardware accelerators for synaptic operations, the interconnection infrastructure will be based on the efficient multicast router that sat at the core of SpiNNaker's exceptional software scalability. This routing infrastructure has been carefully examined as part of the scope of this dissertation, demonstrating the scalability of the SpiNNaker architecture for communication-heavy jobs while identifying the primary constraints limiting context growth. Several of these bottlenecks, such as computation time, DTCM size, and SDRAM access time, should improve with the newer and faster technologies promised by the latest hardware iteration. On the other hand, as seen in Section 5.3.1, the communication efficiency of the neuromorphic platform also heavily depends on

decisions taken on the software side, namely in the placement and routing algorithms. As a result, the size and complexity of the new neuromorphic platforms must necessarily grow in tandem with a greater focus on placement and routing issues, encouraging the development of efficient algorithms that account for the relative physical location of data and computing elements, as well as the quirks of the lower hardware and software layers. This study further validates the utility of digital and fully programmable systems like SpiNNaker, which enable continual evolution by allowing users to create new neural models and middleware algorithms even late in the hardware's lifecycle.

Finally, Chapter 7 reported a few examples of partial and complete neuromorphic pipelines. On a hardware synthesis and system software level, we have demonstrated the interoperability of a neuromorphic processor with a RISC-V CPU located on the same chip, allowing for seamless data interchange and simulation control for edge computing. On a more macroscopic level, we have gradually built a neuromorphic pipeline for classification of time-varying signals. A first experiment (Section 7.2.1) used raw, low-frequency inertial sensor data from human activity monitors to benchmark the behavior of several neural network-based classifiers. This study highlighted an energy/accuracy tradeoff in favor of spiking LMU networks, giving credit to the idea that recurrent networks could enable better results for time-varying data while maintaining the energy advantage connected to spiking computation. In a second work (Section 7.2.2), we incorporated a variety of preprocessing and encoding techniques and expanded our analysis to middle-frequency audio data, allowing us to compare the effects of different data elaboration methods. We also added a model compression step which brought favorable results in terms of classification accuracy and energy savings. In our largest effort, thanks to the collaboration of a group of international researchers, we were able to extend the pipeline to include acceleration of the neural model on neuromorphic hardware, finding a small gap in accuracy when comparing the results of the same RSNN on the Loihi neuromorphic hardware and on the Jetson GPU, overshadowed by a three-order-of-magnitude (1475×) improvement in energy efficiency when compared to the best-performing network on Jetson.

Despite our efforts, some gaps remain in the construction of an end-to-end neuromorphic pipeline for time-varying signals. Due to limited availability of event-based sensors, we have mostly worked with digital sensors and software-based converters to spike encoding. For a truly embedded end-to-end neuromor-

phic system to be deployed on the edge, the computational effort of this encoding step must be taken into account if real time operation is to be guaranteed. Also, in all these pipelines, we primarily relied on transfer learning as a training method. While this method is simple and effective enough for small networks with a fixed task, the potential of SNNs for active adaptability and online learning after deployment remains largely unexplored in this work. However, this unique ability may be one of the decisive aspects for the adoption of this type of technology in the industrial field. Finally, again due to limited reach and availability, we were only able to use first-hand the SpiNNaker hardware platform, thanks to the Politecnico di Torino's involvement in the Human Brain Project, while Loihi results were provided by external collaborators. While these platforms remain more than suitable for experimentation, a convincing proof of concept should use a more miniaturized and energy efficient platform, such as ReckOn or the Kapoho Bay incarnation of Loihi. In light of the above, future work will concentrate on completing a totally neuromorphic pipeline for real-time speech recognition. Leveraging the Neuromorphic Auditory Sensor will allow to sidestep any encoding issues, and the target architecture will shift towards recurrent SNNs, finally focusing on a hardware implementation on the most suitable neuromorphic platform.

This research illustrates a few examples of the many tools now available in neuromorphic engineering, and the difficulties and opportunities in the complicated interplay of the necessary elements constituting a neuromorphic pipeline. For many years, despite booming interest in the development of neuromorphic hardware, the accessibility to these machines has lagged behind, due to the lack of standard and powerful APIs that would ensure the interoperability of the many hardware and software tools being researched as well as a unified front-end that would allow developers to try and combine different solutions. However, first with the introduction of the Neural Engineering Framework (NEF) and Nengo, then with the increased industry involvement on the part of Intel with the creation of Loihi and the Intel Neuromorphic Research Community, collaborative efforts among many entities have brought to light many new tools allowing to explore novel neural architectures and applications with increased modularity. As a result, this dissertation represents a first step toward effortless integration of neuromorphic devices into fully embedded applications, able to independently interact with their surroundings and provide sophisticated inference while respecting very

strict power constraints; the use cases offered here offer a foundation for future research to build upon and expand the possibilities of neuromorphic engineering.

Acknowledgements

Part of the research leading to these results has received funding from:

- the European Union Horizon 2020 Programme (H2020/2014-20), grant number 785907 (HBP-SGA2);
- the ECSEL Joint Undertaking (EU H2020) under the Arrowhead Tools research project, with Grant Agreement No. 826452;
- the Ebrains-Italy project, CUP B51E22000150006;
- the Fluently project with Grant Agreement No. 101058680.

References

[1] Evelina Forno, Vittorio Fra, Riccardo Pignari, Enrico Macii, and Gianvito Urgese. Spike encoding techniques for iot time-varying signals benchmarked on a neuromorphic classification task. *Frontiers in Neuroscience*, 16, 2022.

[2] Vittorio Fra, Evelina Forno, Riccardo Pignari, Terrence C Stewart, Enrico Macii, and Gianvito Urgese. Human activity recognition: suitability of a neuromorphic approach for on-edge aiot applications. *Neuromorphic Computing and Engineering*, 2(1):014006, 2022.

[3] Simon F. Müller-Cleve, Vittorio Fra, Lyes Khacef, Alejandro Pequeño-Zurro, Daniel Klepatsch, Evelina Forno, Diego G. Ivanovich, Shavika Rastogi, Gianvito Urgese, Friedemann Zenke, and Chiara Bartolozzi. Braille letter reading: A benchmark for spatio-temporal pattern recognition on neuromorphic hardware. *Frontiers in Neuroscience*, 16, 2022.

[4] Gianvito Urgese, Francesco Barchi, Enrico Macii, and Andrea Acquaviva. Optimizing network traffic for spiking neural network simulations on densely interconnected many-core neuromorphic platforms. *IEEE Transactions on Emerging Topics in Computing*, 6(3):317–329, 2016. © 2016 IEEE.

[5] Evelina Forno, Alessandro Salvato, Enrico Macii, and Gianvito Urgese. Pagerank implemented with the mpi paradigm running on a many-core neuromorphic platform. *Journal of Low Power Electronics and Applications*, 11(2):25, 2021.

[6] Evelina Forno, Andrea Spitale, Enrico Macii, and Gianvito Urgese. Configuring an embedded neuromorphic coprocessor using a risc-v chip for enabling edge computing applications. In *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 328–332. IEEE, 2021. © 2021 IEEE.

[7] Fredrik Dahlqvist, Mark Patel, Alexander Rajko, and Jonathan Shulman. Growing opportunities in the internet of things. *McKinsey & Company*, pages 1–6, 2019.

[8] Sérgio Branco, André G Ferreira, and Jorge Cabral. Machine learning in resource-scarce embedded systems, fpgas, and end-devices: A survey. *Electronics*, 8(11):1289, 2019.

[9] AS Rajesh, MS Prabhuswamy, and Srinivasan Krishnasamy. Smart manufacturing through machine learning: A review, perspective, and future directions to the machining industry. *Journal of Engineering*, 2022, 2022.

[10] Zeki Murat Çınar, Abubakar Abdussalam Nuhu, Qasim Zeeshan, Orhan Korhan, Mohammed Asmael, and Babak Safaei. Machine learning in predictive maintenance towards sustainable smart manufacturing in industry 4.0. *Sustainability*, 12(19):8211, 2020.

[11] Salima Omar, Asri Ngadi, and Hamid H Jebur. Machine learning techniques for anomaly detection: an overview. *International Journal of Computer Applications*, 79(2), 2013.

[12] Lukas Ruff, Jacob R Kauffmann, Robert A Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G Dietterich, and Klaus-Robert Müller. A unifying review of deep and shallow anomaly detection. *Proceedings of the IEEE*, 109(5):756–795, 2021.

[13] Kamal Kant Verma, Brij Mohan Singh, and Amit Dixit. A review of supervised and unsupervised machine learning techniques for suspicious behavior recognition in intelligent surveillance system. *International Journal of Information Technology*, pages 1–14, 2019.

[14] Christophe Loyez, Kevin Carpentier, Ilias Sourikopoulos, and François Danneville. Subthreshold neuromorphic devices for spiking neural networks applied to embedded ai. In *2021 19th IEEE International New Circuits and Systems Conference (NEWCAS)*, pages 1–4. IEEE, 2021.

[15] TDK. Tdk qeexo – qeexo – enabling the new era of machine learning at the edge. https://qeexo.tdk.com/, 2023.

[16] Microsoft Corporation. The embedded learning library - embedded learning library (ell). https://microsoft.github.io/ELL/, 2018.

[17] Jing Zhang and Dacheng Tao. Empowering Things With Intelligence: A Survey of the Progress, Challenges, and Opportunities in Artificial Intelligence of Things. *IEEE Internet of Things Journal*, 8(10):7789–7817, May 2021.

[18] Nikola K Kasabov and Nikola K Kasabov. From von neumann machines to neuromorphic platforms. *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence*, pages 661–677, 2019.

[19] Dennis V Christensen, Regina Dittmann, Bernabe Linares-Barranco, Abu Sebastian, Manuel Le Gallo, Andrea Redaelli, Stefan Slesazeck, Thomas Mikolajick, Sabina Spiga, Stephan Menzel, et al. 2022 roadmap on neuromorphic computing and engineering. *Neuromorphic Computing and Engineering*, 2(2):022501, 2022.

[20] Hassan N Khan, David A Hounshell, and Erica RH Fuchs. Science and research policy at the end of moore's law. *Nature Electronics*, 1(1):14–21, 2018.

[21] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. A survey of neuromorphic computing and neural networks in hardware. *arXiv preprint arXiv:1705.06963*, 2017.

[22] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Spiking neural networks. *International journal of neural systems*, 19(04):295–308, 2009.

[23] Aaron R Young, Mark E Dean, James S Plank, and Garrett S Rose. A review of spiking neuromorphic hardware communication systems. *IEEE Access*, 7:135606–135620, 2019.

[24] Steve B Furber, David R Lester, Luis A Plana, Jim D Garside, Eustace Painkras, Steve Temple, and Andrew D Brown. Overview of the spinnaker system architecture. *IEEE transactions on computers*, 62(12):2454–2467, 2012.

[25] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[26] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE transactions on biomedical circuits and systems*, 12(1):106–122, 2017.

[27] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothée Masquelier, and Anthony Maida. Deep learning in spiking neural networks. *Neural networks*, 111:47–63, 2019.

[28] Oliver Rhodes, Petruţ A Bogdan, Christian Brenninkmeijer, Simon Davidson, Donal Fellows, Andrew Gait, David R Lester, Mantas Mikaitis, Luis A Plana, Andrew GD Rowley, et al. spynnaker: a software package for running pynn simulations on spinnaker. *Frontiers in neuroscience*, 12:816, 2018.

[29] James C Knight, Anton Komissarov, and Thomas Nowotny. Pygenn: a python library for gpu-enhanced neural networks. *Frontiers in Neuroinformatics*, 15:659005, 2021.

[30] Gianvito Urgese, Francesco Barchi, and Enrico Macii. Top-down profiling of application specific many-core neuromorphic platforms. In *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, pages 127–134. IEEE, 2015.

[31] Chit-Kwan Lin, Andreas Wild, Gautham N Chinya, Tsung-Han Lin, Mike Davies, and Hong Wang. Mapping spiking neural networks onto a manycore neuromorphic architecture. *ACM SIGPLAN Notices*, 53(4):78–89, 2018.

[32] Catherine D Schuman, Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, Prasanna Date, and Bill Kay. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, 2(1):10–19, 2022.

[33] Lahiru Laminda Abeysekara and Hamid Abdi. Short paper: neuromorphic chip embedded electronic systems to expand artificial intelligence. In *2019 Second International Conference on Artificial Intelligence for Industries (AI4I)*, pages 119–121. IEEE, 2019.

[34] László Bako. Real-time clustering of datasets with hardware embedded neuromorphic neural networks. In *2009 International Workshop on High Performance Computational Systems Biology*, pages 13–22. IEEE, 2009.

[35] Yongseok Lee and Moonju Park. Power consumption and accuracy in detecting pedestrian images on neuromorphic hardware accelerated embedded systems. In *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*, pages 1–4. IEEE, 2019.

[36] Minseon Kang, Yongseok Lee, and Moonju Park. Energy efficiency of machine learning in embedded systems using neuromorphic hardware. *Electronics*, 9(7):1069, 2020.

[37] Tiffany Hwu, Jacob Isbell, Nicolas Oros, and Jeffrey Krichmar. A self-driving robot using deep convolutional neural networks on neuromorphic hardware. In *2017 international joint conference on neural networks (IJCNN)*, pages 635–641. IEEE, 2017.

[38] Luca Leonardo Bologna, Jérémie Pinoteau, Jesús Garrido, and Angelo Arleo. Active tactile sensing in a neurorobotic braille-reading system. In *2012 4th IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 1925–1930, 2012.

[39] Ken E Friedl, Aaron R Voelker, Angelika Peer, and Chris Eliasmith. Human-inspired neurorobotic system for classifying surface textures by touch. *IEEE Robotics and Automation Letters*, 1(1):516–523, 2016.

[40] Riccardo Massa, Alberto Marchisio, Maurizio Martina, and Muhammad Shafique. An efficient spiking neural network for recognizing gestures with a dvs camera on the loihi neuromorphic processor. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2020.

[41] Chiara Bartolozzi, Francesco Rea, Charles Clercq, Daniel B Fasnacht, Giacomo Indiveri, Michael Hofstätter, and Giorgio Metta. Embedded neuromorphic vision for humanoid robots. In *CVPR 2011 workshops*, pages 129–135. IEEE, 2011.

[42] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[43] Sander M. Bohte. The evidence for neural information processing with precise spike-times: A survey. *Natural Computing*, 3(2):195–206, 2004.

[44] Samanwoy Ghosh-Dastidar and Hojjat Adeli. Spiking Neural Networks. *International Journal of Neural Systems*, 19(04):295–308, August 2009.

[45] Wolfgang Maass. To Spike or Not to Spike: That Is the Question. *Proceedings of the IEEE*, 103(12):2219–2224, December 2015.

[46] Conrad D. James, James B. Aimone, Nadine E. Miner, Craig M. Vineyard, Fredrick H. Rothganger, Kristofor D. Carlson, Samuel A. Mulder, Timothy J. Draelos, Aleksandra Faust, Matthew J. Marinella, John H. Naegle, and Steven J. Plimpton. A historical survey of algorithms and hardware architectures for neural-inspired and neuromorphic computing applications. *Biologically Inspired Cognitive Architectures*, 19:49–64, January 2017.

[47] Adarsha Balaji, Federico Corradi, Anup Das, Sandeep Pande, Siebren Schaafsma, and Francky Catthoor. Power-Accuracy Trade-Offs for Heartbeat Classification on Neural Networks Hardware. *Journal of Low Power Electronics*, 14(4):508–519, December 2018.

[48] Wenzhe Guo, Mohammed E Fouda, Ahmed M Eltawil, and Khaled Nabil Salama. Neural coding in spiking neural networks: A comparative study for robust neuromorphic systems. *Frontiers in Neuroscience*, 15:638474, 2021.

[49] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, November 2019.

[50] Bipin Rajendran, Abu Sebastian, Michael Schmuker, Narayan Srinivasa, and Evangelos Eleftheriou. Low-power neuromorphic hardware for signal processing applications: A review of architectural and system-level design approaches. *IEEE Signal Processing Magazine*, 36(6):97–110, 2019.

[51] Peter Blouw, Xuan Choo, Eric Hunsberger, and Chris Eliasmith. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th annual neuro-inspired computational elements workshop*, pages 1–8, 2019.

[52] Mike Davies, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R Risbud. Advancing neuromorphic computing with loihi: A survey of results and outlook. *Proceedings of the IEEE*, 109(5):911–934, 2021.

[53] Yexin Yan, Terrence C Stewart, Xuan Choo, Bernhard Vogginger, Johannes Partzsch, Sebastian Höppner, Florian Kelber, Chris Eliasmith, Steve Furber, and Christian Mayr. Comparing Loihi with a SpiNNaker 2 prototype on low-latency keyword spotting and adaptive robotic control. *Neuromorphic Computing and Engineering*, 1(1):014002, September 2021.

[54] Kyle Buettner and Alan D. George. Heartbeat Classification with Spiking Neural Networks on the Loihi Neuromorphic Processor. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 138–143. IEEE, July 2021.

[55] Mostafa Rahimi Azghadi, Corey Lammie, Jason K. Eshraghian, Melika Payvand, Elisa Donati, Bernabe Linares-Barranco, and Giacomo Indiveri. Hardware Implementation of Deep Network Accelerators Towards Healthcare and Biomedical Applications. *IEEE Transactions on Biomedical Circuits and Systems*, 14(6):1138–1159, December 2020.

[56] Peter Blouw and Chris Eliasmith. Event-driven signal processing with neuromorphic computing systems. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8534–8538. IEEE, 2020.

[57] Gianvito Urgese, Francesco Barchi, Emanuele Parisi, Evelina Forno, Andrea Acquaviva, and Enrico Macii. Benchmarking a many-core neuromorphic platform with an mpi-based dna sequence matching algorithm. *Electronics*, 8(11):1342, 2019.

[58] Seoyeon Kim, Jisu Park, Jaehyeok Jeong, Young-Sun Yun, Seongbae Eun, and Jinman Jung. Survey of IoT platforms supporting artificial intelligence. In *Proceedings of the Conference on Research in Adaptive and Convergent Systems*, pages 65–66, New York, NY, USA, September 2019. ACM.

[59] Hongyu An, Dong Sam Ha, and Yang Cindy Yi. Powering next-generation industry 4.0 by a self-learning and low-power neuromorphic system. In *Proceedings of the 7th ACM International Conference on Nanoscale Computing and Communication*, pages 1–6, New York, NY, USA, September 2020. ACM.

[60] Zhuoqing Chang, Shubo Liu, Xingxing Xiong, Zhaohui Cai, and Guoqing Tu. A Survey of Recent Advances in Edge-Computing-Powered Artificial Intelligence of Things. *IEEE Internet of Things Journal*, 8(18):13849–13875, September 2021.

[61] Jan Stuijt, Manolis Sifalakis, Amirreza Yousefzadeh, and Federico Corradi. μbrain: An event-driven and fully synthesizable architecture for spiking neural networks. *Frontiers in neuroscience*, page 538, 2021.

[62] Wei Yu, Fan Liang, Xiaofei He, William Grant Hatcher, Chao Lu, Jie Lin, and Xinyu Yang. A survey on the edge computing for the internet of things. *IEEE access*, 6:6900–6919, 2017.

[63] Jie Lin, Wei Yu, and Xinyu Yang. Towards multistep electricity prices in smart grid electricity markets. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):286–302, 2015.

[64] Qingyu Yang, Dou An, Rui Min, Wei Yu, Xinyu Yang, and Wei Zhao. On optimal pmu placement-based defense against data integrity attacks in smart grid. *IEEE Transactions on Information Forensics and Security*, 12(7):1735–1750, 2017.

[65] Neeraj Kumar, Sherali Zeadally, and Joel JPC Rodrigues. Vehicular delay-tolerant networks for smart grid data management using mobile edge computing. *IEEE Communications Magazine*, 54(10):60–66, 2016.

[66] Guobin Xu, Wei Yu, David Griffith, Nada Golmie, and Paul Moulema. Toward integrating distributed energy resources and storage devices in smart grid. *IEEE internet of things journal*, 4(1):192–204, 2016.

[67] Han Li, Hicham Johra, Flavia de Andrade Pereira, Tianzhen Hong, Jérôme Le Dréau, Anthony Maturo, Mingjun Wei, Yapan Liu, Ali Saberi-Derakhtenjani, Zoltan Nagy, et al. Data-driven key performance indicators and datasets for building energy flexibility: A review and perspectives. *Applied Energy*, 343:121217, 2023.

[68] Jie Lin, Wei Yu, Xinyu Yang, Qingyu Yang, Xinwen Fu, and Wei Zhao. A real-time en-route route guidance decision scheme for transportation-based cyberphysical systems. *IEEE Transactions on Vehicular Technology*, 66(3):2551–2566, 2016.

[69] Ilias Kalamaras, Alexandros Zamichos, Athanasios Salamanis, Anastasios Drosou, Dionysios D Kehagias, Georgios Margaritis, Stavros Papadopoulos, and Dimitrios Tzovaras. An interactive visual analytics platform for smart intelligent transportation systems management. *IEEE Transactions on Intelligent Transportation Systems*, 19(2):487–496, 2017.

[70] Hsin-Te Wu and Gwo-Jiun Horng. Establishing an intelligent transportation system with a network security mechanism in an internet of vehicle environment. *Ieee Access*, 5:19239–19247, 2017.

[71] Kristian Micko, Peter Papcun, and Iveta Zolotova. Review of iot sensor systems used for monitoring the road infrastructure. *Sensors*, 23(9):4469, 2023.

[72] Shivam Mishra, Ghada A Khouqeer, B Aamna, Abdullah Alodhayb, S Jafar Ali Ibrahim, Manish Hooda, and Gaurav Jayaswal. A review: Recent advancements in sensor technology for non-invasive neonatal health monitoring. *Biosensors and Bioelectronics: X*, page 100332, 2023.

[73] Kristina Zovko, Ljiljana Šerić, Toni Perković, Hrvoje Belani, and Petar Šolić. Iot and health monitoring wearable devices as enabling technologies for sustainable enhancement of life quality in smart environments. *Journal of Cleaner Production*, page 137506, 2023.

[74] Alfredo J Perez, Farhan Siddiqui, Sherali Zeadally, and Derek Lane. A review of iot systems to enable independence for the elderly and disabled individuals. *Internet of Things*, page 100653, 2022.

[75] Sina Dami and Mahtab Yahaghizadeh. Predicting cardiovascular events with deep learning approach in the context of the internet of things. *Neural Computing and Applications*, 33:7979–7996, 2021.

[76] Nicole A Capela, Edward D Lemaire, and Natalie Baddour. Feature selection for wearable smartphone-based human activity recognition with able bodied, elderly, and stroke patients. *PloS one*, 10(4):e0124414, 2015.

[77] Hoda Allahbakhshi, Lindsey Conrow, Babak Naimi, and Robert Weibel. Using accelerometer and gps data for real-life physical activity type detection. *Sensors*, 20(3):588, 2020.

[78] Enea Ceolini, Charlotte Frenkel, Sumit Bam Shrestha, Gemma Taverni, Lyes Khacef, Melika Payvand, and Elisa Donati. Hand-gesture recognition based on emg and event-based camera sensor fusion: A benchmark in neuromorphic computing. *Frontiers in Neuroscience*, page 637, 2020.

[79] Evelina Forno, Simone Moio, Michael Schenatti, Enrico Macii, and Gianvito Urgese. Techniques for improving localization applications running on low-cost iot devices. In *2020 AEIT International Conference of Electrical and Electronic Technologies for Automotive (AEIT AUTOMOTIVE)*, pages 1–6. IEEE, 2020.

[80] Erika Covi, Elisa Donati, Xiangpeng Liang, David Kappel, Hadi Heidari, Melika Payvand, and Wei Wang. Adaptive extreme edge computing for wearable devices. *Frontiers in Neuroscience*, 15:611300, 2021.

[81] Md Abdullah Al Hafiz Khan, David Welsh, and Nirmalya Roy. Firearm detection using wrist worn tri-axis accelerometer signals. In *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 221–226. IEEE, 2018.

[82] Gil Boudet, Pierre Chausse, David Thivel, Sylvie Rousset, Martial Mermillod, Julien S Baker, Lenise M Parreira, Yolande Esquirol, Martine Duclos, and Frédéric Dutheil. How to measure sedentary behavior at work? *Frontiers in Public Health*, 7:167, 2019.

[83] Sandeep Kumar, Monika Nehra, Sakina Khurana, Neeraj Dilbaghi, Vanish Kumar, Ajeet Kaushik, and Ki-Hyun Kim. Aspects of point-of-care diagnostics for personalized health wellness. *International journal of nanomedicine*, 16:383, 2021.

[84] Guo Jia, Guiyi Zhang, Xin Yuan, Xiaosong Gu, Heshan Liu, Zhijun Fan, and Lingguo Bu. A synthetical development approach for rehabilitation assistive smart product–service systems: A case study. *Advanced Engineering Informatics*, 48:101310, 2021.

[85] Liam Dawson and Alex Akinbi. Challenges and opportunities for wearable iot forensics: Tomtom spark 3 as a case study. *Forensic Science International: Reports*, 3:100198, 2021.

[86] Zohar Jackson, César Souza, Jason Flaks, Yuxin Pan, Hereman Nicolas, and Adhish Thite. Jakobovski/free-spoken-digit-dataset: v1. 0.8, 2018.

[87] Gary M Weiss. Wisdm smartphone and smartwatch activity and biometrics dataset. *UCI Machine Learning Repository: WISDM Smartphone and Smartwatch Activity and Biometrics Dataset Data Set*, 7:133190–133202, 2019.

[88] Gary M. Weiss, Kenichi Yoneda, and Thaier Hayajneh. Smartphone and Smartwatch-Based Biometrics Using Activities of Daily Living. *IEEE Access*, 7:133190–133202, 2019.

[89] Raffaele Gravina, Parastoo Alinia, Hassan Ghasemzadeh, and Giancarlo Fortino. Multi-sensor fusion in body sensor networks: State-of-the-art and research challenges. *Information Fusion*, 35:68–80, 2017.

[90] Yun-Soung Kim, Musa Mahmood, Yongkuk Lee, Nam Kyun Kim, Shinjae Kwon, Robert Herbert, Donghyun Kim, Hee Cheol Cho, and Woon-Hong Yeo. All-in-one, wireless, stretchable hybrid electronics for smart, connected, and ambulatory physiological monitoring. *Advanced Science*, 6(17):1900939, 2019.

[91] Jean-Francois Daneault, Gloria Vergara-Diaz, Federico Parisi, Chen Admati, Christina Alfonso, Matilde Bertoli, Edoardo Bonizzoni, Gabriela Ferreira Carvalho, Gianluca Costante, Eric Eduardo Fabara, et al. Accelerometer data collected with a minimum set of wearable sensors from subjects with parkinson's disease. *Scientific Data*, 8(1):48, 2021.

[92] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. Activity recognition using cell phone accelerometers. *ACM SIGKDD Explorations Newsletter*, 12(2):74–82, March 2011.

[93] Federico Cruciani, Chen Sun, Shuai Zhang, Chris Nugent, Chunping Li, Shaoxu Song, Cheng Cheng, Ian Cleland, and Paul Mccullagh. A public domain dataset for human activity recognition in free-living conditions. In *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 166–171. IEEE, 2019.

[94] Jorge-L Reyes-Ortiz, Luca Oneto, Albert Samà, Xavier Parra, and Davide Anguita. Transition-aware human activity recognition using smartphones. *Neurocomputing*, 171:754–767, 2016.

[95] Oresti Banos, Rafael Garcia, Juan A Holgado-Terriza, Miguel Damas, Hector Pomares, Ignacio Rojas, Alejandro Saez, and Claudia Villalonga. mhealthdroid: a novel framework for agile development of mobile health applications. In *Ambient Assisted Living and Daily Activities: 6th International Work-Conference, IWAAL 2014, Belfast, UK, December 2-5, 2014. Proceedings 6*, pages 91–98. Springer, 2014.

[96] Oresti Banos, Claudia Villalonga, Rafael Garcia, Alejandro Saez, Miguel Damas, Juan A Holgado-Terriza, Sungyong Lee, Hector Pomares, and Ignacio Rojas. Design, implementation and validation of a novel open framework for agile development of mobile health applications. *Biomedical engineering online*, 14(2):1–20, 2015.

[97] Daniel Roggen, Alberto Calatroni, Mirco Rossi, Thomas Holleczek, Kilian Förster, Gerhard Tröster, Paul Lukowicz, David Bannach, Gerald Pirkl, Alois Ferscha, et al. Collecting complex activity datasets in highly rich networked sensor environments. In *2010 Seventh international conference on networked sensing systems (INSS)*, pages 233–240. IEEE, 2010.

[98] Ricardo Chavarriaga, Hesam Sagha, Alberto Calatroni, Sundara Tejaswi Digumarti, Gerhard Tröster, José del R Millán, and Daniel Roggen. The opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recognition Letters*, 34(15):2033–2042, 2013.

[99] Attila Reiss and Didier Stricker. Introducing a new benchmarked dataset for activity monitoring. In *2012 16th international symposium on wearable computers*, pages 108–109. IEEE, 2012.

[100] Attila Reiss and Didier Stricker. Creating and benchmarking a new dataset for physical activity monitoring. In *Proceedings of the 5th International Conference on PErvasive Technologies Related to Assistive Environments*, pages 1–8, 2012.

[101] Wallace Ugulino, Débora Cardador, Katia Vega, Eduardo Velloso, Ruy Milidiú, and Hugo Fuks. Wearable computing: Accelerometers' data classification of body postures and movements. In *Advances in Artificial Intelligence-SBIA 2012: 21th Brazilian Symposium on Artificial Intelligence, Curitiba, Brazil, October 20-25, 2012. Proceedings*, pages 52–61. Springer, 2012.

[102] Mi Zhang and Alexander A Sawchuk. Usc-had: A daily activity dataset for ubiquitous activity recognition using wearable sensors. In *Proceedings of the 2012 ACM conference on ubiquitous computing*, pages 1036–1043, 2012.

[103] Chen Chen, Roozbeh Jafari, and Nasser Kehtarnavaz. Utd-mhad: A multimodal dataset for human action recognition utilizing a depth camera and a wearable inertial sensor. In *2015 IEEE International conference on image processing (ICIP)*, pages 168–172. IEEE, 2015.

[104] Barbara Bruno, Fulvio Mastrogiovanni, and Antonio Sgorbissa. A public domain dataset for adl recognition using wrist-placed accelerometers. In *the 23rd IEEE International Symposium on Robot and Human Interactive Communication*, pages 738–743. IEEE, 2014.

[105] Sakorn Mekruksavanich and Anuchit Jitpattanakul. Deep convolutional neural network with rnns for complex activity recognition using wrist-worn wearable sensor data. *Electronics*, 10(14):1685, 2021.

[106] Konstantinos Peppas, Apostolos C Tsolakis, Stelios Krinidis, and Dimitrios Tzovaras. Real-time physical activity recognition on smart mobile devices using convolutional neural networks. *Applied Sciences*, 10(23):8482, 2020.

[107] Francisco Javier Ordóñez and Daniel Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115, 2016.

[108] Shaohua Wan, Lianyong Qi, Xiaolong Xu, Chao Tong, and Zonghua Gu. Deep learning models for real-time human activity recognition with smartphones. *Mobile Networks and Applications*, 25:743–755, 2020.

[109] Kun Xia, Jianguang Huang, and Hanyu Wang. Lstm-cnn architecture for human activity recognition. *IEEE Access*, 8:56855–56866, 2020.

[110] Isibor Kennedy Ihianle, Augustine O Nwajana, Solomon Henry Ebenuwa, Richard I Otuka, Kayode Owa, and Mobolaji O Orisatoki. A deep learning approach for human activities recognition from multimodal sensing devices. *IEEE Access*, 8:179028–179038, 2020.

[111] Sakorn Mekruksavanich and Anuchit Jitpattanakul. Smartwatch-based human activity recognition using hybrid lstm network. In *2020 IEEE SENSORS*, pages 1–4. IEEE, 2020.

[112] Sakorn Mekruksavanich, Anuchit Jitpattanakul, Phichai Youplao, and Preecha Yupapin. Enhanced hand-oriented activity recognition based on smartwatch sensor data using lstms. *Symmetry*, 12(9):1570, 2020.

[113] Bolu Oluwalade, Sunil Neela, Judy Wawira, Tobiloba Adejumo, and Saptarshi Purkayastha. Human activity recognition using deep learning models on smartphones and smartwatches sensor data. *arXiv preprint arXiv:2103.03836*, 2021.

[114] Ankita Bose and BK Tripathy. Deep learning for audio signal classification. In *Deep learning research and applications*. Walter de Gruyter GmbH & Co KG, 2020.

[115] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

[116] Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 1990.

[117] Shih-Chii Liu, Andre van Schaik, Bradley A Minch, and Tobi Delbruck. Asynchronous binaural spatial audition sensor with $2 \times 64 \times 4$ channel output. *IEEE transactions on biomedical circuits and systems*, 8(4):453–464, 2013.

[118] Sören Becker, Marcel Ackermann, Sebastian Lapuschkin, Klaus-Robert Müller, and Wojciech Samek. Interpreting and explaining deep neural networks for classification of audio signals. *CoRR*, abs/1807.03418, 2018.

[119] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[120] Lyes Khacef, Laurent Rodriguez, and Benoit Miramond. Written and spoken digits database for multimodal learning, 2019.

[121] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(7):2744–2757, 2020.

[122] Amirhossein Rostami, Bernhard Vogginger, Yexin Yan, and Christian G Mayr. E-prop on spinnaker 2: Exploring online learning in spiking rnns on neuromorphic hardware. *Frontiers in Neuroscience*, 16, 2022.

[123] Dylan G Peterson, Thoshara Nawarathne, and Henry Leung. Modulating stdp with back-propagated error signals to train snns for audio classification. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2022.

[124] Lingli Cheng, Lili Gao, Xumeng Zhang, Zuheng Wu, Jiaxue Zhu, Zhaoan Yu, Yue Yang, Yanting Ding, Chao Li, Fangduo Zhu, et al. A bioinspired configurable cochlea based on memristors. *Frontiers in Neuroscience*, 16, 2022.

[125] Sam Lilak, Walt Woods, Kelsey Scharnhorst, Christopher Dunham, Christof Teuscher, Adam Z Stieg, and James K Gimzewski. Spoken digit classification by in-materio reservoir computing with neuromorphic atomic switch networks. *Frontiers in Nanotechnology*, 3:675792, 2021.

[126] Dylan George Peterson. A biologically inspired supervised learning rule for audio classification with spiking neural networks. Master's thesis, Schulich School of Engineering, 2021.

[127] Shih-Chii Liu, Bodo Rueckauer, Enea Ceolini, Adrian Huber, and Tobi Delbruck. Event-driven sensing for efficient perception: Vision and audition algorithms. *IEEE Signal Processing Magazine*, 36(6):29–37, 2019.

[128] Chiara Bartolozzi, Lorenzo Natale, Francesco Nori, and Giorgio Metta. Robots with a sense of touch. *Nature materials*, 15(9):921–925, 2016.

[129] Chiara Bartolozzi, Paolo Motto Ros, Francesco Diotalevi, Nawid Jamali, Lorenzo Natale, Marco Crepaldi, and Danilo Demarchi. Event-driven encoding of off-the-shelf tactile sensors for compression and latency optimisation for robotic skin. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 166–173. IEEE, 2017.

[130] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128×128 120 dB 15 µs latency asynchronous temporal contrast vision sensor. *IEEE journal of solid-state circuits*, 43(2):566–576, 2008.

[131] Jorg Conradt, Raphael Berner, Matthew Cook, and Tobi Delbruck. An embedded aer dynamic vision sensor for low-latency pole balancing. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 780–785. IEEE, 2009.

[132] Vincent Chan, Shih-Chii Liu, and Andr van Schaik. Aer ear: A matched silicon cochlea pair with address event representation interface. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(1):48–59, 2007.

[133] R Gary Leonard and George Doddington. Tidigits ldc93s10. Web Download. Philadelphia: Linguistic Data Consortium, 1993. https://catalog.ldc.upenn.edu/LDC93S10.

[134] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7243–7252, 2017.

[135] Udaya Bhaskar Rongala, Alberto Mazzoni, and Calogero Maria Oddo. Neuromorphic artificial touch for categorization of naturalistic textures. *IEEE transactions on neural networks and learning systems*, 28(4):819–829, 2015.

[136] Hian Hian See, Brian Lim, Si Li, Haicheng Yao, Wen Cheng, Harold Soh, and Benjamin CK Tee. St-mnist–the spiking tactile mnist neuromorphic dataset. *arXiv preprint arXiv:2005.04319*, 2020.

[137] Jérémie Pinoteau, Luca Leonardo Bologna, Jesús Alberto Garrido, and Angelo Arleo. A closed-loop neurorobotic system for investigating braille-reading finger kinematics. In *International Conference on Human Haptic Sensing and Touch Enabled Computer Applications*, pages 407–418. Springer, 2012.

[138] Force dimension - haptic devices. https://www.forcedimension.com/products, 2022.

[139] Nawid Jamali, Marco Maggiali, Francesco Giovannini, Giorgio Metta, and Lorenzo Natale. A new design of a fingertip for the icub hand. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2705–2710. IEEE, 2015.

[140] Anthony F Jahn and Joseph Santos-Sacchi. Physiology of the ear. *Ear and Hearing*, 11(3):243, 1990.

[141] Richard F Lyon and Carver Mead. An analog electronic cochlea. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1119–1134, 1988.

[142] Angel Jiménez-Fernández, Elena Cerezuela-Escudero, Lourdes Miró-Amarante, Manuel Jesus Domínguez-Morales, Francisco de Asís Gómez-Rodríguez, Alejandro Linares-Barranco, and Gabriel Jiménez-Moreno. A binaural neuromorphic auditory sensor for fpga: a spike signal processing approach. *IEEE transactions on neural networks and learning systems*, 28(4):804–818, 2016.

[143] Manuel Domínguez-Morales, Angel Jimenez-Fernandez, Elena Cerezuela-Escudero, Rafael Paz-Vicente, Alejandro Linares-Barranco, and Gabriel Jimenez. On the designing of spikes band-pass filters for fpga. In *Artificial Neural Networks and Machine Learning–ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part II 21*, pages 389–396. Springer, 2011.

[144] Daniel Gutierrez-Galan, Juan Pedro Domínguez-Morales, Angel Jimenez-Fernandez, Alejandro Linares-Barranco, and Gabriel Jimenez-Moreno. Opennas: Open source neuromorphic auditory sensor hdl code generator for fpga implementations. *Neurocomputing*, 436:35–38, 2021.

[145] Juan P Domínguez-Morales, Angel F Jimenez-Fernandez, Manuel J Domínguez-Morales, and Gabriel Jimenez-Moreno. Deep neural networks for the recognition and classification of heart murmurs using neuromorphic auditory sensors. *IEEE transactions on biomedical circuits and systems*, 12(1):24–34, 2017.

[146] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, 101(23):e215–e220, 2000.

[147] Juan P Domínguez-Morales, Qian Liu, Robert James, Daniel Gutierrez-Galan, Angel Jimenez-Fernandez, Simon Davidson, and Steve Furber. Deep spiking neural network model for time-variant signals classification: a real-time speech recognition approach. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[148] Misha A Mahowald. Silicon retina with adaptive photoreceptors. In *Visual information processing: from neurons to chips*, volume 1473, pages 52–58. SPIE, 1991.

[149] Fuyou Liao, Feichi Zhou, and Yang Chai. Neuromorphic vision sensors: Principle, progress and perspectives. *Journal of Semiconductors*, 42(1):013105, 2021.

[150] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. A 128 × 128 1.5% contrast sensitivity 0.9% fpn 3μs latency 4 mw asynchronous frame-free dynamic vision sensor using transimpedance preamplifiers. *IEEE Journal of Solid-State Circuits*, 48(3):827–838, 2013.

[151] Yuhuang Hu, Hongjie Liu, Michael Pfeiffer, and Tobi Delbruck. Dvs benchmark datasets for object tracking, action recognition, and object recognition. *Frontiers in neuroscience*, 10:405, 2016.

[152] Hongmin Li, Hanchao Liu, Xiangyang Ji, Guoqi Li, and Luping Shi. Cifar10-dvs: an event-stream dataset for object classification. *Frontiers in neuroscience*, 11:309, 2017.

[153] Teresa Serrano-Gotarredona and Bernabé Linares-Barranco. Poker-dvs and mnist-dvs. their history, how they were made, and other details. *Frontiers in neuroscience*, 9:481, 2015.

[154] Shu Miao, Guang Chen, Xiangyu Ning, Yang Zi, Kejia Ren, Zhenshan Bing, and Alois Knoll. Neuromorphic vision datasets for pedestrian detection, action recognition, and fall detection. *Frontiers in neurorobotics*, 13:38, 2019.

[155] Antonio Rios-Navarro, Enrique Piñero-Fuentes, Salvador Canas-Moreno, Aqib Javed, Jin Harkin, and Alejandro Linares-Barranco. Lipsfus: A neuromorphic dataset for audio-visual sensory fusion of lip reading. *arXiv preprint arXiv:2304.01080*, 2023.

[156] Daniel Auge, Julian Hille, Etienne Mueller, and Alois Knoll. A survey of encoding techniques for signal processing in spiking neural networks. *Neural Processing Letters*, 53(6):4693–4710, 2021.

[157] Joseph M Brader, Walter Senn, and Stefano Fusi. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural computation*, 19(11):2881–2912, 2007.

[158] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International joint conference on neural networks (IJCNN)*, pages 1–8. ieee, 2015.

[159] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. *Advances in neural information processing systems*, 28, 2015.

[160] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11:682, 2017.

[161] Shih-Chii Liu, André Van Schaik, Bradley A Minch, and Tobi Delbruck. Event-based 64-channel binaural silicon cochlea with q enhancement mechanisms. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2027–2030. IEEE, 2010.

[162] Zhenshan Bing, Claus Meschede, Florian Röhrbein, Kai Huang, and Alois C Knoll. A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in neurorobotics*, 12:35, 2018.

[163] Jaehyun Kim, Heesu Kim, Subin Huh, Jinho Lee, and Kiyoung Choi. Deep neural networks with weighted spikes. *Neurocomputing*, 311:373–386, 2018.

[164] Bodo Rueckauer and Shih-Chii Liu. Conversion of analog to spiking neural networks using sparse temporal coding. In *2018 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–5. IEEE, 2018.

[165] Lei Zhang, Shengyuan Zhou, Tian Zhi, Zidong Du, and Yunji Chen. Tdsnn: From deep neural networks to deep spike neural networks with temporal-coding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1319–1326, 2019.

[166] Hung Tat Chen, Kwan Ting Ng, Amine Bermak, Man Kay Law, and Dominique Martinez. Spike latency coding in biologically inspired microelectronic nose. *IEEE transactions on biomedical circuits and systems*, 5(2):160–168, 2011.

[167] Tobi Delbrück, Bernabe Linares-Barranco, Eugenio Culurciello, and Christoph Posch. Activity-driven, event-based vision sensors. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 2426–2429. IEEE, 2010.

[168] Daqi Liu and Shigang Yue. Fast unsupervised learning for visual pattern recognition using spike timing dependent plasticity. *Neurocomputing*, 249:212–224, 2017.

[169] Saeed Reza Kheradpisheh, Mohammad Ganjtabesh, Simon J Thorpe, and Timothée Masquelier. Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, 99:56–67, 2018.

[170] Seongsik Park, Seijoon Kim, Hyeokjun Choe, and Sungroh Yoon. Fast and efficient information transmission with burst spikes in deep spiking neural networks. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[171] Alexander Sboev, Alexey Serenko, Roman Rybka, and Danila Vlasov. Solving a classification task by spiking neural network with stdp based on rate and temporal input encoding. *Mathematical Methods in the Applied Sciences*, 43(13):7802–7814, 2020.

[172] Stéphane Loiselle, Jean Rouat, Daniel Pressnitzer, and Simon Thorpe. Exploration of rank order coding with spiking neural networks for speech recognition. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2076–2080. IEEE, 2005.

[173] Benjamin Schrauwen, Michiel D'Haene, David Verstraeten, and Jan Van Campenhout. Compact hardware liquid state machines on fpga for real-time speech recognition. *Neural networks*, 21(2-3):511–523, 2008.

[174] Simei Gomes Wysoski, Lubica Benuskova, and Nikola Kasabov. Text-independent speaker authentication with spiking neural networks. In *Artificial Neural Networks–ICANN 2007: 17th International Conference, Porto, Portugal, September 9-13, 2007, Proceedings, Part II 17*, pages 758–767. Springer, 2007.

[175] Vishal Sharma and Dipti Srinivasan. A spiking neural network based on temporal encoding for electricity price time series forecasting in deregulated markets. In *The 2010 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2010.

[176] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[177] Qiuwen Chen and Qinru Qiu. Real-time anomaly detection for streaming data using burst code on a neurosynaptic processor. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 205–207. IEEE, 2017.

[178] Donald D Greenwood. Critical bandwidth and the frequency coordinates of the basilar membrane. *The Journal of the Acoustical Society of America*, 33(10):1344–1356, 1961.

[179] Jorge E Hachmeister. An abbreviated history of the ear: from renaissance to present. *The Yale journal of biology and medicine*, 76(2):81, 2003.

[180] Florian Gomez and Ruedi Stoop. Mammalian pitch sensation shaped by the cochlear fluid. *Nature Physics*, 10(7):530–536, 2014.

[181] Andrew J Oxenham. How we hear: The perception and neural coding of sound. *Annual review of psychology*, 69:27–50, 2018.

[182] Daniel Schurzig, Markus Pietsch, Peter Erfurt, Max E Timm, Thomas Lenarz, and Andrej Kral. A cochlear scaling model for accurate anatomy evaluation and frequency allocation in cochlear implantation. *Hearing Research*, 403:108166, 2021.

[183] PLM Johannesma. The pre-response stimulus ensemble of neurons in the cochlear nucleus. In *Symposium on Hearing Theory, 1972*. IPO, 1972.

[184] Andreas G Katsiamis, Emmanuel M Drakakis, and Richard F Lyon. Practical gammatone-like filters for auditory processing. *EURASIP Journal on Audio, Speech, and Music Processing*, 2007:1–15, 2007.

[185] Yushi Zhang and Waleed H Abdulla. Gammatone auditory filterbank and independent component analysis for speaker identification. In *Ninth international conference on spoken language processing*, 2006.

[186] James T George and Elizabeth Elias. A 16-band reconfigurable hearing aid using variable bandwidth filters. *Global Journals of Research in Engineering*, 14(F1):1–7, 2014.

[187] Roneel V Sharan, Shlomo Berkovsky, and Sidong Liu. Voice command recognition using biologically inspired time-frequency representation and convolutional neural networks. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 998–1001. IEEE, 2020.

[188] Nik Dennler, Germain Haessig, Matteo Cartiglia, and Giacomo Indiveri. Online detection of vibration anomalies using balanced spiking neural networks. In *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–4. IEEE, 2021.

[189] Sony group portal - ai initiatives - event-based vision sensor (evs). https://www.sony.com/en/SonyInfo/sony_ai/technology/evs.html.

[190] Greg Blackman. Prophesee releases industrial-grade neuromorphic sensor: Greg blackman speaks to prophesee's luca verre about high-speed imaging with event-based cameras. *Imaging and Machine Vision Europe*, 95(95):14–15, 2019.

[191] Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2003.

[192] Qian Liu, Garibaldi Pineda-García, Evangelos Stromatias, Teresa Serrano-Gotarredona, and Steve B Furber. Benchmarking spike-based visual recognition: a dataset and evaluation. *Frontiers in neuroscience*, 10:496, 2016.

[193] Julien Dupeyroux, Stein Stroobants, and Guido CHE De Croon. A toolbox for neuromorphic perception in robotics. In *2022 8th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)*, pages 1–7. IEEE, 2022.

[194] Elisa Donati, Melika Payvand, Nicoletta Risi, Renate Krause, and Giacomo Indiveri. Discrimination of emg signals using a neuromorphic implementation of a spiking neural network. *IEEE transactions on biomedical circuits and systems*, 13(5):795–803, 2019.

[195] Chang Gao, Stefan Braun, Ilya Kiselev, Jithendar Anumula, Tobi Delbruck, and Shih-Chii Liu. Real-time speech recognition for iot purpose using a delta recurrent neural network accelerator. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

[196] Tobi Delbruck and Patrick Lichtsteiner. Fast sensory motor control based on event-based hybrid neuromorphic-procedural system. In *2007 IEEE international symposium on circuits and systems*, pages 845–848. IEEE, 2007.

[197] Nikola Kasabov, Nathan Matthew Scott, Enmei Tu, Stefan Marks, Neelava Sengupta, Elisa Capecci, Muhaini Othman, Maryam Gholami Doborjeh, Norhanifah Murli, Reggio Hartono, et al. Evolving spatio-temporal data machines based on the neucube neuromorphic framework: Design methodology and selected applications. *Neural Networks*, 78:1–14, 2016.

[198] Jacob Wiren and Harold L Stubbs. Electronic binary selection system for phoneme classification. *The Journal of the Acoustical Society of America*, 28(6):1082–1091, 1956.

[199] Benjamin Kedem. Spectral analysis and discrimination by zero-crossings. *Proceedings of the IEEE*, 74(11):1477–1493, 1986.

[200] Michael Hough, Hugo De Garis, Michael Korkin, Felix Gers, and Norberto Eiji Nawa. Spiker: Analog waveform to digital spiketrain conversion in atr's artificial brain (cam-brain) project. In *International conference on robotics and artificial life*, volume 92. Citeseer, 1999.

[201] Benjamin Schrauwen and Jan Van Campenhout. Bsa, a fast and accurate spike train encoding scheme. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 4, pages 2825–2830. IEEE, 2003.

[202] Balint Petro, Nikola Kasabov, and Rita M Kiss. Selection and optimization of temporal spike encoding methods for spiking neural networks. *IEEE transactions on neural networks and learning systems*, 31(2):358–370, 2019.

[203] John J Hopfield. Pattern recognition computation using action potential timing for stimulus representation. *Nature*, 376(6535):33–36, 1995.

[204] Simon Thorpe and Jacques Gautrais. Rank order coding. *Computational Neuroscience: Trends in Research, 1998*, pages 113–118, 1998.

[205] Roland S Johansson and Ingvars Birznieks. First spikes in ensembles of human tactile afferents code complex spatial fingertip events. *Nature neuroscience*, 7(2):170–177, 2004.

[206] Marcelo A Montemurro, Malte J Rasch, Yusuke Murayama, Nikos K Logothetis, and Stefano Panzeri. Phase-of-firing coding of natural visual stimuli in primary visual cortex. *Current biology*, 18(5):375–380, 2008.

[207] Seongsik Park, Seijoon Kim, Byunggook Na, and Sungroh Yoon. T2fsnn: deep spiking neural networks with time-to-first-spike coding. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[208] John E Lisman. Bursts as a unit of neural information: making unreliable synapses reliable. *Trends in neurosciences*, 20(1):38–43, 1997.

[209] Eugene M Izhikevich, Niraj S Desai, Elisabeth C Walcott, and Frank C Hoppensteadt. Bursts as a unit of neural information: selective communication via resonance. *Trends in neurosciences*, 26(3):161–167, 2003.

[210] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[211] Rodrigo Quian Quiroga and Stefano Panzeri. Extracting information from neuronal populations: information theory and decoding approaches. *Nature Reviews Neuroscience*, 10(3):173–185, 2009.

[212] Patrik O Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(9), 2004.

[213] José M de la Rosa. Sigma-delta modulators: Tutorial overview, design guide, and state-of-the-art survey. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(1):1–21, 2010.

[214] Kashu Yamazaki, Viet-Khoa Vo-Ho, Darshan Bulsara, and Ngan Le. Spiking neural networks and their applications: A review. *Brain Sciences*, 12(7):863, 2022.

[215] Wulfram Gerstner. A framework for spiking neuron models: The spike response model. In *Handbook of biological physics*, volume 4, pages 469–516. Elsevier, 2001.

[216] Nicholas J Pritchard, Andreas Wicenec, Mohammed Bennamoun, and Richard Dodson. A bibliometric review of neuromorphic computing and spiking neural networks. *arXiv preprint arXiv:2304.06897*, 2023.

[217] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[218] Charlotte Frenkel, Martin Lefebvre, Jean-Didier Legat, and David Bol. A 0.086-mm^2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos. *IEEE transactions on biomedical circuits and systems*, 13(1):145–158, 2018.

[219] Nitin Rathi, Indranil Chakraborty, Adarsh Kosta, Abhronil Sengupta, Aayush Ankit, Priyadarshini Panda, and Kaushik Roy. Exploring neuromorphic computing based on spiking neural networks: Algorithms to hardware. *ACM Computing Surveys*, 55(12):1–49, 2023.

[220] Louis Lapicque. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *Journal de physiologie et de pathologie générale*, 9:620–635, 1907.

[221] Doron Tal and Eric L Schwartz. Computing with the leaky integrate-and-fire neuron: logarithmic computation and multiplication. *Neural computation*, 9(2):305–318, 1997.

[222] Jason K Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, Mohammed Bennamoun, Doo Seok Jeong, and Wei D Lu. Training spiking neural networks using lessons from deep learning. *arXiv preprint arXiv:2109.12894*, 2021.

[223] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *Advances in neural information processing systems*, 31, 2018.

[224] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):3625, 2020.

[225] Intel Corporation. Dynamics, neurons, and spikes - lava documentation. https://lava-nc.org/lava-lib-dl/slayer/notebooks/neuron_dynamics/dynamics.html#2.1-Adaptive-Leaky-Integrator-and-Fire-Neuron, 2021.

[226] Robert Urbanczik and Walter Senn. Learning by the dendritic prediction of somatic spiking. *Neuron*, 81(3):521–528, 2014.

[227] Luca Peres and Oliver Rhodes. Parallelization of neural processing on neuromorphic hardware. *Frontiers in Neuroscience*, 16, 2022.

[228] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[229] Bojian Yin, Federico Corradi, and Sander M Bohté. Accurate and efficient time-domain classification with adaptive spiking recurrent neural networks. *Nature Machine Intelligence*, 3(10):905–913, 2021.

[230] Friedemann Zenke and Tim P Vogels. The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks. *Neural computation*, 33(4):899–925, 2021.

[231] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

[232] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *Autodiff Workshop*, 2017.

[233] Howard Eichenbaum. Time cells in the hippocampus: a new dimension for mapping memories. *Nature Reviews Neuroscience*, 15(11):732–744, 2014.

[234] Aaron R Voelker and Chris Eliasmith. Improving spiking dynamical networks: Accurate delays, higher-order synapses, and time cells. *Neural computation*, 30(3):569–609, 2018.

[235] Aaron Voelker, Ivana Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. *Advances in neural information processing systems*, 32, 2019.

[236] Aaron R Voelker and Chris Eliasmith. Programming neuromorphics using the neural engineering framework. *Handbook of Neuroengineering*, pages 1–43, 2020.

[237] Peter Blouw, Gurshaant Malik, Benjamin Morcos, Aaron R Voelker, and Chris Eliasmith. Hardware aware training for efficient keyword spotting on general purpose and specialized hardware. *arXiv preprint arXiv:2009.04465*, 2020.

[238] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508, 2016.

[239] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

[240] Sen Song, Kenneth D Miller, and Larry F Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 3(9):919–926, 2000.

[241] Sumit B Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. *Advances in neural information processing systems*, 31, 2018.

[242] Charlotte Frenkel and Giacomo Indiveri. Reckon: A 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 65, pages 1–3. IEEE, 2022.

[243] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113:54–66, 2015.

[244] Jyotibdha Acharya, Aakash Patil, Xiaoya Li, Yi Chen, Shih-Chii Liu, and Arindam Basu. A comparison of low-complexity real-time feature extraction for neuromorphic speech recognition. *Frontiers in neuroscience*, 12:160, 2018.

[245] Jithendar Anumula, Daniel Neil, Tobi Delbruck, and Shih-Chii Liu. Feature representations for neuromorphic audio spike streams. *Frontiers in neuroscience*, 12:23, 2018.

[246] Enea Ceolini, Jithendar Anumula, Stefan Braun, and Shih-Chii Liu. Event-driven pipeline for low-latency low-compute keyword spotting and speaker verification system. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7953–7957. IEEE, 2019.

[247] Ahana Gangopadhyay and Shantanu Chakrabartty. A sparsity-driven backpropagation-less learning framework using populations of spiking growth transform neurons. *Frontiers in neuroscience*, 15:715451, 2021.

[248] James Paul Turner, James C Knight, Ajay Subramanian, and Thomas Nowotny. mlgenn: accelerating snn inference using gpu-enabled neural networks. *Neuromorphic Computing and Engineering*, 2(2):024002, 2022.

[249] Andrew P Davison, Daniel Brüderle, Jochen M Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. Pynn: a common interface for neuronal network simulators. *Frontiers in neuroinformatics*, page 11, 2009.

[250] Germany Electronic Vision(s) Group at the Kirchhoff-Institute for Physics, Heidelberg University. Pynn for brainscales-2. https://github.com/electronicvisions/pynn-brainscales, 2020.

[251] SpiNNaker University of Manchester. spynnaker - pynn simulations on spinnaker hardware. https://github.com/SpiNNakerManchester/sPyNNaker, 2021.

[252] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7:48, 2014.

[253] Daniel Rasmussen. Nengodl: Combining deep learning and neuromorphic modelling methods. *Neuroinformatics*, 17(4):611–628, 2019.

[254] Jozsef Suto. The effect of hyperparameter search on artificial neural network in human activity recognition. *Open Computer Science*, 11(1):411–422, 2021.

[255] Microsoft. Neural Network Intelligence. https://github.com/microsoft/nni, 1 2021.

[256] Evelina Forno, Andrea Acquaviva, Yuki Kobayashi, Enrico Macii, and Gianvito Urgese. A parallel hardware architecture for quantum annealing algorithm acceleration. In *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 31–36. IEEE, 2018.

[257] Francesco Barchi, Gianvito Urgese, Andrea Acquaviva, and Enrico Macii. Directed graph placement for snn simulation into a multi-core gals architecture. In *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 19–24. IEEE, 2018.

[258] Francesco Barchi, Gianvito Urgese, Enrico Macii, and Andrea Acquaviva. Mapping spiking neural networks on multi-core neuromorphic platforms: Problem formulation and performance analysis. In *VLSI-SoC: Design and Engineering of Electronics Systems Based on New Computing Paradigms: 26th IFIP WG 10.5/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2018, Verona, Italy, October 8–10, 2018, Revised and Extended Selected Papers 26*, pages 167–186. Springer, 2019.

[259] Javier Navaridas, Mikel Luján, Luis A Plana, Steve Temple, and Steve B Furber. Spinnaker: Enhanced multicast routing. *Parallel Computing*, 45:49–66, 2015.

[260] Giacomo Indiveri. Neuromorphic engineering. *Springer Handbook of Computational Intelligence*, pages 715–725, 2015.

[261] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[262] Charlotte Frenkel, Jean-Didier Legat, and David Bol. Morphic: A 65-nm 738k-synapse/mm^2 quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning. *IEEE transactions on biomedical circuits and systems*, 13(5):999–1010, 2019.

[263] Jia-Qin Yang, Ruopeng Wang, Yi Ren, Jing-Yu Mao, Zhan-Peng Wang, Ye Zhou, and Su-Ting Han. Neuromorphic engineering: From biological to spike-based hardware nervous systems. *Advanced Materials*, 32(52):2003610, 2020.

[264] Youhui Zhang, Peng Qu, and Weimin Zheng. Towards" general purpose" brain-inspired computing system. *Tsinghua Science and Technology*, 26(5):664–673, 2021.

[265] Johannes Schemmel, Andreas Grübl, Stephan Hartmann, Alexander Kononov, Christian Mayr, Karlheinz Meier, Sebastian Millner, Johannes Partzsch, Stefan Schiefer, Stefan Scholze, et al. Live demonstration: A scaled-down version of the brainscales wafer-scale neuromorphic system. In *2012 IEEE international symposium on circuits and systems (ISCAS)*, pages 702–702. IEEE, 2012.

[266] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[267] Louis Blin, Ahsan Javed Awan, and Thomas Heinis. Using neuromorphic hardware for the scalable execution of massively parallel, communication-intensive algorithms. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 89–94. IEEE, 2018.

[268] Indar Sugiarto, Gengting Liu, Simon Davidson, Luis A Plana, and Steve B Furber. High performance computing on spinnaker neuromorphic platform: A case study for energy efficient image processing. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2016.

[269] Xin Jin, Steve B Furber, and John V Woods. Efficient modelling of spiking neural networks on a scalable chip multiprocessor. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 2812–2819. IEEE, 2008.

[270] Andrew D Brown, Steve B Furber, Jeffrey S Reeve, Jim D Garside, Kier J Dugan, Luis A Plana, and Steve Temple. Spinnaker—programming model. *IEEE Transactions on Computers*, 64(6):1769–1782, 2014.

[271] Francesco Barchi, Gianvito Urgese, Enrico Macii, and Andrea Acquaviva. An efficient mpi implementation for multi-coreneuromorphic platforms. In *2017 New Generation of CAS (NGCAS)*, pages 273–276. IEEE, 2017.

[272] Andrew GD Rowley, Christian Brenninkmeijer, Simon Davidson, Donal Fellows, Andrew Gait, David R Lester, Luis A Plana, Oliver Rhodes, Alan B Stokes, and Steve B Furber. Spinntools: the execution engine for the spinnaker platform. *Frontiers in neuroscience*, 13:231, 2019.

[273] Francesco Barchi, Gianvito Urgese, Alessandro Siino, Santa Di Cataldo, Enrico Macii, and Andrea Acquaviva. Flexible on-line reconfiguration of multi-core neuromorphic platforms. *IEEE Transactions on Emerging Topics in Computing*, 9(2):915–927, 2019.

[274] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[275] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[276] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146, 2010.

[277] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[278] Yue Geng and Xinyu Luo. Cost-sensitive convolution based neural networks for imbalanced time-series classification. *arXiv preprint arXiv:1801.04396*, 2018.

[279] Joan Serrà, Santiago Pascual, and Alexandros Karatzoglou. Towards a universal neural network encoder for time series. In *CCIA*, pages 120–129, 2018.

[280] Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, 2017.

[281] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F Schmidt, Jonathan Weber, Geoffrey I Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, 2020.

[282] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

[283] Arnab Neelim Mazumder, Morteza Hosseini, Tinoosh Mohsenin, et al. 2021 roadmap on neuromorphic computing and engineering. *UMBC Student Collection*, 2021.

[284] Chris Yakopcic, Nayim Rahman, Tanvir Atahary, Tarek M Taha, and Scott Douglass. Solving constraint satisfaction problems using the loihi spiking neuromorphic processor. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1079–1084. IEEE, 2020.

[285] Juan P Domínguez-Morales, Stefano Buccelli, Daniel Gutierrez-Galan, Ilaria Colombi, Angel Jimenez-Fernandez, and Michela Chiappalone. Real-time detection of bursts in neuronal cultures using a neuromorphic auditory sensor and spiking neural networks. *Neurocomputing*, 449:422–434, 2021.

[286] Federico Corradi, Sandeep Pande, Jan Stuijt, Ning Qiao, Siebren Schaafsma, Giacomo Indiveri, and Francky Catthoor. Ecg-based heartbeat classification in neuromorphic hardware. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.

[287] Kenneth Stewart, Garrick Orchard, Sumit Bam Shrestha, and Emre Neftci. On-chip few-shot learning with surrogate gradient descent on a neuromorphic processor. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 223–227. IEEE, 2020.

[288] Travis DeWolf, Pawel Jaworski, and Chris Eliasmith. Nengo and low-power ai hardware for robust, embedded neurorobotics. *Frontiers in Neurorobotics*, 14:568359, 2020.

[289] Rasmus Karnøe Stagsted, Antonio Vitale, Alpha Renner, Leon Bonde Larsen, Anders Lyhne Christensen, and Yulia Sandamirskaya. Event-based pid controller fully realized in neuromorphic hardware: A one dof study. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10939–10944. IEEE, 2020.

[290] Raphaela Kreiser, Gabriel Waibel, Nuria Armengol, Alpha Renner, and Yulia Sandamirskaya. Error estimation and correction in a spiking neural network for map formation in neuromorphic hardware. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6134–6140. IEEE, 2020.

[291] Guangzhi Tang, Arpit Shah, and Konstantinos P Michmizos. Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4176–4181. IEEE, 2019.

[292] Patrick Farr, Aaron M Jones, Trevor Bihl, Jayson Boubin, and Ashley DeMange. Waveform design implemented on neuromorphic hardware. In *2020 IEEE International Radar Conference (RADAR)*, pages 934–939. IEEE, 2020.

[293] Cosmas Ifeanyi Nwakanma, Jae-Woo Kim, Jae-Min Lee, and Dong-Seong Kim. Edge ai prospect using the neuroedge computing system: Introducing a novel neuromorphic technology. *ICT Express*, 7(2):152–157, 2021.

[294] Adarsha Balaji, Shihao Song, Twisha Titirsha, Anup Das, Jeffrey Krichmar, Nikil Dutt, James Shackleford, Nagarajan Kandasamy, and Francky Catthoor. Neuroxplorer 1.0: An extensible framework for architectural exploration with spiking neural networks. In *International Conference on Neuromorphic Systems 2021*, pages 1–9, 2021.

[295] Andrea Spitale. Interfacing a neuromorphic coprocessor with a risc-v architecture. Master's thesis, Politecnico di Torino, 2021.

[296] Alexander Dörflinger, Mark Albers, Benedikt Kleinbeck, Yejun Guan, Harald Michalik, Raphael Klink, Christopher Blochwitz, Anouar Nechi, and Mladen Berekovic. A comparative survey of open-source application-class risc-v processor implementations. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*, pages 12–20, 2021.

[297] UC Berkeley Architecture Research. Chipyard framework. https://github.com/ucb-bar/chipyard, 2021.

[298] Charlotte Frenkel. Odin spiking neural network (snn) processor. https://github.com/ChFrenkel/ODIN, 2019.

[299] Joshua Arul Kumar Ranjan, Titus Sigamani, and Janet Barnabas. A novel and efficient classifier using spiking neural network. *The Journal of Supercomputing*, 76(9):6545–6560, September 2020.

[300] Christoph Kayser, Marcelo A Montemurro, Nikos K Logothetis, and Stefano Panzeri. Spike-phase coding boosts and stabilizes information carried by spatial and temporal spike patterns. *Neuron*, 61(4):597–608, 2009.

[301] Eliathamby Ambikairajah, Julien Epps, and Lee Lin. Wideband speech and audio coding using gammatone filter banks. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*, volume 2, pages 773–776. IEEE, 2001.

[302] Nafiul Rashid, Berken Utku Demirel, and Mohammad Abdullah Al Faruque. Ahar: Adaptive cnn for energy-efficient human activity recognition in low-power edge devices. *IEEE Internet of Things Journal*, 2022.