Politecnico di Torino

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

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (38^{th}cycle)

From Neuron Models to Edge Intelligence: A Multi-Level Exploration of Neuromorphic Computing

By

Riccardo Pignari

******

Supervisors:
Prof. Gianvito Urgese, Supervisor
Prof. Enrico Macii, Co-Supervisor

Doctoral Examination Committee:
Luca Peres PhD, Referee, University of Manchester
Prof. Catherine Schuman, Referee, University of Tennessee
Petruț Antoniu Bogdan PhD, Innatera Nanosystems
Prof. Santa Di Cataldo, Politecnico di Torino
Prof. Thomas Nowotny, University of Sussex

Politecnico di Torino
2026

Declaration

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

Riccardo Pignari
2026

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

Abstract

The work presented in this thesis falls within the field of neuromorphic computing, an emerging paradigm that offers an alternative to the traditional von Neumann model. Inspired by the functioning of the human brain, this approach aims to combine energy efficiency, massive parallelism and distributed robustness, opening up new perspectives for intelligent processing in real-world scenarios. The research was conducted on several complementary levels. At the low-level analysis, the research introduce the Neuronal Phase Map method for representing the dynamics of spiking models, facilitating their interpretation and design. On the higher level, neuromorphic applications were developed for combinatorial optimisation problems, such as MAX-CUT and the Latin Square Problem, demonstrating how spiking neural networks can implement dynamics inspired by simulated annealing and constraint satisfaction. In parallel, neuromorphic pipelines were designed and analysed for classification tasks, particularly in Human Activity Recognition and audio signal processing, evaluating different spike coding strategies and demonstrating the potential of SNNs as a compromise between accuracy and energy sustainability on edge devices. In addition, micro-benchmarking was initiated on the Lava framework and Loihi 2 hardware, aimed at systematically measuring performance at different architectural levels. Overall, the results obtained show that neuromorphic computing can offer concrete and efficient solutions in heterogeneous domains, confirming its relevance as an alternative and competitive paradigm. Future prospects are to be extended to more complex application scenarios, the development of more general design methodologies, and experimentation on multi-chip platforms, helping to bring these technologies from the laboratory to the real world.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last few decades, the evolution of computational systems has been strongly guided by the possibilities offered by the traditional architectural paradigm, known as the von Neumann paradigm. Proposed by John von Neumann in the 40s, this model is based on a linear and compartmentalised architecture, in which the memory and the processing unit are physically and functionally distinct components [1]. This model, characterised by a clear structural separation, has formed the theoretical and practical basis on which modern computers and processing systems have been developed [2].

The linear structure of the architecture is reflected in a sequential processing of data flow, using the imperative programming paradigm [3, 4]. The programmer defines step by step the instructions on "what to do" and "how to do it", using a programming language [5]. These instructions are executed sequentially, modifying the state of the system through operations on variables and control structures. The imperative programming paradigm lays the foundations for the theory of computability and the study of algorithm complexity [6]. An emblematic example is the Turing machine [7], which is fundamental for identifying the capabilities and limitations of real computing systems.

The von Neumann architectural paradigm and the imperative programming paradigm represented a milestone in the development of computer science, serving as the theoretical and engineering foundation for the design of general-purpose computing units namely CPUs [8]. Although these processors form the basis of modern hardware structures, their capabilities encounter structural limitations when

addressing large-scale computational problems with low latency and high parallelism requirements [9]. The general-purpose nature of these processors imposes constraints on scalability, reducing their energy efficiency and preventing their use in more complex contexts.

To address these limitations, the paradigm of parallel [10–12] and distributed computing has gradually been adopted alongside that of object-oriented programming [13], made possible by multi-core hardware and Graphics Processing Units (GPUs) [14–16], as well as distributed computing infrastructures such as clusters [17–19], clouds [20, 21] and edge computing [22–25]. In this scenario, processing is divided among multiple computational units, which operate in parallel on subcomponents of the task or on different data. The goal is to increase performance in terms of throughput, latency and improve energy efficiency [26–28], especially in computationally intensive areas such as deep learning, scientific simulation, and real-time analysis of large amounts of data.

However, despite its widespread adoption, parallel and distributed computing remains in most cases an extension of the von Neumann paradigm rather than a successor to it. The basic logic - data transfer between memory and processor, execution controlled by a central clock and the deterministic algorithmic programming - continues to represent a bottleneck in terms of scalability, energy efficiency and adoption in asynchronous application contexts [9].

These critical factors have motivated the exploration of new architectural and computational models that are better suited to address the challenges of modern data processing. It is precisely these needs that have led to the emergence of new computational paradigms, either alternative or complementary to the classical one [29–31]. These include quantum computing, which exploits quantum properties [32–34], and neuromorphic computing, which is directly inspired by non-sequential, non-symbolic and bio-inspired models [29–31]. These paradigms offer a discontinuity with respect to the classical von Neumann paradigm.

1.1 Paradigm of neuromorphic computing

In contrast to traditional computational paradigms, based on a separation between memory and processing units and a rigidly orchestrated sequential or parallel com-

puting logic, the neuromorphic computing paradigm draws inspiration from the organisation and functioning of biological nervous systems, offering advantages such as: integration of memory and computation, energy efficiency, massive and asynchronous parallelism [35]. These advantages are not limited to incremental improvements in efficiency, but represent a real discontinuity from classical paradigms, which justify why the scientific community is increasingly interested in developing neuromorphic systems [36–39]. Figure 1.1 shows the main differences and strengths of the neuromorphic paradigm compared to the von Neumann paradigm.

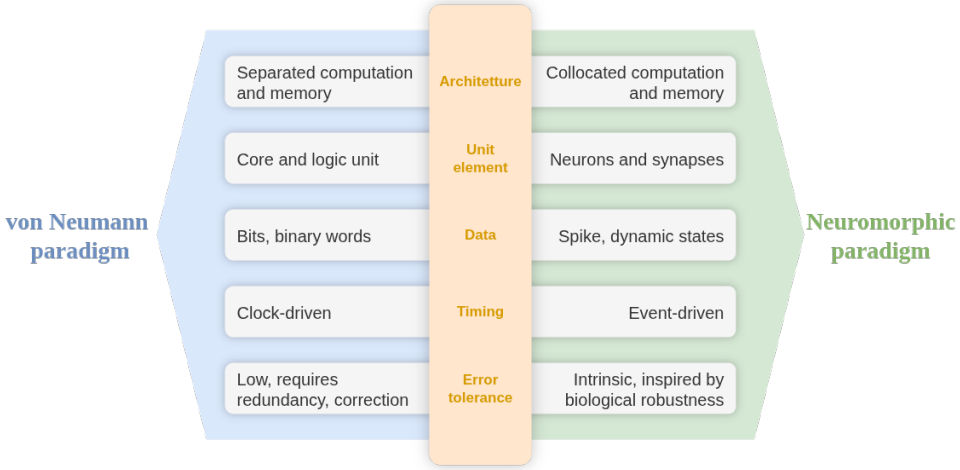

Fig. 1.1 In neuromorphic architectures, memory is distributed within the synapses, eliminating the access overhead and drastically reducing energy consumption. Computation is massively parallel and distributed, with each neuron acting as an autonomous unit. Information is encoded temporally, through the relative distribution of spikes, allowing the representation of sensory sequences and patterns in a bio-inspired and efficient manner.

In this case, memory is distributed within the synapses, which represent the strength of the connections between neurons. This eliminates the overhead of accessing memory and allows for a drastic reduction in energy consumption. This is in strong contrast to the classical architecture, in which memory is separate and must be continuously consulted by the processor to transfer data and instructions.

By contrast traditional architectures, which achieve parallelism through multiple cores synchronised by a central clock, the neuromorphic paradigm implements massive, distributed parallelism, in which each neuron acts as an autonomous computational unit. Each elementary unit locally combines inputs from other neurons and produces an output only when its activity exceeds a threshold, producing single-bit information, 1 in the presence of information 0 otherwise. In the most modern neu-

romorphic applications and platforms, the concept of spike is extended to a broader meaning than the biological one, associating more complex information such as a real value or an array, while maintaining its asynchronous and event-driven nature.

Compared to conventional digital models, which represent information using static values such as bits and numerical vectors, neuromorphic systems use temporal encoding, in which information is coded in the relative temporal distribution of spikes. This allows temporal sequences and sensory patterns to be represented and processed in a bio-inspired and efficient manner. Furthermore, the use of mechanisms based on the membrane potentials dynamics allows complex relationships between stimuli to be expressed without the need for explicit memory structures.

In addition, these systems are inherently robust, thanks to their distributed and redundant architecture. The loss of a neuron or synapse in a sufficiently large system does not compromise the overall functioning of the network. This feature makes neuromorphic platforms particularly well suited to operating in non-deterministic environments, where classical systems would require complex fault-tolerance and error correction strategies.

These strengths collectively highlight how neuromorphic computing is not simply an optimisation of existing approaches, but a new computational paradigm capable of tackling problems for which classical models would be inefficient or conceptually inadequate.

1.2 Approaches to the design Spiking Neural Network

Within the neuromorphic paradigm, there are two distinct approaches to designing computational models: one approach involves manually designing specific neural circuits [40, 41]; and the other based on machine learning techniques, through the use of Spiking Neural Networks (SNNs) [42–44]. These two approaches represent not only different strategies for addressing the design of neuromorphic systems, but also define two distinct perspectives: on the one hand, there is an engineering, deductive and modelling approach; on the other, a data-driven, adaptive method based on machine learning.

Although the two share the same constituent elements - spiking neurons, synapses, temporal dynamics - the two approaches differ profoundly in terms of applicability,

methodology, generalisation capabilities and computational requirements. In the first case, the focus is on the construction and interpretability of neural circuits capable of performing specific tasks, through the explicit design of populations of neurons, synaptic connections, and temporal dynamics. In the second, the system is designed as a plastic structure capable of learning from the data provided, internally modelling the representations necessary to solve a given task through biologically inspired optimisation algorithms or those adapted from traditional machine learning. The choice between one approach or the other depends on multiple factors such as: level of control, task complexity, data availability, hardware and energy constraints.

The coexistence of these two approaches reflects the exploratory nature of the neuromorphic paradigm, which is still in its infancy in terms of architectural standards research and development.

1.2.1 Manual design of neural circuits approach

The first approach, closer to computational logic and classical neuromorphic electronics, involves the manual design of ad hoc neural circuits, geared towards performing specific functions. In this context, neurons and synapses are modelled and interconnected to reproduce logical-mathematical or dynamic behaviours. This type of development is particularly suitable for simple and well-specified tasks, such as counters, oscillators and integrators; or for embedded/edge computing systems, with stringent constraints on energy consumption or memory capacity [45]. In other words, for all those applications where interpretability and predictability of behaviour are essential.

An emerging and exploratory field in manual design concerns the implementation of neural circuits capable of solving optimisation problems [46–49]. In this scenario, populations of neurons are configured to represent state variables or a domain, while synaptic connections and network dynamics can be designed to model the constraints that must be satisfied to reach a valid solution state. The interaction between neurons, through excitatory and inhibitory synapses, leads the system towards a status of exploration of the states in the candidate space, converging towards stationary or cyclic states that represent the solution configurations of the problem.

Manual design represents a powerful and controllable approach to neuromorphic computing, especially for systems that require logical operations and low complexity.

Its strength lies in the possibility of directly creating circuits dedicated to logical and symbolic computation.

While this approach allows for a high degree of control, it suffers from poor scalability. As the complexity of the problem increases, manual circuit design quickly becomes prohibitive, preventing its large-scale use for complex or high-dimensional tasks. Furthermore, adapting to changes in constraints or problem parameters often requires a complete redesign of the network.

1.2.2 Machine learning and data-driven approach

The second approach to building neuromorphic systems is inspired by machine learning principles, in which SNNs are trained to learn the tasks to be performed autonomously. The use of machine learning techniques within the neuromorphic paradigm has opened up new application frontiers in various fields, particularly in contexts that require efficient processing of event-driven temporal data or multi-channel sensory data.

The potential of integrating machine learning and neuromorphic hardware is highlighted by several use cases, such as audio classification [50, 36], Human Activity Recognition (HAR) [43], and embedded systems [51, 52].

Studies on audio signals have shown that appropriately trained SNN architectures can achieve accuracies comparable to those of traditional neural networks, while demonstrating dramatic energy savings when implemented on neuromorphic hardware such as Loihi 2 [53, 54] or DYNAP-SE [55, 56]. The adoption of neuromorphic models for HAR is particularly advantageous in wearable environments, where energy and computational constraints require efficient use of resources. Furthermore, the temporal dynamics of SNNs make them naturally suited to handling non-stationary data sequences with minimal latency.

A distinctive advantage of SNNs over traditional Artificial Neural Networks (ANNs) is their operational efficiency on specialised neuromorphic hardware, which is perfectly suited for use in edge computing. Hardware such as Intel Loihi [37], IBM TrueNorth [57], BrainScaleS [58], and open-source projects with SpiNNaker 2 [38] demonstrate that it is possible to build complex SNNs on chips that operate at lower orders of magnitude in terms of power consumption than traditional CPUs or GPUs.

1.3 Spike encoding: The Interface between data and spiking neural networks

One of the most important aspects of using SNNs lies in the way information is represented, i.e. how raw data is transformed into spike trains for processing. Unlike traditional neural models, neuromorphic systems base the elaboration process on discrete events known as spikes, which encode information in the temporal distribution of events using quantities such as frequency, Inter-Spike Interval (ISI) or activation pattern [35].

This need has led to the development of various encoding methods, which can be classified into two broad categories: algorithmic encoding and neural encoding.

1.3.1 Algorithmic spike encoding

In the following approach, information is encoded in a temporal spike sequence, using algorithms that attempt to reproduce - through a series of instructions and signal transformations - the response pattern of a neuron undergoing various stimuli derived from signal analysis obtained from in vivo experiments. The main algorithmic coding schemes include categories belonging to: rate coding, where the rate of spikes is proportional to the intensity of the stimulus; or categories of temporal coding, where the coding characteristics extend those of rate coding using relative spike timing and ISI [42].

These encoding methods allow for a high degree of algorithm customisation, but at the same time introduce a significant limitation on the variety of encoding patterns that the algorithm can provide.

1.3.2 Model-based spike encoding

The alternative approach to the algorithmic encoding is the model-based encoding. With these methodologies the spike coding is entrusted directly to the neuron model, to which the data to be coded is supplied directly or by synaptic models.

In this context, the neuron is not only a computing unit, but also a signal transducer. Model-based encoding is more biologically plausible not only because it

mimics the mechanisms by which real sensory neurons transform physical stimuli into spikes, but also because real neuron models are used for this purpose. Furthermore, this type of encoding allows for a more fluid integration between the encoding and processing phases, especially when the model is simulated or implemented directly on neuromorphic hardware, minimising the latency and bottlenecks introduced by preprocessing steps [59].

Although closer to biological world, this type of coding is not without implementation challenges. Possible design difficulties include calibrating neural parameters such as: threshold; membrane time constant; refractory period; adaptation of the data domains.

1.4 Objective and motivation

In light of the new possibilities introduced by the neuromorphic computing paradigm, this research aims to explore new computational models and approaches capable of extending the possibilities of tools that can operate on systems and approaches belonging to the neuromorphic world. In this context, the study objective is divided into four main areas, each aimed at exploring the opportunities that have emerged in the applications of neuromorphic architectures:

1. **Low level analysis strategy**: low-level analyses of neuron models were conducted with the aim of understanding and measuring their computational properties. In particular, I studied the behaviour of spiking neuron models using mathematical tools of nonlinear dynamics and phase space representations, developing methodologies to make the complexity of neuron models more interpretable by introducing the concept of Neuronal Phase Map [60]. A graphical representation that makes it possible to visualise how the neurocomputational features of neuron models vary as parameters change. This technique proved useful not only for deepening the understanding of neural dynamics, but also as a practical tool to support the design and prototyping of neuromorphic solutions, especially in the spike train input encoding phases.

2. **Design of functional neural circuits**: through a manual design approach of a spiking networks, we addressed the problem of combinatorial optimisation in the neuromorphic field, with the aim of explicitly mapping the dynamics of

the problem onto the behaviour of neurons and synapses. First, we explored architectures inspired by annealing models, such as NeuroSA [40], in which pairs of ON-OFF neurons integrate adaptive threshold mechanisms based on quantum tunnelling to faithfully reproduce the dynamics of simulated annealing and ensure convergence towards the ground state of Ising problems. This has made it possible to solve classic benchmarks such as MAX-CUT and Max Independent Set with performance comparable to or superior to the state-of-the-art, without the need for specific hyperparameters for each graph. Secondly, I designed spiking pipelines dedicated to constraint satisfaction problems, such as Sudoku, introducing additional neural layers capable of implementing logical control functions, solution validation and network state storage [41]. This strategy, validated first in simulation with GeNN and then on neuromorphic platforms SpiNNaker, showed three main advantages: increased success rates on particularly complex instances, significant reduction in communication traffic thanks to the transmission of the final state of the network rather than the entire spike flow, and a drastic reduction in the time required for spike extraction. Overall, the manual design approach adopted has highlighted how accurate engineering of spiking dynamics can translate into more robust, efficient and energy-sustainable solutions for optimisation in neuromorphic computing.

3. **Training spiking neural networks using machine learning**: we addressed classification problems based on temporal signals through a data-driven approach on neuromorphic networks. In particular, we focused on signal classification from digital sensors and application of HAR, both highly relevant to edge computing and the Internet of Things [42]. The comparative analysis between different encoding techniques and different datasets (audio and inertial sensors) has made it possible to build a generalisable methodology for selecting the best trade-off between accuracy, efficiency, and energy consumption, thus expanding the range of applications that can be achieved with neuromorphic artificial intelligence in embedded and IoT contexts. Further studies led to the development of neuromorphic pipelines that utilise SNNs to efficiently process time-dependent signals, demonstrating how these networks can achieve competitive performance with respect to traditional deep models, but with significantly reduced energy consumption. The approach included application-specific optimisation of hyperparameters, the definition of encod-

ing strategies for spike train signals, and the use of specialised architectures, such as spiking CNNs and hybrid models with memory units (e.g. LMUs), capable of capturing complex temporal relationships [43].

4. **Micro-benchmarking suite**: finally we implemented benchmarking activities, contributing to the development of a micro-benchmarking suite for the Lava framework and Loihi 2 neuromorphic hardware, inspired by MPI tests but adapted to the peculiarities of neuromorphic architectures. This suite allows for the separate evaluation of both the framework messaging infrastructure and the efficiency of the Loihi 2 architecture, including tests for host CPUs, embedded processors, and neuron cores [44]. This tool provides detailed metrics on execution times, resource utilisation, and communication performance, offering a useful benchmark for both framework developers and end users.

Chapter 2

Neuro-computational features of spiking neuron

Neuron modelling is a central aspect of the study and development of neural networks [35]. Over the decades, numerous models have been developed with the aim of capturing the electrophysiological dynamics of neurons more or less accurately. Depending on the level of abstraction, models range from biologically accurate but computationally expensive ones to more abstract and efficient ones designed for use in large networks or neuromorphic hardware [61].

The historical starting point in neuron modelling is represented by the Hodgkin-Huxley (H-H) model [62]. This model provides a detailed description of the currents that pass through the neuron membrane and the mechanisms underlying the generation and propagation of the action potential. Its accuracy is such that it can reproduce a wide range of behaviours observed in biological neurons, capturing up to approximately 17 different dynamic characteristics [63].

This level of fidelity has made it a milestone in modelling and a benchmark for research in computational neuroscience. However, this descriptive richness comes with high mathematical complexity and significant computational cost. Consequently, although the H-H model is the ideal tool for in-depth studies of individual neurons, it is impractical for large-scale simulations or applications that require efficiency and low latency [63].

Over the decades, this limitation has led to the development of simplified models, obtained through H-H approximations. These models represent a compromise

between biological fidelity and computational simplicity, offering researchers more versatile tools with which to work [63].

An important aspect is the choice of a model based on the intended use. When the focus is on studying electrophysiological mechanisms, biologically detailed models such as FitzHugh-Nagumo [64], Hindmarsh-Rose [65], Morris-Lecar [63] or Wilson polynomial models [66] are used. These models allow complex phenomena such as adaptation, bursting, chaotic oscillations and resonance phenomena to be observed and analysed [63]. However, their descriptive richness translates into a very high computational cost, which makes it difficult to simulate networks composed of thousands or millions of neurons. On the other hand, in large-scale network simulations, it becomes essential to use simpler and computationally lighter models, such as the LIF model and its variants, or the Izhikevich model [67], which represents a good compromise between efficiency and reproducible behaviour. Although these models do not capture all biological phenomena, they allow essential dynamics such as temporal integration, threshold-dependent firing, adaptation and bursting to be reproduced, while keeping simulation times low. Although these two categories of models differ in complexity and realism, they share the goal of making the study of neuron behaviour more accessible, balancing biological accuracy and computational costs [61].

However, the neuron model represents only part of the description; the parameters that compose it are equally important, since the dynamic properties depend on them. The parameters that characterise the model - which may include time constants, activation thresholds, synaptic conductances or adaptation factors - are not simple numerical values, but directly regulate the dynamics of the neuron by defining the type of response to external stimuli. Even the slightest variation in these parameters can cause the same model to exhibit profoundly different activity patterns, ranging from tonic spiking to phasic bursting, from adaptation to frequency resonance [61, 68–71].

In this sense, parameters play a dual role. On the one hand, they are an indispensable analytical tool for understanding and classifying observable behaviours. On the other hand, they become a strategic design element during the development of SNNs. In fact, while the model defines the structure and the basic equations that describe the behaviour of the neuron, it is the parameters that determine the actual dynamics during simulations. Minimal, seemingly negligible variations can produce

macroscopic effects on the activity of the network [67]. In fact, correct functioning depends not only on the choice of an appropriate model, but above all on the precise calibration of its parameters, which act as fine control elements that distinguish a functional network from an unusable one. It is through this synergy between model and parameters that it becomes possible to conduct a comparative study of neural dynamics and orient the design of SNNs towards solutions optimised for real-world scenarios [61, 68–71].

The work analysed addresses the topic of low-level investigation of neuromorphic systems with the aim of understanding the behaviour of neuron models. The problem of classifying the behaviour of neuron models is addressed through an analysis of the parameter space, with the aim of producing maps in which the dynamics of the membrane potential are classified for each combination of parameters. In this way, it becomes possible to visualise immediately and intuitively how small variations in parameters lead to radically different neuronal behaviours, ranging from regular regimes to bursting or silencing phenomena. This approach has been applied to different models, including the Izhikevich model, the LIF model and its variant LIF Loihi 2, a model specifically adopted by Intel Loihi 2 neuromorphic hardware [60].

2.1 Analyzing the behavior of spiking neuron models through parameter space

2.1.1 Neuron models

The advancement of neuromorphic technologies depends heavily on the development of hardware platforms capable of faithfully emulating the behaviour of neurons and synapses. However, these platforms, together with their simulation frameworks, evolve at different rates and often incorporate heterogeneous design choices. Consequently, unless an intermediate representation is introduced to act as a level of abstraction [72], it becomes essential - for effective hardware implementation - to accurately identify neural models and parameters that define their operating point. This operation, in addition to ensuring simulation consistency across different platforms, can at the same time facilitate network learning and optimisation processes [73, 74].

Among the wide range of neuron models found in the literature, the H-H model represents a fundamental reference point in the neuromorphic field [62], but its mathematical complexity and high computational cost make it difficult to apply in hardware. For this reason, simplified models are generally adopted in neuromorphic applications, capable of capturing the two essential dynamics of a neuron: stimulus integration and spike generation. Among these, the LIF model is the most widely used and is implemented in various neuromorphic platforms such as SpiNNaker 2 [75], Dynap-SE [76], Xylo [77], and Loihi 2 [78]. However, LIF is unable to reproduce most of the biological behaviours described by the H-H model. To overcome this limitation, alternative formulations have been developed that aim to preserve a wider range of neuronal dynamics. Among these, Izhikevich model [67] is a particularly interesting solution, having demonstrated reduced computational cost on platforms such as SpiNNaker [79], Loihi 2 [80] and Odin [81].

When we talk about information processing by neurons, we are referring to the generation of action potentials - commonly known as spikes - following an external stimulus. Spikes are produced as a result of rapid depolarisation of the membrane potential, which causes an increase in the flow of ionic charges transmitted to connected neurons. The study of the depolarisation process allows to identify the two modes of neuron excitation: the alternation between resting and spiking states, and the presence of sub-threshold oscillations [61]. The dominant characteristic determined by the model parameters defines the operating point of the neuron. The evolution of the temporal dynamics described by an operating point of a physical system can be described by studying the nonlinear dynamics in phase space, thus allowing the identification of behaviours also known as neuro-computational features as a function of the modes of excitation [61]. As shown in [63], through the analysis of these modes, it is possible to identify up to twenty neuro-computational features, which describe the variety of dynamic regimes observable in neuronal models.

Although the H-H model is capable of reproducing most of these behaviours, the presence of four coupled differential equations and a large number of parameters makes the direct application of the nonlinear dynamics approach difficult, since the phase space is four-dimensional. To facilitate the study of these phenomena, several research groups have proposed simplified models obtained through appropriate approximations. Among the main examples are the FitzHugh-Nagumo [64], Hindmarsh-Rose [65], Morris-Lecar [63] or Wilson polynomial models [66] models, each of which represents a compromise between biological fidelity and analytical-

computational tractability. These simplified models represent a significant step forward in the study and classification of different neuronal behaviours.

To overcome these limitations, Izhikevich proposed an innovative approach based on the concept of topological equivalence [61], with the aim of constructing a reduced model that is dynamically equivalent to the H-H model. Through bifurcation methodologies, he demonstrated that it is possible to reduce the latter to a two-dimensional system of ODEs [61, 67]. This reduction allows for a more mathematically tractable model, while preserving a high degree of biological plausibility and maintaining the ability to reproduce a wide range of neuro-computational features.

Several studies have analysed the dynamics of neuron models by studying nonlinear dynamics using phase space, in which the evolution of the system is described as a function of the model parameters [82, 69–71, 61]. Here, we propose an alternative approach by introducing the Neuronal Phase Map (NePhaM). Through two-dimensional maps constructed within the parameter space, NePhaM provides a compact representation of the behaviours that a neuron model can exhibit in relation to the operating point.

The NePhaM can be used as a tool to identify the parameter configurations necessary to achieve the different behaviours of a neuron model, enabling both task-aware and hardware-aware design of spiking neurons. This approach represents a potential advance in the design of SNN-based encoding phase, which is one of the main challenges in effectively interfacing SNNs with real-valued signals from conventional sensors and hardware.

The most common methods for spike encoding are based on algorithmic techniques that can be divided into six main categories: Rate Coding, Temporal Contrast, Deconvolution-based, Global Referenced, Latency/ISI, and Correlation & Synchrony [83, 84]. As an alternative to these approaches, neuron models can be adopted directly as an encoding mechanism, with the aim of increasing the biological plausibility of the process. It is clear, however, that such a strategy is closely linked to - and intrinsically dependent on - the parameters of the neuron model. This implies the need for careful design and appropriate choice of operating point. The impact that each individual parameter has on the behaviour of the neuron can be visualised directly through NePhaM, thus making it possible to identify the most suitable configurations for encoding based on neural models.

The NePhaM analysis was conducted using three different neuron models: the Izhikevich model, the LIF model, and the bit-accurate implementation of LIF that replicates the behaviour performed on the Loihi 2 platform. During the simulations, a common input term, denoted by I, was applied to ensure a uniform stimulation structure across the different models.

Izhikevich model

The model proposed by Izhikevich [63] is described by Equation 2.1:

$$\begin{aligned} \frac{dV}{dt} &= 0.04V^2 + 5V + 140 - U + I \\ \frac{dU}{dt} &= a(bV - U) \end{aligned} \quad ; \quad \text{if } V \geq 30 \text{ mV, then} \begin{cases} V \leftarrow c \\ U \leftarrow U + d \end{cases} \tag{2.1}$$

in this formulation, V defines the membrane potential of the neuron, U represents the auxiliary variable for the resonator dynamics. Four quantities a, b, c, and d determine the operating point and the type of response to a stimulus. a represents the time scale of the recovery variable U; b characterizes how sensitive U is to the sub-threshold fluctuations of the membrane potential V; c denotes the post-spike reset value of V; d specifies the post-spike reset of U.

Leaky integrate-and-fire model

The LIF model in its simplest representation includes only the membrane potential and the reset rule, as illustrated in Equation 2.2

$$C\frac{dV}{dt} = I - \frac{V - V_l}{R}; \qquad \text{if } V \geq V_{th}, \text{ then } V \leftarrow V_l \tag{2.2}$$

where V is the membrane potential, V_l is the leaky component, C is the membrane capacitance, while R is the membrane resistance.

LIF model on Loihi 2

The Loihi 2 platform utilizes a LIF model that incorporates not only the neuron but also a synaptic contribution, with a slight change in the reset condition in contrast to the Equation 2.2. The resulting model is described in Equation 2.3:

$$C\frac{dV}{dt} = u - \frac{V - V_l}{R}; \qquad \tau_d \frac{du}{dt} = -u + I; \qquad \text{if } V > V_{th}, \text{ then } V \leftarrow 0 \tag{2.3}$$

the parameters have the same meaning as for the LIF, except for two additional contributions: the synaptic current u and its temporal decay constant τ_d.

The Lava neuromorphic framework offers a bit-accurate implementation of the model that runs on the Loihi 2 chip in its hard-coded version, described by Equation 2.3, through the Euler approximation method in Equation 2.4:

$$\begin{aligned} V[t] &= V[t-1](1 - dV) + u[t] + bias \\ u[t] &= u[t-1](1 - du) + I \end{aligned} \tag{2.4}$$

the scaling parameters dv, du and $bias$ are used to represent the parameters C, R, τ_d and V_l, while the reset condition formulation remains unchanged.

2.1.2 Neuronal phase map

NePhaM represents a graphical methodology for visualising the neuro-computational features that a neuron model can exhibit as a function of the parameters that define its operating point. Each point on the map corresponds to a specific behaviour of the neuron model, determined by the parametric coordinates associated with that point.

nD to 2D graphical representation

The dimensionality of the space represented in a phase map is equal to the number of parameters selected as variables, which may vary depending on the model considered. For example, in the case of the LIF model, the parametric space is 4D, defined by the quantities C, R, V_l and the input I, as shown in Equation 2.2. In theory, a four-

dimensional representation would not be an obstacle, as one of the variables could be interpreted as the time axis, allowing the dynamic evolution of the representation to be observed [69]. However, this approach becomes impractical in the presence of more complex models, such as Izhikevich model, in which the dimensionality of the space rises to 5D, determined by the parameters a, b, c, d and the input I, as described in Equation 2.1.

In addition to issues related to the dimensionality of space, it is important to emphasise that characterising the behaviour of a model based solely on the configuration of its parameters is a difficult task. In fact, in order to describe the entire parametric space exhaustively, it would be necessary to characterise each point individually, with a total of $O(p^n)$ points, where p represents the number of values sampled for each parameter and n represents the number of parameters considered as variables. This implies that the total number of points to be analysed grows exponentially as the dimensionality increases, quickly making the problem intractable.

For this reason, it is essential to identify an alternative strategy that allows the parametric space to be accurately represented while maintaining computational tractability. The approach adopted for the construction of a three-dimensional NePhaM is illustrated in Figure 2.1.

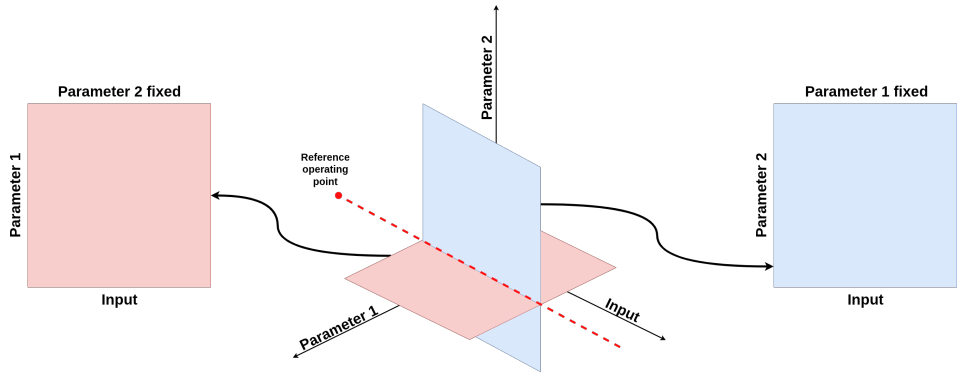

Fig. 2.1 NePhaM generation for a three-dimensional space. The maps are obtained by using the bifurcation parameter as input and treating one model parameter as variable, while all others remain fixed. Setting the ranges of both input and parameters defines the hyperplanes, whose intersection corresponds to the straight line passing through the *reference operating point*.

First, a bifurcation parameter is identified, i.e. a parameter whose value, when changed, causes a transition in the model dynamics and which can be controlled

by the user. In the specific case analysed here, this parameter coincides with the input signal I. Next, the *reference operating point* is defined, consisting of the set of fixed values for the model parameters. This *reference operating point* is useful for defining a control line in the parameter space obtained by varying the bifurcation parameter while keeping the other parameters fixed at the *reference operating point*. This reference axis allows to identify a series of mutually orthogonal hyperplanes in the parameter space, within which it is possible to vary two parameters at a time while keeping the others unchanged at the value of the *reference operating point*.

In these configurations, two-dimensional maps are obtained in which the input parameter is shown on the abscissa axis and the variation of one of the other parameters is shown on the ordinate axis, while the remaining parameters remain fixed. In this way, it is possible to construct a coherent set of maps that describes the behaviour of the model as a function of the main parametric combinations, while preserving readability and computational tractability.

The final graphical representation of NePhaM for a model characterised by n parameters consists of a total of $n-1$ maps. In each of these, the abscissa shows the values of the bifurcation parameter, while the ordinate represents the range of variation of the parameter analysed. The remaining $n-2$ parameters are kept fixed at the values defined by the *reference operating point*.

This approach allows for a functional representation of the model under examination, reducing the complexity of characterisation from $O(p^n)$ to $O((n-1)p^2)$. While not offering an exhaustive description of the entire parametric space, this graphical representation enables intuitive yet meaningful visualisation, providing valuable insights into the model neuro-computational features.

Map production

The NePhaM generation process follows the scheme shown in Figure 2.2a. The first step consists in selecting the neuron model and the parameter variation range. The resolution of the map is determined by the number of values sampled for each parameter: a higher density of points allows for more detailed observations, which is essential for the clear identification of different behavioural regions.

Once the model and its parameters have been defined, the input signal is selected. Different types of stimuli allow different information to be highlighted, such as the

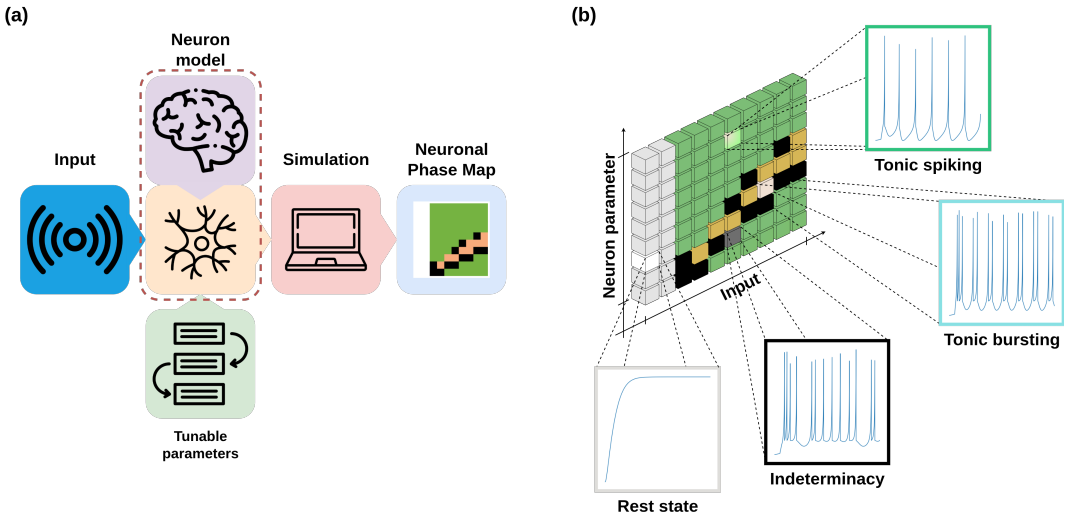

Fig. 2.2 Building blocks of the pipeline for generating a single-parameter NePhaM (a) and an example of the final result (b). In (a), the neuron model is simulated over a fixed time with specific input and parameter ranges, producing the NePhaM through complete exploration of the defined region. In (b), a discretized representation highlighting *rest* state, *tonic bursting*, *tonic spiking*, and indeterminacy conditions.

types of behaviours, the presence of latency or bistability for a given set of parameters. In the specific case of this analysis, a step function with variable intensity was used in order to highlight integrator and resonator behaviours.

Figure 2.2b shows the structure of a single-parameter NePhaM and how each pixel is interpreted. Each behavioural pattern is associated with a distinct colour. The example shows four main cases: absence of spikes, periodic spiking, burst-type behaviour, and a condition in which it is not possible to identify a single dominant behaviour. Each elementary block of the map corresponds to a pixel, obtained following a finite-time simulation of the selected model, performed with the specific set of parameters and stimuli.

The most commonly used finite time numerical simulation methods for integrating differential equations are the Euler method and the Runge-Kutta method. In this analysis, the fourth-order Runge-Kutta method was adopted in order to minimise the numerical error introduced by integration, in accordance with the recommendations reported by Guo *et al.* [68].

Neuro-computational feature classification

As reported in [63], neuron models can exhibit up to 20 different neuro-computational features, which can be divided into subgroups based on the nature of the stimulus and the resulting activity pattern. In this project, we focus exclusively on the four primary characteristics observable in response to a step signal, together with the resting state:

- *rest* state: condition in which the neuron does not generate spikes and the membrane potential remains at equilibrium in the absence of disturbances;
- *tonic spiking*: response characterised by continuous and regular spike generation throughout the duration of the stimulus;
- *phasic spiking*: emission of a single spike at the beginning of the stimulus, followed by inactivity even in the presence of the stimulus itself;
- *tonic bursting*: periodic production of groups of closely spaced spikes, organised into distinct packets, which constitute a burst;
- *phasic bursting*: intermediate condition between *phasic spiking* and *tonic bursting*, characterised by the emission of a single initial burst followed by inactivity.

The classification of these five characteristics, together with the additional case of indeterminacy due to the overlap of different behaviours, was carried out through a statistical analysis based on the distribution of ISIs and the number of spike trains generated in response to the applied stimulus.

2.1.3 Experimental results

To ensure uniform generation of NePhaM maps between the Izhikevich, LIF, and Loihi 2 LIF models, common simulation criteria were defined. Specifically, each parameter was discretised into 1024 intervals to allow for high resolution in the parameter space.

The stimulus used is also discretised with the same level of granularity and consists of a step function with variable amplitude in the interval $[0, 10]$, kept

identical for all simulations. The step function is applied for the entire duration of the simulation, equal to 20 s, with zero intensity 0 in the time interval $[0\ s, 1\ s]$ and with the selected value in the time range defined by the interval $[1\ s, 20\ s]$.

Izhikevich model

In the context of the NePhaM representation, the parameters selected for Izhikevich model are a, b, c and d, according to Equation 2.1. The values of these parameters are chosen in accordance with Izhikevich [67] specifications for the relevant parameter ranges:

- $a \in [0.02, 0.1]$;
- $b \in [0.2, 0.26]$;
- $c \in [-65.0, -50.0]$;
- $d \in [0.05, 8.0]$.

For the *reference operating point*, the following values were adopted: $a = 0.06$, $b = 0.23$, $c = -57.5$, $d = 3.975$. The numerical integration process was implemented using the fourth-order Runge-Kutta algorithm, with a time step of 0.05.

Figure 2.3 shows the resulting NePhaM. The four maps produced represent the six behaviours identified above: *rest* state, *tonic spiking*, *tonic bursting*, *phasic bursting*, *phasic spiking*, as well as behavioural transitions.

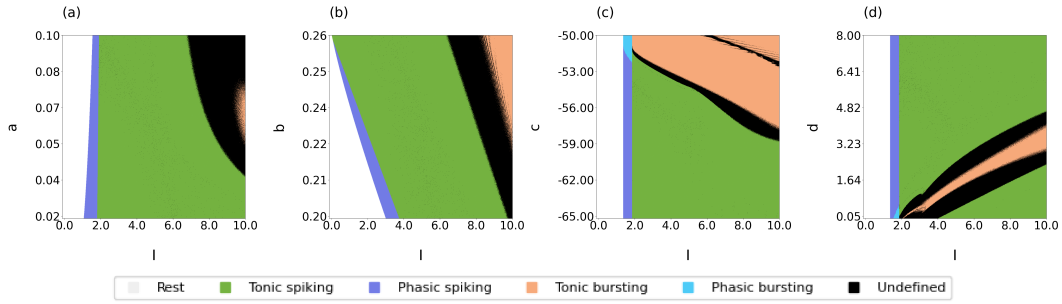

Fig. 2.3 NePhaM generated for the Izhikevich model. In panel (c), the chosen input and parameter values make it possible to observe all the behavioral regimes under consideration.

Each map summarises the set of behaviours that the neuron model can exhibit as the input I and a selected parameter vary, keeping the other parameters constant

at the values set in the *reference operating point*. For example, Figure 2.3a shows the behaviours of the Izhikevich model as I and a vary, with b, c and d kept fixed. A case of particular interest is illustrated in Figure 2.3c, where, as the bifurcation parameter value increases, the neuron can exhibit the entire range of behaviours when the parameter c is around the value -52.

Furthermore, all four maps highlight transition zones from *tonic spiking* to *tonic bursting* behaviour. These areas are characterised by irregular margins and large undefined regions, which indicate the presence of ambiguous behaviour.

Consequently, in the practical application of neuromorphic technologies based on complex models, such as Izhikevich, it becomes crucial to precisely define the neuron operating point. This definition must take into account both the values of the model internal parameters and the type and range of inputs applied, in order to ensure a stable and effective spiking response capable of consistently encoding the information coming from the source.

Leaky integrate-and-fire model

For the construction of the NePhaMs related to the LIF model, described by Equation 2.2, the following parameter variation ranges were considered, in accordance with Dominguez-Morales *et al.* [50]:

- $C \in [0.025, 2.5]$;
- $R \in [8.0, 800.0]$;
- $V_l \in [-75.0, -55.0]$.

The values $C = 1.26$, $R = 404.0$, $V_l = -65.0$ were set as the *reference operating point*. The numerical integration was performed using the fourth-order Runge-Kutta algorithm, with a time step of 0.05.

Figure 2.4 illustrate the NePhaMs obtained for the LIF model. It highlights the presence of only two distinct behaviours: the *rest* state and *tonic spiking*. This result is consistent with the mathematical approximations underlying the model, which was designed to describe only the process of integrating the input stimulus and the subsequent generation of spikes when the threshold of membrane potential is exceeded.

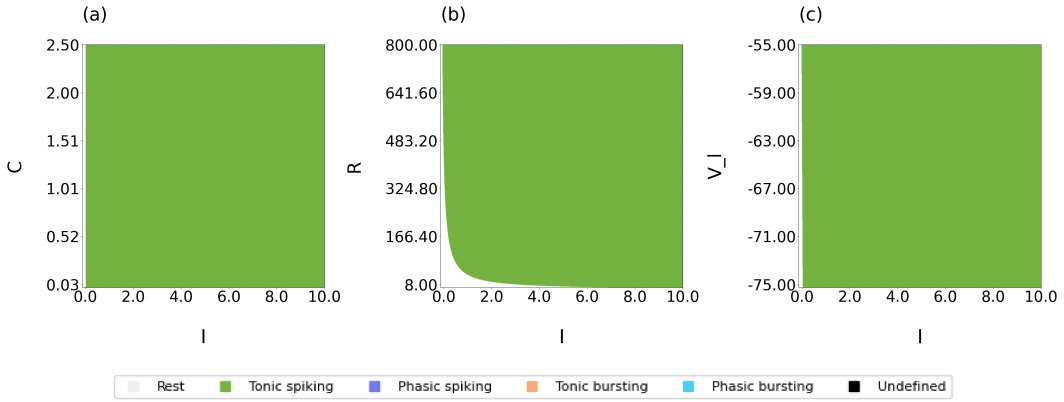

Fig. 2.4 NePhaM generated for the LIF model. Only two behaviors - *rest* state and *tonic spiking* - are observed, which aligns with the approximations used in the model. The LIF model captures just the essential features: integration of the input signal and spike generation above a threshold.

LIF model on Loihi 2

In the bit-accurate LIF model provided by the Lava library for the Loihi 2 chip [85], described by Equation 2.4, the configurable parameters are dv and du, both with values between 0.0 and 1.0, and the parameter $bias$, variable in the interval $[0.0, 50.0]$. The values $dv = 0.5$, $du = 0.5$, and $bias = 25.0$ were assumed as the *reference operating point*.

Figure 2.5 shows the behavioural maps for the LIF model on Loihi 2. The results show that the behaviours faithfully reproduce what was observed in the case of the standard LIF model, showing only the *rest* state and *tonic spiking*.

Although the Loihi 2 chip shares similarities in neuro-computational features with the standard LIF model, there are also significant differences. In particular, only Figures 2.5a and Figures 2.5b represent the membrane potential trend, while Figure 2.5c highlights the synaptic contribution associated with the transmission of the stimulus to the neuron.

The first two maps in Figure 2.5a and Figure 2.5b show a separation of behavioural conditions according to a non-linear trend, an effect attributable to the reorganisation of parameters C, R and V_l of Equation 2.2 into the parameters dv and $bias$ of Equation 2.4. Furthermore, the transition from the *rest* state to the *tonic spiking* state is characterised by a step-like structure. This phenomenon is

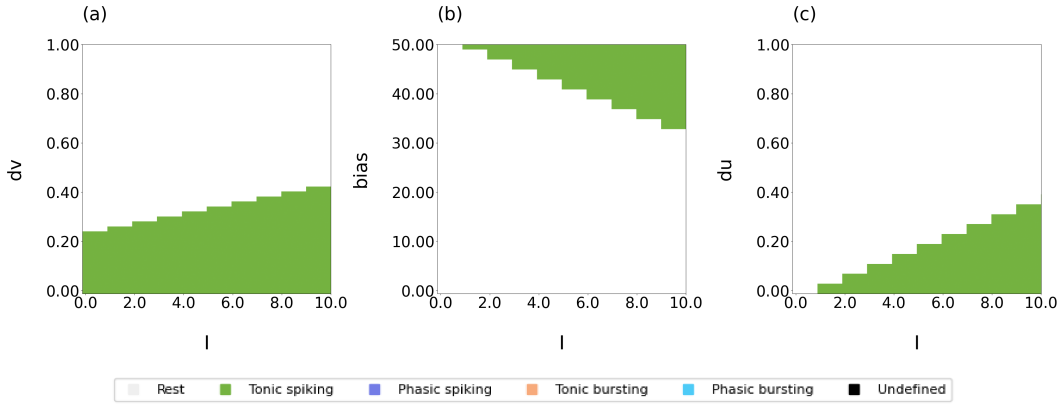

Fig. 2.5 NePhaM obtained for the LIF model of Loihi 2. As in the standard LIF case, only *rest* state and *tonic spiking* are observed. A distinctive step-like structure emerges, resulting from the quantization of parameters during simulation.

attributable to the parameter quantisation process, which introduces a discrete set of values represented with a depth of 12 bits.

2.1.4 Discussion

As highlighted by the three previous examples, NePhaM's potential lies in its ability to represent different neuro-computational features based on neuron model parameters through two-dimensional visualisations. The identification of neuronal behaviours plays an important role in the practical applications of SNNs, both in the spike encoding phase and in the operational phase. The former is crucial from the sensory integration, while the latter concerns the overall activity of the network and the resulting energy consumption.

In order to facilitate the identification of the desired operating range, the added value offered by NePhaM consists in being a graphical tool capable of highlighting information useful for defining the *reference operating point*. The graphical interface, illustrated in Figure 2.6 and Figure 2.9, has the following components: the upper part shows the different map configurations depending on the input signal and the *reference operating point*, the latter being adjustable using the sliders located in the lower right section. The lower left part shows the input signal used as a stimulus to generate the neuron response, together with the membrane potential trend. This interactive application allows to observe in real time how the neuron response varies

depending on both the type of stimulus and the values of the characteristic parameters adjusted using the corresponding sliders.

The dense production of NePhaM maps, which is characterised by high computational costs - as previously shown for the various models analysed - can be carried out following a multi-stage procedure in order to simplify exploration and selection operations. In an initial phase, it is possible to adopt a reduced resolution, based on a coarser subdivision of the intervals of the parameters considered. Subsequently, in order to improve and refine the identification of the operating region of interest and the relative *reference operating point*, the resolution can be increased by narrowing the analysis intervals to allow for a more detailed and accurate study. The graphical tool was applied to Izhikevich model considering the *reference operating point* in the following configurations: *phasic spiking*, *tonic spiking*, *tonic bursting*, and the undefined case. For demonstration purposes, the parameter intervals were divided into 30 sub-intervals, using the same fourth-order Runge-Kutta integration method with a time step of 0.05.

Figure 2.6 shows the *phasic spiking* with configuration parameters $a = 0.06$, $b = 0.205$, $c = -57.5$, $d = 3.975$, and $I = 3.0$, characterized by a membrane potential that produces a single spike.

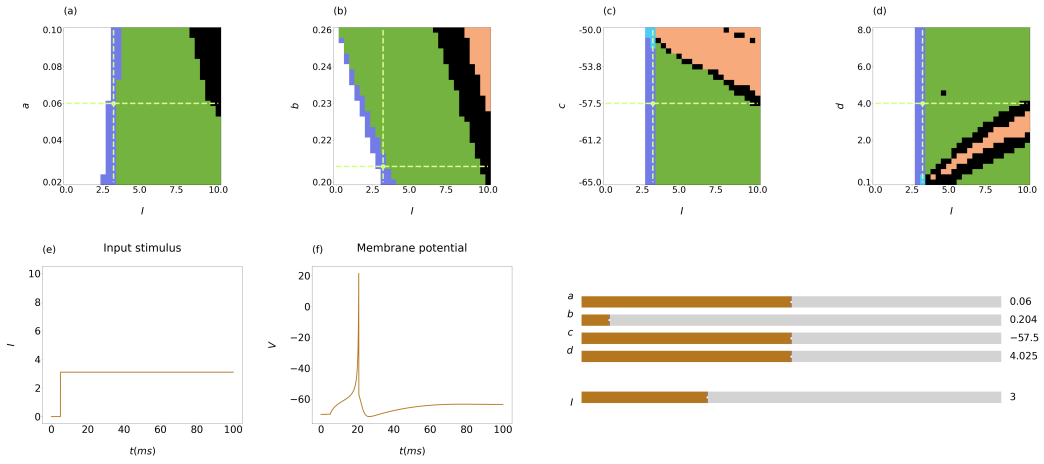

Fig. 2.6 NePhaM for the *reference operating point* in the phasic spiking configuration are reported in (a), (b), (c), and (d). The type of stimulus to which the neuron is subjected is shown in (e), while its membrane potential in output is plotted in (f).

We report an example of *tonic spiking* in Figure 2.7, with parameters set as $a = 0.082$, $b = 0.253$, $c = -61.667$, $d = 3.583$, and $I = 6.667$, where closely spaced

spikes can be identified in the first part of the stimulus, followed by an increase in the time interval for subsequent spikes until a stationary condition is reached.

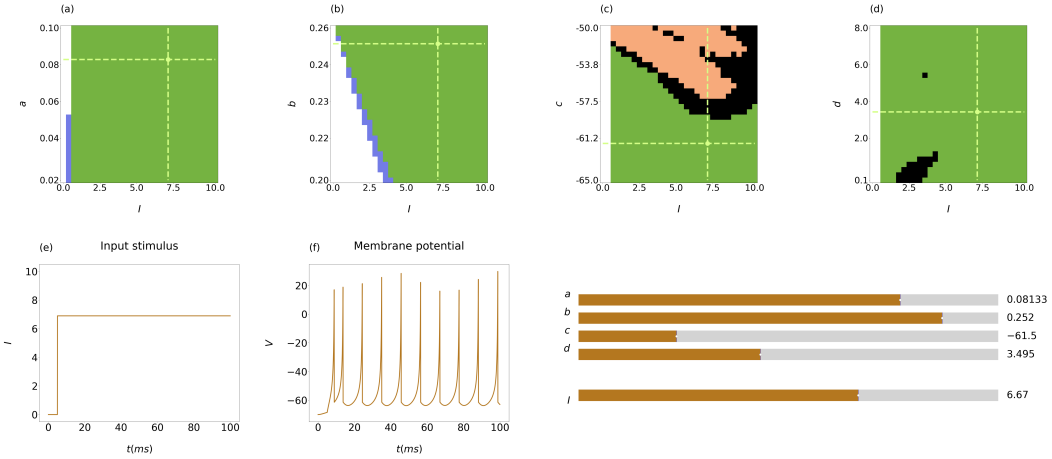

Fig. 2.7 NePhaM for *reference operating point* in the *tonic spiking* configuration.

Figure 2.8 shows a case of *tonic bursting* with configuration parameters $a = 0.082$, $b = 0.253$, $c = -50.0$, $d = 4.467$, and $I = 4.444$. Groups, or bursts, of spikes are produced, separated by a time interval larger than the intra-burst one. Both the ISI and the inter-burst interval show a structure that repeats regularly over time.

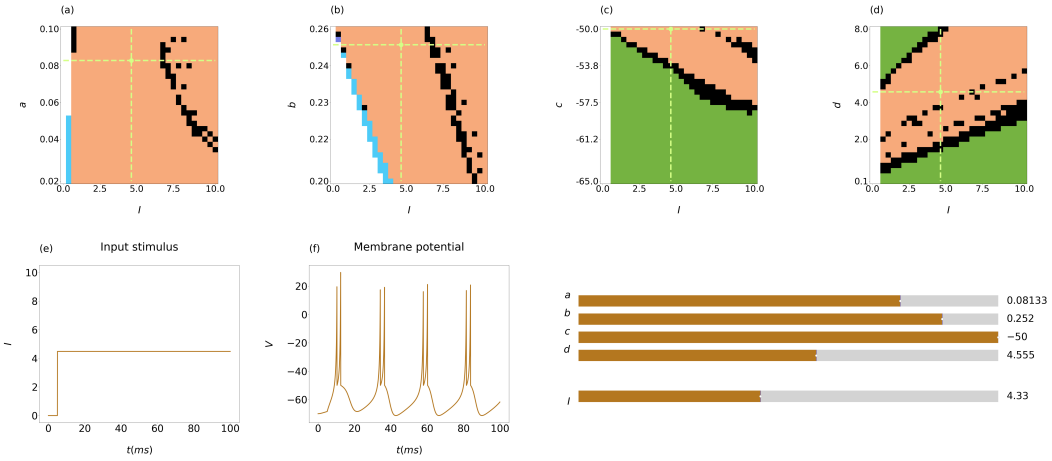

Fig. 2.8 NePhaM for *reference operating point* in the *tonic bursting* configuration.

Lastly, Figure 2.9 shows a condition where defining the observed behaviour unambiguously is not possible, with configuration parameters $a = 0.100$, $b = 0.227$, $c = -51.667$, $d = 4.467$, and $I = 10.0$. The presence of spike production in bursts

of non-homogeneous length and intervals suggests a possible transition from *tonic spiking* to *tonic bursting*.

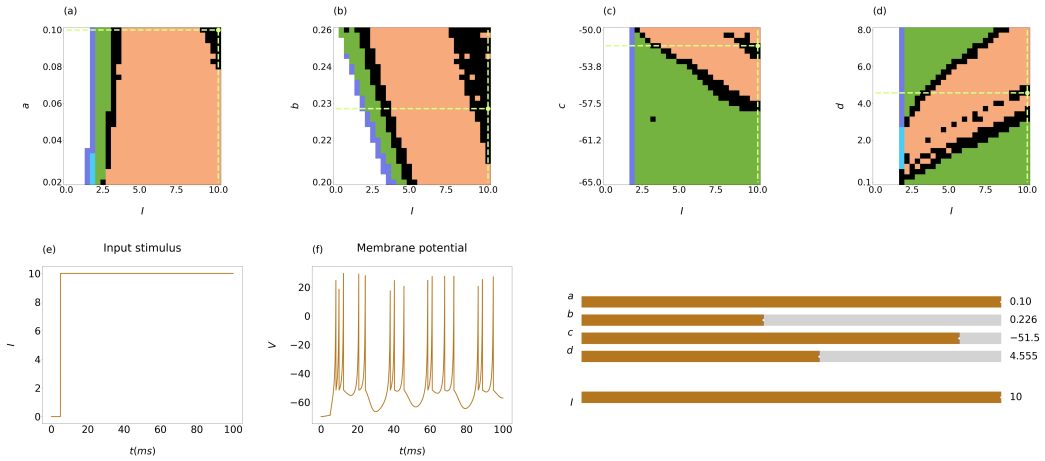

Fig. 2.9 NePhaM for *reference operating point* in the undefined configuration.

The images shown here highlight the four main configurations: *phasic spiking*, *tonic spiking*, *tonic bursting*, and an undefined condition. This analysis emphasises the crucial role of the *reference operating point* in determining the behaviour of the neuron, which ensures a unique response to stimuli. This point must be set in a region sufficiently distant from regions characterised by undefined configurations in the parameter space.

2.2 Summary

This chapter approached the neuromorphic paradigm from its most fundamental level, focusing on the analysis of spiking neuron models as the basic computational units underlying neuromorphic systems. By studying the dynamic and computational characteristics of individual neurons, the theoretical and methodological foundations upon which more complex neuromorphic architectures and applications can be built have been established. The central focus of this study was to ascertain the influence of variations in model parameters on neuronal behaviours such as spiking, bursting and other dynamic regimes.

In this context, the introduction of the NePhaM provides an intuitive and interpretable tool for visualising and analysing the neurocomputational properties

of spiking neuron models across a large parameter space. By offering a compact graphical representation of complex, high-dimensional dynamics, NePhaM bridges the gap between formal mathematical analysis and practical design considerations, supporting both the interpretation of neuronal behaviour and the informed selection of models and parameters for neuromorphic applications. This is particularly relevant in phases such as spike encoding, where the interplay between external, real-valued signals and spiking dynamics assumes a pivotal role.

The knowledge gained at this level constitutes a bottom-up exploration of neuromorphic computation. A solid understanding of single-neuron dynamics is essential to motivate and justify the design of network-level computational mechanisms and strategies. Consequently, the concepts and tools introduced in this chapter provide a solid foundation for the subsequent study of spiking neural networks and their utilisation in solving concrete computational tasks. In such applications, the emergent behaviour of interconnected neurons can be exploited for optimisation, learning, and real-world applications.

Chapter 3

Optimization problem in neuromorphic computing

Optimisation problems have always occupied a prominent place in the field of research due to their importance in many areas, from logistics [86–88] to development of new drugs [89, 90], from warehouse management [91–94] to scheduling [95], from structural optimization [96–98] to resource allocation [99–101]. Optimisation problems are central to both information theory [102] and the creation of intelligent systems in the real world, such as automatic planning in robotics [103–105], the configuration of distributed systems [106, 107], and the training of neural networks [108–110].

An even more fundamental aspect is the distinction of problems based on computational complexity, which allows the most suitable solution methodology to be identified. This classification allows problems to be divided into two broad categories: easy problems defined as Polynomial (P) because they can be solved in polynomial time, and difficult problems known as Nondeterministic Polynomial-time problems (NP-hard) which do not admit efficient deterministic methods of resolution [111].

Thanks to the above classification, two approaches to solving these problems can be identified in the field of optimisation: deterministic methods and heuristic or meta-heuristic methods [112–114]. Each of these families meets different needs in terms of scalability, solution quality and computational cost.

Deterministic methods suitable for class P aim to find the optimal solution to a problem, or to prove its non-existence, through rigorous and deterministic procedures.

Examples of such approaches are linear programming [115], branch and bound [116], cutting planes [117], grid search [118] and Lagrange multipliers [119].

However, these methods prove to be inefficient for high-dimensional problems with a large number of variables, constraints, or complex objective function, as in the case of NP-hard problems. In such contexts, the search for exact solutions can be prohibitive in terms of both time and computational resources. For this class of problems, heuristic or metaheuristic techniques are used [112–114]. These do not guarantee the achievement of the optimal solution, but allow approximate solutions to be obtained in computationally sustainable times. Unlike exact methods, heuristics are based on search strategies in the solution space, inspired by physical, evolutionary or social mechanisms, such as Simulated Annealing (SA) [120], Quantum Simulated Annealing (QSA) [121], genetic algorithms [122], particle swarm optimisation [123], ant colony optimisation [124] and machine learning methods [125].

Alongside classic techniques - deterministic or heuristic - for solving optimisation problems, a new line of research is emerging in the field of neuromorphic computing, which aims to exploit the temporal evolution dynamics of SNNs [41, 40, 49, 126–129].

A distinctive feature of neuromorphic approaches to solving optimisation problems lies in the way the problem is encoded directly into the network structure. This encoding is not just an input, as in traditional ANN models, but constitutes the topology and dynamics of the network itself. In a context where the neural network is called to reflect the structure of the problem, the manual design approach of SNNs proves particularly effective, especially for classes of problems with a well-defined mathematical structure [126].

In this representation, the nodes of the SNN correspond to the variables or elements of the system to be optimised. Each neuron, or population of neurons, represents a possible value, configuration, or attribute of the solution. The synaptic connections between neurons are designed to model constraints, mutual relationships, or violation costs between variables. The excitatory or inhibitory synaptic weight and synaptic delay can be used to define incompatibilities between choices or relationships between states [49].

The network is not just a learning or classification mechanism, but becomes a computational structure that directly incorporates the logic of the problem to be solved. From this, the solution emerges spontaneously as a stable state or limit cycle

in a phase space, following processes of self-organisation or temporal evolution guided by local connections that describe an attractor model. This differs from classical approaches, where the solution is obtained by executing a sequence of instructions [45].

This chapter presents two case studies that aim to explore the application of the neuromorphic paradigm to solving discrete optimisation problems through the use of manually designed SNNs. Although these studies address different issues and refer to distinct theoretical models, they share a common goal: to demonstrate the compatibility and effectiveness of neuromorphic computing in the field of optimisation.

The first case study addresses the active search for a solution to an optimisation problem inspired by the Ising model, widely used in statistical physics to model interacting systems. In this scenario, an SNN is designed to simulate the energy minimisation dynamics of the system. To this purpose, the optimiser was tested in a simulated environment using Python, in order to validate its behaviour in a controlled context. Subsequently, the implementation was transferred to SpiNNaker 2, the new neuromorphic platform. This step made it possible not only to evaluate the performance of the approach in a realistic hardware scenario, but also to explore its potential in terms of scalability and parallelism, paving the way for possible applications in real domains of complex combinatorial optimisation [40].

The second case study, on the other hand, focuses on the dynamic validation of solutions to Constraint Satisfaction Problems (CSPs). In this context, the behaviour of an SNN network is designed to reflect the logical structure of the problem, verifying whether a candidate configuration satisfies the imposed constraints. The approach is explored both through the GPU-enhanced Neuronal Network (GeNN) simulation framework and through execution on SpiNNaker neuromorphic hardware, evaluating the robustness, scalability and interpretability of the network [41].

Overall, the two approaches analysed illustrate two complementary ways of addressing optimisation problems with neuromorphic networks: on the one hand, dynamic search for optimal states, and on the other, topological verification of discrete solutions. Both works contribute to highlighting the potential of the neuromorphic paradigm as a plausible alternative for addressing the optimisation problem from different perspectives.

3.1 Neuromorphic Ising machines

3.1.1 Quadratic unconstrained binary optimization

Quadratic Unconstrained Binary Optimization (QUBO) [130–133] models are very powerful tools for solving a wide range of Combinatorial Optimization Problems (COPs). Their definition is given in Equation 3.1:

$$\begin{aligned} f_{\mathbf{Q}}(\mathbf{x}) &= \mathbf{x}^T \mathbf{Q}\mathbf{x} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} Q_{ij}x_i x_j \end{aligned} \tag{3.1}$$

where:

- $f_{\mathbf{Q}}(\mathbf{x})$ defines the function to be optimised;
- $\mathbf{x} = \{x_1, ..., x_n\}$ represents the set of binary variables for which $x_i \in \{0, 1\}$ defines the state of the system;
- $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is an upper triangular matrix whose elements Q_{ij} define the weight associated with the pair of variables x_i, x_j with indices $i, j \in \{1, ..., n\}$.

QUBO solvers are used in both software simulators and hardware accelerators inspired by classical [134] and quantum phenomena [135]. These accelerators use different forms of annealing to guide the collective dynamics of the optimisation variables towards the global minimum of the COP, where in the case of the physical system corresponds to the fundamental energy states. Relevant examples of such platforms include: quantum annealers based on superconducting qubits [136], optical [137] and digital coherent Ising machines [138], CMOS-based oscillator networks [139, 140], Hopfield networks implemented using memristors [141–143], and SA through digital circuits [144–147].

Analog Ising machines such as quantum ones based on tunnelling processes in theory can guarantee the identification of the optimal solution for problems formulated through QUBO models. However, their scalability is still limited, preventing direct application to large-scale problems [148–151]. In contrast, classical QUBO

solvers such as those based on analogue Ising machine technologies exploit the intrinsic non-linear dynamics of the problem to investigate the solution space and avoid confinement in local minima. Among the various implementations are simulated bifurcation machines [152], stochastic solvers based on magnetic tunnel junctions that use the thermal noise of the hardware to conduct state exploration [153, 154]. Similarly, memristor-based Hopfield networks [141] exploit the intrinsic noise of the device, enabling low energy consumption and reduced solution times, making them particularly suitable for energy-efficient optimisation scenarios. However, when the goal is to generate solutions that systematically approach State-Of-The-Art (SOTA) metrics, or even exceed them by identifying better optimal configurations, the performance of analogue Ising machines is limited by the computational accuracy and dynamic range of the peripheral readout circuits [40].

As shown in Figure 3.1, which reports the three classes of COP problems - low complexity (L), medium complexity (M) and high complexity (H) - for low-complexity problems, most solvers are able to achieve SOTA or the ground state in each run, generating a distribution of solutions that is highly concentrated around SOTA. Conversely, in high-complexity COPs, the distribution of solutions sampled by the various models tends to show high variance. The reasons for this behaviour can be identified in two main causes: the ground state may be significantly distant from SOTA; even the calculation of an approximate solution can be an NP-hard problem, thus requiring exponential times or high calculation precision.

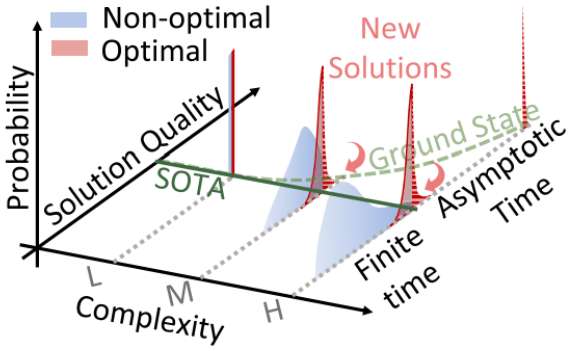

Fig. 3.1 Visualisation of solution distributions produced by optimal and non-optimal Ising machines across different COP complexities: Low (L), Medium (M), and High (H). An ideal QUBO solver yields solutions clustered near the SOTA, with the potential to discover novel configurations closer to the true Ising ground state. The figure is taken from [40].

Compared to analogue architectures, digital Ising machines do not have the same problems related to numerical precision and scalability. Recent advances in neuromorphic hardware have led to the development of platforms capable of simulating SNNs with billions of neurons and trillions of synapses. The implementations of these neuromorphic supercomputers range from solutions based on commercial CPUs and GPUs [51, 155] to customised platforms based on FPGAs, multicore architectures and ASICs [156, 157, 37]. A notable example is the SpiNNaker 2 microchip [38], used in the experiments presented below, which is capable of integrating over 152,000 programmable neurons and more than 152 million synapses.

Although the main motivation behind the development of such platforms has been the emulation and study of neurobiological functions [158, 159], as well as the implementation of energy-efficient artificial intelligence tasks [160, 161, 75, 162], it has been observed that neuromorphic advantages can also emerge in activities that exploit the nonlinear dynamics and intrinsic noise of neuromorphic systems. These approaches exploit the emerging properties associated with energy minimisation [163, 164], phase transition and criticality phenomena [165], as well as chaos [166] and stochastic dynamics [167–169], both at the single neuron unit and at the global network [152, 170, 171].

In recent years, the advantages of neuromorphic platforms in terms of energy efficiency in addressing optimisation problems and simulating stochastic systems have been demonstrated [161, 172]. These specific implementations exploit the high degree of intrinsic parallelism of neuromorphic architectures both to efficiently perform state sampling and to implement Markov processes, both of which are fundamental for solving Ising-type problems [173].

Another promising approach involves the use of neuromorphic architectures for solving Semi-Definite Programming (SDP), a widely used approximation technique for tackling numerous COPs [174]. SDPs have been used to obtain optimal solutions compared to all algorithms in polynomial time [174, 175]. A notable example is the mapping of the Goemans-Williamson (GW) SDP on Intel Loihi for solving MAX-CUT instances [176, 177]. However, the GW algorithm and other SDP variants only provide lower bounds on the quality of the solution, without providing any indication of the distribution of the solutions obtained. Furthermore, the numerical precision required to solve the SDP can be high if the aim is to reach or exceed the SOTA. Consequently, even when the SDP approximation bound is exactly achieved

or exceeded in specific cases and for particular graph instances, this still requires an exhaustive search - exponential or sub-exponential - in the vicinity of the SOTA. It is important to note that, for COPs such as MAX-CUT, even a single cut improvement represents significant progress, as it can only be achieved through the discovery of a new solution.

However, a neuromorphic architecture that is operationally isomorphic to the SA algorithm, equipped with an optimal schedule, should be able to produce high-quality solutions [120, 178, 179]. This behaviour is illustrated in the Figure 3.1, where the ideal or optimal, distribution of solutions is concentrated in the proximity of the SOTA. As highlighted in the same figure, the guarantee of asymptotic optimality also implies that, with extended execution times, the algorithm can generate a solution better than the current SOTA, if the latter does not already coincide with the ground state. However, the inherently slow dynamics of SA are a significant motivation for its implementation and acceleration on dedicated neuromorphic architectures [142, 143].

3.1.2 NeuroSA on Ising model

In its general formulation, the Ising problem [132, 180] consists in minimising the Hamiltonian function $H(\mathbf{s})$ (Equation 3.2) through the identification of the spin state vector $\mathbf{s} = \{s_1, ..., s_n\}$ subjected to an external field or bias vector $b \in \mathbb{R}^D$, the latter can also be used to introduce additional constraints within the formulation of the problem.

$$\min_{s \in \{-1,+1\}} H(s) = \frac{1}{2}\mathbf{s}^T \mathbf{Q}\mathbf{s} + \mathbf{b}^T \mathbf{s} \tag{3.2}$$

In this case, for simplification purposes, attention is focused on a particular case of the Ising model in which the vector $\mathbf{b} = 0$. Under this condition, this model becomes equivalent to the MAX-CUT problem, which is particularly intuitive to visualise and analyse.

In this formulation the graph $G = (V, E)$ constitutes the MAX-CUT problem. The aim is to partition the vertices $V = \{v_1, ..., v_n\}$ into two sets in order to maximise the total number of edges E connecting vertices belonging to different sets. In the case of the Ising formulation applied to MAX-CUT, each vertex v_i is associated with the

spin variable $s_i \in \{-1, +1\}$, while the arcs of the set E are associated with a weight matrix $Q \in \mathbb{R}^{V \times V}$, such that Q_{ij} indicates the weight of the arc connecting vertices i and j. A simple MAX-CUT problem is illustrated in Figure 3.2.

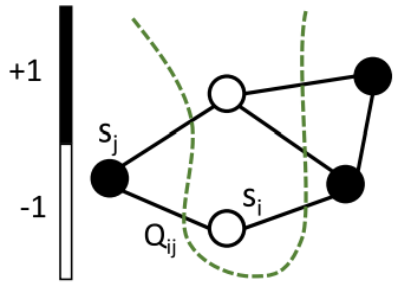

Fig. 3.2 Illustrative example of a graph with edge weights defined by Q_{ij} and spin state determined by s_i. The figure is taken from [40].

Assuming ideal asynchronous operation provided by the neuromorphic platform, at each time instant t only one spin - indicated as p^{th} - can change its state by an amount $s'_{p,t} = -s_{p,t}$. In this scenario, the energy variation of the Hamiltonian function $H(s)$ is equal to:

$$\Delta H(\mathbf{s})_{p,t} = s'_{p,t} \left[\sum_{j=1}^{V} Q_{pj} s_{j,n-1} \right] \tag{3.3}$$

The condition of Equation 3.3, combined with the SA probabilistic acceptance/rejection criterion [120], allows for neuromorphic mapping based on pairs of integrated-and-fire neurons in a coupled ON-OFF configuration, put into mutual competition to ensure that the pair assumes only one possible state, as illustrated in the Figure 3.3.

The pair of ON-OFF neurons p^{th} are differentially connected to the pair related to spin j^{th}, through synaptic weights Q_{pj} and $-Q_{pj}$. The spikes produced by the post-synaptic pair p^{th} differentially encode the variation in the state of spin j^{th}.

To ensure that the spike activity of the SNN is functionally isomorphic to the acceptance/rejection dynamics typical of an SA algorithm, the activation threshold $\mu_{p,t}$ of the p^{th} neuron is modified over time using the tunnelling-based annealing model described by the Fowler-Nordheim (FN) phenomena. To determine the correct cooling schedule, we use the results obtained from the SA algorithm, which define

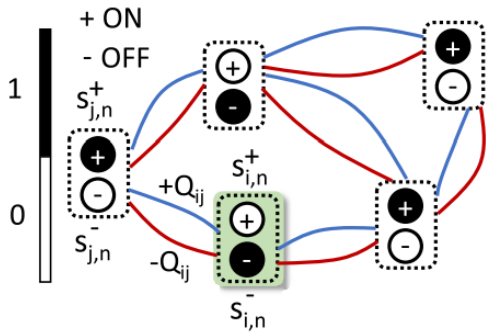

Fig. 3.3 A graph with weights Q_{ij}, decomposed by NeuroSA into pairs of ON-OFF neurons. The figure is taken from [40].

that the Ising model will reach asymptotic convergence to the ground state if the temperature at step t is proportional to $T_t \propto \frac{1}{\log(1+t)}$ [178, 179]. The FN dynamics can produce a variable thresholds by exploiting the model illustrated in the Figure 3.4.

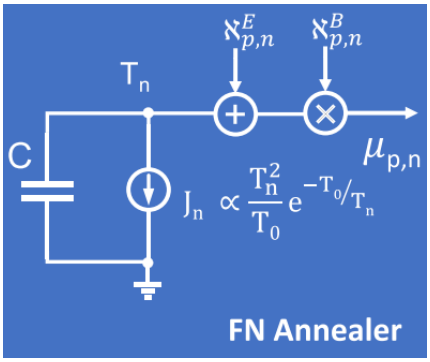

Fig. 3.4 The FN annealer consists of an integrator, and an exponentially distributed noise random variable $\mathcal{N}^E$, and a Bernoulli noise source $\mathcal{N}^B$. The figure is taken from [40].

Based on a current density variable J_t [181] this model is capable to generate the cooling schedule that can modify the threshold $\mu_{p,t}$ by controlling the temperature schedule according to Zhou *et al.* [182] Equation 3.4:

$$T_t = \frac{T_0}{\log(1+t/c)} \tag{3.4}$$

where the parameter t represents the step of the annealing process, while T_0 and c are hyperparameters to be optimised derived from the FN model.

To generate the activation threshold of neurons, the FN dynamic variable is combined with two additional stochastically independent variables. The firing

threshold of the neurons is defined by an exponential distribution $\mathcal{N}^{E}_{p,t} \sim Exp(\lambda)$, and a Bernoulli distribution $\mathcal{N}^{B}_{p,t} \sim \mathcal{B}(q)$. This choice ensures that each neuron has a finite probability of activating or deactivating. A condition equivalent to satisfying the irreducibility and aperiodicity requirements specified in the SA.

The ON-OFF neuron pair, integrated with the FN annealer, constitutes the elementary computational unit of the NeuroSA solver, which can be exploited to solve Ising instances on different neuromorphic computing platforms. Thanks to the functional isomorphism of our implementation, neuromorphic hardware can accelerate execution and asymptotically approach the ground state. The results show that, even in finite-duration executions, the implementation produces solution distributions around the SOTA, without requiring intensive Hyperparameter Optimisation (HPO).

3.1.3 Experimental results

To thoroughly explore the solving capabilities of NeuroSA, we first analyse the solver performance on small-scale MAX-CUT instances for which the exact solution can be easily determined using brute-force search, and then evaluate larger-scale problems on instances for which SOTA is widely documented in the literature [183, 184].

Benchmarking on small-scale graphs

The NeuroSA architecture is initially applied to a MAX-CUT graph consisting of 10 nodes, defined by a random adjacency matrix $\mathbf{Q}$. In this case, the problem has two fundamental degenerate states, as a consequence of the gauge symmetry $s \leftrightarrow -s$. The two degenerate states, identifiable by brute-force search, are defined as Global optimum 1 and Global optimum 2. As shown in Figure 3.5, similarly to what was observed in the dynamics of SA, two main phases can also be identified for NeuroSA: a high-temperature regime and a low-temperature regime, identifiable by the value assumed by the activation threshold μ_t.

To analyse and visualise the different temperature regimes, the aggregate spiking rate of each ON-OFF neuron is calculated using a moving window, as shown in Figure 3.6. The firing dynamics of the entire neuronal population were then projected into a reduced three-dimensional space using Principal Component Analysis (PCA) [185] (Figure 3.6).

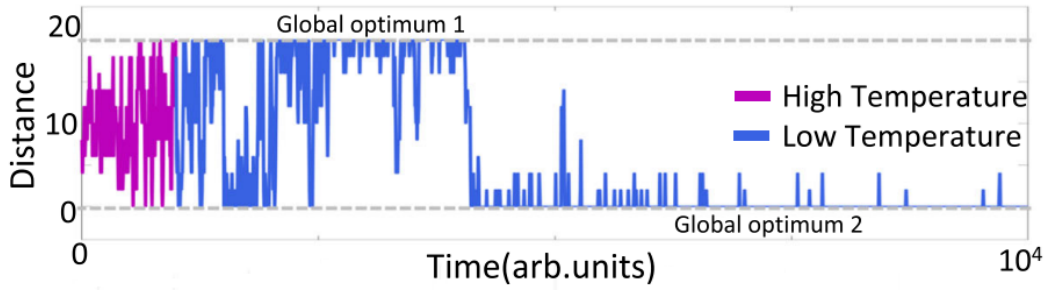

Fig. 3.5 Temporal evolution of the Manhattan distance between NeuroSA-generated solutions and the two known ground states. The dynamic reveals two distinct escape mechanisms in both high-temperature and low-temperature regimes. The figure is taken from [40].

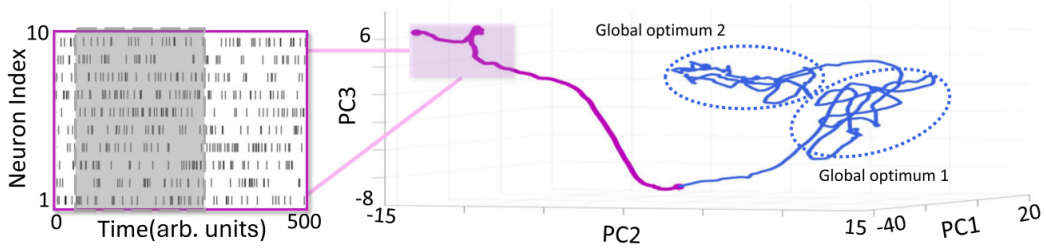

Fig. 3.6 Visualization of NeuroSA trapping and escape dynamics through a PCA-based projection of network spiking activity within a moving time window. The figure is taken from [40].

In the high-temperature regime, the dynamics of the network follow a stochastic evolution behaviour in the form of a random walk, which allows for ergodic exploration of the candidate space. With the progressive reduction in temperature, the exploration of NeuroSA tends to decrease, describing trajectories that orbit near the two global optima. As the temperature decreases further, there is a convergence towards one of the two states corresponding to the optimum, thus ending the exploratory phase [186]. The full dynamics is shown in Figure 3.6.

It is important to emphasise that in the low-temperature regime, the exploration of the state space represents a significant critical issue for SA algorithms. To mitigate this limitation, advanced heuristics have been proposed, including hybrid quantum-classical methods, with the aim of accelerating the escaping process from the local minima. In NeuroSA, FN dynamics intrinsically introduce a finite escape probability even under low-temperature conditions, preserving the ability to explore new regions of the solution space.

Benchmarking on medium-scale graphs

As a second case study, NeuroSA was used to solve a MAX-CUT instance on a graph from the dataset produced by Ye *et al.* [187]. The analysed instance is the G15, a graph formed by a weighted binary planar structure consisting of 800 nodes [187], for which the fundamental state is unknown, but the reported SOTA value is 3050 cuts [152].

For this benchmark, the NeuroSA solver was simulated on a CPU platform. The dynamics of the activation threshold μ_t appear to be limited by the temperature scheduling trend described in Equation 3.4, as shown in Figure 3.7.

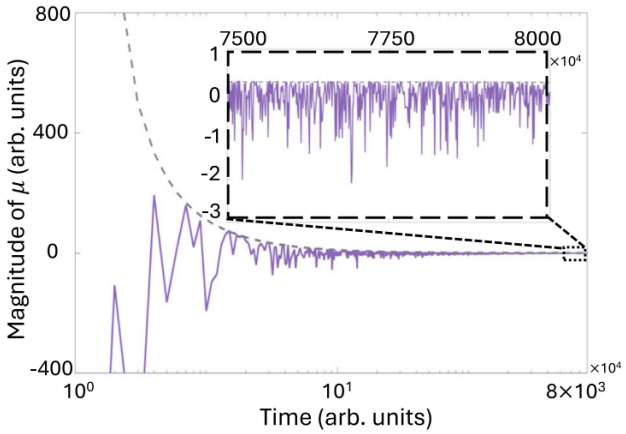

Fig. 3.7 Dynamics of the activation threshold μ_t. The figure is taken from [40].

With the progressive decrease in the threshold envelope, the probability of neuron activation is reduced, as shown by the histogram in Figure 3.8. In the initial stages of convergence, the envelope between the number of active neurons has a wide margin with respect to the upper bound $\#neuron \propto \frac{1}{\log(t)}$. However, this discrepancy progressively decreases as the simulation steps increase, highlighting the influence of the FN-based tunnelling mechanism.

Similar to the results obtained for the small-scale graph shown in the Figure 3.5, here the trajectory described by the state of the cuts as a function of the simulation steps reveals two exploration regimes: the one at high temperature and the second at low temperature. During the high-temperature regime, the exploration follows a path of progressive increase in cuts, indicative of dynamics driven by the Ising energy gradient. Near convergence, with decreasing temperature, the dynamics are

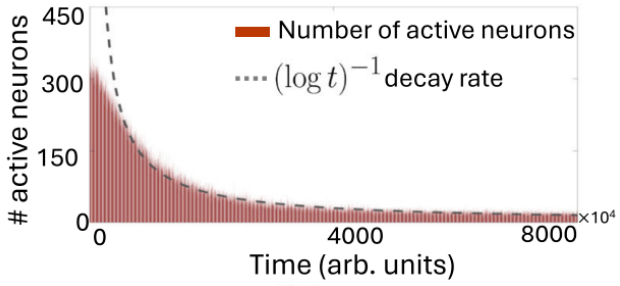

Fig. 3.8 Number of active neurons, showing decay proportional to $\frac{1}{\log t}$ when the contribution of the Bernoulli random variable $\mathcal{N}^B$ is excluded. The figure is taken from [40].

dominated by Brownian motion and sporadic escape mechanisms with no preferred direction, as demonstrate in Figure 3.9.

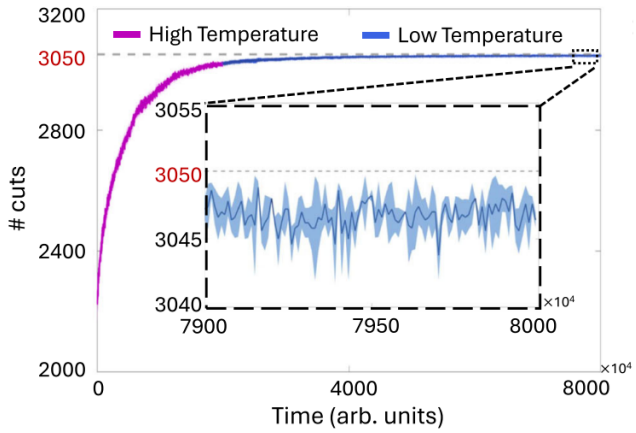

Fig. 3.9 Convergence of the solution quality. The highlights show the fluctuations near 3050 cuts, which represents the current SOTA for this graph. The figure is taken from [40].

Extensive benchmark on a set of graphs

Additional experiments were conducted by subjecting the NeuroSA solver to MAX-CUT instances belonging to the Gset graph family. These experiments were conducted using both a traditional CPU-based architecture and the SpiNNaker 2 neuromorphic platform. The former consists of an Intel® Core Ultra 9 185H processor with a maximum clock frequency of 2.5 GHz, whereas the latter operates at 300 MHz. To make the resolution performance of the different graphs comparable, the number of iterations of NeuroSA running on both platforms is fixed. This methodological

choice demonstrates the robustness and independence of NeuroSA's performance with respect to the topological complexity of the MAX-CUT graphs considered.

To obtain an analysis of the solution capacity of the NeuroSA optimiser, the MAX-CUT instances were divided into classes L, M, and H using the following metrics: graph size; average fan-out per node; graph entropy [188]; network transitivity [189].

The size of the graph, expressed by the number of vertices, allows graphs to be divided according the size of the variable domain. The average fan-out per node quantifies the average number of direct connections as outgoing arcs associated with each node, providing a basic indication of the overall connectivity of the graph and the potential for information diffusion. The entropy of the graph expresses the degree of disorder or randomness by analysing the distribution of connections across all nodes. A high entropy value indicates a more complex or disordered structure, characterised by a less uniform distribution of connections. Transitivity - also known as the global clustering coefficient - measures the tendency of nodes to form cohesive groups, assessing the overall propensity of nodes to create dense interconnected local structures.

Although the average fan-out per node provides a plain measure of connectivity, it does not capture the details related to the specific configuration of these connections, which are instead analysed using graph entropy and transitivity. Graph entropy complements average fan-out by assessing variability in node connectivity, thereby highlighting inequalities or irregularities in connection distribution. In contrast, the global clustering coefficient targeting on the propensity to form local clusters, providing a measure of the structural compactness of the graph and the likelihood of closely interconnected subnetworks being created. Overall, these indicators offer a multidimensional view of the complexity of the graph, describing not only the quantity of connections present, but also their spatial organisation and how they contribute to the formation of aggregate structures and the overall resilience of the network.

The results obtained, depicted in Figure 3.10, describe the normalised distributions with respect to the SOTA of the solutions obtained by NeuroSA on the Gset family. It is important to note that, for the MAX-CUT problems considered, the SOTA solution derives from results previously reported in the literature [152, 183]. However, as can be seen, the solutions produced by NeuroSA consistently reach 99%

of SOTA quality, regardless of the complexity of the graph. Although, as expected, the variance of the distribution increases with increasing complexity, implicitly suggesting a correlation with the intrinsic complexity of the COP problem itself.

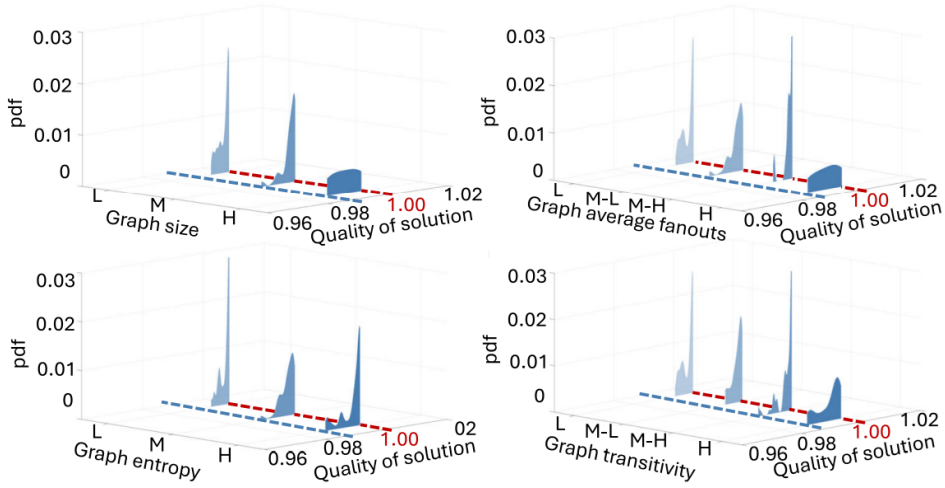

Fig. 3.10 Empirical probability density functions of solutions on Gset benchmarks. The solutions are obtained after 10^8 iterations and fall within the interval $98.9\% \div 100\%$ of the current SOTA., highlighted by red and blue dotted lines. Results are organized in ascending complexity: Low (L); Medium-Low (M-L); Medium (M); Medium-High (M-H); High (H). The figure is taken from [40].

Figure 3.11 also demonstrates the significant algorithmic advantage in implementing NeuroSA on a parallel architecture, such as the SpiNNaker 2 platform. The comparison is made between five instances of NeuroSA running in parallel, each with 10^8 iterations, and a single instance running 5×10^8 iterations in sequence. The results of the five parallel instances are aggregated by calculating their average and plotting the probability distribution of the quality of the solutions obtained for both cases. As evidenced, parallel search produces a distribution that is more concentrated around the SOTA solution than the long-running sequential approach. In essence, the parallel instances of NeuroSA explore different trajectories in the solution space, and the combination of multiple results significantly increases the probability of reaching, within a finite time limit, the region close to the SOTA/ground state optimum.

It is important to note that optimal annealing scheduling guarantees global convergence only in the asymptotic limit of exponential time. Consequently, any improvements over the current solution for reaching the ground state, even using a parallel approach, still depend on a prolonged execution time of a single NeuroSA instance. We therefore extended the execution time to 10^11 iterations to verify whether

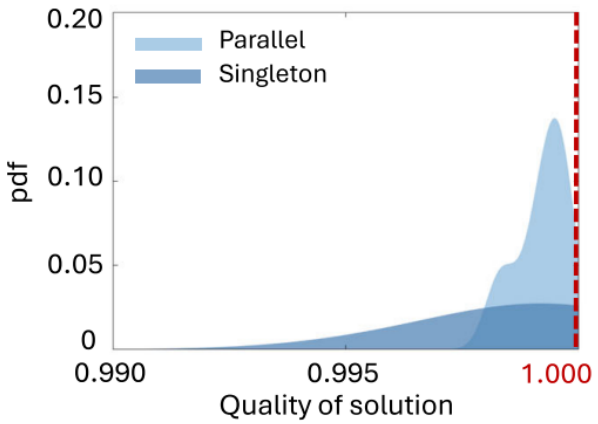

Fig. 3.11 A parallel search comprising five NeuroSA instances generates more consistent results compared to a single search executed for five times the duration of the parallel run. The figure is taken from [40].

NeuroSA could find new solutions for some sets of MAX-CUT problems. However, within a finite execution time - both on CPU and SpiNNaker 2 - NeuroSA failed to discover solutions that exceeded the SOTA on the Gset/MAX-CUT benchmarks.

To appreciate the challenge in finding new solutions, Figure 3.12 offers an analysis of the time required per unit of improvement in the solution on three different Gset benchmarks. The graphs show an increase in the computational cost required to achieve marginal improvements in the quality of the solution. The metric reported is defined as the ratio between the time required to achieve a unit increase and the total execution time. As the difficulty in finding better solutions increases, the execution time grows exponentially or sub-exponentially. This dynamic is illustrated by the extrapolation curve and transition point **A**, which could represent the limit beyond which diminishing returns are encountered. This point could be interpreted as a hardware-agnostic stopping criterion for combinatorial optimisation problems.

The performance advantage of NeuroSA mapping on SpiNNaker 2 compared to a traditional platform using CPU architecture is shown in Figure 3.13, which reports the time-to-solution, energy consumption, and finally the energy required for the solution. The comparison covers three different 800-node MAX-CUT problems belonging to the Gset.

As demonstrated in Figure 3.13 in the right side, the time taken to reach the solution is dependent on the platform. In order to ensure a fair comparison in the case of the CPU platform, a linear extrapolation of the time taken to reach the solution

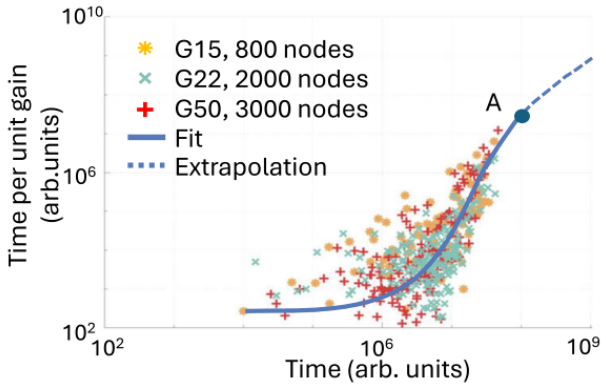

Fig. 3.12 The instantaneous time per unit gain in solution for three Gset benchmarks, approaches an exponential or sub-exponential run-time scaling. The figure is taken from [40].

was performed in order to account for the difference in clock frequencies when compared to the SpiNNaker 2 platform. The linear extrapolation, which is employed as an optimistic lower bound for the execution time of the CPU with a lower clock frequency, disregards the delay caused by communication between the CPU and memory. It is evident from the findings of this study that, even when the absolute execution time is disregarded due to the CPU bottleneck, the current implementation on SpiNNaker2 exhibits superior performance in terms of comparable clock times when compared with the CPU implementation.

As illustrated in Figure 3.13, the central part of the figure presents the energy consumption of NeuroSA on both the CPU and SpiNNaker 2. For the CPU system, the consumption was measured using the following profilers: the HWiNFO®, Intel SOC Watch and Intel Vtune Profiler. It has been mesured that when the NeuroSA algorithm is executed, the CPU system consumes 15.93 W. Of this, 11.63 W is attributable to the CPU core at a normal clock frequency. Performing a linear extrapolation analogous to the preceding one on the energy consumption per CPU core with respect to the SpiNNaker 2 clock frequency, the energy consumption per core would be 1.39 W. Conversely, the SpiNNaker 2 implementation of NeuroSA consumes 561.9 mW. The integrated hardware sensors measure the power consumption breakdown as follows: the PEs (Processing Elements) and the Network on Chip (NoC) consume 455.95 mW, the I/O 85.7 mW, and the remaining 20.26 mW account for unused DRAM leakage and clock generation. The power consumption analysis demonstrates a marked power advantage in utilising the SpiNNaker 2 platform in comparison to the CPU platform.

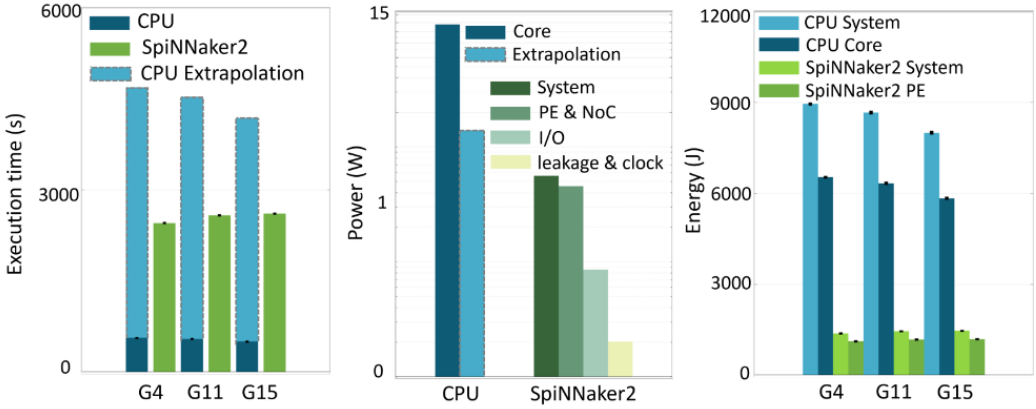

Fig. 3.13 Shows the timing and energy performance of CPU and SpiNNaker2 systems when running identical NeuroSA MAX-CUT workloads with 800 nodes. (left) Real time-to-solution (in seconds) for both platforms, together with extrapolated CPU execution time as a function of the SpiNNaker2 clock frequency. (center) Instantaneous power consumption and its breakdown for the SpiNNaker2 and CPU systems. (right) Energy-to-solution for both platforms; error bars indicate the minimum and maximum energy consumption observed across multiple runs. The figure is taken from [40].

Finally, the energy-to-solution which compares the energy consumption of CPU-based and SpiNNaker 2 systems, with no extrapolations made. As demonstrated in the right side of the Figure 3.13, SpiNNaker 2 exhibits superior performance in terms of energy efficiency when executing the same NeuroSA workloads as the CPU. Notwithstanding certain suboptimalities in the present implementation of NeuroSA in SpiNNaker2, low-power neuromorphic hardware has already demonstrated substantial energy advantages in executing the optimisation algorithm.

Benchmarking on MIS problems

In subsequent experiments, NeuroSA performance in solving Maximum Independent Set (MIS) problems was evaluated. This problem consists of finding the largest subset of vertices in a graph such that no pair of vertices within the same set is connected by an arc. The solutions to the MIS problem are subject to restrictive constraints, making it intrinsically more complex than the MAX-CUT problem, where every configuration is a valid solution.

The distributions of the solutions, is depicted in Figure 3.14, were obtained through simulations on graphs of varying complexity from the NeuroBench benchmark suite [184].

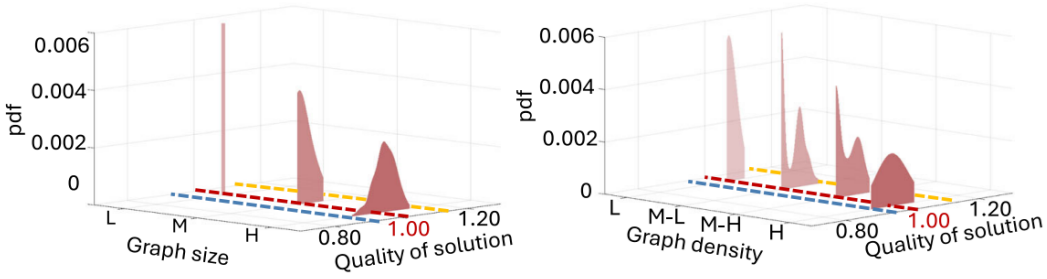

Fig. 3.14 Distribution of results for MIS problems according to complexity metrics: (left) the number of graph vertices Low (L) = 10, 25, 50, Medium (M) = 100, 250, 500, High (H) = 1000, 2500, 5000; (right) graph density L = 0.01, M-L = 0.05, M-H = 0.1, H = 0.25. The figure is taken from [40].

In this case, the classification metrics used to classify the various graphs are the size and density of the graph. As defined above, the size of the graph indicates the number of vertices that make up the graph. The density of the graph represents the number of connections between the vertices, directly influencing the dimensionality of the system state space and, consequently, the effective dimensionality of the energy landscape described by the hamiltonian. An increase in density also modifies the spectrum of the eigenvalues of the Hamiltonian, broadening it and generating a richer set of critical points in the energy landscape of the states.

Similar to the results obtained for MAX-CUT problems, the results for MIS problems also show distributions of solutions concentrated around the SOTA value, obtained without the need for optimisation of hyperbolic parameters. However, unlike the MAX-CUT benchmarks, NeuroSA consistently manages to find solutions that improve the current SOTA in MIS problems. It is also noted that the distributions shown are bounded below by 95% of the SOTA value. Although the variance of the distributions increases with the complexity of the problem, a significant portion of the solutions obtained exceed the current SOTA, thus confirming the effectiveness of the NeuroSA approach.

3.1.4 Discussion

This section proposes a methodology called NeuroSA, which defines the mapping of dynamics derived from optimisers inspired by the physical principles of annealing on SNNs. The isomorphism resulting from the mapping process allows all properties related to annealing processes to be inherited, ensuring asymptotic convergence to the ground state of the hamiltonian.

The computational unit of NeuroSA consists of a pair of LIF ON-OFF neurons, which can be implemented on any standard neuromorphic platform. As a result, NeuroSA can exploit the computational capacity of both current and next-generation neuromorphic platforms.

Within each pair of ON-OFF neurons there is an annealer whose stochastic dynamics are regulated by the FN dynamic transition method. The combination of the LIF neural model with the FN annealer produces overall population activity that faithfully reproduces the accept/reject dynamics typical of the SA algorithm [120]. This dynamic is encoded directly by the spike neurons, which transmit their activity to all connected neurons via the synaptic connections described by the adjacency matrix $\mathbf{Q}$.

Most large-scale neuromorphic platforms use event routing mechanisms, such as Address Event Routing, to transmit spikes across the network, which introduces some latency. As a result, when the spike frequency is high, event packets may not be routed correctly or may be lost. However, given the stochastic nature of the NeuroSA algorithm, such artefacts or errors are tolerable in the both two phases of the convergence process. In the initial phase, the optimisation process follows the steepest gradient descent and updates the neural states in parallel in a manner similar to SDP. The expected value of the solution obtained in this phase can be described by limits similar to those of GW [180]. Subsequently, in the second phase, the search proceeds with an asymptotic approximation method to the fundamental state while exploring new and potentially better solutions than SOTA [178, 179].

One of the most interesting features of NeuroSA mapping on SNN models lies in the possibility of easily implementing and scaling the network on existing neuromorphic platforms such as SpiNNaker 2, thanks to the large-scale systems such as the 5 million core supercomputer in Dresden [38]. The algorithmic advantage of mapping NeuroSA onto a parallel architecture, such as a neuromorphic one,

means that multiple instances can be launched simultaneously on SpiNNaker 2 with minimal, provided that local memory overload is avoided.

In addition to the algorithmic advantage derived from the use of neuromorphic hardware, a significant improvement in terms of energy per solution and time per solution has been observed compared to CPU implementations, as shown in Figure 3.13. This is made possible by the presence of local memory close to the PE and on-chip random number generators that facilitate the acceleration of NeuroSA.

Although this improvement has been observed, the main bottleneck, both for NeuroSA and for other neuromorphic architectures that execute random-walk algorithms, is represented by the process of generating independent and identically distributed random variables within each neuron. As reported in the literature, the generation of high-quality random noise involves significant energy consumption, which is why many neuromorphic architectures adopt the intrinsic physical noise of devices as an efficient source of randomness [190]. However, in this work, digital emulation was chosen to ensure better scalability.

In this work, two families of COP problems, namely MAX-CUT and MIS, were selected as reference benchmarks. These are widely studied classes with SOTA results that are well documented in the literature [191]. During the various experiments conducted, NeuroSA was able to consistently find solutions that approached 99% of the SOTA metrics for several MAX-CUT benchmarks, while an improvement in SOTA was observed for the most recent MIS benchmark suite.

The main objective of this work is to present an algorithmic advancement in the development of an asynchronous neuromorphic architecture capable of exploiting FN annealing dynamics. Since the SA algorithm is inherently slow, the advantages of NeuroSA over other COP solvers with polynomial complexity, such as the GW algorithm, are particularly evident in the regime where minimum gradient information is available to guide the optimisation process.

3.2 Efficient solution validation for constraint satisfaction problems

3.2.1 Constraint satisfaction problems

CSPs represent a broad class of discrete optimisation problems, widely used in numerous application areas including: logistics optimisation [86–88], drug design [89, 90], warehouse management [91–94], task scheduling [95], structural optimisation [96–98], and resource allocation [99–101].

Formally a CSP can be defined as a triplet of elements $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$, where:

- $\mathbf{X} = \{X_1, X_2, ..., X_n\}$ is the set of variables of the problem;
- $\mathbf{D} = \{D_1, D_2, ..., D_n\}$ represents the domains of admissible values for each variable;
- $\mathbf{C} = \{C_1, C_2, ..., C_n\}$ is the set of constraints that specify the admissible relationships between the variables.

The resolution process consists of determining the configuration of the elements $\mathbf{X}$ within the domain $\mathbf{D}$, such that the constraints $\mathbf{C}$ are satisfied.

The complexity of a CSP depends directly on the number of variables and the nature of the constraints: as the degrees of freedom of the system increase, the solution space grows exponentially, making a brute force approach to finding the optimal solution impractical. For this reason, numerous methods have been developed over time to solve them, ranging from analytical techniques, such as those based on Lagrange multipliers or linear programming, to more modern strategies based on statistical approaches or machine learning algorithms.

An emerging area of research is the use of SNNs to address CSP problems [126, 49]. These computational models allow the solution space to be explored dynamically through a dynamics of attractors that guides the network towards stable configurations known as fixed points, which correspond to the admissible solutions of the problem. This approach has the potential to significantly reduce computational and energy costs, particularly when performed on neuromorphic hardware.

Jonke *et al.* [126] proposed a methodology for mapping CSPs on SNNs, modelling the energy landscape of the problem through the topological structure of the neural network. In this configuration, the temporal evolution of the network follows the dynamics of an attractor system, in which the stable states of the network - fixed or cyclic - represent the possible solutions to the problem. This approach has proven effective in addressing classic CSPs such as the travelling salesman problem, the 3-SAT problem, graph colouring, Latin squares and Ising spin models.

In the work of Fonseca *et al.* [49], a strategy is presented in which the CSP is translated directly into the structure of the SNN, using neurons and synaptic connections to represent variables and constraints. The network thus designed performs a stochastic search in the configuration space. This method can be applied to address several problems, including graph colouring, the Latin square problem and the Ising model, demonstrating the feasibility of the neuromorphic approach in the field of optimisation problems [47, 46, 48].

One of the main advantages of SNNs in the context of CSPs lies in the possibility of running these models directly on specialised neuromorphic hardware such as SpiNNaker or Intel Loihi. However, early attempts at implementation on neuromorphic hardware have shown some practical limitations. More specifically, three main issues can be highlighted:

- the impossibility of interrupting the simulation once the solution to the problem has been found;
- extraction and validation on external platforms requires data preparation and transfer, slowing down the process;
- reliability issues may arise in the mapping process if appropriate design choices are not made when defining constraints.

This not only increases computational and energy costs, but also introduces delays and complexity into the pipeline.

In response to these limitations, our work proposes a fully spiking pipeline, depicted in Figure 3.15, which integrates solution validation within the SNN, performed directly on neuromorphic hardware, without the need for external components. To this end, we have designed a series of functional blocks that implement logical

control operations on the constraints, namely the *Polisher* block: filters network activity to stabilise its dynamics; *NetChecker* and *If* blocks: verify the correctness of the solution and activate shutdown mechanisms; *Memory* block: stores the final state of the network corresponding to the solution.

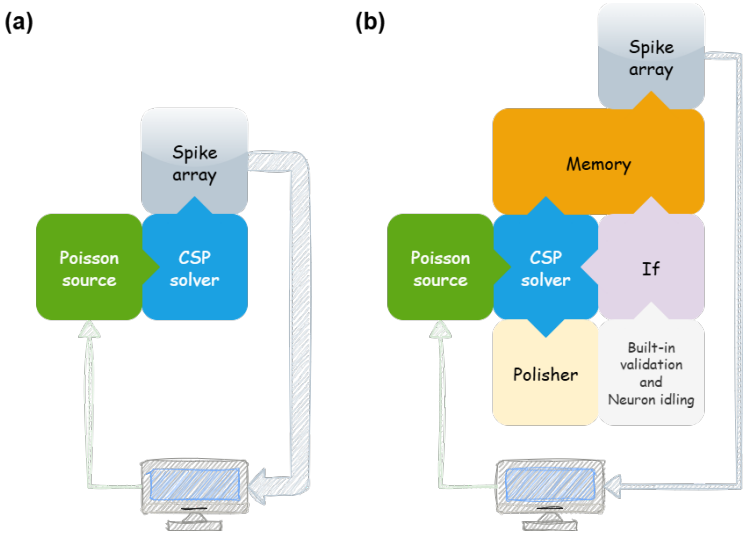

Fig. 3.15 The original (a) and enhanced (b) pipelines are composed of the following building blocks. In the original pipeline (a), the solution proposed in [49] is computed by the CSP solver, with Poisson source and Spike array blocks representing spiking input and output. Output must be transmitted off the neuromorphic platform for validation. The enhanced pipeline (b) integrates a stop condition and on-chip solution validation, allowing fully spike-based computation for both problem solving and validation. This reduces data transfer requirements and eliminates the computational cost of off-chip solution verification.

To demonstrate the effectiveness of this approach, we focused on a variant of the Latin square problem, the Sudoku puzzle, tested on a set of nine Sudoku puzzles belonging to three different difficulty classes: easy, medium, and hard [49, 192].

In addition to its educational and theoretical value, this problem has numerous practical applications, such as in cryptography (Hill algorithm), scheduling, error correction codes, and agricultural research.

3.2.2 Implementation of the fully spike pipeline

Among the examples of CSPs mentioned above, the Latin Square problem involves an $n \times n$ matrix, partially filled with n distinct elements such that, once solved, each symbol appears only once per row and per column.

In the version of Sudoku puzzle, the problem matrix consists of 81 cells arranged in a 9×9 square grid, divided into 9 blocks of 3×3 sub-squares. Each Sudoku cell must be filled with a number between 1 and 9. Each cell represents the fundamental unit of the puzzle, i.e. its basic constituent. The relationships between cells, defined along rows, columns and within sub-squares, together with the initial clues provided by the puzzle, determine the structure and nature of the constraints that characterise this specific case of CSP.

Neuron populations as elemental units

Each Sudoku cell is modelled using 9 distinct populations of Leaky Integrate-and-Fire (LIF) neurons, each corresponding to one of the possible numerical values from 1 to 9. These populations are connected to each other through an internal inhibition mechanism, which creates a winner-take-all configuration [45]. Only one population can be active at a time suppressing the activity of the others, in order to ensure that each cell takes on only one value at a time.

Between different cells, populations representing the same numerical value are interconnected to implement a lateral inhibition mechanism. This scheme prevents the same value from appearing more than once in the same row, column or sub-squares, thus respecting the constraints of Sudoku. In addition, each neural population receives additional synaptic connections from specific stimulus populations, whose purpose is to keep the values potentially assignable to each cell available. A schematic representation of the connections between populations is illustrated in Figure 3.16.

The specific neural model adopted is the Current-Based (CuBa) LIF, whose dynamics are described by Equation 3.5 for the membrane potential and Equation 3.6 for the synaptic current:

$$\frac{dV_m(t)}{dt} = \frac{I_{syn}(t) + I_{offset}}{C_m} - \frac{V_m(t) - V_{rest}}{\tau_m} \tag{3.5}$$

$$\tau_d \frac{dI_{syn}(t)}{dt} = -I_{syn}(t) + I_{spike}(t - t_s) \tag{3.6}$$

where:

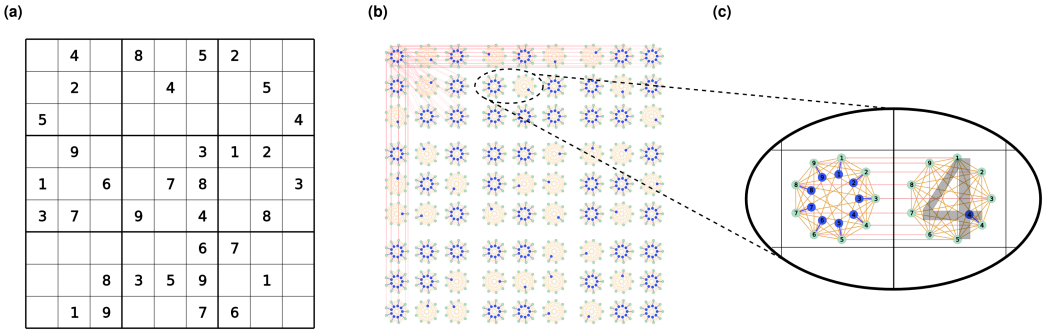

Fig. 3.16 Illustration of a Sudoku puzzle (a) with its corresponding SNN-based CSP solver (b). Internal inhibition connections are depicted in orange, lateral inhibition in red, and stimulus populations with their connections in blue. Initial clues are encoded via single stimulus populations per specified cell. Lateral inhibition is shown for the top-left cell only for clarity. In (c), the nine possible values of each Sudoku cell are represented by nine neuron populations (green), with a winner-take-all configuration, (orange inter-population connections) ensuring a single active population per cell. Stimulus populations and their connections are shown in blue. The right-hand side depicts an example of clue encoding for a single cell with value four. Reworking based on [41].

- $V_m(t)$ is the membrane potential;
- V_{rest} represents the leaky component;
- C_m models the membrane capacitance;
- τ_m is the time decay constant of the membrane potential;
- I_{offset} is the bias current that regulates the internal dynamics of the neuron,
- $I_{spike} = w\delta(t - t_s)$ represents the incoming synaptic activity with weight w at time t_s,
- τ_d is the time decay constant of the synaptic component.

Each time the membrane potential reaches the threshold voltage V_{th}, a spike is generated and the potential is reset to a value V_{reset} for an inactivity time τ_{reset}. The specific values of the neural parameters used are shown in Table 3.1.

The SNN system representing Sudoku evolves stochastically in terms of spike activity. Neural dynamics are maintained slightly above the activation threshold in order to allow exploration of possible configurations. This system is referred to as a CSP solver, as it is responsible for actively searching for the solution.

Neuronal parameter	Value
C_m	0.25 nF
I_{offset}	0.1 (*) nA
V_{rest}	-65.0 mV
V_{th}	-50.0 mV
V_{reset}	-70.0 mV
τ_m	25.0 ms
τ_{refrac}	2.0 ms
τ_d	5.0 ms

(*) For the CSP solver, 0.3 is used to ensure neural activity in the absence of input stimuli.

Table 3.1 Summary of the neuronal parameters utilized for the implementation of the CUBA-LIF model. The table is taken from [41].

The initial state of the network is defined, as in [49], by a random initialisation of synaptic weights, distributed uniformly within a defined range, both for connections to stimulus neurons and for those related to internal and lateral inhibition, as shown in Table 3.2.

	Neurons per population	Synaptic connection weight				
		Stimulus	Internal	Lateral	to CSP solver	to Memory
CSP solver	27 (*)	[1.4, 1.6]	[-0.08, 0.00]	[-0.08, 0.00]	/	/
Polisher	1	1.0	-1.0	/	/	/
NetChecker	10	1.00 0.15 (check pop.)	/	-1.2	/	/
If	10	11.00 0.02 (val. pop.)	0.5 (`True`) 0.0 (`False`)	-1.0	-2.0	-0.6
Memory	3	1.0	0.4	-0.3	/	/

(*) The solution for two of the three puzzles belonging to the easy class (#2 and #3 specifically) has been simulated by using 28 neurons per populations in the CSP solver due to an observed gain in performance.

Table 3.2 Summary of parameters used for the different components of the full model in GeNN. The table is taken from [41].

Starting from this initial configuration, the network evolves over time. At each simulation step, the most active population is determined for each cell, corresponding to the assigned value, thanks to the winner-take-all dynamic. Figure 3.16a illustrates the generic case of an empty cell, in which all populations are connected to the stimuli, thus representing the initial state of uncertainty. In the case of cells belonging to the initial clues of Sudoku, it is necessary to preserve their value throughout the evolution

of the network. In this scenario, as shown in Figure 3.16c, only the population corresponding to the assigned value (e.g., the number 4) remains connected to the stimulus, while the others are excluded. This forces the cell to maintain its state unchanged, preventing changes during execution.

Attractor dynamics

The architecture described above allows a Sudoku puzzle to be mapped into a sparsely connected graph, as illustrated in Figure 3.16b, in which the variables and constraints of the problem are encoded through synaptic connections between neural populations.

Due to the complexity of synaptic interactions, this network does not allow for a direct prediction of overall behaviour based on the analysis of individual components. Instead, the evolution of the system must be interpreted as an attractor dynamic, where the attractive fixed points of the network represent stable configurations of the system, and in particular the solution to the puzzle. However, although the puzzle has a single valid solution, the mapping procedure can introduce secondary attractors that are interpretable as locally minima, which do not represent correct configurations but can nevertheless temporarily stabilise the activity of the network, trapping it in invalid states.

This phenomenon reflects the stochastic and high-dimensional nature of the dynamic system implemented by the SNN. The quality of this dynamic is strongly influenced by the size of the neural populations. If the populations are too small, the network tends to exhibit noisy and chaotic behaviour, with a low capacity to converge towards valid solutions. Conversely, excessively large populations lead to general hyperactivation, with an indiscriminate increase in the neuron activity, which can compromise the inhibitory relationships between cells and distort the evolution towards a correct solution.

Constraint stabilisation

One of the observable effects of using large neural populations is the modification of cells defined by initial clues during the dynamic evolution of the network. However, the main cause of this phenomenon does not lie solely in the size of the populations,

but rather in excessive spiking activity. It has been verified that such alterations can be induced by acting on the parameters of the neuron model, such as the activation threshold, the bias current, or by increasing the weight of synaptic connections, all factors that increase the probability of generating spontaneous or undesired spikes.

A particularly interesting behaviour is that, even after the initial constraints have been altered, the network can still converge towards a correct solution, but one that refers to a modified problem, not the one originally defined. In this sense, the network demonstrates a certain adaptive and generative capacity, reinterpreting the problem based on the new initial configuration. However, rather than being considered an advantage, this capacity constitutes a critical issue: the solution obtained does not solve the desired problem, but an alternative, uncontrolled instance.

Among the possible causes of the violation of the initial constraints - namely threshold voltage, bias current, and the weight of synaptic connections - action on synaptic connections has proven to be the most effective method for mitigating the problem.

As shown in Figure 3.16, the lateral inhibition connections represented in red are designed to transmit information relating to initial constraints through bidirectional connections between cells subject to the same constraints, i.e. belonging to the same row, column or sub-squares. However, to prevent that bound cells are altered by inhibitory feedback from unbound cells, a unidirectional inhibition mechanism has been introduced. Populations associated with initial constraints can inhibit empty cells, but do not receive inhibition input. This preserves the integrity of the initial conditions and ensures consistency with the original problem.

Neuron idling and built-in validation

To address the limitations encountered in the CSP solver and improve the solution validation process [49], an auxiliary network consisting of four functional blocks was introduced alongside the main solver, as illustrated in Figure 3.17. This network is tasked with reducing unnecessary neural activity and verifying the correctness of the solution reached by the CSP solver.

The *Polisher* block filters the activity of the main network, keeping only the spikes coming from the most active populations in each cell. It is structured similarly to the CSP solver, but with two fundamental differences: a reduced number of

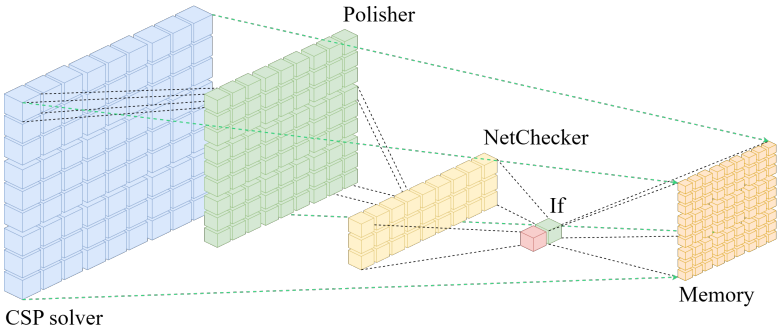

Fig. 3.17 The fully spiking pipeline incorporates four blocks the *Polisher*, *NetChecker*, *If*, and *Memory* to implement neuron idling and built-in validation. Dashed lines highlight the inter-block interaction scheme by showing connections of individual sub-units. The connection between the CSP solver and *Memory* is depicted in green, as one of the two components of the *If* block, indicating that it is active only under specific conditions. Reworking based on [41].

neurons per population, and direct synaptic connections from the corresponding populations of the CSP solver, which act as stimuli.

The *NetChecker* block is responsible for verifying compliance with the constraints between the rows, columns and sub-squares of each population. It consists of 27 units $(3 \times 9) : 3$ for the types of constraints per rows, columns and sub-blocks; 9 for the units of constraints across row, column and sub-square. Each unit contains 10 populations of neurons, 9 control populations and 1 *check population*. The 9 control populations are connected to the corresponding populations in the *Polisher*, one for each digit from 1 to 9. The *check population* is activated only if all 9 digits are represented correctly and without repetition in the monitored constraint. For example, in Figure 3.17, the unit in position (1,4) monitors the fourth row of the Sudoku: its 9 control populations receive input from the respective populations of the *Polisher*, while the *check population* receives input from all the populations in the row and is activated only if the constraint is fully respected.

The *If* block is responsible for: the global constraint validation of the solution; for monitoring the evolution of the system. It consists of two populations: *True*, which is activated only if all the *check population* of the *NetChecker* are active, indicating that all constraints are met; *False*, is activated if at least one of the *check population* is inactive. The two populations compete with each other through a winner-take-all configuration, to ensure that only one of the two is active. In the

event of a positive outcome, namely the *True* population being active, the *If* block interrupts the activity of the CSP solver by inhibiting its populations and activates the *Memory* block. If, on the other hand, *False* is activated, the system continues its evolution in search of a valid solution.

The *Memory* block has a structure similar to that of the CSP solver, but with populations reduced to 3 neurons each. The *Memory* block preserves the final state of the main network, i.e. the solution validated by direct synaptic stimulation from the CSP solver populations.

The entire pipeline supports two key features: the reduction of residual neural activity, stabilising the evolution of the main network once a solution has been found; and the built-in validation of the problem constraints, ensuring that the final configuration is consistent with the initial conditions of the Sudoku puzzle. The topological organisation of the validator is generated from the structure of the CSP problem, mapped directly into an SNN network. Although the implementation of the Neuron idling and built-in validation blocks are designed for Latin Square class problems, it is independent of the specific puzzle and can be scaled to include new types of constraints by adjusting the configuration of the *NetChecker*.

3.2.3 Experimental results

Software simulation on GeNN framework

Our fully spiking architecture was tested on a set of nine Sudoku puzzles belonging to three different difficulty classes: easy, medium, and hard. Specifically, part of the puzzles for the easy and hard classes proposed by Fonseca *et al.* [49] were used, along with an additional selection of puzzles from the collection by Mantere *et al.* [192].

Through software simulation, experiments were conducted using the GeNN environment, running on a hardware platform made up of an Intel 11th Gen i7-11700KF (16) @ 5 GHz processor, 32GB of RAM, NVIDIA RTX A4000 graphics card with 16GB of VRAM and Ubuntu 20.04.5 LTS x86_64 operating system.

To perform a comprehensive comparative analysis, each puzzle was solved using four different configurations, all based on the same CSP solver:

a) basic configuration, as presented by Fonseca *et al.* [49], used as a reference for comparison with our fully spiking approach;

b) version with constraint stabilisation only;

c) version with only neuron idling and built-in validation;

d) complete version, which integrates both (b) and (c).

For each configuration, 300 simulations were performed for each puzzle, for a total of 10,800 experiments.

The introduction of the constraint stabilisation mechanism made it possible to completely correct the problem encountered in the original implementation. It was verified that, with the inclusion of this additional component in the solution process, the initial clues were never altered, unlike what was observed in the solver by Fonseca *et al.* [49], where 171 changes to the constraints were detected. The effectiveness of this strategy, identified as strategy *b*, is documented in detail in both Table 3.3 and Figure 3.18, which clearly show its positive impact on system performance.

Class		**Easy**				**Medium**				**Hard**			
Constraint stabilization		✗		✓		✗		✓		✗		✓	
Neuron idling and built-in validation		✗	✓	✗	✓	✗	✓	✗	✓	✗	✓	✗	✓
Puzzle:	#1	293	289	292	297	5	5	210	202	0	0	121	147
	#2	188	203	216	233	0	1	126	136	0	0	0	0
	#3	248	245	273	282	0	0	7	7	0	0	91	89

Table 3.3 Summary of the solutions obtained from 300 simulations in GeNN for three distinct puzzles within each class. The table is taken from [41].

Specifically for medium-level puzzles, the success rate increased from 0.56% to 38.11%, with an increase of more than an order of magnitude in the number of correct solutions found. For difficult puzzles, a similar improvement was observed, rising from 0.00% to 23.56%. Even for easy puzzles, which already had high performance, the success rate improved, rising from 81.00% to 86.78%. These results show that the constraint stabilisation not only prevents the alteration of initial clues, but also contributes significantly to the success of the solution process, especially in more complex cases.

The adoption of strategy *c* demonstrated the effectiveness of the neuron idling mechanism in a fully spiking pipeline that includes both the puzzle-solving phase

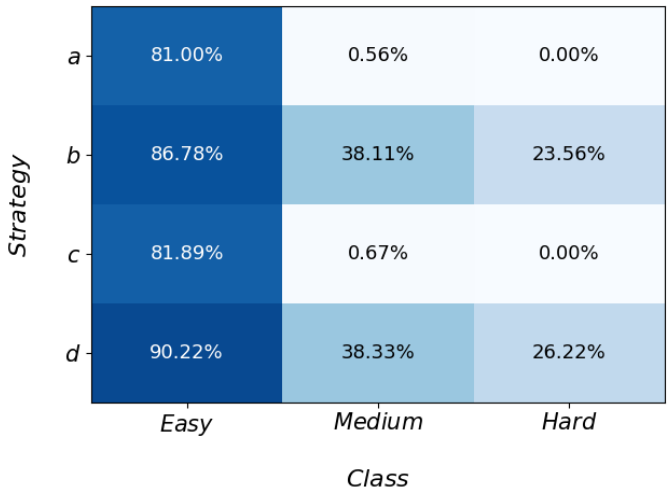

Fig. 3.18 Comparison of the various strategies evaluated through GeNN simulations. The reported values correspond to the success rate, defined as the percentage of solutions obtained for three distinct puzzles per class, each simulated 300 times. Reworking based on [41].

and the solution validation phase. In particular, this mechanism is effective in significantly reducing energy consumption when a solution is found in the early stages of the simulation.

For easy puzzles, a median reduction in the number of spikes of 59.87% was observed compared to strategy *a*, as shown in Figure 3.19. However, this improvement was not observed for medium and difficult puzzles. The main reason for this is the low number of valid solutions found in these two cases, which implies a significantly longer network evolution time. The neuron idling mechanism is only activated after a valid solution has been identified. If no solution is found, the network's spiking activity continues without limitation. Consequently, the increase in the number of spikes observed for the medium and difficult classes in strategy *c* as shown in Figure 3.19 is not due to the idling mechanism itself, but is rather a side effect of the CSP solver's inability to reach solution states, which results in prolonged simulations and greater computational effort.

In terms of effectiveness, strategy *c* led to a modest increase in the number of valid solutions found: the success rate increased by 0.89% for easy puzzles and 0.11% for medium puzzles, as shown in Table 3.3 and Figure 3.18. This improvement, although limited, is attributable exclusively to the integrated validation system, which replaces

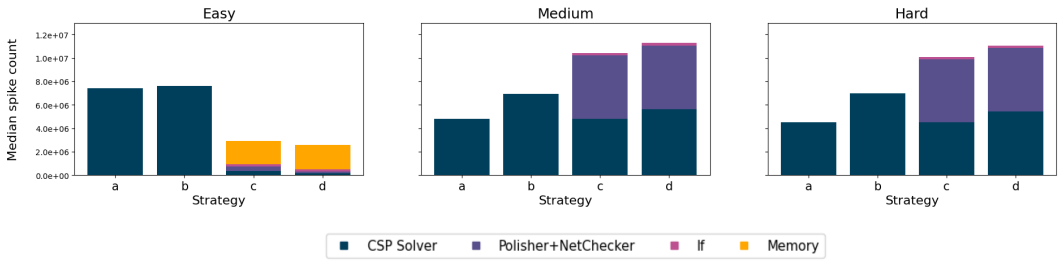

Fig. 3.19 The introduction of neuron idling and built-in validation significantly lowers the median number of spikes produced by the full pipeline for easy-class puzzles. For medium and hard classes, the limited performance of the CSP solver, which frequently utilizes the entire simulation time, conceals this effect. Reworking based on [41].

the bin-based approach adopted in strategy *a*, while maintaining the architecture of the CSP solver unchanged.

The integration of strategies *b* and *c* led to the definition of the complete pipeline, referred to as strategy *d*, in which constraint stabilisation, neuron idling and built-in validation mechanisms are jointly adopted. As illustrated in Figure 3.18, this synergy between the two approaches translates into significant improvements in system performance compared to the original strategy *a*, for all difficulty classes considered. In particular: for easy puzzles, the success rate increases from 81.00% to 90.22%; for medium difficulty puzzles, the increase is from 0.56% to 38.33%; for difficult puzzles, there is an improvement from 0.00% to 26.22%. These results confirm the effectiveness of the combined use of the two mechanisms as an overall strategy for efficient and reliable exploration of the solution space in a fully neural spiking context.

Hardware simulation on SpiNNaker platform

Following the simulations conducted in the GeNN environment, we evaluated the actual performance of the pipeline on the SpiNNaker neuromorphic system. In particular, strategies *a* and *d* were compared, performing 100 simulations for each puzzle belonging to each difficulty class in both cases, for a total of 1800 experiments.

As is well known, the fundamental and distinctive feature of neuromorphic platforms such as SpiNNaker compared to GPU-based platforms lies in the spike-based computing paradigm. Consequently, while simulations on GeNN allow for detailed analysis through direct observation of the activity of various neural populations,

executions on SpiNNaker require explicit collection and transmission of spikes in order to perform any type of subsequent inspection.

To avoid introducing computational overhead and keep the execution as close as possible to the native behaviour of the system, a minimal approach was adopted: spikes were collected and transmitted only at the end of each simulation. This was the most effective compromise to ensure reliable validation without altering the computational dynamics of the network during evolution.

The performance of the fully spiking pipeline on neuromorphic hardware is reported in Table 3.4 and Figure 3.20, which show the number of correct solutions and the associated success rate, respectively.

Class		**Easy**		**Medium**		**Hard**	
Constraint stabilization		✗	✓	✗	✓	✗	✓
Neuron idling and built-in validation		✗	✓	✗	✓	✗	✓
Puzzle:	#1	97	85	0	12	0	14
	#2	69	64	0	0	0	0
	#3	81	87	0	0	0	5

Table 3.4 Summary of the solutions obtained from 100 SpiNNaker experiments for three distinct puzzles within each class. The table is taken from [41].

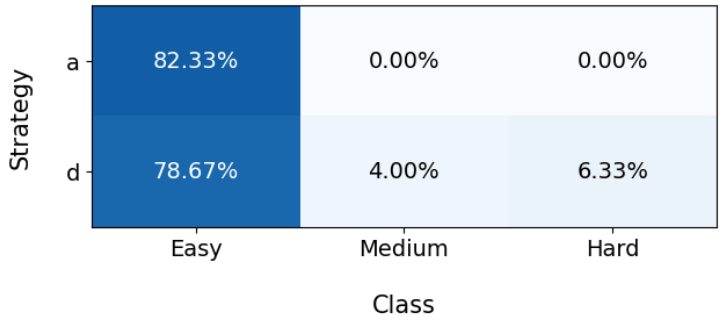

Fig. 3.20 Comparison of the original pipeline by strategy *a* with the enhanced, fully spiking pipeline for strategy *d*. The reported values correspond to success rates obtained from 100 SpiNNaker experiments for three distinct puzzles per class. Reworking based on [41].

In line with the results obtained on GeNN, strategy *d* introduces an improvement in the success rate for the medium and hard classes, although to a lesser extent than observed on GPUs: from 0.00% to 4.00% for the medium class, and from 0.00% to 6.33% for the hard class. On the contrary, for the easy class, there was a slight

reduction in resolution capacity, with a drop in the success rate from 82.33% to 78.67%.

These differences compared to the behaviour observed on GPUs, although undesirable, are consistent with expectations. They can be attributed to the limited numerical precision used by the SpiNNaker platform during the synaptic weight quantization process compared to the floating-point precision of GeNN. A similar phenomenon was documented by Ostrau *et al.* [46], who analysed the performance of three neuromorphic platforms - SpiNNaker, Spikey and BrainScaleS - in comparison with the NEST software framework, highlighting discrepancies in Sudoku solving capabilities due to the different numerical precisions used between the software and hardware environments.

To verify whether this hypothesis could also be extended to our work, we conducted additional simulations on GeNN, adopting two different numerical representations as alternatives to the default 32-bit floating-point format the `float16` and `float64`. For each class, each puzzle was simulated using strategy *d*, performing 300 simulations per puzzle, for a total of 5400 new experiments.

The results, summarised in Figure 3.21, confirm that the success rate is indeed sensitive to the numerical precision used, showing an almost linear decrease as quantisation increases. In particular, the transition from `float64` to `float32` results in an overall reduction of 1.11% (from 52.70% to 51.59%), while the subsequent transition from `float32` to `float16` results in a further decrease of 1.59% (from 51.59% to 50.00%).

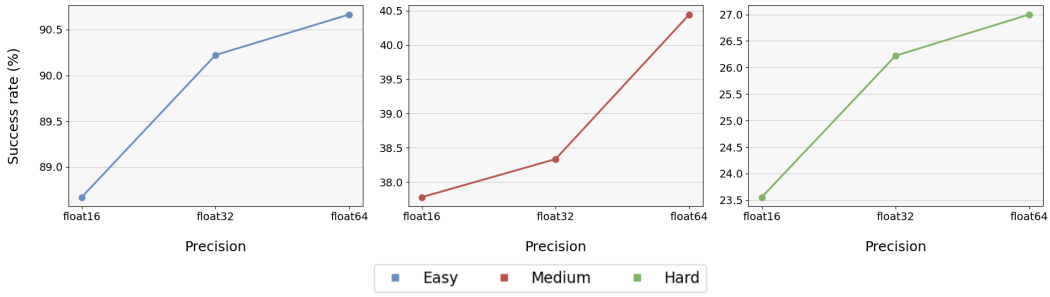

Fig. 3.21 The success rate is shown as a function of the numeric precision employed in GeNN simulations. Across all puzzle classes, reducing the floating-point precision from `float64` to `float16` leads to a decline in the percentage of successfully solved Sudoku puzzles. Reworking based on [41].

A further advantage offered by our fully spiking pipeline in the implementation of strategy *d* on SpiNNaker is highlighted in Figure 3.22, which shows both the number of spikes to be extracted and the associated extraction time, compared to strategy *a*. The reductions observed - ranging from 54.63% to 99.98% for the number of spikes and from 88.56% to 96.41% for the extraction time - clearly show how strategy *d* allows for a significant increase in the overall efficiency of the system. At the end of the simulation, the amount of data to be transmitted from the SpiNNaker platform is drastically reduced, resulting in a decrease in the time and energy consumption associated with this operation.

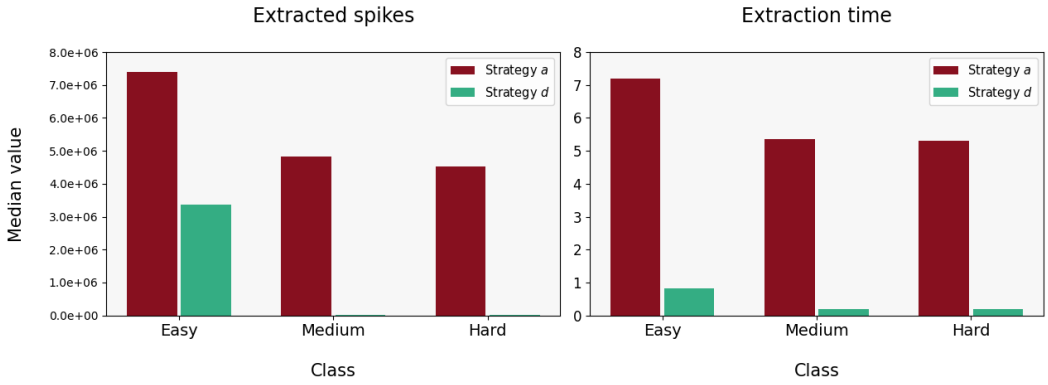

Fig. 3.22 The comparison of strategies *a* and *d* in terms of spike extraction from SpiNNaker, over 100 experiments for three different puzzles per class, the combined implementation of built-in validation and neuron idling markedly reduces both the number of spikes to extract and the associated extraction time. This leads to significant efficiency improvements by lowering the computational cost for processing on external, non-neuromorphic platforms. Reworking based on [41].

3.2.4 Discussion

The attractor network analysed in this study derives from the work of Fonseca *et al.* [49] and stems from the intention to further exploit the computational efficiency offered by the spike-based paradigm. Compared to the original implementation, two substantial changes have been introduced: the integration of a constraint stabilisation mechanism and the use of four functional modules - *Polisher*, *NetChecker*, *If* e *Memory* - in order to create a fully spiking pipeline for solving and validating Sudoku puzzles entirely on neuromorphic hardware. Simulations conducted in the

GeNN environment are used to independently evaluate the impact of each of these changes.

The constraint stabilisation mechanism, introduced through a modification to the lateral inhibition, and analysed through experiments associated with strategy *b*, shows a significant impact on the success rate, as reported in Figure 3.18. The main function of this mechanism is to ensure the integrity of the problem formulation throughout the dynamic evolution of the network.

In the original implementation by Fonseca *et al.* [49], a change in the initial clues was observed during the evolution of the system. This phenomenon was attributed to the inhibition of cells containing the initial values, which should remain unchanged. In particular, due to the inhibitory nature of synaptic connectivity, the state of these cells can be altered if other neuronal populations generate sufficiently high activity. To prevent the modification of the initial clues, it is therefore necessary to prevent the inhibitory action on these cells. This aim is achieved by eliminating the inhibitory synaptic connections directed towards the neuronal populations corresponding to the clues, thus safeguarding the original constraints during the entire evolution of the system. The introduction of the constraint stabilisation mechanism prevents the original formulation of the problem from being altered during the stochastic evolution of the system, thus increasing the ability of the CSP solver to find the correct solution for the proposed Sudoku. As shown in Figure 3.18, this mechanism plays a decisive role in all difficulty classes, and is particularly crucial in solving hard Sudoku puzzles, for which it is an essential element in achieving a valid solution.

The neuron idling and built-in validation mechanisms are implemented through the four blocks *Polisher*, *NetChecker*, *If* and *Memory*. These modules are the distinctive elements, compared to the architecture proposed by Fonseca *et al.* [49], in the definition of our fully spiking pipeline, and their impact is analysed by adopting strategy *c*. As summarised in Figure 3.18 and Figure 3.19, the effect of these mechanisms is closely related to the efficiency of a CSP solver in producing correct solutions. In particular, the analyses clearly show that their contribution is significant for easy puzzles, i.e. those with the highest success rate.

The explanation for this result lies in the specific operating modes and activation conditions of the *If* block. Considering a CSP solver tasked with solving a given Sudoku puzzle, the *Polisher* block operates by continuously receiving the activity generated by the network during its evolution, with the aim of identifying the

populations with the highest activity. This operation can be interpreted as a noise suppression mechanism applied to the spike dynamics of the CSP solver. The clean activity status of all populations is then forwarded to the *NetChecker* block, which verifies that all Sudoku constraints are satisfied, i.e. those relating to rows, columns and sub-squares. This phase constitutes the newly introduced built-in validation mechanism.

When all conditions are met, the *If* block comes into play, namely the *True* population, which enables the current state of the CSP solver to be written to the *Memory* block and the subsequent idling of the solver itself. The writing phase corresponds to copying the last activity state of the CSP solver populations into *Memory*, an operation made possible by an intentional increase in the synaptic delay in the connections between the *If* block and the latter. Figure 3.23 provides a visual representation of the crucial role played by this synaptic delay.

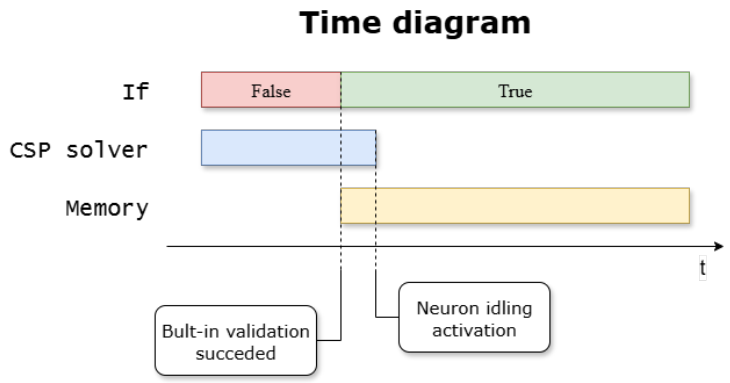

Fig. 3.23 To correctly store the final activity state of the CSP solver upon validation (activation of the *True* population in the *If* block), a temporal overlap is needed between the *If* block operation and the *Memory* block activation. This delay, representing the interval between successful built-in validation and neuron idling, is implemented through synaptic connections linking the *If* block and the CSP solver. Reworking based on [41].

When the *True* population of the *If* block is activated, the *False* population is deactivated, the *Memory* block is activated, and the CSP solver is stopped. The last two operations must be appropriately delayed to ensure the correct acquisition of the final activity status of the CSP solver and its writing within the *Memory* block. Writing can only be successful if both blocks are active simultaneously. It is therefore essential to define a time overlap interval between the solver shutdown phase and the activation of the *Memory*. This overlap is achieved by introducing a synaptic delay in the connections between the *True* population of the *If* block and the CSP solver,

thus ensuring the temporal coexistence necessary for the correct completion of the write operation.

A crucial aspect in the development of our fully spiking pipeline and its implementation on SpiNNaker closely concerns the management of synaptic delay, for which a solution specifically adapted to the hardware characteristics of the platform was necessary. To this end, the required delay was achieved by emulating an axonal structure using a population called *Delay*, consisting of a sequence of smaller, concatenated neural populations, whose properties are summarised in Table 3.5. Similar to what happens in a biological axon, the *Delay* population introduces a delay in the propagation of spikes, in this case between the *If* block and the CSP solver. The axonal length, i.e. the number of concatenated populations, was designed to exceed the maximum limit of 144 time steps imposed on individual synapses by SpiNNaker, a value that proved insufficient to faithfully reproduce the behaviour observed in the simulations conducted in GeNN. In order to effectively integrate these changes with respect to GPU-based simulations in GeNN - where the implementation of synaptic delay could be easily achieved as a single variable - it was necessary to perform a specific optimisation of the size of the *Delay* population and the synaptic weights related to the *If* and *Memory* blocks.

	Neurons per population	**Synaptic connection weight**				
		Stimulus	**Internal**	**Lateral**	**to CSP solver**	**to Memory**
CSP solver	27 (*)	[1.4, 1.6]	[-0.08, 0.00]	[-0.08, 0.00]	/	/
Polisher	10	1.0	-1.0	/	/	/
NetChecker	10	1.00 0.15 (check pop.)	/	-1.2	/	/
If	10	**1.00** **0.11** (val. pop.)	**1.1** (`True`) 0.0 (`False`)	-1.0	/	-0.6
Delay	**10** (×**18**)	**2.5**	**/**	**-1.0**	**-2.0**	**/**
Memory	3	1.0	**0.8**	-0.3	/	/

(*) The solution for two of the three puzzles belonging to the easy class (#2 and #3 specifically) has been simulated by using 28 neurons per populations in the CSP solver due to an observed gain in performance.

Table 3.5 Summary of the parameters utilized for the various components of the entire model implemented on SpiNNaker. Where ranges instead of values are reported, uniform distribution of values within such ranges must be considered. The optimal values changed with respect to Table 3.2 are highlighted in bold. For the *Delay* population, both the number of inner populations (18) and their dimension (10) are reported. The table is taken from [41].

This optimisation phase was conducted using the Neural Network Intelligence (NNI) toolkit, which allowed us to systematically explore the parameter space in order to identify the best performing configuration. The optimal values obtained are

shown in bold in Table 3.5, highlighting the changes introduced with respect to the configuration previously illustrated in Table 3.2.

The importance of the neuron idling mechanism emerges particularly clearly from the comparison between strategy *a* and strategy *d* on the SpiNNaker platform. As shown in Figure 3.22, the adoption of the entirely spiking pipeline results in significant gains - particularly for the medium and hard classes - both in terms of the number of spikes extracted and the extraction time. Both of these quantities are closely related to the activity of the CSP solver through the state stored in the *Memory* block. The number of spikes extracted corresponds to the total number of spikes generated by *Memory*, while the extraction time represents the duration of the internal operations performed by SpiNNaker to collect and package these spikes.

The built-in validation introduced in our fully spiking pipeline eliminates the need for the binning procedure adopted by Fonseca *et al.* [49] for solution validation. Substituting this approach with the *NetChecker* module allows us to completely avoid the computational load associated with analysing spikes extracted to external hardware.

This change produces an increase in overall efficiency in two main ways: a significant reduction in data transmission from the SpiNNaker platform, as it is no longer necessary to export the entire spiking activity of the solver; and zero computational cost for validating solutions on external hardware, as this operation is performed internally, directly during execution on the neuromorphic platform.

The construction of the CSP solver depends strictly on both the specific configuration of the puzzle in question and the synaptic connections defined by probability distributions. In this approach, each puzzle instance requires a customised mapping of the solver populations. In contrast, our advanced pipeline adopts a process of topological mapping of constraints and their validation, allowing the modules responsible for neuron idling and built-in validation to have a unique definition of both synaptic connections and their weights. This enables a validation process that is independent of the specific puzzle.

3.3 Summary

This section explores the use of neuromorphic computing as a computational substrate for solving optimization and constraint satisfaction problems, moving from analysing the dynamics of individual neurons to designing structured spiking neural networks capable of exhibiting global computational behaviour. In the subsequent chapter, the study of spiking neuron models was expanded upon, with the emphasis being transferred to the utilisation of emergent network dynamics for the purpose of addressing complex, NP-hard problems in a distributed and energy-efficient manner.

The proposed approaches demonstrate how classic optimization paradigms, such as simulated annealing and constraint satisfaction, can be reinterpreted and implemented within fully spiking architectures. Specifically, the NeuroSA framework demonstrates the feasibility of mapping stochastic escape mechanisms and annealing processes onto integrate-and-fire neural networks through adaptive threshold dynamics. This enables asymptotic convergence to near-optimal or optimal solutions without the necessity of problem-specific hyperparameter tuning. In a similar manner, the constraint satisfaction pipeline developed for the purpose of solving Sudoku problems demonstrates how control logic, validation, and state storage can be directly integrated into spiking networks. This integration leverages temporal dynamics and event-driven computation, as opposed to relying on external supervision mechanisms.

Beyond algorithmic performance, the practical implications of neuromorphic optimization systems when implemented on real hardware platforms are also highlighted. The implementation and validation of such models on SpiNNaker-based architectures demonstrates not only the feasibility of mapping these models to neuromorphic accelerators, but also the benefits in terms of reduced communication overhead, efficient spike management, and scalability. These aspects underscore a salient benefit of neuromorphic computing: the capacity to co-design algorithms and architectures in which computation, control, and validation are inherently integrated.

Chapter 4

Machine learning approach in neuromorphic computing

The development of ANN-based machine learning approaches stems from the need to overcome the intrinsic limitations of classical programming paradigm represented by deterministic or heuristic algorithms in tackling complex tasks [193]. Traditional methods, while highly effective in contexts with well-defined rules and reduced solution spaces, tend to fail or become impractical in scenarios characterised by:

- *high-dimensional data*: when the number of variables and relationships grows exponentially, finding an optimal solution becomes computationally prohibitive.
- *complex and non-linear structures*: many real-world applications involve interactions and constraints that cannot be modelled using simple analytical functions.
- *incomplete or noisy data*: deterministic methods often require clean and complete data, while practical applications operate under conditions of uncertainty or in the presence of noise;
- *absence of an explicit formulation*: in several cases, there is no direct mathematical description of the problem, making an explicit algorithmic approach impossible.

ANNs allow these limitations to be overcome through a data-driven learning approach. In recent years, this approach has found particularly fertile ground thanks to the ever-increasing global availability of large and high-quality datasets [194–196]. Access to rich and diverse datasets has drastically reduced the need to manually model every aspect of the problem, allowing learning algorithms to infer hidden relationships and patterns autonomously. This transition from classical methods to ANNs, enhanced by the availability of data, has made it possible to tackle previously inaccessible problems such as visual recognition [197–199], natural language understanding [200, 201], predictive analysis in complex systems [202–204], and robot control in dynamic [205–208].

Although ANNs have provided and continue to provide solutions to increasingly complex tasks, their use in edge computing systems [209–211], the Internet of Things (IoT) [212–214] and autonomous battery-powered devices can be prohibitive in terms of energy costs and memory requirements. The introduction of SNNs offers a more resource-efficient solution [35], capable of maintaining high performance even in the most demanding contexts, where traditional ANNs encounter practical limitations such as:

- *energy efficiency*: event-driven processing of SNNs allows calculations to be performed only when necessary, reducing energy consumption compared to synchronous and dense ANN calculations;
- *adaptability to hardware constraints*: SNNs are the ideal choice for embedded devices, where memory resources and computing power are limited;
- *scalability in distributed environments*: thanks to asynchronous communication, SNNs can operate efficiently on architectures with variable latencies or on sensor networks.

The introduction of SNNs and the asynchronous computing paradigm redefines the adoption of SNNs over ANNs as not just a simple change in architecture but a new paradigm [35]. It allows us to move from dense to sparse data representation through the use of spike trains. This difference requires that data be encoded in a spike stream consistent with the dynamics of the neural network used. The quality and efficiency of this translation directly influence the performance of the SNN, as much as the architecture and the learning algorithm used.

The following sections will present two case studies that highlight the crucial role of data encoding and the ability of SNN architectures to tackle complex tasks in real-world contexts.

The first study analyses in detail the impact of different spike coding techniques on two types of signals from heterogeneous domains: Free Spoken Digit (FSD), consisting of a set of audio data containing voice recordings of spoken digits, and Smartwatch Activity and Biometrics Dataset from the Wireless Sensor Data Mining (WISDM) Lab, a sensor-type dataset that collects measurements from accelerometers and gyroscopes for HAR applications. Through the use of spiking Convolutional Neural Networks (sCNN), the study compares the performance obtained with different encoding techniques, evaluating how the nature of the temporal signal, spectral components, and available channels influence the choice of the most suitable encoding strategy [42].

The second study focuses on the direct application of an SNN model - based on the Legendre Memory Unit (LMU) - to the HAR problem using the WISDM dataset. In particular, it compares the capacity of the spiking LMU (sLMU) with other architectures from both traditional machine learning field and the neuromorphic field. The comparison is conducted in terms of classification accuracy and energy consumption, with the aim of identifying solutions that maximise efficiency without sacrificing performance [43].

These two case studies provide a comprehensive experimental framework linking coding choices, network architecture, and performance metrics, outlining guidelines for implementing SNNs in real-world applications.

4.1 Spike encoding techniques for IoT time-varying signals

4.1.1 Spike encoding

In recent years, there has been growing interest in the development of neuromorphic models as candidates for the analysis of temporal data related to human activities. Although deep learning techniques have achieved significant results in the classification of time-dependent data, their implementation on hardware platforms with

limited resources encounters critical issues related to the signal pre-processing and the management of short/long-term temporal dependencies, factors that influence the efficiency of the model [215].

In order to be processed by an SNN, input data - whether analogue or digital - must be transformed into a spike stream. Biological research shows that, even in nature, sensory information can be converted into spike signals in multiple ways. Inspired by these mechanisms, various coding schemes have been developed [83]. Among these, rate-based coding [216] is one of the most widely used strategies in SNNs, proving particularly effective in converting ANNs trained in SNNs for classification applications [73, 217, 218]. In contrast, temporal-based coding techniques have attracted renewed interest, with the emergence in recent years of a wide range of applications that natively exploit time-based coding [219–221]. These include bio-inspired olfactory sensors [222], visual sensors such as event-based cameras [223], fully spiking architectures for image classification [224–227], voice authentication systems [228, 229], time series prediction [230], and anomaly detection [231, 232].

Despite the constant growth in the availability of event-based neuromorphic sensors - as demonstrated by the commercialisation of silicon retina cameras by Sony and Prophesee [233] - SNNs are still frequently used to process data from conventional sensors. In such cases, a spike encoding process is therefore necessary to convert the input signals into sparse event-based signals. However, in the context of IoT applications that exploit neuro-inspired methodologies, the absence of dedicated neuromorphic hardware can be compensated for by the use of algorithmic encoding methods. These methods enable the signal to be adapted for processing by SNNs [234].

The coding algorithms for generating spikes can be divided into two broad categories: Rate Coding and Temporal Coding, which differ in the amount of degrees of freedom allowed in the coding process. In Rate Coding, the information of a signal is represented by the number of spikes generated per unit of time; in contrast, Temporal Coding comprises a heterogeneous set of approaches in which information is conveyed mainly by the timing of the spikes [83]. An example of conversion to a spike train is illustrated in Figure 4.1.

Fig. 4.1 Example of spike trains produced by the various encoding techniques, given an arbitrary input signal. The figure is taken from [42].

Rate Coding

Poisson Rate Among the various algorithms belonging to the Rate Coding category [235, 83], this study adopted an approach based on Poisson distribution for the generation of spike trains. In this scheme, the probability of generating $n \in \mathbb{N}$ spikes within a time interval Δt is defined as [236]:

$$P_n(\Delta t) = \frac{(r\Delta t)^n}{n!} e^{-r\Delta t} \tag{4.1}$$

where $r \in \mathbb{R}$ represents the value to be encoded. Spike production is based on the definition of ISI:

$$ISI_i = \frac{-\log(1 - x_i)}{r} \tag{4.2}$$

where the interval between the i^{th} peaks is described as the i^{th} time interval in which the probability of having $n = 0$ peaks is proportional to the uniform random value x_i.

Temporal Coding

As mentioned above, temporal coding includes coding mechanisms in which the information representation strategy is based on multiple characteristics simultaneously [237]. A distinctive feature of this approach is the ability to transmit information through the exact moment of generation, in addition to the possibility of considering the number of peaks per unit of time. Additional exploitable properties include relative spike timing and the time interval between consecutive spikes with ISI. Depending on the temporal feature considered, it is possible to distinguish five main categories of Temporal Coding algorithms: Temporal Contrast, Deconvolution-based, Global Referenced, Latency/ISI, and Correlation & Synchrony [83].

Temporal Contrast Algorithms belonging to this category focus mainly on signal variations over time, producing positive or negative spikes. Since temporal variation is the predominant characteristic encoded, this approach is less suitable for purely spatial data, such as static images. On the contrary, it finds application in domains where information is in the temporal domain, such as audio signals [36], electromyography data [238], speech recognition [235], fault prediction based on mechanical vibrations [239], and robotic Braille reading systems [240].

Threshold-Based Representation (TBR) The TBR can be considered the constitutive model of the Temporal Contrast category [241]. The algorithm encodes information by generating spikes based on the absolute variation of the signal with respect to a pre-set threshold. Operational implementation involves defining a reference value:

$$Threshold = mean(Variation) + \gamma \cdot std(Variation) \tag{4.3}$$

where γ is a tunable parameter that directly affects the amplitude of the noise-reduction band, which in turn affects the $Variation$ values between $-Threshold$ and

$+Threshold$. The timesteps for spike trains are defined by dividing the interval Δt over which the spikes are generated by the length (L) of the input signal. At each timestep, if the absolute value of the *Variation* exceeds the $Threshold$, a spike is emitted with a polarity defined by the signs of both the *Variation* and the $Threshold$.

Moving Window (MW) The principle behind MW is similar to that of TBR, i.e. the use of a threshold as a criterion for spike emission. However, unlike the previous strategy, the threshold is used in conjunction with a value called *Base*, defined as the average of the signal within a fixed-length moving window.

$$\begin{aligned} Threshold &= mean(Variation) \\ Base &= mean(Signal[1:Window]) \end{aligned} \tag{4.4}$$

In contrast to TBR, the spike generation condition is based directly on the instantaneous value of the signal and not on its variation. Specifically, when the signal exceeds a value equal to $Base + Threshold$, a positive spike is emitted, while for values lower than $Base - Threshold$, a negative spike is generated. The use of a moving window for calculating the average introduces greater noise robustness compared to TBR [242].

Step-Forward (SF) The SF, proposed by Kasabov *et al.* [242] as an evolution of the encoding scheme used in the artificial silicon retina [241]. Also in this case as well this method is based on an iteratively updated reference value, with *Base* and $Threshold$ as fundamental parameters:

$$\begin{aligned} Threshold &= mean(Jump)/\gamma \\ Base &= Signal[1] \end{aligned} \tag{4.5}$$

where *Jump* represents the maximum-to-minimum difference of the signal and γ is a calibration parameter. As with TBR and MW, SF produces both positive and negative spikes: the former when the signal exceeds $Base + Threshold$, the latter when it falls below $Base - Threshold$.

Zero-Crossing Step-Forward (ZCSF) A variant of SF is ZCSF, which introduces half-wave rectification behaviour through the condition $Signal > 0$ [243, 244]. In this formulation, the calculation of *Base* is not required: the encoding preserves the definition of $Threshold$, but the spike is emitted only for positive signal values above the threshold. In this way, the ZCSF produces only positive spikes, differing from previous coding schemes.

Deconvolution-based This category of techniques includes the Hough Spiker Algorithm (HSA) [245] and its variants, the Modified-HSA (MHSA) and Bens Spiker Algorithm (BSA) [246]. These encoding methods originate from the inverse problem of reconstructing an analogue signal from a spike train using a Finite Impulse Response (FIR) filter. In this context, by reversing the reconstruction process, it is possible to obtain an analogue-to-spike conversion using the convolution operator in a subtractive procedure [245]. Similar to the ZCSF scheme, these techniques also generate only unipolar spikes.

Hough Spiker Algorithm (HSA) The HSA implements an iterative procedure of progressive subtraction based on the comparison between the value of the analogue signal to be encoded and the result of a predefined convolution. If the signal exceeds this convolved value, the convolution term is subtracted from the signal itself. The operation performed by the algorithm at time step i^{th} is expressed as:

$$Signal[i+j-1] = Signal[i+j-1] - filter[j] \tag{4.6}$$

where $filter$ represents the result of the convolution, and j indicates its sampling indices. In the analysis presented here, the convolution function adopted corresponds to a rectangular window.

Modified Hough Spiker Algorithm (MHSA) The modified version of HSA retains the subtractive approach based on deconvolution, but introduces a threshold value $Threshold$ to regulate spike emission. In this case, the subtraction operation only takes place when the accumulated variable *error* is less than or equal to the set threshold. The value *error* is incremented in the time steps in which the input

signal does not exceed the value resulting from the convolution, according to the relationship:

$$error = error + (filter[j] - Signal[i+j-1]) \tag{4.7}$$

Bens Spiker Algorithm (BSA) Compared to previous techniques, the Bens Spiker Algorithm introduces two cumulative error metrics in addition to the value $Threshold$. These parameters, called $error1$ and $error2$, allow for a more detailed evaluation of the signal deviation from the reference value generated by the convolution.

$$error1 = error1 + abs(Signal[i+j-1] - filter[j]) \tag{4.8}$$

$$error2 = error2 + abs(Signal[i+j-1]) \tag{4.9}$$

In the original formulation of the BSA [246], the condition for applying the subtraction operation defined in Equation 4.6 requires that $error1$ does not exceed the value $error2 - Threshold$. In this work, however, we adopt the variant proposed by Pedro *et al.* [247], in which the condition is reformulated as:

$$error1 \leq error2 \cdot Threshold \tag{4.10}$$

This modification improves the stability of the encoding process, reducing the likelihood of generating unwanted spikes in the presence of short-lived local variations in the signal.

Global Referenced The third category of temporal coding algorithms includes techniques in which the spike generation mechanism is based on a global temporal characteristic of the signal provided as input. One case is Phase Encoding (PHASE), in which information is coded as a function of the temporal difference with referred to an oscillatory reference [248]; another is Time-to-First-Spike (TTFS), which uses the time elapsed since the onset of the stimulus as the main parameter [249, 250]. Similar to deconvolution-based techniques, both PHASE and TTFS produce unipolar spikes.

Phase Encoding (PHASE) The possibility of implementing a coding method based on phase evaluation with respect to an oscillatory reference was introduced by Montemurro *et al.* [251]. In this work, reference is made to the implementation proposed by Kim *et al.* [219], in which the binary representation of the input using fractional β bits is used as the oscillatory reference. Preliminarily, the signal is rectified and normalised in the interval $[0, 1]$ before applying phase coding.

Time-to-First-Spike (TTFS) Rueckauer *et al.* [218] analysed various strategies for implementing Time-to-First-Spike coding, mainly differentiated based on the definition of the threshold for membrane potential. Here the threshold is modelled as an exponentially decreasing function, in accordance with the approach proposed by Park *et al.* [252]:

$$P_m(t) = \theta_0 e^{-t/\tau_m} \tag{4.11}$$

where θ_0 is a constant and τ_m represents the time decay constant of the membrane potential. In our experiments, we used the values $\theta_0 = 1$ and $\tau_m = 0.1$. Compared to other implementations, we also adopted a bitwise strategy similar to that used in Phase Encoding, which allows to obtain a representation of the signal values in binary form based on bin-based interval subdivision [253].

Latency/ISI Neural communication via bursts of spikes - i.e., the transition from emitting a single spike to sending a variable number N of spikes to represent the data - is known to increase the reliability of information transmission. In addition to the number of spikes, the latency between them can also be exploited for encoding [254]. On this basis, the Latency/ISI class of algorithms is defined, of which Burst Encoding is a significant example.

Burst Encoding (BURST) As highlighted by Guo *et al.* [255], Burst Encoding is an effective technique for conveying information by simultaneously exploiting two temporal characteristics of a spike train, i.e. the number of spikes and their ISI. The algorithm uses three main parameters: N_{max}, which indicates the maximum number of spikes per burst; t_{min}, which represents the minimum time interval between consecutive spikes; and t_{max}, which defines the maximum ISI value. The parameter

ratio, determined by a signal normalisation procedure, defines the value to be encoded. These parameters define both the number of spikes per burst and their temporal spacing:

$$SpikeNumber = \lceil ratio \cdot N_{max} \rceil \tag{4.12}$$

$$ISI = \begin{cases} \lceil t_{max} - ratio(t_{max} - t_{min}) \rceil & \text{if } SpikeNumber > 1 \\ t_{max} & \text{otherwise} \end{cases} \tag{4.13}$$

Similar to the two previous classes of algorithms, Burst Encoding also produces unipolar spike trains.

4.1.2 Classification of FSD and WISDM

Figure 4.2 illustrates the steps of the pipeline implemented to conduct the analysis of the encoding capacity of encoding techniques. First, the block called filter bank aims to separate the raw data into frequency channels using a set of filters, allowing for an increase in the number of features available for analysis. Next, the data is converted into the spike domain using encoding algorithms belonging to the rate coding and temporal coding families.

The feature extraction process allows to obtain the sonogram, an image representation created using a time-binning procedure of the signal encoded in the spike domain. This sonogram constitutes the data used for the training phase through transfer learning, which enables us to indirectly train an SNN network using techniques typically employed in ANNs. To evaluate performance in terms of accuracy, the sonogram is then converted back to the spike domain and subjected to sCNN. As a final step to test the resilience of the network, a process of compressing the SNN model was applied by progressively eliminating low-weight synaptic connections in order to reduce its complexity and size.

The adoption of a convolutional architecture is a well-established choice for processing time-varying signals, as it avoids the use of recurrent neural networks, which are characterised by greater structural complexity and computational overhead.

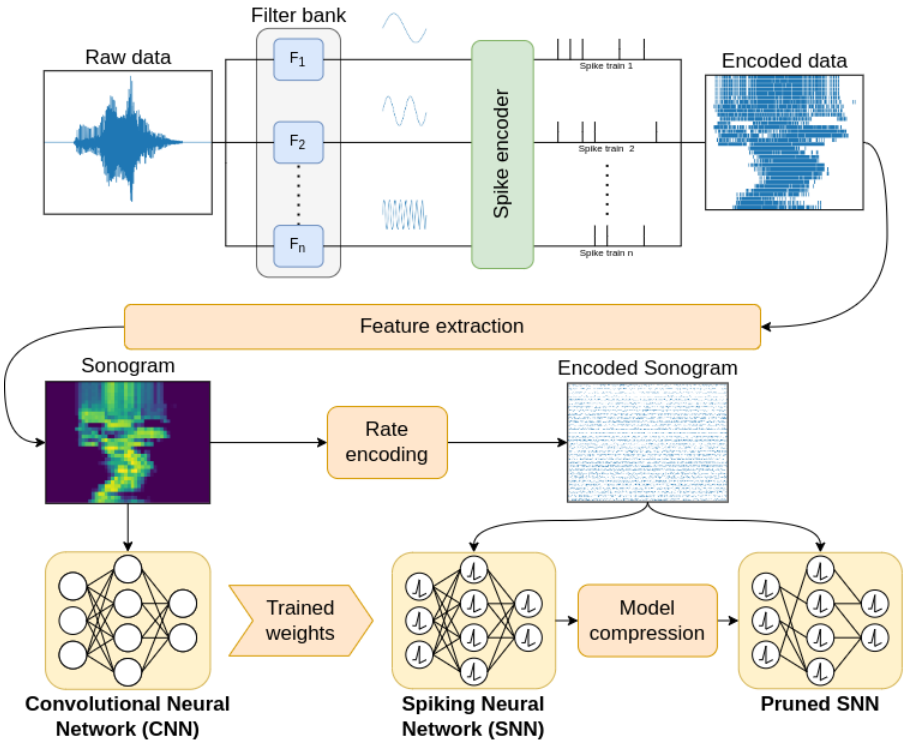

Fig. 4.2 Block diagram of the pipeline for benchmarking encoding strategies. The workflow includes frequency decomposition via a filter bank, spike encoding, feature extraction through sonogram generation, transfer learning with a non-spiking network, and subsequent model compression. The figure is taken from [42].

Dataset

The datasets selected to conduct this analysis are: the FSD dataset consisting of a collection of audio signals and the WISDM commonly used for HAR applications [43].

The FSD Dataset [195] consists of a set of audio signals sampled at a frequency of 8 kHz. In the latest version available, each number pronounced is recorded 50 times by six speakers with English pronunciation but different accents. All samples have been pre-processed by trimming to ensure uniform silence intervals at both the beginning and end of the track. This dataset has been widely used in previous studies focusing on the analysis of spike coding techniques in the neuromorphic context [256].

The WISDM dataset [194, 257], published in 2019 by the Wireless Sensor Data Mining Lab, collects signals acquired via accelerometer and gyroscope, coming from smartphones and smartwatches. The dataset covers 18 different activities of

daily living, ranging from eating to folding clothes, performed by 51 subjects, with a sampling frequency of 20 Hz and a duration of 3 minutes per activity. Unlike previous versions [258], this release ensures high class balance, with a percentage distribution for each activity ranging from 5.3% to 5.8% of the total 15,630,426 samples.

As can be easily seen, the first distinction between the two types of data mainly refers to the content and application context of the data, while the second distinction is related to the neuro-inspired processing phases applied to the signals, and the third distinction is linked to the frequency content of the data. According to this last category, the analysed datasets can be divided based on the frequency of the signal with respect to the human audible spectrum:

- very low frequencies: below 20 Hz;
- low frequencies: between 20 Hz and 500 Hz;
- medium frequencies: between 500 Hz and 2 kHz;
- high frequencies: from 2 kHz to 20 kHz.

Based on these definitions the signals in the FSD dataset fall into the medium frequency category, while those in the WISDM dataset belong to the very low frequency category. This initial categorisation provides an initial criterion for identifying the most suitable encoding technique based on the data analysed. As depicted in Figure 4.3, the spectral density of the datasets is shown, highlighting how the two datasets occupy distinct and non-overlapping portions of the frequency spectrum.

Pre-processing

It is well known that biology and nature are an extremely rich source of inspiration for the development of technologies. In this context, the implementation of pre-processing techniques dedicated to the analysis of time-varying signals as FSD is inspired by the human auditory system.

In particular, a procedure has been developed that replicates the operating principle of the cochlea, located at the end of the auditory system, is characterised by a spiral structure in which nerve cells are distributed along the basilar membrane.

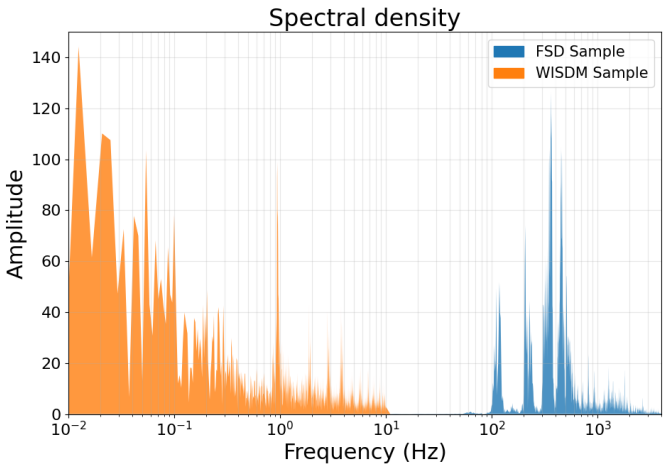

Fig. 4.3 Spectral densities of sample from the FSD dataset and WISDM. The accelerometer-based WISDM signal is concentrated in the very-low frequency range, whereas the FSD sample lies in the mid-frequency range, referred to the typical human-audible range. The figure is taken from [42].

This arrangement gives the cochlea a behaviour similar to that of a filter bank, allowing the acoustic stimulus to be divided into frequency channels. Each channel is linked to the excitation of specific regions of the basilar membrane, each with its own characteristic frequency, generating electrical impulses that are processed by the brain [259–263]. In the literature, it has been indicated that gammatone and Butterworth filters are effective solutions for reproducing this biological mechanism [264–268].

Furthermore, by appropriately adjusting the frequency range, these filter banks are also suitable for extracting features from other types of time-varying signals, such as vibrations acquired by an accelerometer present in the WISDM dataset [239].

In this work, these filters were used as a pre-processing stage, allowing the analysed signals to be divided into distinct frequency channels.

Transfer Learning

Currently, there are numerous algorithms available for training SNNs, which can be classified according to the approach adopted in global or local methods. In global methods, network parameters are updated collectively across all layers at each training step, similar to what happens in conventional ANN architectures,

examples of such techniques include Backpropagation Through Time (BPTT) [269]. In contrast in local methods only a subset of parameters is modified at each iteration, this approach includes Spike-Time-Dependent Plasticity (STDP) [225], Hebbian learning [270] and E-prop [271].

In the context of neuromorphic SOTA for audio signal classification, the most widely adopted approach for training a SNN is through the transfer learning method [272, 273, 50, 274]. This methodology involves the initial training of an ANN and the subsequent transfer of the weights obtained to a structurally identical SNN [275]. To ensure correct transferability from CNN, used in this work, to the corresponding sCNN precautions must be taken when choosing the architecture of the layers and the neuron model, thus ensuring equivalent behaviour between the two networks. The method followed is the one proposed by Liu *et al.* [276] and applied by Dominguez-Morales *et al.* [50].

The neural model used in the convolutional layers and fully connected layers is a modified version of the ReLU activation function [276]. The operation for dimensional reduction in pooling layer is the average pooling instead of a max pooling. The classification layer uses a Softmax activation function. All biases of neurons across the network are fixed to 0 value.

The training methods uses the classical approach such as Backpropagation or Stochastic Gradient Descent (SGD). At the end of CNN training, a structurally identical SNN network is constructed in which the neural model used is LIF and the synaptic weights are initialised with the values obtained from the CNN. The input of SNN consists of the encoded form of the sonogram [273] using Poisson rate encoding, which allows pixel intensities to be represented in terms of spike rates.

This procedure ensures a uniform and impartial evaluation method for the various coding techniques, as all coded data is processed through an identical processing flow.

Model Compression

As reported in Figure 4.2, at the final stage of the process a model compression is applied. The use of compression techniques has many advantages, both in terms of quality and quantity, such as the generation of more compact models that are suitable for embedded systems. Compression reduces memory usage and computational

costs, resulting in faster simulations on non-neuromorphic hardware. Furthermore, through fine-tuning, in some cases this phase can also lead to an increase in classification accuracy, attributable to the reduction of the stimulus transmitted through the synapses, a potential source of noise in the network decision-making process.

The synaptic reduction process adopted consists of selectively removing connections between neurons in the convolutional and fully-connected layers of the CNN, based on the absolute value of the synaptic weight. The elimination is performed by calculating the distribution of synaptic weight and progressively removing connections with increasingly heavier weights. This strategy allows the elimination of synapses that make a marginal contribution to spike generation.

Following synaptic reduction, classification accuracy generally tends to decrease. To recover initial performance, a fine-tuning phase is performed whereby the resulting CNN with the remaining connections is retrained for 5 epochs and, finally, the updated weights are transferred back to the corresponding SNN version.

4.1.3 Experimental results

In order to evaluate the capacity of different encoding techniques in the classification of time-varying data, various configurations of the pipeline illustrated in Figure 4.2 were tested. The aim of these comparisons is not to identify a universally optimal solution, but rather to provide a detailed analysis to guide developers and researchers in selecting the most appropriate encoding methods for specific applications.

Frequency decomposition

The division of the input signal into frequency channels is an important step for encoding performance, as it can increase the number of extractable features and generate sonograms with richer information content. To assess the impact of this operation, extensive tests were conducted on the effect of frequency filtering in relation to the accuracy of the sCNNs tested.

The generic structure of the CNN architecture used includes: a first convolutional layer with a number of feature maps equal to I, followed by an average pooling layer; a second convolutional layer with J feature maps, followed by further average pooling; finally, K fully-connected layers. Each configuration is identified by

the acronym C*I*-C*J*-F*K*. An example, relating to the C6-C12-F2 configuration, is illustrated in Figure 4.4.

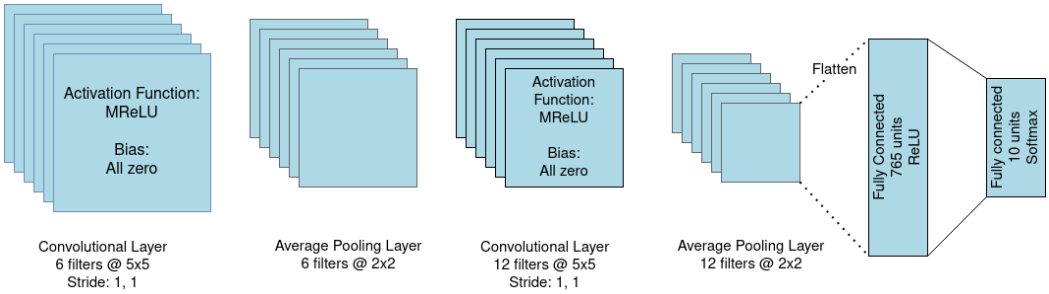

Fig. 4.4 Architecture of the convolutional neural network consisting of layers C6, C12, and F2. The figure is taken from [42].

From a global perspective experiments conducted using the gammatone filter showed overall superior performance compared to the Butterworth filter. When the classification of data is performed on FSD dataset, 32- and 64-channel decomposition configurations were tested, using the C6-C12-F2 architecture for all tests. This latter result can be attributed to the presence of redundant components in the frequency response for gammaton filter bank, which cause a greater number of spikes, thus allowing a greater amount of information to be represented. The average results obtained on the different channel configurations indicate a median accuracy of 77.50% for the Butterworth filter and 84.00% for the gammatone. In particular, the highest accuracy values were observed with Phase Encoding, equal to 83.00% with Butterworth and up to 93.00% with gammatone.

As regards the WISDM dataset, the limited sampling frequency allowed only 4, 8 and 16-channel separation configurations to be tested. In this case, for all types of encoding, average accuracy values were lower than those of the FSD: 66.67% with a Butterworth filter and 46.67% with a gammatone filter, using a C12-C24-F2 network architecture.

Comparisons between classes of encoding algorithms

Although the encoding phase is an essential step in the use of SNNs, choosing the encoding technique best suited to the characteristics of the signal to be analysed can significantly affect performance in terms of accuracy. Figure 4.5 shows a comparison between the median accuracies obtained by the different families of coding

algorithms, combined with all the channel separation variants, feature extraction and network architectures considered.

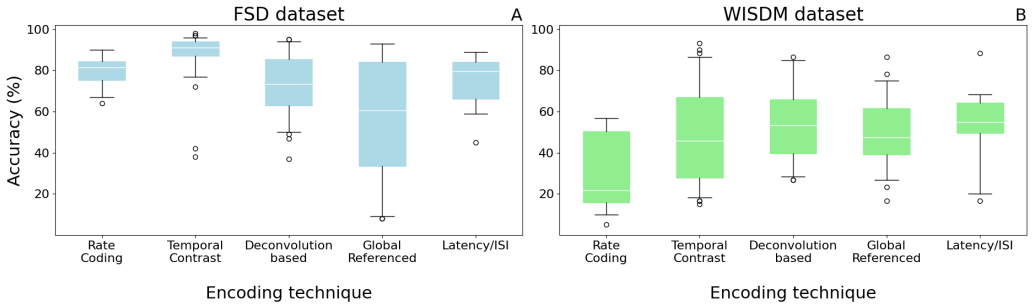

Fig. 4.5 Median accuracy values for each coding class in different combinations of network architecture, filter type, number of channels, and feature extraction bins, for the FSD dataset (A) and the WISDM dataset (B). The figure is taken from [42].

For the FSD dataset, the Temporal Contrast class performed best, with a median accuracy of 91% (Figure 4.5a). At the opposite extreme, the Global Referenced family showed the lowest median result, with a value of around 53% accompanied by high variance. This behaviour can be attributed to the marked difference in performance between the two algorithms belonging to this class: Phase Encoding produced satisfactory results with a median of 77%, with a maximum peak of 93%, while TTFS showed very low accuracy values of around 35%, with a minimum of around 8%. This gap is probably due to the reduced number of spikes generated by TTFS, which results in insufficient stimulation of the network.

In the case of the WISDM dataset classification, the different algorithms show rather heterogeneous performance; however, the median accuracy, aggregated by algorithm class, remains around 48% for all families, with the exception of Rate Coding, which records the worst median performance at 21.67% and the overall minimum value at 5%. The best median accuracy is achieved by Burst Encoding, with a value of 55%. The absolute maximum result is achieved, instead, by the ZCSF algorithm, in combination with a 16-channel Butterworth filter for a C6-C12-F2 network architecture, with an accuracy of 93%.

Spike density

The spike density value can be used to estimate energy consumption, as a low spike density leads to a decrease in communications between the various layers of the

network, resulting in low energy consumption. However, an excessively low density may be insufficient to encode information effectively, causing loss of information content. Our analyses show that this quantity is strongly influenced by the encoding algorithm and by the refractory period, causing significant variations in the density of spikes generated.

To evaluate the impact of using the refractory period τ_{ref}, preliminary experiments were conducted using different values equal to 3 ms, 2 ms, and 1 ms. In all cases, the usage of this parameter in the encoding phase resulted in an excessive reduction in spike density, compromising the correct stimulation of the SNN layers. This phenomenon led to a drastic drop in classification performance. The median FSD accuracy for the test - considering all architectures, channel decomposition configurations, and encoding techniques - was 22.00%. In the case of the WISDM dataset, the value of τ_{ref} is constrained by the low sampling frequency f_s = 20 Hz of the signals, imposing a lower limit of 50 ms. Given the performance degradation observed even for low values of τ_{ref}, all results presented in this document were obtained by setting $\tau_{ref} = 0$.

The graphs shown in Figure 4.6 and Figure 4.7 illustrate the distribution of the number of spikes produced by each coding technique, after separation into channels using a Butterworth and gammatone filter bank, respectively in the right and left columns. In all scenarios considered, the family of Deconvolution-based encoding - HSA, MHSA, BSA - is the one with the highest spike count.

Feature Extraction

The feature extraction phase is an essential step in applying the transfer learning method, as it makes it possible to process data encoded in the spike domain by convolutional layers.

The term sonogram, borrowed from Dominguez-Morales *et al.* [50], was originally used to describe the binned representation of an audio signal; in this work, the same definition is adopted for both the corresponding representation of the FSD dataset and that of the WISDM dataset.

The number of bins - i.e., the number of intervals into which the spike-encoded signal is divided - is the parameter that defines the resolution of the sonogram and, consequently, the quality of feature extraction. Excessively large or small

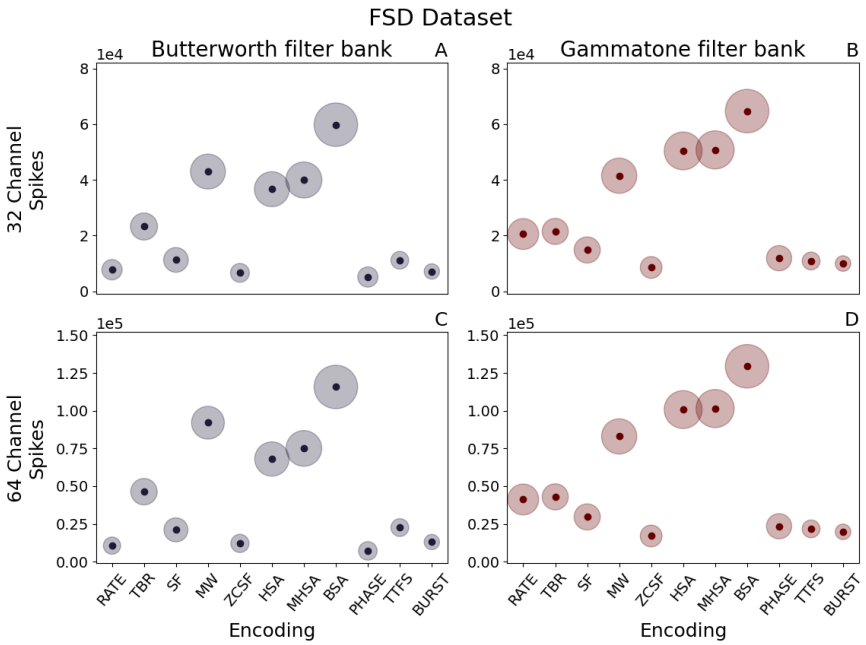

Fig. 4.6 Median spike counts per sample generated for the FSD dataset using different encoding techniques, numbers of channels, and filter types. Results are shown for: (A) Butterworth filter with 32 channels, (B) Gammatone filter with 32 channels, (C) Butterworth filter with 64 channels, and (D) Gammatone filter with 64 channels. The figure is taken from [42].

values for this parameter tend to generate an almost uniform pattern with limited information content, as the difference in intensity among the pixels of the sonogram is significantly reduced, as shown in Figure 4.8, which compares different types of binning sizes.

In the case of the FSD dataset, binning intervals of 50 and 250 were tested for the 32-channel filter bank, and 50 and 125 for the 64-channel filter bank. For the WISDM dataset, the values used are 24, 18, and 18 bins for 4, 8, and 16 frequency channel respectively. These values were selected as the most suitable because other binning options showed unsatisfactory accuracy performance. The results of the comparison of this process are shown in Figure 4.9.

In this case too, the WISDM dataset generally presents lower accuracy than the FSD. Furthermore, regardless of the number of channels used in the pre-processing phase, worse performance is recorded for high bin counts.

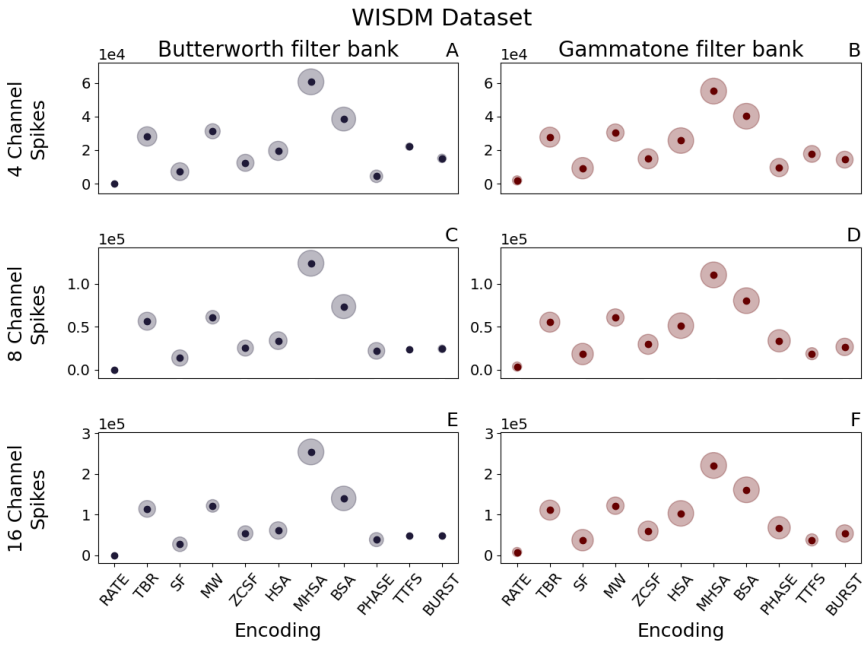

Fig. 4.7 Median spike counts per sample generated for the WISDM dataset using different combinations of encoding techniques, filter types, and channel numbers. Results are shown for: (A) Butterworth filter with 4 channels, (B) Gammatone filter with 4 channels, (C) Butterworth filter with 8 channels, (D) Gammatone filter with 8 channels, (E) Butterworth filter with 16 channels, and (F) Gammatone filter with 16 channels. The figure is taken from [42].

CNN and sCNN architecture

The type and the architecture of the network are a determining factor for performance in terms of accuracy. A CNN architecture was adopted for the training and weight transfer phase. The use of this type of network in the classification of audio signals is motivated by the SOTA results achieved through the silicon cochlea [50]. Further studies conducted by Fra *et al.* [43] have confirmed the computational and energy efficiency of CNNs in the analysis of time-varying signals.

In this study, several configurations of the CNN structure were developed, starting from the model proposed in Dominguez-Morales *et al.* [50] and proceeding to optimise the structural parameters. All the networks tested share the basic structure illustrated in Figure 4.4, differing in the number of filters used in the convolutional layers and the number of fully-connected layers. The networks with the best performance are configurations C12-C24-F1, C6-C12-F2 and C12-C24-F2.

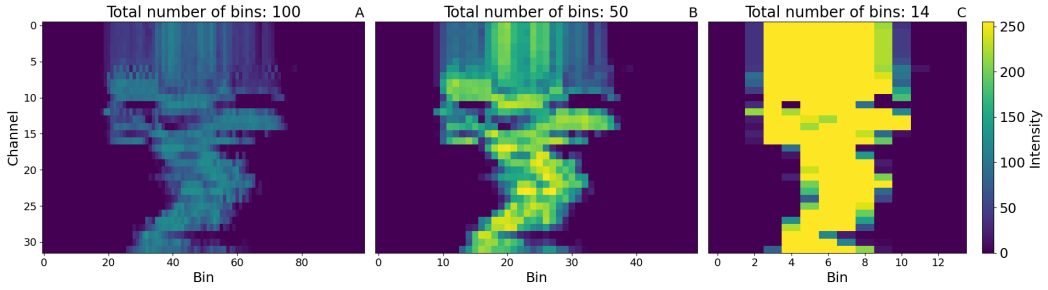

Fig. 4.8 Visual representation of a 32-channel sonogram subdivided into 100 (A), 50 (B), and 14 (C) time bins. The 50-bin case offers the best trade-off between resolution and information density. The figure is taken from [42].

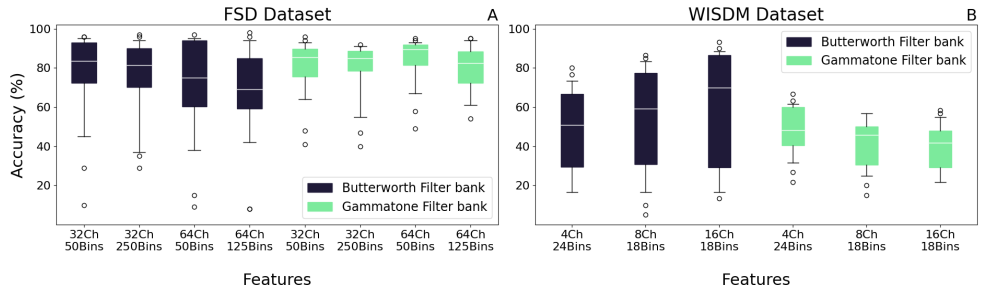

Fig. 4.9 Median accuracy for the feature extraction class in different configuration of network architecture and encoding strategies for the FSD dataset (A) and the WISDM dataset (B). The figure is taken from [42].

As regards the classification of the FSD dataset, the C12-C24-F1 configuration achieves a median accuracy of 53.00%, while the other two architectures show significantly higher performance, with values of 82.50% for C6-C12-F2 and 84.00% for C12-C24-F2. For the WISDM dataset, the C6-C12-F2 and C12-C24-F2 architectures are also confirmed as the best, achieving median accuracies of 45.00% and 52.50%, respectively. These relatively low values are not attributable to the structure of the networks, but are due to the lower efficiency of the coding algorithms analysed when applied to data characterised by lower frequencies.

Model Compression

Once the CNN structure with the best performance was identified, model compression techniques were applied to reduce connectivity within the network, thereby decreasing memory, computing and energy requirements.

To evaluate the effectiveness of these techniques, the synapse reduction process was applied to the best-performing network configurations that emerged from previous tests, with the C6-C12-F2 and C12-C24-F2 structures as the best candidates. The procedure involves the progressive elimination of connections based on the weight of the synapses, calculated on the distribution of absolute values. Initially, connections with a weight less than or equal to the first quartile are removed, followed by those up to the median and, finally, those up to the third quartile.

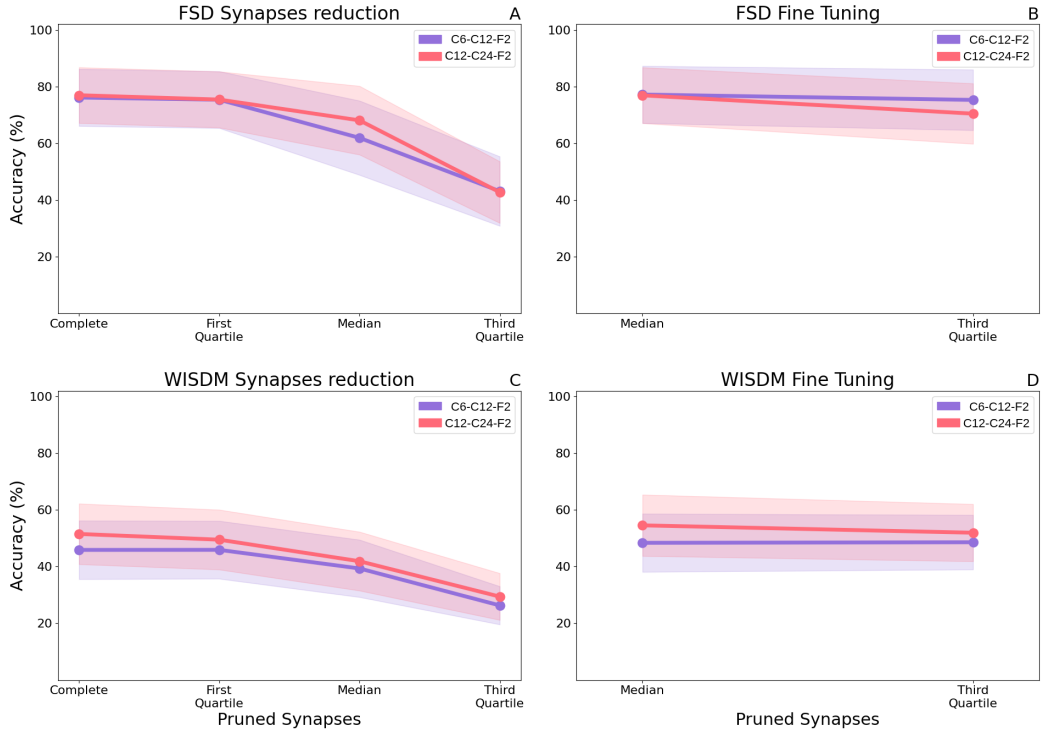

Fig. 4.10 Median test accuracy for all combinations of encoding class, filter type, number of channels, and feature extraction bins, using architectures C6-C12-F2 and C12-C24-F2 on FSD and WISDM datasets. Panels show results after synapse reduction (A,C) and after fine-tuning (B,D). The figure is taken from [42].

Figure 4.10a illustrates the effect of synapse reduction on the classification of the FSD dataset. As the amount of connections reduced, classification performance worsens. This behavior is due to the reduced number of spikes in the network, which makes it more challenging to stimulate neurons through the layers correctly. To optimize the model based on residual synapses, a fine-tuning process is applied to the smaller C6-C12-F2 network, which is retrained for approximately 5 epoch. Subsequently, the new weights are transferred to the final version of the sCNN, achieving

accuracy comparable to that of the complete network. Some configurations are able to slightly exceed the classification capabilities of the original model, achieving an increase of 1.75% in accuracy. The compressed networks achieve a median test accuracy of 81.00% while maintaining only 25% of the original network size, through pruning to the third quartile Figure 4.10b. This median value is calculated considering all filter bank configurations, feature extraction and encoding algorithms for the given architecture.

In the case of the WISDM dataset, the reduction in synapses also leads to a decrease in accuracy (Figure 4.10c). Also in this case, after applying fine-tuning to the C12-C24-F2 network, some configurations show an increase in the maximum achievable accuracy, leading to superior performance of the compressed network compared to the complete one. For example, the ZCSF algorithm with 16 channels and 18 bins, pruned to the third quartile, shows a 1.7% increase in classification capacity, reaching 91.7%; Similarly, the SF algorithm with 16 channels and 18 bins, reducing synapses to the median, achieves an increase of 8.3%, reaching an accuracy of 95.0%.

This improvement can be attributed to the synergy of synapse reduction and fine-tuning. This has the effect of decreasing the number of synapses between the layers reducing the noise through the network with positive effects on the classification process, while maintaining a high classification capability.

The configurations that show improved performance after model compression are illustrated in Figure 4.11.

4.1.4 Discussion

Through these experiments, we conducted an in-depth analysis evaluating different combinations of filters, encoding algorithms, feature extraction parameters, and network architectures. The goal of this work is to create a comprehensive guide designed to provide neuromorphic system designers with useful information on the comparative performance of various encoding techniques.

The results show that the performance of the coding techniques considered is strongly influenced by the signal frequency. In the case of FSD - characterised by intermediate frequencies and a wide band - it is possible to extract a greater number of features from the signal, making the effectiveness of the coding classes that

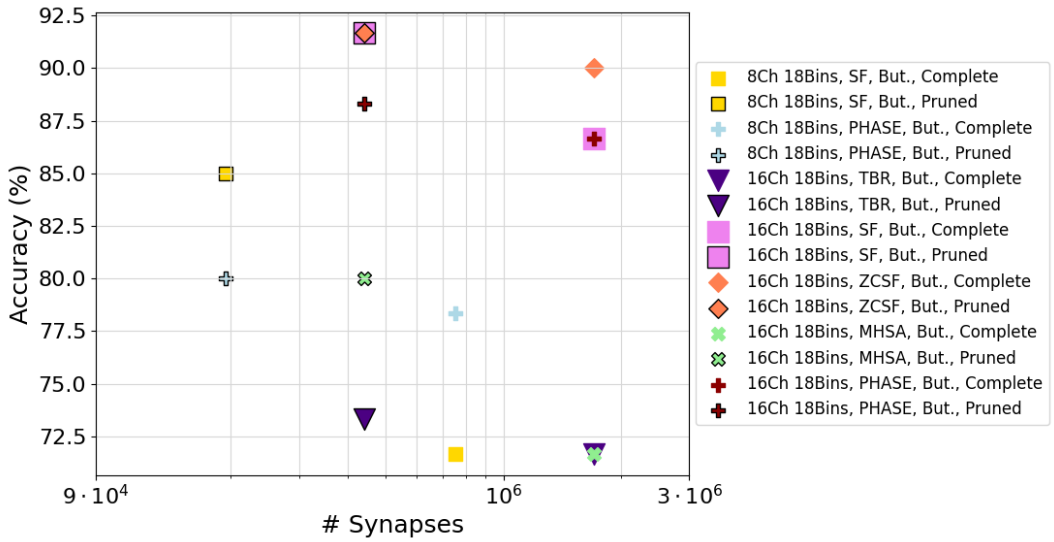

Fig. 4.11 Overview of network configurations that achieved higher performance following model compression, in the case of the WISDM dataset. The figure is taken from [42].

allow for more accurate classification more evident. On the contrary, for very low frequency signals such as those in the WISDM dataset, there is no clear advantage of one class of algorithms over the others. However, some configurations based on temporal coding, such as the ZCSF algorithm have clearly outperformed rate coding, demonstrating that, although the choice of algorithm must be carefully calibrated, temporal coding techniques have a greater ability to extract characteristics suitable for neuromorphic processing from very low frequency signals.

Finally, it has been observed that the number of spikes generated by each encoding method must be sufficiently high to adequately stimulate all downstream layers of the SNN. Consequently, any strategies to reduce the spike count for energy saving purposes must be carefully balanced with the need to preserve an adequate amount of information.

Figure 4.12 shows a quantitative and comparative representation of all the coding techniques examined. Each method is described using five fundamental metrics: $\mathcal{S}$ Shannon entropy [277] of the encoded signal, $\mathcal{MI}_{\mathcal{S}}$ mutual information [278] between the encoded signal and the original input normalised with respect to entropy, $\mathcal{HS}$ Hoyer sparsity of the encoded signal [279], $\boldsymbol{\mathcal{E}}$ spiking efficiency [237], and $\mathcal{O}(f)$ computational complexity.

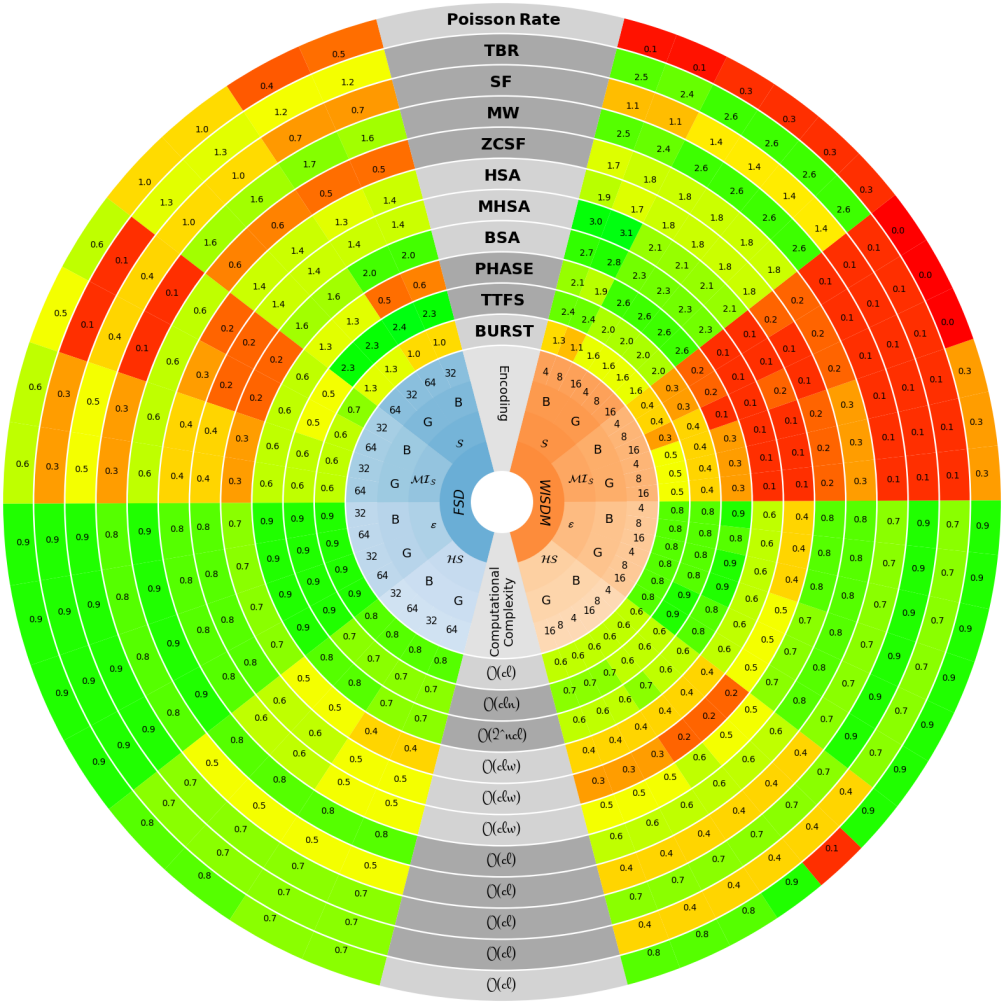

Fig. 4.12 Each encoding technique is represented along specific rings of a circular graph. The bottom central section shows computational complexity, defined by signal length (l), number of channels (c), bitwise representation length (n), and convolution width (w). Values on the left correspond to FSD data, while those on the right correspond to WISDM. Four signal-related metrics ($\mathcal{S}$, $\mathcal{MI}_{\mathcal{S}}$, $\mathcal{HS}$, $\boldsymbol{\mathcal{E}}$) are arranged symmetrically across the vertical axis, with results for both Butterworth (B) and Gammatone (G) filters reported according to the number of channels used. The figure is taken from [42].

Table 4.1 presents a summary of the recommendations, correlating each encoding technique with the frequency of the time-varying input signal. This table is the central result of this study and represents a first step towards a systematic, large-scale analysis of the tools available for representing signals in the neuromorphic domain.

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	✓	✓	✗
	Filter & Optimizer	HSA	–	–	✗
		MHSA	–	–	✗
		BSA	✓	–	✗
	Global Referenced	PHASE	✗	✓	✓
		TTFS	✗	✓	✓
	Latency/ISI	BURST	✗	✓	✓

Table 4.1 Summary of the various encoding techniques, with an evaluation of their performance according to input data type. The table is taken from [42].

4.2 A neuromorphic approach for on-edge AIoT application

4.2.1 On-edge computing advantage

The constant process of technological miniaturisation and greater integration of a large number of sensors in wearable devices has significantly expanded the integration of Body Sensor Networks (BSNs) in the non-invasive monitoring of activities and physiological signals [280–282]. The growing availability and spread of BSNs, combined with the possibility of applying machine learning techniques, has enabled the creation of applications based on wearable sensors for technologies in the fields of IoT, edge computing and Human-Centred Computing, for digital healthcare, elderly care, fitness monitoring and gesture recognition [283–286].

The ability to implement machine learning models directly on the remote device brings a number of significant advantages. Edge devices enable real-time inferences, reducing system latency in time-sensitive applications such as autonomous driving [287–291]. Locally stored data increases system reliability by avoiding the consequences of network outages, while also improving user privacy, as there is no need to transmit information externally. Removing dependence on external infrastructure reduces both the energy consumption associated with communication and overall maintenance costs.

As a result, numerous studies have adopted approaches based on deep learning techniques, proposing solutions that exploit CNNs, Recurrent Neural Networks (RNNs) or a combination of both [292–298], specifically geared towards recognising human activities in scenarios designed for wearable platforms [299–302]. The development of these applications is supported by dedicated end-to-end solutions that cover the entire design flow: from data collection and annotation, model definition and training, to optimisation and deployment. These solutions are provided both by hardware manufacturers - for example, Qeexo AutoML [303] - and in open-source form, such as the Embedded Learning Library.

However, despite significant advances in machine learning for ANN-based technologies, crucial challenges remain, as the adoption of devices designed to be extremely compact and portable introduces new constraints that cannot be overlooked. Most ANN models have excessive computational requirements to run directly on embedded devices, as they require large amounts of memory, dedicated hardware, and high energy consumption.

As Covi *et al.* [304] have thoroughly pointed out, achieving the ultimate goals of edge computing for wearable devices requires a real paradigm shift, which translates into a reduction in the computational effort required for data processing. The sensors typically integrated into wearable devices are burdened by severe energy limitations; at the same time, the conventional approach, based on transmitting data to remote servers for off-chip processing, introduces an additional constraint in terms of time latency. Overcoming these challenges would mean enabling real-time data processing directly on the device, paving the way for a wider range of personalised services that can be implemented efficiently and widely on smart edge devices [84].

In the field of edge computing, the body monitoring sector has undoubtedly attracted the attention of the scientific and industrial communities, with HAR playing a key role. This application stands out for its intrinsic reachness of information and consequent adaptability to multiple usage scenarios. A reliable and responsive classification system for ongoing activities is useful not only for monitoring Activities of Daily Living (ADL), but also for other purposes. [305], but can also play a decisive role in critical safety situations where response time is essential.

In the race for the best classification results, the vast majority of proposed solutions focused almost exclusively on accuracy performance, neglecting the possibility of adopting models closer to biological inspiration and leaving the alternative per-

spective offered by a neuromorphic approach in the background. In this context, the adoption of brain-inspired neural paradigms, such as SNNs [306], represents a candidate of great interest for the creation of low-energy solutions [307, 308].

By relying on spike-based encoding, SNNs have a natural predisposition for processing temporal information [309–313]. Complementary to neural computing, whose main objective is the implementation of ANNs for solving practical tasks, neuromorphic computing shifts the focus towards emulating neural processes through new and alternative computing architectures [314–317, 37, 318, 319, 319–321]. This approach has the direct consequence of orienting research and development towards the creation of dedicated, compact, low-power neuromorphic hardware solutions, such as SpiNNaker [38], Intel Loihi [37], and Dynap-SE [76].

Alongside these platforms, considerable interest has also been shown in neuromorphic simulators, which are powerful tools capable of supporting the growth of neuromorphic computing even in scenarios where specific hardware is not immediately available. Among the various tools available, Nengo is a particularly interesting example [322, 234, 323].

In this work, we propose a comparative analysis of different recurrent and convolutional architectures, in their spiking and non-spiking variants for the HAR task, using the WISDM dataset [257, 194], with the aim of investigating a classification approach based solely on raw data. In particular, through the use of Nengo, we evaluate the positive effect of adopting a neuromorphic paradigm as an alternative to classical deep learning solutions, presenting bio-inspired models for the recognition of human activities directly from raw data.

4.2.2 Human activity recognition

In recent years, the dominance of HAR has generated a vast amount of scientific output, in which machine learning techniques have been widely applied and tested on different datasets. Numerous studies have adopted approaches based on deep learning techniques, proposing solutions that exploit CNNs, RNNs or combinations thereof [292–298], specifically geared towards the recognition of human activities in scenarios designed for wearable platforms. Through a data acquisition approach using personal and non-invasive devices, there has been a gradual convergence towards the use of wearable sensors, often integrated into smart devices, which have progres-

sively taken on a central role in the field of HAR. At the same time, neuromorphic computing has attracted growing interest, encouraging increasing attention towards the use of bio-inspired networks and the development of applications in HAR [324].

The comparison is conducted at the end of the optimisation pipeline, schematically summarised in Figure 4.13. In it, the vertical arrows represent the preliminary phases, which include the selection of the dataset (a) and the design of the optimisation experiment (c, d). The horizontal arrows, on the other hand, describe the subsequent phases that constitute the backbone of the study: the selection of neural network architectures (b), the optimisation of hyperparameters (e) and, finally, the creation of classifiers specifically adapted to the HAR task (f).

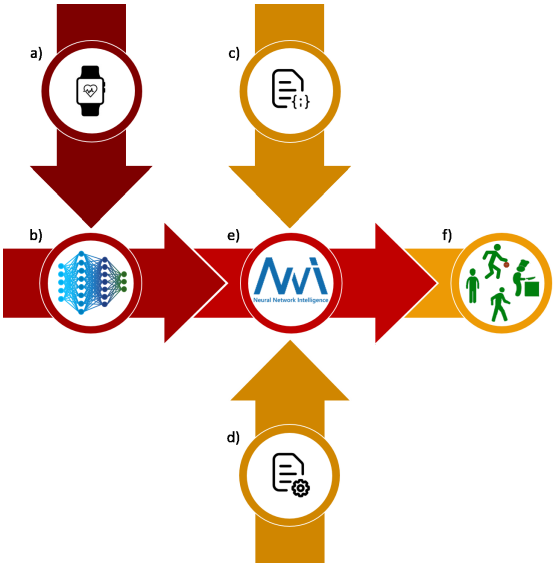

Fig. 4.13 The adopted procedure can be divided into two complementary phases. The vertical arrows highlight preliminary steps: (a) dataset selection, (c) definition of the hyperparameter search space, and (d) configuration of the optimization experiment. The horizontal arrows summarize the backbone of the pipeline: (b) neural network architecture selection, (e) hyperparameter optimization, and (f) classifier evaluation. The figure is taken from [43].

Dataset WISDM

In an increasing number of cases, datasets are collected using Inertial Measurement Units (IMUs) integrated in smartphones and smartwatches, as is the case with the WISDM [257, 194] and UCI-HAR datasets [325]. Alongside these well-established

datasets, a more recent version of the WISDM dataset is gaining increasing interest thanks to better class balancing and the inclusion of signals from smartwatches.

The WISDM dataset is composed of records from 51 subjects engaged in 18 activities. The dataset collects signals from the accelerometer and gyroscope of a smartphone and smartwatch. Each activity was recorded for a duration of 3 minutes, with an acquisition frequency of 20 Hz. Furthermore, the data can be divided into three subsets depending on the type of activity: non-hand-oriented activities, hand-oriented activities (general), and hand-oriented activities (eating). As an example, the kernel density estimation of the three-dimensional data from the smartwatch accelerometer and gyroscope, relating to the general hand-oriented subset, is shown in Figure 4.14, where a clear overlap between the raw signal values can be observed, clearly showing that the classification of raw signals is not trivial when analysed in real time, without any form of data processing, filtering or aggregation.

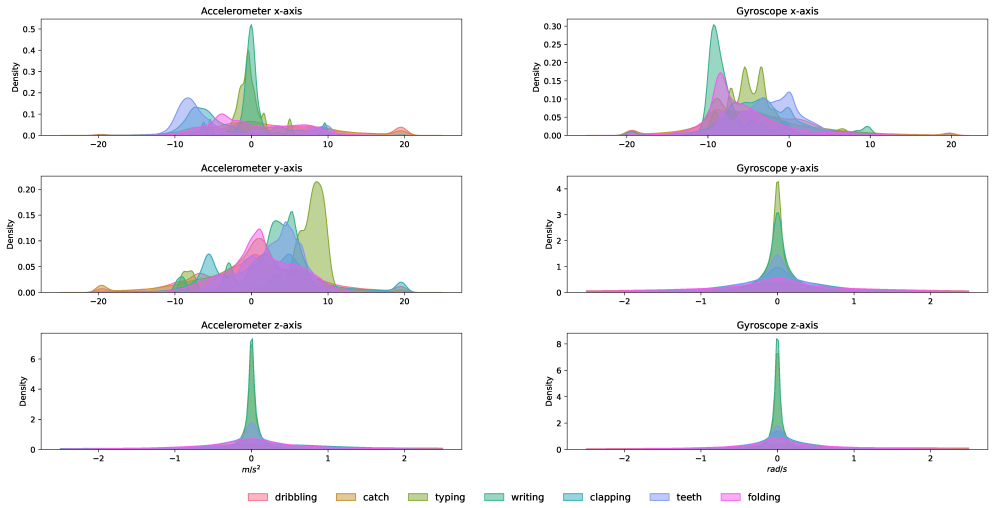

Fig. 4.14 Kernel density estimates of smartwatch data collected along the six IMU sensor axes for the seven classes within the general hand-oriented subset of the WISDM dataset. The figure is taken from [43].

For the experiments carried out a subset of hand-oriented activities (general) was selected from the complete dataset, including: dribbling in basketball, playing catch with a tennis ball, typing, writing, clapping, brushing teeth and folding clothes. An example of the raw three-axis signals from the accelerometer and gyroscope present in the WISDM dataset is illustrated in Figure 4.15.

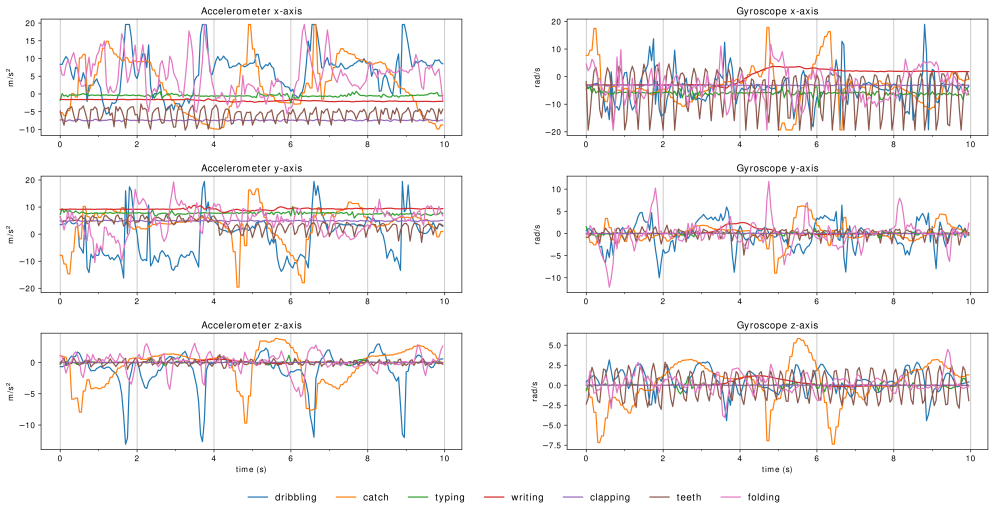

Fig. 4.15 Comparison of representative 10 s smartwatch samples from six IMU sensors for the seven classes in the general hand-oriented WISDM subset. The figure is taken from [43].

Impact of data segmentation

The actual demonstration of improved energy efficiency introduced by the use of neuro-inspired and neuromorphic approaches has been the subject of a growing number of experiments. Blouw *et al.* [326] performed benchmarking on different hardware platforms for the task of keyword spotting, highlighting a significant reduction in energy consumption through the use of the Intel Loihi chip, while Yan *et al.* [327] conducted a direct comparison with a SpiNNaker 2 prototype.

In the work of Buettner *et al.* [328], the effectiveness of the conversion from ANN to SNN for the heartbeat classification task is evaluated, subsequently implemented on Loihi. A more extensive benchmarking analysis was proposed by Azghadi *et al.* [329], who tested different neuromorphic platforms in biomedical applications. Finally, Blouw *et al.* [324] highlighted the advantages of the neuromorphic approach, demonstrating the reduction in computational cost offered by SNNs developed in Nengo, compared to structurally identical Deep Neural Networks (DNNs).

The IoT is considered one of the areas set to benefit most from the development of neuromorphic models and technologies. In this regard, Kim *et al.* [330] proposed a survey of IoT platforms that enable artificial intelligence applications, while An *et al.* [331] investigated the impact of neuromorphic systems in the context of Industry

4.0. The role of edge computing in Artificial IoT (AIoT), as well as in healthcare applications and smart environments, was explored in depth by Chang *et al.* [332] Finally, promising results for event-driven on-edge AI applications in IoT scenarios were presented by Stuijt *et al.* [321], using the μBrain neuromorphic integrated circuit.

An example of benchmarking network architectures on different datasets is presented in the work of Mekruksavanich *et al.* [333–335], in which an in-depth analysis is conducted based on the use of different techniques and types of data. A particularly relevant contribution of this study concerns observations on the impact of data segmentation in relation to classification accuracy. Identifying the optimal time window size for time-varying signals, such as those typical of HAR, is crucial from two perspectives: on the one hand, it can significantly increase the accuracy of the classifier, making it more reliable; on the other hand, in terms of classification time, it is a fundamental parameter for evaluating the suitability of the model in real-time applications. The work of Peppas *et al.* [336] provides a summary of the most commonly used window sizes in HAR tasks, highlighting that the most common choices are between 1 s and 10 s. The exceptions are the works of Ordoñez *et al.* [337], Wan *et al.* [338] and Xia *et al.* [339], in which time windows of up to 0.25 s were used, depending on the dataset considered. In contrast, Mekruksavanich *et al.* [333–335], as well as Oluwalade *et al.* [340] and Ihianle *et al.* [341], adopted signal segmentation based on 10 s windows.

For the segmentation implemented for the development of our experiments, the signals were divided into non-overlapping time windows of 2 s. This choice is the result of a preliminary exploration that considered longer windows, equal to 5 s and 10 s. Compared to the latter, the 2 s window proved to offer an optimal compromise between the need for a sufficiently high number of time data per sample and the goal of ensuring fast-response classifiers.

The process led to the generation of 36,201 samples based exclusively on raw data, without any feature extraction or pre-processing, which were then divided into three sets for training, validation and testing in a ratio of 60:20:20 respectively.

Nengo

Nengo was developed as a simulator based on the Neural Engineering Framework (NEF) [342] guiding principle for building complex neural models capable of generating sophisticated networks with cognitive abilities, starting from models of individual neurons, providing access to low-level neural archetypes for performing high-level functional tasks.

The three principles of NEF - representation, transformation and dynamics - are embodied in Nengo into the fundamental units for network construction, defined through three main objects: ensemble, node and connection. Their combination leads to the definition of two further objects, network and model. The Probe represents the element dedicated to data collection during simulations. This set of six objects constitutes the toolkit necessary for designing the neural model, which is required to run the network simulation.

A distinctive feature of Nengo is the flexibility of its simulator, made possible by its ability to adapt to specific and, if necessary, highly specialised hardware. An example of this is NengoLoihi, a back-end developed specifically for running Nengo models on the Intel Loihi neuromorphic chip. Furthermore, thanks to this adaptability, it is possible to directly integrate models from other frameworks: the NengoDL Converter module allows the translation of deep learning models by change standard activation functions with Nengo spiking neurons [343].

Legendre Memory Unit

The mechanisms of neural communication based on the complex processes of transmitting and filtering spikes through synapses can be modelled using Ordinary Differential Equations (ODEs) integrated over time, which enable the behaviour of the cells to be approximated [344, 345].

The LMU is a recurrent cell that can perform this approximation for continuous-time delay [346]. The fundamental property of the LMU network consists in the ability to decode a delayed signal $u(t-\theta')$, contained within a time window of length θ, through a high-dimensional projection of the input $u(t)$, suitably orthogonalised using translated Legendre polynomials [347]. The i^{th} translated Legendre polynomial is defined in Equation 4.14:

$$P_i(r) = (-1)^i \sum_{j=0}^{i} \binom{i}{j} \binom{i+1}{j} (-r)^j \tag{4.14}$$

and is used to delay the input signal as shown in Equation 4.15:

$$u(t-\theta') \approx \sum_{i=0}^{d-1} P_i\left(\frac{\theta'}{\theta}\right) m_i(t) \tag{4.15}$$

The maximum degree $d-1$ of the expansion series is linked to the dimension of the state vector $\mathbf{m}(t)$, defined from the input $u(t)$ according to Equation 4.16:

$$\theta \dot{\mathbf{m}}(t) = \mathbf{A}\mathbf{m}(t) + \mathbf{B}u(t) \tag{4.16}$$

where **A** and **B** represent the state matrices derived using Padé approximants, as reported in *Eq.4.17* and *Eq. 4.18*.

$$\mathbf{A} = [a]_{ij} \in \mathbb{R}^{d\times d}, \quad a_{ij} = (2i+1) \begin{cases} -1 & i < j \\ (-1)^{i-j+1} & i \geq j \end{cases} \tag{4.17}$$

$$\mathbf{B} = [b]_i \in \mathbb{R}^{d\times 1}, \quad b_i = (2i+1)(-1)^i, \quad i,j \in [0, d-1] \tag{4.18}$$

Although the literature available on LMU applications is still limited, the results are very promising in the field of keyword spotting [326]. This architecture has demonstrated SOTA performance in terms of accuracy, accompanied by a surprisingly small number of parameters.

Network architecture

The HAR task can be successfully addressed by using convolutional or recurrent architectures. In phase *b* of Figure 4.13, we examined both types in order to benchmark the possible options offered by neuro-inspired solutions.

As regards convolutional networks, the structure adopted includes two convolutional layers, one max pooling layer, and two fully connected layers, as shown

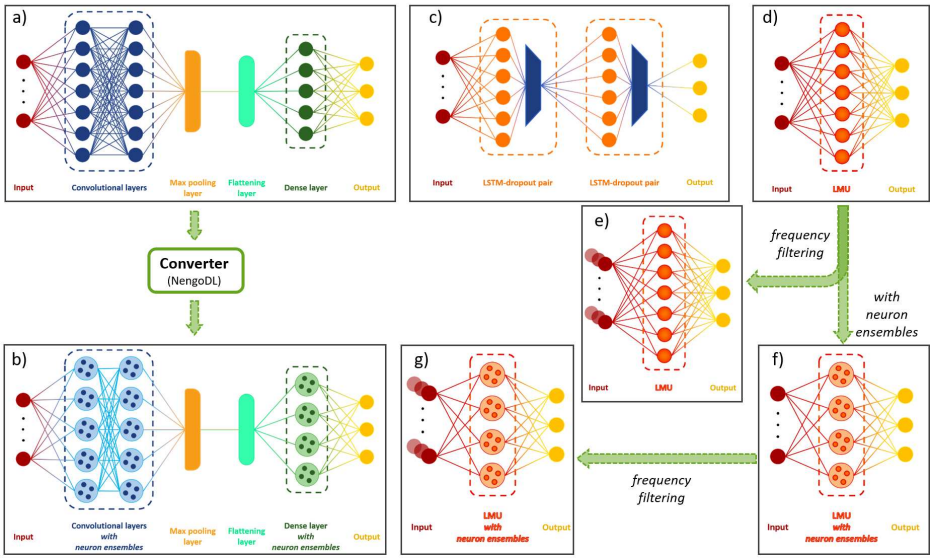

Fig. 4.16 Summary of the networks under investigation. The convolutional architecture used in the non-spiking domain (a) was translated to a spiking domain version (b) using NengoDL. Recurrent architectures differ between domains: LSTM units with a dropout layer were used for the non-spiking implementation (c), while the spiking recurrent network was built using the LMU (f), which was also adopted in the non-spiking domain (d). Additional variations with frequency-filtered inputs were tested for both non-spiking (e) and spiking (g) LMU-based architectures. The figure is taken from [43].

in Figure 4.16a. This configuration has been implemented both in the traditional version (non-spiking CNN) and in the corresponding version with spike neurons sCNN Figure 4.16b. For the recurrent architecture, we implemented a network based on two cells of Long Short-Term Memory (LSTM) layers, each followed by a dropout layer and a final dense layer, as shown in Figure 4.16c. Unlike the convolutional case, the spiking implementation of recurrent networks does not reproduce the same architecture as its non-spiking counterpart. As displayed in Figure 4.16f, the LMU was used to replace the LSTM, with a single LMU layer instead of the LSTM-dropout pair. To further enrich the comparison and benchmarking activity, the LMU was also adopted in a non-spiking version, Figure 4.16e.

The sCNN, LMU and its spiking variant sLMU networks were implemented using the Nengo neural simulator, exploiting the NengoDL Converter to automatically build the spiking version of the CNN from the corresponding non-spiking model.

On LMU-based networks, we also used Nengo to analyse the impact of introducing a filter bank inspired by the functioning of the human auditory system. By

analogy with the cochlea, a frequency filter (ff) was introduced and applied to the input, as shown in Figure 4.16e and Figure 4.16g, with the aim of breaking down the original signal into five distinct channels.

In all the spiking networks studied, the Rectified Integrate-and-Fire (RIF) neuron model was used, which is available in Nengo and compatible with the Loihi neuromorphic chip.

Hyperparameter optimisation

ANNs can be described from two opposite perspectives. On the one hand, there is the architecture, which is defined by the number, type and connection mode of the layers. On the other hand, there are the hyper-parameters, which uniquely characterise each network, determining its intrinsic behaviour. Consequently, as also highlighted by the work of Suto *et al.* [348], HPO is an essential step when analysing and comparing different network topologies, with the aim of maximising their modelling capabilities while minimising unnecessary complexity.

The steps from (c) to (e) in Figures 4.13 illustrate the hyperparameter tuning procedure we adopted, conducted using the NNI toolkit [349], exploiting the integrated Annealing algorithm. For each architecture, an NNI optimisation experiment was designed, defined within a specific search space identified by step (c). Each experiment comprises 1.000 trials, interspersed with four equidistant random reinitialisations of the tuner, in order to partially mitigate the risk of convergence to local minima typical of annealing algorithms.

Each test consists of 100 training epochs; at the end of which, the weights that provided the highest accuracy in training are selected and used to evaluate test accuracy, chosen as the optimisation objective. In all cases, training was conducted with the Adam optimiser at a constant learning rate, which was also included among the parameters optimised during the experiments. The overall configuration of the NNI experiments is summarised in step in Figure 4.13d, while an overview of the hyperparameters included in the search spaces is shown in Table 4.2.

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 weights 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

Table 4.2 Summary and description of the optimized hyperparameters. For spiking networks, the hyperparameters of the corresponding non-spiking implementations are included as well. The table is taken from [43].

4.2.3 Experimental results

To ensure a comprehensive comparison between networks, multiple metrics were adopted in addition to classification accuracy among the verified network architectures. An initial differentiation was made as follows: for all non-spiking networks, the number of Floating Point Operations (FLOPs) and the respective estimated energy consumption on the Intel Movidius Neural Compute Stick 2 board were evaluated; for spiking networks, the number of neurons, the number of Synaptic Operations (SOPs) and the estimated energy consumption on the Intel Loihi platform were analysed; while for all the network categories, the number of parameters and the memory footprint were taken into account. The energy estimates reported here are based on the results reported in the work carried out by Blouw *et al.* [350].

	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

Table 4.3 Summary of the evaluated metrics. All reported values were obtained using the optimal hyperparameter configuration for each network. The table is taken from [43].

Table 4.3 summarises the metrics considered together with the corresponding values obtained for each network. Classification accuracy is the first parameter to be examined. From this point of view, the best result is provided by the LSTM-based network, which achieves a test accuracy of $96.42 \pm 0.03\%$. Of particular interest

is the fact that the second best performance is obtained by the recurrent network based on the spiking version of LMU implemented with RIF neurons, achieving a test accuracy of $94.51 \pm 0.15\%$, thus surpassing the performance of convolutional architectures, regardless of whether they are borrowed in the spiking or non-spiking domain. The same is observed in the sLMU enriched with a frequency filter inspired by the auditory system also ranks above both sCNNs and CNNs, with a test accuracy of $94.39 \pm 0.13\%$.

In Table 4.3 the second metric considered is the total number of parameters, which is directly connected to memory footprint. From this viewpoint, what might have seemed to confirm the superiority of the LSTM-based network - suggested by the classification accuracy results - is completely reversed. With over two million parameters, this architecture is by far the most memory-intensive, with a footprint of 8.50 MB.

At the opposite extreme is the network built on the non-sLMU, characterised by a memory footprint that is more than an order of magnitude lower than the LSTM, at only 0.30 MB. Comparable values are also found for the LMU with frequency filtering and its spiking counterpart sLMU. While the spiking version with frequency filtering shows a slight increase, approaching the values recorded for convolutional architectures.

The relative difference in the sizes of the various networks can be further appreciated by observing the diameter of the circles shown in Figures 4.17, which visually represent the scale of the memory footprints.

The last two rows of Table 4.3 highlight the energy consumption evaluated with reference to two specialised hardware platforms: Intel Movidius Neural Compute Stick 2 for non-spiking networks, and Intel Loihi for spiking networks. In both cases, the quantitative estimate provides the energy cost associated with a single operation. The same table shows both the number of operations and the energy consumption per inference for each of the architecture considered. The equivalent results are shown in Figure 4.17, where the y-axis represents energy consumption, while the number of operations is encoded by the colour of the circles. As clearly visible - consistent with expectations - all spiking networks are less computationally expensive, with a minimum energy consumption value of $5.49\mu J$ for the sCNN network.

It is also important to note that the estimated energy consumption for all LMU-based architectures is at least one order of magnitude lower than that of CNNs and

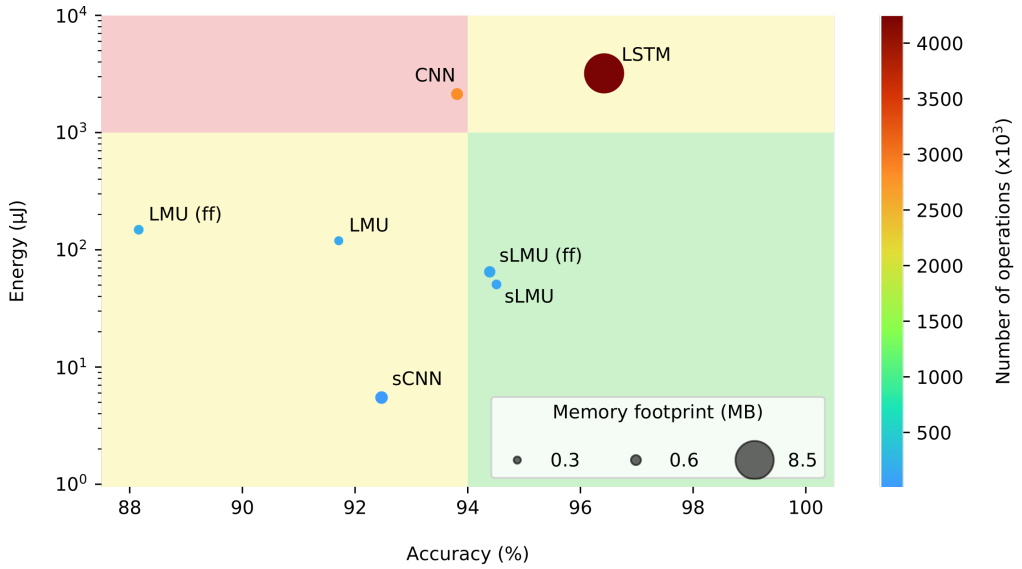

Fig. 4.17 An energy versus accuracy analysis highlights the advantages of neuromorphic approaches for temporal signal classification. The results show that all studied SNNs and LMU-based networks achieve at least an order of magnitude lower energy consumption than conventional DNNs. Similarly, memory requirements are reduced, with CNNs and LSTMs exhibiting the largest footprints. Accuracy-wise, spiking LMUs demonstrate performance comparable to, or exceeding, that of CNNs and LSTMs. The figure is taken from [43].

LSTMs. Once again, the maximum value is provided by LSTM, which exceeds $3000\mu J$: a consumption that makes it almost three orders of magnitude more energy-intensive than sCNN.

Within the range defined by these two extremes, sLMU emerges as the solution offering the best compromise, with consumption of just $50.66\mu J$ and the second-best overall accuracy, making it approximately two orders of magnitude more energy efficient while maintaining performance comparable to that of the LSTM network.

4.2.4 Discussion

At the end of the proposed pipeline, corresponding to step (f) in Figure 4.13, a classifier trained with optimised hyperparameters for each architecture was obtained. Subsequently, these classifiers were compared with the aim of evaluating the benefits introduced by neuro-inspired approaches, while avoiding a reductive perspective based solely on performance in terms of accuracy.

Benchmarking neural networks, and more generally comparing classifiers, is inherently subject to the risk of oversimplification, which often results in accuracy being evaluated as the only significant metric. In the neuromorphic context, this reductive approach is even more misleading. Although classification accuracy undoubtedly plays a central role, it cannot be considered independently of other comparison metrics, such as energy consumption and memory footprint. Both of these metrics provide crucial information for a more in-depth and informed evaluation of neuromorphic solutions to classification problems.

In particular, when SNN achieve classification performance comparable to that of other non-spiking DNNs, accuracy alone is not sufficient to ensure a fair comparison.

By combining the information that emerged from the results discussed above, it is easy to identify the network with the best accuracy and the one with the smallest memory footprint. Nevertheless, these two solutions do not coincide; in fact, they are very far apart in terms of both metrics. It follows that, in the absence of an assessment of energy consumption, the conclusions that can be drawn remain partial.

By taking this further step, i.e. quantifying the advantage of adopting a neuromorphic approach in the task under consideration, a trio of fundamental quantities is extracted from each network. In this way, the proposed benchmarking is not reduced to a simple numerical comparison, but also becomes a useful tool for guiding possible future application scenarios for neuro-inspired solutions.

The compromise between high classification accuracy and low energy consumption offered by sLMU, combined with its small memory footprint, makes this structure a particularly promising competitor for potential on-edge applications of neuromorphic classifiers in real-time processing scenarios.

With a view to comparing the different architectures in terms of specific tasks and applications, the results obtained are also summarised in Figure 4.18, where the use of a radar chart further highlights the strengths and limitations of the networks analysed.

4.3 Summary

The present analysis examines neuromorphic computing from the perspective of machine learning, with a particular focus on data-driven methodologies for processing

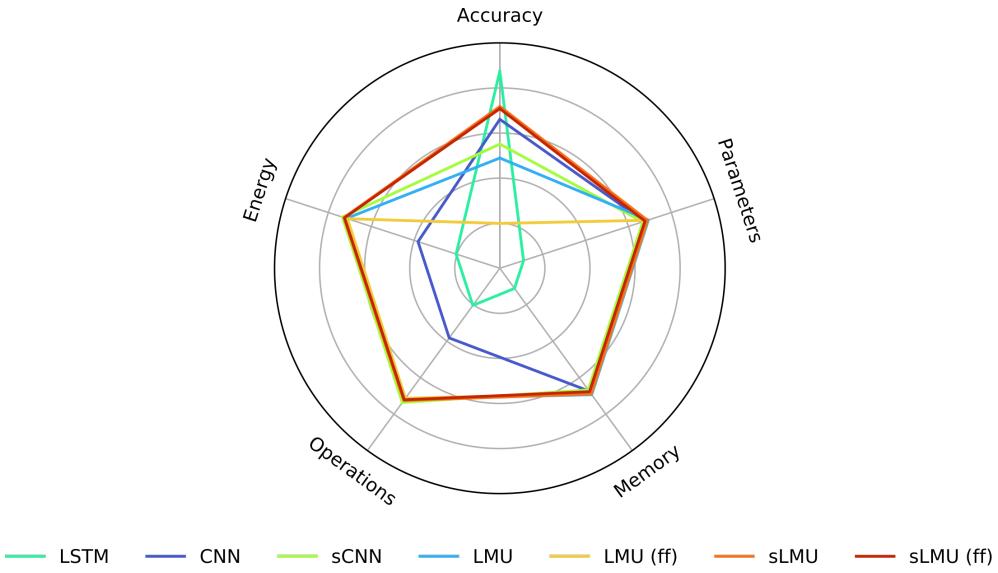

Fig. 4.18 Radar chart comparing the investigated networks across evaluated metrics. LMU-based architectures outperform traditional DNNs (LSTM, CNN) in energy and memory, while spiking CNN also shows significant improvements over LSTM and non-spiking CNN. The figure is taken from [43].

real-world, time-dependent signals. As a progression from the manually designed spiking architectures that were explored in the preceding chapter for the purposes of optimisation and constraint compliance, the focus here shifts to learning-based approaches. In these approaches, spiking neural networks are trained to extract meaningful representations directly from sensory data. This transition signifies a progression towards scalable, general-purpose neuromorphic systems that are capable of operating autonomously in realistic application scenarios.

The central theme of this chapter is the role of spike coding as a key interface between conventional digital sensors and neuromorphic processing. By means of a systematic comparison of multiple coding strategies applied to audio and inertial datasets, this work highlights how the choice of coding method significantly influences both classification accuracy and computational efficiency. The proposed end-to-end pipeline, which includes biologically inspired preprocessing, spike-based feature extraction, transfer learning from equivalent ANN models, and model compression, provides a structured framework for evaluating and implementing SNNs in embedded and edge-oriented contexts.

The application of these concepts to human activity recognition further demonstrates the suitability of neuromorphic machine learning for energy-constrained

edge devices. A comparison of SNN-based classifiers with SOTA deep learning models and alternative temporal architectures, such as LMUs, was conducted. The results demonstrate that spiking networks can achieve competitive performance while significantly reducing energy consumption. Furthermore, the adoption of an application-based hyperparameter optimisation strategy demonstrates the capacity for neuromorphic classifiers to be systematically tailored to specific domains, thereby enhancing their potential for personalised, on-device intelligence in AIoT scenarios.

The contributions presented in this chapter demonstrate that neuromorphic computing is not confined to manual network designs, but can also support contemporary machine learning workflows for the analysis of temporal data.

Chapter 5

Exploring neuromorphic hardware

The approaches presented so far focus on analysing SNNs as a whole, examining the neuron models, the applications, architectures and comparison metrics [351, 306]. However, to fully understand the opportunities and potential offered by neuromorphic systems, it is essential to extend this study to how software frameworks and hardware systems process information [35].

In this context, benchmarking is an essential tool for the systematic and comparative evaluation of the performance for a software framework [352, 353] or hardware architecture [354, 355]. From a methodological point of view, benchmarking constitutes a reference practice for the validation of emerging technologies, as it allows heterogeneous solutions to be compared using objective and reproducible criteria. In particular, the definition of targeted benchmarks makes it possible to verify the correctness of system behaviour with respect to theoretical specifications, to assess the impact of different hardware or software configurations, and to identify bottlenecks that limit overall performance. Such information is fundamental both in the development and optimisation phase of architectures, and in the subsequent adoption phase in real application contexts [356, 357].

In computer science and engineering [358], benchmarking consists of the development and execution of a structured and repeatable set of tests, designed to objectively and comparably evaluate the performance of a system. The tests are defined to measure quantitative and qualitative parameters related to different aspects of system behaviour, such as computational efficiency, energy efficiency, execution speed, scalability, resource utilisation - memory, processing cores, communication

bandwidth - and overall reliability. The results obtained are then statistically analysed in order to quantify the system performance and compare it with that of alternative solutions or baseline values. This procedure makes it possible not only to provide a set of numerical indicators, but also to systematically describe the dynamic behaviour of the system under different operating conditions, highlighting structural limitations and potential margins for improvement [359].

The aim of benchmarking is therefore not limited to the production of performance measures, but extends to the global characterisation of the architecture or framework analysed, providing useful indications for its optimisation. In this sense, benchmarking is configured as a methodology of engineering analysis capable of combining experimental rigour and applicative relevance, acting as a pivotal element in the process of development, validation and technological maturation of complex systems. In addition to simple quantitative measurement, benchmarking also performs a methodological and diagnostic function: it makes it possible to identify performance bottlenecks [360], inefficiencies in resource management and structural limits of architectures or frameworks. Through the comparative analysis of the results, it is possible to direct design choices towards better performing solutions, optimise the balance between energy consumption, latency and throughput, and verify the functional correctness and scalability of the system in realistic operating scenarios.

A crucial aspect of benchmarking is the appropriate choice of metrics and test cases, which must be sufficiently representative of the type of applications for which the system is designed. Within this landscape, a particularly relevant role is played by micro-benchmarking [361–363], a specialised category of tests that aims to measure elementary and low-level system performance by isolating individual components or internal mechanisms that contribute to the overall behaviour. Unlike application benchmarks, which assess overall efficiency in an end-use context, micro-benchmarks focus on specific functional units - such as memory management, transfer latency between cores or synchronisation between processes - thus providing a detailed, analytical view of the system. This approach makes it possible to precisely identify the causes of any inefficiencies and to analyse how different hardware, software or network configurations affect overall performance [364].

Particularly in neuromorphic systems, micro-benchmarking is essential for understanding the behaviour of asynchronous communication mechanisms, distributed

scheduling and spike management, and mapping of neurons to cores, elements that differ profoundly from classical computing paradigms. Thanks to this granularity of analysis, micro-benchmarking not only provides quantitative data on basic performance, but also contributes to predictive modelling of system behaviour, allowing the impact of architectural changes or new mapping strategies of neural networks to be estimated. In this sense, it represents an indispensable tool for the experimental validation and progressive optimisation of neuromorphic platforms, acting as a link between hardware experimentation and algorithmic design.

The study presented here focuses on evaluating the capabilities of the Lava simulation framework. It focuses on aspects of communication between neuron processes through a series of experiments based on MPI and micro-benchmarks. Unlike ANNs, where operations can be distributed evenly across GPUs or clusters, in SNNs spike communication is an asynchronous process that is highly dependent on network connectivity. This means that inadequate mapping of neurons to cores can introduce significant bottlenecks, increasing latency times and energy consumption due to frequent exchanges. [365, 366]. Spike traffic management therefore becomes a design factor that directly affects accuracy and energy efficiency. From this perspective, the framework is not just a software tool, but a critical link between the computational model and the hardware architecture, capable of transforming theory into practice with efficiency and reliability. The tests were conducted on different data exchange scenarios. This laid the foundations for the creation of a test suite capable of providing a systematic measurement of performance, highlighting the strengths and limitations of the framework [44].

5.1 Micro-benchmarking on Lava-Loihi neuromorphic ecosystem

5.1.1 Benchmarking on neuromorphic hardware

Neuromorphic computing requires architectures and design criteria inspired by the functioning of the brain [351]. In recent decades, various efforts have been made to create hardware devices and software frameworks inspired by these principles [315, 367, 155]. Among these devices, a notable example is the Loihi chip,

developed by Intel and first unveiled in 2018 [37], which is now in its second generation [78]. Alongside the Loihi 2 hardware system, Intel has introduced the open-source programming and simulation framework Lava [368, 85], which allows users to develop neuro-inspired applications and run them on both traditional hardware and dedicated platforms.

With the increasing availability and development of neuromorphic technologies, benchmarking plays a fundamental role as a tool for defining common objectives and rigorous methodologies for measuring progress. Consequently, expanding the range of benchmarking tools in the field of neuromorphic computing is of primary importance.

Since both the algorithmic and physical components are still under development, there is an opportunity to advance cross-cutting benchmarking activities through a two-stage approach [369]. Specifically, in the first case, by setting the task and dataset, the algorithm is optimised; in the second case, by setting the dataset and algorithm, the neuromorphic system is optimised. In line with this vision, neuromorphic system benchmarking activities have been discussed by Davies *et al.* [370], who have developed tests on keyword spotting applications, MNIST classification, graph path optimisation, spatio-temporal tactile pattern recognition, CSP optimisation problems such as the Latin square problem and map colouring [371], as well as a comparison between Loihi, SpiNNaker 2 hardware and traditional architectures [324, 327].

However, these benchmarks are adequate for providing an overview of the platform operation through high-level heterogeneous applications. Nevertheless, we believe it is necessary to develop a benchmarking approach that provides a low-level view of the operation of neuromorphic processes simulated or implemented through hardware. This level of detail can be achieved through micro-benchmarking implementations, already widely used to evaluate the performance of the Message Passing Interface (MPI) standard [372, 373] and to measure the efficiency of different architectures, including GPUs [374], clusters [375], memories [362] and applications in cloud environments [376]. Micro-benchmarking has also been adopted for the performance analysis of the SpiNNaker neuromorphic architecture [377, 378].

In this study, we present an activity currently under development: the Lava Micro-Benchmarking Suite (MBS), consisting of a set of tests designed to evaluate the performance of Lava and the Loihi 2 chip. The proposed suite is provided as an extension of the Lava framework and offers a configurable set of tests.

The experiments conducted can be divided into two main types:

- tests aimed at measuring the communication performance of Lava processes running on traditional hardware such as host CPU and embedded CPU on Loihi 2;
- tests aimed at evaluating communication performance and hardware utilisation in the distribution of neural networks within Loihi 2 Neuron Cores (NCs).

It is expected that the results and information obtained from running the suite will contribute significantly to improving both the algorithmic and systemic levels in the field of neuromorphic computing.

Through the Lava MBS, we offer a set of elementary micro-operations that form the building blocks for more complex tasks. The adoption of these micro-operations allows Intel neuromorphic ecosystem to be characterised at different levels, providing information on:

- Lava as a message passing framework;
- Lava compilation process for mapping networks on Loihi 2 chips;
- Loihi 2 hardware architecture, with particular attention to communication mechanisms between NCs;
- hardware backends used during the execution of Lava processes;
- network interconnection between different hardware backends.

Micro-benchmarking activity is an ideal complement to the high-level benchmarking suggested in [240, 370, 379, 371, 324, 327]. This is because, while high-level benchmarking allows to identify situations in which a task or algorithm does not achieve the expected performance on a given system, it does not always provide precise indications on the specific aspect of the architecture that needs optimisation. The results obtained from running micro-benchmarks have the capacity to fill this gap by providing detailed metrics on the behaviour of individual system elements. For instance, the latency in spike transmission as a function of the amount neurons.

Neuromorphic computing inherently involves sparse, asynchronous communication between elementary units designed to implement neurobiologically inspired functionalities. The resulting improvement in energy-delay product achieved through the use of dedicated hardware highlights the importance of evaluating communication performance within the system [240]. In particular, in real-time on-edge application scenarios, it is essential to analyse the exchange of information between nodes in order to estimate - and where possible reduce - latency in local communications. An effective approach to achieving this goal is to exploit elementary micro-operations in order to measure the fundamental contributions that contribute to higher-level activities related to data transmission.

Although Lava is not directly based on MPI and integrates specific features for neuromorphic execution that adopts message passing mechanisms via channels between asynchronous processes based on the communicating sequential processes model, the concepts derived from MPI benchmarking are still useful for evaluating the communication performance between processes that share data via message exchange and micro-benchmarking implementations [85]. In particular, the Intel MPI Benchmarks (IMB) [372] and the Ohio State University Micro-Benchmarks (OSU-MB) [373] are taken as references.

To characterise the Loihi 2 hardware, we use the profiling tools available within the Lava framework, which allow to obtain various execution metrics, including the duration of each synchronisation phase at each timestep. In this analysis, we focus in particular on the hardware utilisation ratio during the deployment of SNNs and the duration of the spiking phase by testing different sizes and topologies.

Lava neuromorphic ecosystem

Lava is an open-source framework developed in Python by Intel, designed for the creation and deployment of neuromorphic applications on heterogeneous architectures. The main features of Lava that are relevant to the MBS are presented below.

Processes and models In Lava, *Processes* represent abstract entities that define their interface exclusively in terms of communication *Ports* and internal variables. The concrete implementation of a process is provided through *Process Models*. A single process can therefore be associated with multiple models, each characterised

by specific properties and intended for different hardware backends. The framework selects the most appropriate *Model* for each *Process*, based on the available hardware configuration and the preferences set by the user. *Processes* can in turn be interconnected to compose complex applications.

Types of models Since Lava supports different hardware backends, different types of models are required. Currently, there are three main types: `PyLoihiProcessModel`, for Python models running on host CPUs; `CLoihiProcessModel`, for models in the `C` language running on the Lakemont (LMT) embedded processor integrated into Loihi 2; `NcModel`, for models running directly on Loihi 2 NCs. Depending on the type of scenario, the user must define the corresponding models for each hardware backend they intend to support.

Compiler Since different backends require specific compilation procedures, in order to execute *Process Models*, Lava is able to delegate this operation to external libraries, so that they can be executed on the corresponding hardware backend. For example, for *Models* written in the `C` language, compilation is performed using `gcc`, while for models intended for execution on NCs, the `NxCore` library is used.

Processes interconnection The interconnection between *Processes* occurs through *Ports* connections by means of *Channels*. An operational connection is guaranteed even if the *Processes* are executed on different backends, provided that this type of connection is supported. Currently, Lava allows the following connection types: `CPU-CPU`, `CPU-LMT`, `LMT-NC` and *NC-NC*.

Types of communication Ports can be divided into two main categories, corresponding to two different modes of communication:

- *Point-to-Point* using `InPorts` and `OutPorts`: the communication channel is establish by connecting the output port of a process with the input port of the receiver process. With this binding using the `send()` and `recv()` APIs the two processes can exchange information via message-passing in a *point-to-point* fashion;

- *One-sided* using `RefPorts` and `VarPorts`: reference port is connected to the variable port, this allow a process to directly access the internal variable of another process, using the APIs `read()` and `write()`. However this communication is known as *One-sided* because only the accessor process is explicitly involved.

Data representation The two modes of data transmission between `Process` can be `dense` or `sparse`. In the `dense` case, the entire array is sent in its complete form, thus including null elements. In the `sparse` representation, on the other hand, only certain elements of the array that are not null are transmitted, thus reducing the volume of data to be transferred. This second mode requires the management of two separate arrays: the first containing the actual values; the second containing the indices that specify their position within the original array.

Synchronization protocol In the Lava system, the communication between *Processes* operating in parallel and communicating asynchronously via the exchange of message tokens enables the identification of distinct *Phases*. The progression of these phases is contingent upon the completion of all *Processes* operations within each timestep, prior to the progression to the subsequent *Phases*. In particular, Lava implements the `LoihiProtocol`, which adheres to the execution phases defined by the Loihi 2 architecture. The main phase is called *Spiking*, in which the NCs perform the calculation and verify the activation conditions for generating a spike. This phase is mandatory, unlike the other optional phases of the protocol, namely: *Pre-management*, *Learning*, *Post-management* and *Host*. For processes running on a host CPUs, Lava also supports the `AsyncProtocol` protocol, which does not require synchronisation phases. In this case, processes can operate at different speeds, exchanging messages at any time, without time alignment constraints.

Loihi 2 architecture

The Loihi 2 chip consists of 128 fully asynchronous NCs and 6 embedded LTM `x86` microprocessors, supporting up to 1 million neurons and 120 million synapses [368].

NCs enable the implementation of spiking neuron groups and associated synaptic connections, communicating with each other via a network-on-chip that uses data exchange exclusively in the form of spike messages.

The microprocessor cores also enable spike-based communication and execute standard `C` code, performing I/O interfacing, network configuration, management and monitoring tasks. Compared to the first generation, Loihi 2 introduces new features, including: the ability to implement custom neuron models using assembly microcode instructions; support for the generation and transmission of graded spikes; compatibility with three-factor learning rules.

Experimental setup

The Intel vLab server was used to run the tests. It acts as a host CPU and provides access to neuromorphic boards, including the Oheo Gulch board, which is used for experiments requiring a Loihi 2 backend. This board is equipped with a single Loihi 2 chip, enabling characterisation and debugging operations. The software infrastructure is based on Python (version 3.10.4) as the main language and the open-source Lava library (version 0.6.0), available in the GitHub repository.

5.1.2 Micro-benchmarking suite architecture

The benchmarking suite was developed in Python, which offers established libraries for data management, analysis and visualisation, examples of which include NumPy, Pandas and Matplotlib. The development of micro-operations was conducted following the official guidelines of the Lava framework, with the aim of making the suite potentially integrable into the official repository of the framework. These guidelines include: adoption of the PEP8 style, use of linting tools, systematic use of type hints, use of docstrings in NumPy format, and creation of unit tests [85].

From a structural point of view, the MBS consists of two main modules: the first is responsible for executing the tests selected by the user; the second is the data analysis tool, which processes the results obtained and allows the generation of graphs and visualisations according to the user needs.

Python natively supports object-oriented programming, a feature that is particularly advantageous for ensuring code modularity and extensibility. These are

fundamental aspects in a benchmarking suite that must be easily expandable and maintainable over time. The basic components are implemented as classes, which can be extended to adapt to the specifics of each test using inheritance mechanisms. Where possible, the same basic modules are reused in multiple tests, thus reducing redundancy in the code.

To ensure statistically significant results, each test can be performed multiple times, with the number of repetitions configurable by the user. For each test category, a class called *test maker* has been defined, responsible for the automated execution of the test according to the different combinations of parameters and iterations required. This class collects all the measurements produced and organises them using basic Python data structures such as lists and dictionaries, which can also be saved to disk for further processing. The collected data is then passed to the analysis and visualisation module, which converts it into a Pandas DataFrame for easy manipulation.

Through the application of statistical methods, the measurements are processed to extract summary information relevant to the production of interactive graphs, which are used to generate an HTML report on the test results.

MPI-inspired tests

Table 5.1 summarizes the parameter configurations of the MPI-inspired tests. These type of tests cover specific features of the Lava framework. Example of that are:

- data representation in *Ports* that can either be *dense* or *sparse*;
- synchronization protocol can be `LoihiProtocol` or `AsyncProtocol`. This second is not supported by LMT processes, but it is only applicable for the `CPU-CPU` connection;
- communication type *point-to-point* or *one-sided*.

Single transfer tests Single transfer tests are inspired by the IMB benchmarks [372] *single transfer* and the OSU-MB benchmarks [373] *point-to-point* tests. They involve the interaction of two active processes, called *Player A* and *Player B*, which

Category	Tests	Parameters	Connection	Variants
Single transfer	- PingPong - PingPing	Message size	CPU-CPU(*) CPU-LMT	Dense or Sparse Loihi or Async(**) P2P or One-sided
Multiple transfer	- Unidirectional - Bidirectional	Message size No. of message		
Parallel single transfer	- Par. PingPong - Par. PingPing - Periodic chain - Bidirectional periodic chain	Message size No. of processes		
Parallel multiple transfer	- Parallel unidirectional	Message size No. of messages No. of processes		
Collective transfer	- One to many - Many to one	Message size No. of processes		

(*) Only one process can be configured to run on the LMT processor.
(**) The Asynchronous protocol is only supported for CPU-CPU communication.

Table 5.1 MPI implemented tests. The table is taken from [44].

communicate exclusively with each other through the exchange of elementary messages. In *Process* Lava messages are defined by the user, who has the option of configuring the size of the array by specifying the number of 32-bit integers to be sent, or supplying a list of sizes, in which case the test is executed iteratively for each value indicated.

PingPong test In the *PingPong* test, the process *Player A* sends a single message *Ping* to the process *Player B*. The latter, upon receiving the message, responds by sending a return message *Pong* of the same size. The time measurement is performed by *Player A*, calculating the interval between sending the *Ping* and receiving the corresponding message *Pong*.

PingPing test In the *PingPing* test, the process *Player A* sends a single message (*Ping 1*) to the process *Player B*, while the latter simultaneously sends a message (*Ping 2*) to *Player A*. The time measurement is performed by *Player A*, considering the interval between the sending of *Ping 1* and the receiving of the message *Ping 2* (Figure 5.1).

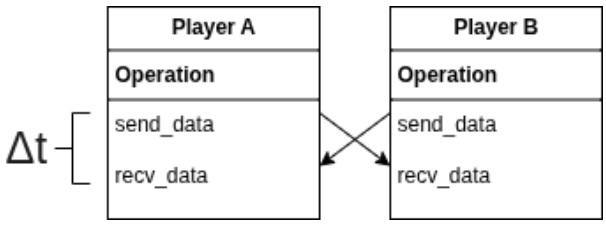

Fig. 5.1 PingPing structure test. Reworking based on [44].

Multiple transfer tests The multiple transfer tests are inspired by the *point-to-point multiple transfer* benchmarks of the OSU-MB suite [373]. In this scenario, two active processes participate in the exchange of multiple information in which one or both processes send multiple messages in sequence. In this type of test, the user can define the size of the individual messages, and the message stack to be transmitted, i.e. the number of consecutive messages sent in a single test run. To enable the test to be executed with different configurations, the user may provide a list of integer pairs, each specifying the size of the messages and the total number of messages to be sent.

Unidirectional In the unidirectional test, the process *Player A* sends a sequence of consecutive messages, all of the same size. The process *Player B* receives the messages one after the other, sequentially. Once reception is complete, *Player B* responds with a single return message, having the same size as the messages sent by *Player A*. Time is measured by considering the interval between the sending of the first message of the stack from *Player A*, and the reception of the reply message *Player B*. This test makes it possible to estimate the sustained transmission rate obtainable between two isolated processes (Figure 5.2).

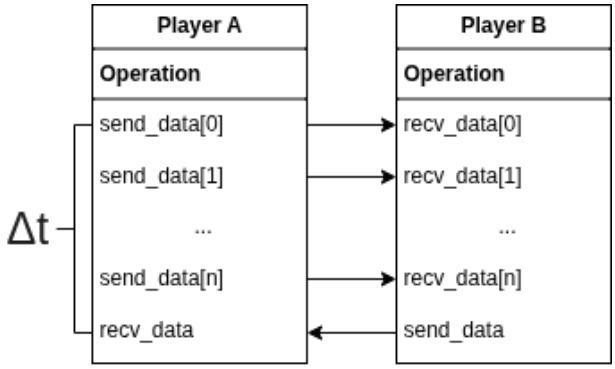

Fig. 5.2 Unidirectional structure test. Reworking based on [44].

Bidirectional In the bidirectional test, both *Player A* and *Player B* send a sequence of consecutive messages, all of the same size. Each process receives the messages sent by its counterpart and, once reception is complete, responds with a single return message. The time is measured by *Player A*, considering the interval between sending the first message in the stack and receiving the reply. This test makes it possible to estimate the maximum aggregate bandwidth sustainable in the exchange of data between two processes.

Parallel transfer tests The parallel transfer tests take inspiration from the *parallel transfer* benchmarks of the IMB suite [372], combined with the *multi-pair* tests of the OSU-MB suite [373]. In this case, more than two processes are active simultaneously, performing data transfer operations in parallel.

Parallel PingPong, PingPing and Unidirectional tests Processes are divided into two groups *rank A* and *rank B*. Each process belonging to *rank A* communicates exclusively with its counterpart in *rank B*. The pairs of processes independently and simultaneously perform the required test (*PingPong*, *PingPing* or *Unidirectional*). Time is measured separately for each pair, allowing performance to be analysed both individually and in global terms.

Periodic chain Processes are organised in a way to form a periodic communication chain. Each process sends a message exclusively to the first consecutive neighbour and receives a message from the first preceding neighbour, realising a clockwise circular transmission. Time is measured locally in each process, considering the interval between the instant of sending and the next reception (Figure 5.3).

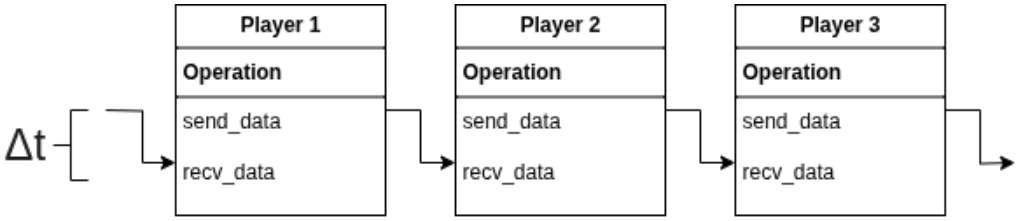

Fig. 5.3 Periodic chain with three processes. Reworking based on [44].

Bidirectional periodic chain In this case, the processes are arranged according to the same circular structure as described for the *periodic chain* tests, but each process simultaneously sends a message to both previous and consecutive first neighbours, waiting to receive a response from both. Also in this scenario, time measurement is conducted within each process, calculating the interval between sending the first message and receiving the second.

Colletive transfer tests This category of tests exploits Lava ability to split or merge communication channels to support *one-to-many* and *many-to-one* configurations. The approach is inspired by the *collective benchmarks* defined in the IMB [372] and OSU-MB suites [373].

One-to-many A single process, referred as *Player A*, sends a message to a set of n recipient processes {*Player* B_1, ..., *Player*B_n}, Lava allows a single output port to be connected to multiple input ports, thus ensuring automatic distribution. Each receiving process responds in turn with a single message to *Player A*. Time is quantified by *Player A*, considering the interval between sending the message and receiving the last reply.

Many-to-one In this configuration, a set of n transmitting processes {*Player* A_1, ..., *Player* A_n} send their messages to a single receiving process, named *Player B*. The latter receives data via a single input port through which incoming messages are combined by Lava using an element-by-element summing operation. Once the data has been received and aggregated, *Player B* sends a reply to *Player* A_1, which takes care of the time measurement, calculating the interval between sending its message and receiving the reply.

Loihi 2 tests

To estimate the performance of the Loihi 2 hardware, a series of specific tests were designed, summarised in Table 5.2, using SNNs characterised by different topologies, sizes and neuronal patterns as analysis tools, covering the use of both NCs and LMT processors.

Category	Tests	Parameters	Connection	Variants
NC SNNs	- Fully connected	NC-NC	No. of layers No. of neurons Active neurons	Hcoded LIF Ucoded LIF Ucoded SDN
	- Convolutional - One-to-one	NC-NC	No. of layers No. of neurons Kernel size	
SNNs with LMT	LMT as spiker	LMT-NC-NC	No. of layers No. of neurons	Hcoded LIF
	LMT as receiver	NC-NC-LMT		
	LMT as adapter	CPU-LMT-NC		

Table 5.2 Loihi 2 specific tests. The table is taken from [44].

The Lava framework provides a set of tools called *Profilers*, which allow the collection of metrics related to the execution of a network configured and executed by a supported backend, such as the Loihi 2 chip. Data collected includes execution time, resource utilisation, memory utilisation, spiking activity and power consumption. Measurements are performed directly by the Loihi 2 hardware, with negligible or no overhead. Results are transmitted to the host PC via Lava, from which they can be acquired and analysed by the MBS.

During the analysis, the focus was brought to the execution time metric. The objective is to analyse how the calculation performed by processes and the spiking activity influence the duration of a timestep via the synchronisation protocol `LoihiProtocol`. By means of this protocol, execution is divided into sequential phases and advancement to the next instant only occurs when all processes in the network have completed the current phase. The duration of a timestep depends on the time of the executed phases. In the case of SNNs, the most important phase is the *spiking* phase, in which both neural processing and spike transmission take place. The tests have been designed in a way that for each timestep it is easy to identify which neurons have generated spikes.

The tests are conducted by defining SNNs such that the parameters associated with them allow the generation of an equal number of timesteps containing spikes and timesteps without spikes. From these measurements, the timesteps without spikes are then evaluated using the average of the timesteps to assess the impact of the computation performed by each node on the overall duration, while the timesteps with spikes allow the communication performance between nodes to be analysed.

SNNs running on Loihi 2 neuron cores In this first set of micro-benchmarking on SNNs, configurations of networks exploiting only the Loihi 2 NCs were realised, with the aim of probing the number of neurons per NC, the amount of memory allocated to each NC, evaluating how the networks are mapped by the compiler onto the hardware, and the execution time and resource consumption. The Loihi 2 platform provides a maximum number of neurons together with a maximum amount of memory that can be allocated to each NC. To probe this limit in the benchmarks, various network configurations were created by acting on the number of layers and the number of neurons, testing networks with increasing size up to the limits imposed by the architecture and observing which resource is saturated first. On the other hand, resource consumption is tested by means of neurone models that modify the execution time in each NC. Currently, Lava provides two implementations of the LIF neuron: the first in hardware, called *hard-coded*; and the second via software simulation, called *micro-coded*. A further model supported is the Sigma-Delta Neuron (SDN) model via a *micro-coded* implementation. Running the same tests with different models allows a direct comparison of performance in terms of time efficiency and resource utilisation. The input with which the network is stimulated is generated by a Lava process of type *Spiker*, configured to emit spikes at regular intervals and executed on a single NC. The time interval between consecutive spikes is chosen so that each pulse has time to propagate to the last layer before the next one is generated. Synaptic weights and neuronal parameters are configured so that a neuron produces only one spike in response to the received stimuli. This ensures that the spikes generated by the *Spiker* process propagate neatly from one layer to the next until they reach the last layer of the network.

Fully connected SNN In this topology, all layers that compose the network contain the same number of neurons, both parameters being user-definable during the configuration phase. Each neuron of a layer is connected to all neurons of the next layer using a Lava *Dense* process. An example with LIF neurons is shown in Figure 5.4. This configuration is designed to evaluate the performance of the Loihi 2 hardware in scenarios characterised by a high density of connections between neurons, in which the available synaptic memory tends to be exhausted before the maximum capacity of allocable neurons in each NC. The suite also makes it possible to reduce the percentage of active neurons in each layer, making it possible to analyse the extent to which the level of spiking activity affects communication performance.

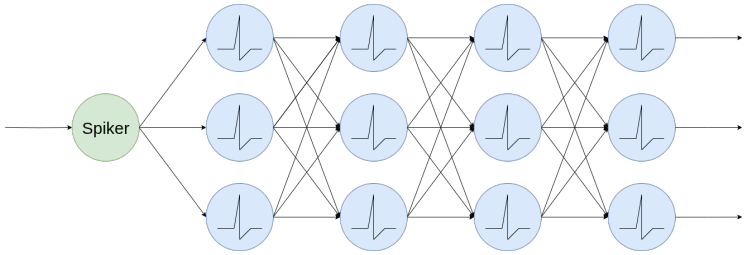

Fig. 5.4 Fully connected network with 3 layers and 3 neurons per layer. Reworking based on [44].

Convolutional SNN In this topology, the connection between layers is realised by means of the *Convolutional* process, which allows connections to be defined by means of a multidimensional matrix, called *filter*, which contains the weights used during the convolutional operation. This configuration allows the user to analyse the performance of convolutional connections as the network and filter sizes vary.

SNN One-to-One This test represents a special case of the convolutional configuration, in which the filter adopted is a 1×1 matrix. The resulting topology, illustrated in Figure 5.5, provides that the i^{th} neuron of one layer is exclusively connected to the i^{th} neuron of the next layer. This significantly reduces the overall number of connections, and consequently the amount of synaptic memory required by each neuron. This configuration is useful for verifying the behaviour of the platform in the event that the maximum allowed number of neurons is mapped onto the individual NC.

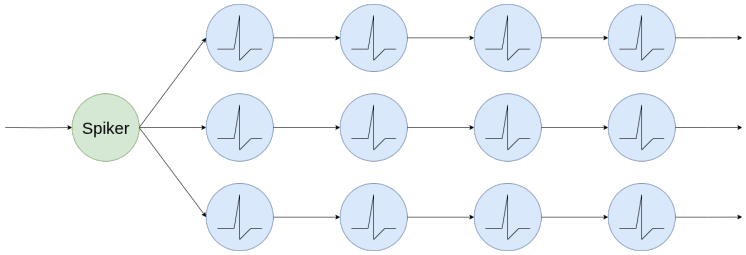

Fig. 5.5 One-to-one network with 3 layers and 3 neurons per layer. Reworking based on [44].

SNN with LMT This category of tests is only aimed at evaluating the LMT processor when integrated within an SNN, testing only the fully connected topology consisting of only LIF neurons. Here, code execution on the LMT, communication

between the LMT and NCs, and communication between the LMT and the host CPU are analysed.

LMT as Spiker In this scenario, the LMT processor is configured via *C* code to execute a *Process* that generates spikes at regular intervals, which are then propagated to the rest of the network. The obtained performance is compared with network executions performed exclusively on NCs, in order to verify whether the communication between the first layer of neurons and the communication between the LMT processor introduce potential bottlenecks.

LMT as Receiver In this configuration, the LMT is connected to the network output and receives the spikes produced by the last layer. Since the processor does not perform any processing on the received data, this test allows the impact of the communication between the NCs and the LMT processor on the duration of the spiking phase to be evaluated in isolation.

LMT as Bridge This test is designed to evaluate the spike transfer between Python models running on the host CPU and NCs, using the LMT as an interface element. The Lava processes *PyToNxAdapter* and *NxToPyAdapter* are used for this purpose. The network is divided into two sections in which one part is executed on the CPU by means of Python templates and the remainder on the core NCs of Loihi 2, while LMT takes care of the translation and transfer of messages between the two parts.

Time measurements

CPU-CPU For communication between CPU processes, Lava uses `SharedMemory` objects provided by Python `multiprocessing` module. The measurement of transmission times is carried out by means of the `perf_counter_ns()` function of the `time` module of Python, which provides a high-resolution measurement in nanoseconds.

CPU-LMT Communication between the Python process running on the host CPU and the *C* process running on the LMT processor is fully managed by Lava, provided

the declared interfaces are compatible. Timing is measured using the same approach as for the `CPU-CPU` tests, i.e. by the process or processes running on the host PC.

NC-NC and NC-LMT These two types of connections take place entirely within the Loihi 2 hardware, so metrics are captured using the profiling tools provided by Lava.

LMT-LMT Lava supports the execution of only one process running on the LMT embedded processor. Consequently, it is not possible to perform single or multiple transfer tests with both processes running on the LMT. This limitation of Lava also affects MPI-inspired parallel and collective tests, as only one of the processes involved can be executed on the LMT.

CPU-NC Lava does not allow direct communication between the host CPU and the NCs of Loihi 2.

5.1.3 Experimental results

This represents the first micro-benchmarking effort designed specifically for the Lava framework and Loihi 2 hardware, to the best of our knowledge. Although the realisation is still in its infancy, progress has already been made in architectural design and the realisation of a pivotal set of tools. The first implemented tests cover the three available hardware backends: host CPU, embedded LMT processor and NCs.

Execution examples

The MPI tests belonging to the *PingPong PingPing* and *Unidirectional*, conducted using the host CPU and the LMT embedded processor of Loihi 2, are reported below. No detailed analysis of the results is carried out, as this is beyond the scope of this work: the interpretation of the data is left to the end user, depending on the specific objectives and characteristics of the tests performed.

Single transfer tests Figure 5.6 and Figure 5.7 show, respectively, the transmission time as a function of message size for connections of type `CPU-CPU` and `CPU-LMT`. The tests were performed with data sizes ranging from 1 to 2048 elements, repeating each configuration 25 times. The results show no clear correlation between transmission time and message size. Instead, a clear difference between the two connection types can be observed: the `CPU-LMT` communication is approximately three orders of magnitude slower than its `CPU-CPU` counterpart.

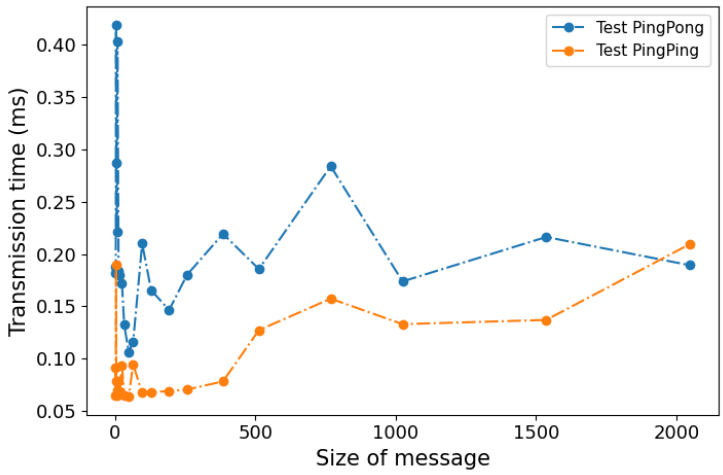

Fig. 5.6 Transmission time for CPU-CPU single transfer tests as a function of the message size. Reworking based on [44].

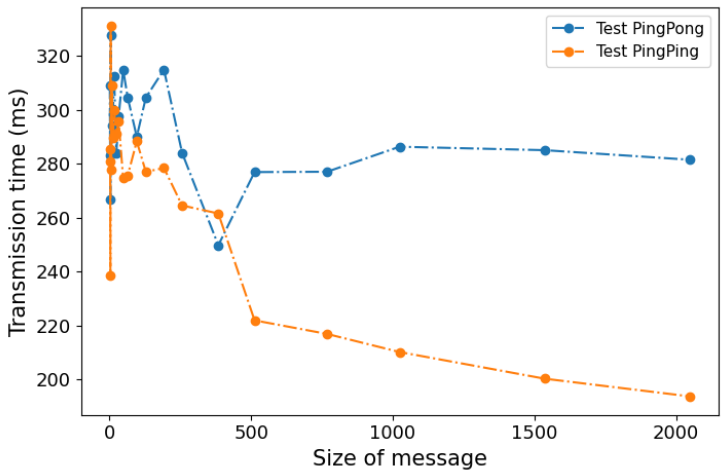

Fig. 5.7 Transmission time for CPU-LMT single transfer tests as a function of the message size. Reworking based on [44].

Unidirectional tests Figure 5.8 and Figure 5.9 illustrate the transmission time as a function of the message size and the number of messages sent, respectively, for the connections `CPU-CPU` and *CPU-LMT*. In both cases, the data size varies between 1 and 512 elements. For the `CPU-CPU` connection, the number of messages was configured between 1 and 512, while for the *CPU-LMT* connection, this range was limited by the Lava constraints that supports a maximum of 32 messages for this type of communication. Each combination of size and number of messages was executed 25 times. The results show a correlation between transmission time and number of messages, with an exception in the case of the *CPU-LMT* connection when 32 messages are used. Similarly to what was observed in the single-transfer tests, the *CPU-LMT* communication in this scenario is also approximately three orders of magnitude slower than the *CPU-CPU* communication.

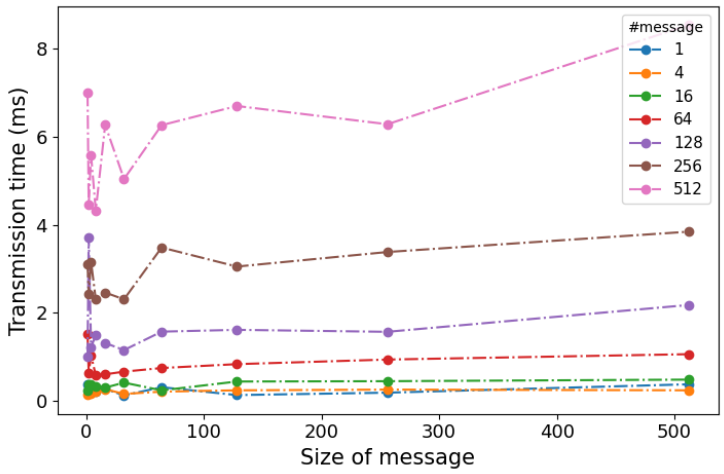

Fig. 5.8 Transmission time for CPU-CPU unidirectional multiple transfer test as a function of the message size, for several number of messages. Reworking based on [44].

5.1.4 Discussion

The suite is modular and extensible, allowing the addition of new tests or support for additional hardware backends. Interaction with the suite is via high-level Python APIs, which allow users to select the tests to be run and generate HTML reports containing the collected data. Although the suite, in its current state, is fully functional and executable for a set of already defined tests, as it is still a project in the development phase, there is room for improvement.

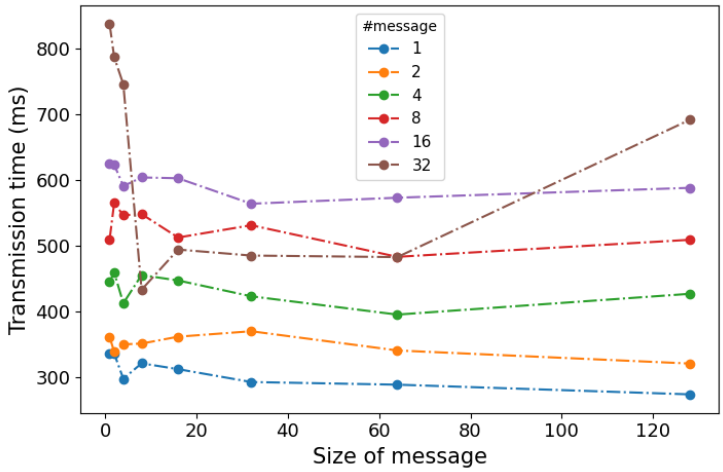

Fig. 5.9 Transmission time for CPU-LMT unidirectional multiple transfer test as a function of the message size, for several number of messages. Reworking based on [44].

Further tests will have to be designed to extend the coverage of the functionalities offered by Lava and the Loihi 2 hardware, including the use of hierarchical processes in Lava, the use of virtual gates and transformations on the data, on-chip learning with particular reference to *Dense* and *LIF* processes, and the adoption of customised neurons described in microcode and multi-chip connectivity.

The report generation system can also be further improved, relating to the configuration parameters of the tests performed. A planned extension consists of the integration of more expressive types of graphs and additional statistical information. Finally, we intend to increase the flexibility available to the user by allowing customisation of the graphs to be generated via a dedicated configuration file.

We briefly present below two possible scenarios that show how the results obtained from running the suite can be used.

Lava/NxCore developer

A developer in Lava or `NxCore` may be interested in study the behaviour of the Lava framework and the `NxCore` compiler as the number of neurons and layers to be mapped on the Loihi 2 hardware changes. By running SNN tests with different parameter configurations, the developer can identify potential compiler criticalities, e.g. in terms of sub-optimal memory or core utilisation.

Lava user

A user of the Lava framework engaged in the development of an application requiring real-time data processing may need to understand how the spike propagation time varies according to the size of the network and the neuron model employed. Performing SNN tests allows to determine, for each neuron model, the maximum network size that still allows the time constraints required by the application to be met.

5.2 Summary

The aim of this chapter is to explore neuromorphic hardware platforms and software frameworks, addressing a key aspect of practical neuromorphic computing: systematic performance evaluation through benchmarking. Whilst preceding analyses have examined neuromorphic systems from the perspectives of neural dynamics, manual network design and data-driven machine learning applications, the present part will focus on the underlying computational infrastructure that enables such approaches to be implemented and evaluated under real-world conditions.

The present work addressed the paucity of specific micro-benchmarking tools for neuromorphic systems by proposing a dedicated benchmarking suite for the Lava framework and the Loihi 2 neuromorphic architecture. In contrast to application-level benchmarks, which focus on end-to-end performance for specific tasks, the proposed micro-benchmarks are designed to isolate and characterise individual system components, such as message-passing mechanisms, compilation processes, and execution backends. This detailed approach facilitates a more profound comprehension of the manner in which architectural and software design decisions influence performance, scalability, and efficiency.

By targeting multiple execution backends, including the host CPU, the embedded CPU, and the Loihi 2 neuromorphic cores, the benchmarking effort provides a comprehensive view of system behaviour across heterogeneous computational layers. The results of the study highlight the strengths and limitations of the current neuromorphic ecosystem, offering concrete insights for developers and users of Lava and Loihi 2. Moreover, the proposed benchmarking suite modular and extensible nature

establishes a framework for future expansion, thereby facilitating the evaluation of additional functionalities and emerging neuromorphic features.

The capacity to interpret neuronal dynamics, design effective spiking networks, and implement learning-based neuromorphic pipelines is contingent on a comprehensive understanding of the hardware and software platforms on which these systems operate. In this sense, benchmarking serves not only as a measurement tool but also as a methodological bridge between theory, algorithm design, and practical implementation. The insights gained through this exploration contribute to a more informed and systematic approach to neuromorphic systems design, supporting the maturation of neuromorphic computing from experimental prototypes to robust and scalable computational platforms.

Chapter 6

Conclusions

The research presented is part of an analysis of the neuromorphic computing paradigm, which represents a discontinuity with respect to the traditional computing model proposed by von Neumann, that has guided the development of processing systems for over seventy years. Unlike the classical approach, characterised by a clear separation between memory and processing and a sequential execution model, neuromorphic computing is inspired by the functioning of the human brain, proposing a distributed, asynchronous and event-driven architecture capable of combining energy efficiency, massive parallelism and intrinsic robustness.

The primary goal of the research activity has been to investigate the potential of this emerging paradigm through the design and development of applications capable of highlighting the strengths and challenges of neuromorphic computing. The study was conducted on several complementary levels, ranging from the theoretical analysis of neural models and their dynamical properties to experimental validation on latest-generation neuromorphic hardware platforms. The research activity included: low-level study of the behaviour of individual spiking neurons and network dynamics, addressed through mathematical and simulation tools; the design of applications aimed at solving optimisation problems; classification and benchmarking.

This multi-level approach has not only allowed us to deepen our understanding of the theoretical foundations of neuromorphic computing, but also to test its effectiveness in practice, comparing it with traditional models in terms of both performance and energy efficiency. In particular, the objective was twofold: on the one hand, to demonstrate the validity and effectiveness of neuromorphic architectures

in addressing complex problems in different application domains and, on the other, to contribute to the definition of design methodologies that can make such systems more accessible and transferable to real-world scenarios.

This cross-cutting perspective has made it possible to explore a wide range of application fields, ranging from embedded machine learning for edge and IoT devices to solving combinatorial and complex optimisation problems. The focus has been on identifying contexts where neuromorphic computing not only represents a computational alternative, but can also be a truly competitive and sustainable solution compared to conventional architectures, both in terms of efficiency and scalability.

The areas analysed cover different levels of the neuromorphic paradigm, demonstrating not only the versatility but also the concrete potential of these technologies in addressing problems of a heterogeneous nature. In the domain of combinatorial optimisation, approaches based on manual SNN design have been developed and tested, capable of effectively addressing complex problems such as MAX-CUT and the validation of instances of the Latin Square Problem. These solutions have highlighted how mechanisms inspired by SA, attractor dynamics and constraint satisfaction principles can be translated into forms of distributed neural dynamics, maintaining the robustness and efficiency typical of neuromorphic systems. This result highlights the possibility of using SNNs not only as biologically plausible models, but also as actual computational tools for solving NP-hard problems, opening up interesting prospects from both a methodological and an applicative point of view.

In the field of classification, addressed through machine learning approaches oriented towards the neuromorphic paradigm, attention has focused on applications such as HAR and audio signal processing. In this context, neuromorphic pipelines have been designed and analysed, including pre-processing stages inspired by biological models such as cochlear filter banks, the transformation of signals into spike trains through different encoding strategies, and the use of sCNNs for classification. The systematic comparison of different spike encoding techniques and network configurations has highlighted their respective strengths and limitations, thus providing a useful methodological framework to guide the choice of the most suitable pipeline depending on the application domain and the nature of the input data. The results obtained demonstrate how SNNs can provide an advantageous compromise between prediction accuracy, energy efficiency, and scalability on edge devices, showing the

concrete possibility of using neuromorphic computing in real-world scenarios of distributed artificial intelligence and IoT.

Finally, in terms of low-level analysis, an in-depth investigation was conducted on spiking neuron models and their dynamics, with particular attention to the methods of representation and interpretation of neuro-computational features. To this end, the NePhaM tool was introduced and applied, offering an intuitive yet rigorous visualisation of the dynamic transitions of neural models as parameters vary. This approach not only improves understanding of the internal mechanisms of spiking models, but also more effectively supports the design and prototyping of neuromorphic networks for specific applications, acting as a bridge between theoretical analysis and practical implementation. At the same time, extensive micro-benchmarking was carried out on next-generation hardware platforms and software frameworks, such as Loihi 2 and Lava, aimed at systematically measuring performance at different computational levels (host CPU, embedded CPU and neural cores). This dual approach has made it possible to highlight both architectural and software strengths and weaknesses, contributing on the one hand to a more solid understanding of the real capabilities of neuromorphic technology, and on the other to outlining more clearly the challenges still facing its large-scale deployment.

Although the results obtained represent only a first step in a field of research that is still young and rapidly evolving, they constitute a set of concrete contributions of functioning applications, capable of demonstrating not only the technical feasibility but also the potential practical effectiveness of the neuromorphic paradigm. The experiments conducted have made it possible to validate both manual and data-driven design methodologies, explore real application scenarios, and analyse behaviour at the architectural and model level in depth. These results testify to the growing maturity of these technologies and confirm their relevance as an alternative to traditional architectures, paving the way for their use in contexts where energy efficiency, massive parallelism, and distributed robustness are fundamental requirements.

The activities carried out have made it possible to explore the neuromorphic paradigm in various forms, demonstrating how this approach can provide efficient and versatile solutions to heterogeneous problems, such as in the field of optimisation, classification and low-level analysis. The developments presented certainly do not exhaust the possibilities offered by neuromorphics, but they provide a clear picture of the progress made and the potential still to be explored. Looking ahead, a

natural extension is to strengthen the link between theoretical models and practical applications, with the aim of translating the tools and frameworks available today into operational solutions in real contexts, with a particular focus on scenarios characterised by time and resource constraints, typical of edge computing. The extension of benchmarks to industrial use cases, the exploration of more sophisticated coding techniques and the scalability of architectures to multi-chip systems are just some of the promising directions. In this perspective, the research presented here can be seen as part of a broader process that aimed at bringing neuromorphic computing from the laboratory to the world of applications, thus helping to consolidate its role as a new-generation computational paradigm.

References

[1] John von Neumann. Introduction to "the first draft report on the edvac". *Annals of the History of Computing*, 15(1):11–21, 1993.

[2] Gerard O'Regan and O'Regan. *A brief history of computing*. Springer, 2008.

[3] Kurt Normark. Overview of the four main programming paradigms. *Aalborg University*, 2013.

[4] Peter Van Roy et al. Programming paradigms for dummies: What every programmer should know. *New computational paradigms for computer music*, 104:616–621, 2009.

[5] Brian W Kernighan and Dennis M Ritchie. The c programming language second. *Prentice Hall*, 1988.

[6] Marcelo Arenas, Martin Muñoz, and Cristian Riveros. Descriptive complexity for counting complexity classes. *Logical Methods in Computer Science*, 16, 2020.

[7] Alan Mathison Turing et al. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[8] Mark Skilton and Felix Hovsepian. *The 4th industrial revolution*. Springer, 2018.

[9] Xingqi Zou, Sheng Xu, Xiaoming Chen, Liang Yan, and Yinhe Han. Breaking the von neumann bottleneck: architecture-level processing-in-memory technology. *Science China Information Sciences*, 64(6):160404, 2021.

[10] Gil Speyer, Natalie Freed, Richard Akis, Dan Stanzione, and Eric Mack. Paradigms for parallel computation. In *2008 DoD HPCMP Users Group Conference*, pages 486–494. IEEE, 2008.

[11] Marco Vanneschi. Parallel paradigms for scientific computing. In *Reaction and Molecular Dynamics: Proceedings of the European School on Computational Chemistry, Perugia, Italy, July (1999)*, pages 168–181. Springer, 2000.

[12] Koushik Mondal and Paramartha Dutta. Big data parallelism: challenges in different computational paradigms. In *Proceedings of the 2015 Third International Conference on Computer, Communication, Control and Information Technology (C3IT)*, pages 1–5. IEEE, 2015.

[13] Tim Rentsch. Object oriented programming. *ACM Sigplan Notices*, 17(9):51–57, 1982.

[14] Jeremy Fowers, Greg Brown, John Wernsing, and Greg Stitt. A performance and energy comparison of convolution on gpus, fpgas, and multicore processors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 9(4):1–21, 2013.

[15] Sergio Bernabe, Sergio Sanchez, Antonio Plaza, Sebastián López, Jón Atli Benediktsson, and Roberto Sarmiento. Hyperspectral unmixing on gpus and multi-core processors: A comparison. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 6(3):1386–1398, 2013.

[16] Gerassimos Barlas. *Multicore and GPU Programming: An integrated approach*. Elsevier, 2014.

[17] Mark Baker and Rajkumar Buyya. Cluster computing at a glance. *High Performance Cluster Computing: Architectures and Systems*, 1(3-47):12, 1999.

[18] Chee Shin Yeo, Rajkumar Buyya, Hossein Pourreza, Rasit Eskicioglu, Peter Graham, and Frank Sommers. Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers. In *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies*, pages 521–551. Springer, 2006.

[19] Giorgio Luigi Valentini, Walter Lassonde, Samee Ullah Khan, Nasro Min-Allah, Sajjad A Madani, Juan Li, Limin Zhang, Lizhe Wang, Nasir Ghani, Joanna Kolodziej, et al. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, 16(1):3–15, 2013.

[20] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. Cloud computing: An overview. In *IEEE international conference on cloud computing*, pages 626–631. Springer, 2009.

[21] Won Kim. Cloud computing: Today and tomorrow. *J. Object Technol.*, 8(1):65–72, 2009.

[22] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE international conference on smart cloud (SmartCloud)*, pages 20–26. IEEE, 2016.

[23] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE access*, 8:85714–85728, 2020.

[24] Hongxing Li, Guochu Shou, Yihong Hu, and Zhigang Guo. Mobile edge computing: Progress and challenges. In *2016 4th IEEE international conference on mobile cloud computing, services, and engineering (MobileCloud)*, pages 83–84. IEEE, 2016.

[25] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[26] Xiaoyan Hu, Lifeng Wang, Kai-Kit Wong, Meixia Tao, Yangyang Zhang, and Zhongbin Zheng. Edge and central cloud computing: A perfect pairing for high energy efficiency and low-latency. *IEEE Transactions on Wireless Communications*, 19(2):1070–1083, 2019.

[27] José M Cecilia, Juan-Carlos Cano, Juan Morales-García, Antonio Llanes, and Baldomero Imbernón. Evaluation of clustering algorithms on gpu-based edge computing platforms. *Sensors*, 20(21):6335, 2020.

[28] Piergiorgio Vitello, Andrea Capponi, Claudio Fiandrino, Guido Cantelmo, and Dzmitry Kliazovich. Mobility-driven and energy-efficient deployment of edge data centers in urban environments. *IEEE Transactions on Sustainable Computing*, 7(4):736–748, 2021.

[29] Herbert Jaeger. Towards a generalized theory comprising digital, neuromorphic and unconventional computing. *Neuromorphic Computing and Engineering*, 1(1):012002, 2021.

[30] 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.

[31] Jeewaka Perera, Sasitharan Balasubramaniam, Samitha Somathilaka, Qu Wen, Xu Li, Dharshana Kasthurirathna, Arman Roohi, and Tyler Nelson. Wet-neuromorphic computing: A new paradigm for biological artificial intelligence. *IEEE Intelligent Systems*, 2025.

[32] Andrew P Woolnough, Lloyd CL Hollenberg, Phillip Cassey, and Thomas AA Prowse. Quantum computing: a new paradigm for ecology. *Trends in Ecology & Evolution*, 38(8):727–735, 2023.

[33] Feng Hu, Ban-Nan Wang, Ning Wang, and Chao Wang. Quantum machine learning with d-wave quantum computer. *Quantum Engineering*, 1(2):e12, 2019.

[34] Siddhant Dutta, Pavana P Karanth, Pedro Maciel Xavier, Iago Leal de Freitas, Nouhaila Innan, Sadok Ben Yahia, Muhammad Shafique, and David E Bernal Neira. Federated learning with quantum computing and fully homomorphic encryption: A novel computing paradigm shift in privacy-preserving ml. *arXiv preprint arXiv:2409.11430*, 2024.

[35] Carver Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, 2002.

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

[37] 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.

[38] Christian Mayr, Sebastian Hoeppner, and Steve Furber. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning-keynote presentation. In *Communicating Process Architectures 2017 & 2018*, pages 277–280. IOS Press, 2019.

[39] Muhammad Aitsam, Sergio Davies, and Alessandro Di Nuovo. Event camera-based real-time gesture recognition for improved robotic guidance. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2024.

[40] Zihao Chen, Zhili Xiao, Mahmoud Akl, Johannes Leugering, Omowuyi Olajide, Adil Malik, Nik Dennler, Chad Harper, Subhankar Bose, Hector A Gonzalez, et al. On-off neuromorphic ising machines using fowler-nordheim annealers. *Nature communications*, 16(1):3086, 2025.

[41] Riccardo Pignari, Vittorio Fra, Enrico Macii, and Gianvito Urgese. Efficient solution validation of constraint satisfaction problems on neuromorphic hardware: the case of sudoku puzzles. *IEEE Transactions on Artificial Intelligence*, 2025.

[42] 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*, 16:999029, 2022.

[43] 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.

[44] Walter Gallego Gomez, Andrea Pignata, Riccardo Pignari, Vittorio Fra, Enrico Macii, and Gianvito Urgese. First steps towards micro-benchmarking the lava-loihi neuromorphic ecosystem. In *2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 462–469. IEEE, 2023.

[45] Mikail Khona and Ila R Fiete. Attractor and integrator networks in the brain. *Nature Reviews Neuroscience*, 23(12):744–766, 2022.

[46] Christoph Ostrau, Christian Klarhorst, Michael Thies, and Ulrich Rückert. Comparing neuromorphic systems by solving sudoku problems. In *2019 International Conference on High Performance Computing & Simulation (HPCS)*, pages 521–527. IEEE, 2019.

[47] B Boreland, G Clement, and Herb Kunze. Set selection dynamical system neural networks with partial memories, with applications to sudoku and kenken puzzles. *Neural Networks*, 68:46–51, 2015.

[48] Liying Tao, Pan Li, Meihua Meng, Zonglin Yang, Xiaozhuang Liu, Jinhua Hu, Ji Dong, Shushan Qiao, Tianchun Ye, and Delong Shang. Blended glial cell's spiking neural network. *IEEE Access*, 11:43566–43582, 2023.

[49] Gabriel A Fonseca Guerra and Steve B Furber. Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems. *Frontiers in neuroscience*, 11:714, 2017.

[50] Juan P Dominguez-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.

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

[52] Mishal Fatima Minhas, Rachmad Vidya Wicaksana Putra, Falah Awwad, Osman Hasan, and Muhammad Shafique. Continual learning with neuromorphic computing: Theories, methods, and applications. *arXiv e-prints*, pages arXiv–2410, 2024.

[53] Kenneth Michael Stewart, Timothy Shea, Noah Pacik-Nelson, Eric Gallo, and Andreea Danielescu. Speech2spikes: Efficient audio encoding pipeline for real-time neuromorphic systems. In *Proceedings of the 2023 annual neuro-inspired computational elements conference*, pages 71–78, 2023.

[54] Sumit Bam Shrestha, Jonathan Timcheck, Paxon Frady, Leobardo Campos-Macias, and Mike Davies. Efficient video and audio processing with loihi 2. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 13481–13485. IEEE, 2024.

[55] Mattias Nilsson. Monte carlo optimization of neuromorphic cricket auditory feature detection circuits in the dynap-se processor, 2018.

[56] Jens Magnusson, Mattias Nilsson, and Olof Skogby Steinholtz. Pattern recognition with neuromorphic sensor system, 2018.

[57] Hsin-Pai Cheng, Wei Wen, Chunpeng Wu, Sicheng Li, Hai Helen Li, and Yiran Chen. Understanding the design of ibm neurosynaptic system and its tradeoffs: A user perspective. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 139–144. IEEE, 2017.

[58] Stefanie Czischek, Andreas Baumbach, Sebastian Billaudelle, Benjamin Cramer, Lukas Kades, Jan M Pawlowski, Markus Oberthaler, Johannes Schemmel, Mihai A Petrovici, Thomas Gasenzer, et al. Spiking neuromorphic chip learns entangled quantum states. *SciPost Physics*, 12(1):039, 2022.

[59] Terrence C Stewart, Bryan Tripp, and Chris Eliasmith. Python scripting in the nengo simulator. *Frontiers in neuroinformatics*, 3:359, 2009.

[60] Riccardo Pignari, Vittorio Fra, Enrico Macii, Gianvito Urgese, et al. Exploring spiking neuron model behaviours through the analysis of parameter space. *COMMUNICATIONS IN COMPUTER AND INFORMATION SCIENCE*, 2024.

[61] Eugene M Izhikevich. *Dynamical systems in neuroscience*. MIT press, 2007.

[62] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

[63] Eugene M Izhikevich. Which model to use for cortical spiking neurons? *IEEE transactions on neural networks*, 15(5):1063–1070, 2004.

[64] Jinichi Nagumo, Suguru Arimoto, and Shuji Yoshizawa. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50(10):2061–2070, 2007.

[65] RM Rose and JL Hindmarsh. The assembly of ionic currents in a thalamic neuron i. the three-dimensional model. *Proceedings of the Royal Society of London. B. Biological Sciences*, 237(1288):267–288, 1989.

[66] Hugh R Wilson. Simplified dynamics of human and mammalian neocortical neurons. *Journal of theoretical biology*, 200(4):375–388, 1999.

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

[68] Wenzhe Guo, Hasan Erdem Yantır, Mohammed E Fouda, Ahmed M Eltawil, and Khaled Nabil Salama. Toward the optimal design and fpga implementation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 33(8):3988–4002, 2021.

[69] Hidekazu Fukai, Taishin Nomura, Shinji Doi, and Shunsuke Sato. Hopf bifurcations in multiple-parameter space of the hodgkin-huxley equations ii. singularity theoretic approach and highly degenerate bifurcations. *Biological Cybernetics*, 82(3):223–229, 2000.

[70] Saeed Farjami, Ryan PD Alexander, Derek Bowie, and Anmar Khadra. Switching in cerebellar stellate cell excitability in response to a pair of inhibitory/excitatory presynaptic inputs: a dynamical system perspective. *Neural Computation*, 32(3):626–658, 2020.

[71] Dragos Calitoiu, B John Oommen, and Doron Nussbaum. Spikes annihilation in the hodgkin-huxley neuron. *Biological cybernetics*, 98(3):239–257, 2008.

[72] Jens E Pedersen, Steven Abreu, Matthias Jobst, Gregor Lenz, Vittorio Fra, Felix Christian Bauer, Dylan Richard Muir, Peng Zhou, Bernhard Vogginger, Kade Heckel, et al. Neuromorphic intermediate representation: A unified instruction set for interoperable brain-inspired computing. *Nature Communications*, 15(1):8122, 2024.

[73] 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.

[74] Simon F Müller-Cleve, Fernando M Quintana, Vittorio Fra, Pedro L Galindo, Fernando Perez-Peña, Gianvito Urgese, and Chiara Bartolozzi. Walin-gui: a graphical and auditory tool for neuron-based encoding. *arXiv preprint arXiv:2310.16983*, 2023.

[75] Hector A Gonzalez, Jiaxin Huang, Florian Kelber, Khaleelulla Khan Nazeer, Tim Langer, Chen Liu, Matthias Lohrmann, Amirhossein Rostami, Mark Schöne, Bernhard Vogginger, et al. Spinnaker2: A large-scale neuromorphic system for event-based and asynchronous machine learning. *arXiv preprint arXiv:2401.04491*, 2024.

[76] Ole Richter, Chenxi Wu, Adrian M Whatley, German Köstinger, Carsten Nielsen, Ning Qiao, and Giacomo Indiveri. Dynap-se2: a scalable multi-core dynamic neuromorphic asynchronous spiking neural network processor. *Neuromorphic computing and engineering*, 4(1):014003, 2024.

[77] Hannah Bos and Dylan Muir. Sub-mw neuromorphic snn audio processing applications with rockpool and xylo. In *Embedded Artificial Intelligence*, pages 69–78. River Publishers, 2023.

[78] Garrick Orchard, E Paxon Frady, Daniel Ben Dayan Rubin, Sophia Sanborn, Sumit Bam Shrestha, Friedrich T Sommer, and Mike Davies. Efficient neuromorphic signal processing with loihi 2. In *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 254–259. IEEE, 2021.

[79] Chinmay Chiplunkar, Nishant Gautam, Ishita Mediratta, Andrew Gait, Sujith Thomas, Andrew Rowley, Teresa Serrano-Gotarredona, and Basabdatta Sen-Bhattacharya. A reduced-scale cortical network with izhikevich's neurons on spinnaker. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[80] Recep Buğra Uludağ, Serhat Çağdaş, Yavuz Selim İşler, Neslihan Serap Şengör, and İsmail Aktürk. Bio-realistic neural network implementation on loihi 2 with izhikevich neurons. *Neuromorphic Computing and Engineering*, 4(2):024013, 2024.

[81] Charlotte Frenkel, Martin Lefebvre, Jean-Didier Legat, and David Bol. A $0.086-\text{mm}^2$ 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.

[82] Hatsuo Hayashi and Satoru Ishizuka. Chaotic nature of bursting discharges in the onchidium pacemaker neuron. *Journal of Theoretical Biology*, 156(3):269–291, 1992.

[83] 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.

[84] 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.

[85] Intel. Lava framework, 1 2021.

[86] Bruno Machado, Carina Pimentel, and Amaro de Sousa. Integration planning of freight deliveries into passenger bus networks: Exact and heuristic algorithms. *Transportation Research Part A: Policy and Practice*, 171:103645, 2023.

[87] Jianjun Dong, Wanjie Hu, Shen Yan, Rui Ren, and Xiaojing Zhao. Network planning method for capacitated metro-based underground logistics system. *Advances in civil engineering*, 2018(1):6958086, 2018.

[88] Xiaoshan Bai, Andres Fielbaum, Maximilian Kronmüller, Luzia Knoedler, and Javier Alonso-Mora. Group-based distributed auction algorithms for multi-robot task assignment. *IEEE Transactions on Automation Science and Engineering*, 20(2):1292–1303, 2022.

[89] Lingzhi Hu, Chengzhou Fu, Zhonglu Ren, Yongming Cai, Jin Yang, Siwen Xu, Wenhua Xu, and Deyu Tang. Sselm-neg: spherical search-based extreme learning machine for drug–target interaction prediction. *BMC bioinformatics*, 24(1):38, 2023.

[90] Hengwei Chen and Jürgen Bajorath. Designing highly potent compounds using a chemical language model. *Scientific reports*, 13(1):7412, 2023.

[91] René De Koster, Tho Le-Duc, and Kees Jan Roodbergen. Design and control of warehouse order picking: A literature review. *European journal of operational research*, 182(2):481–501, 2007.

[92] Oluwadare Badejo and Marianthi Ierapetritou. A mathematical modeling approach for supply chain management under disruption and operational uncertainty. *AIChE Journal*, 69(4):e18037, 2023.

[93] Popuri Srinivasarao and Aravapalli Rama Satish. Multi-objective materialized view selection using flamingo search optimization algorithm. *Software: Practice and Experience*, 53(4):988–1012, 2023.

[94] Lu Zhen, Jingwen Wu, Haolin Li, Zheyi Tan, and Yingying Yuan. Scheduling multiple types of equipment in an automated warehouse. *Annals of Operations Research*, 322(2):1119–1141, 2023.

[95] Banu Çaliş and Serol Bulkan. A research survey: review of ai solution strategies of job shop scheduling problem. *Journal of Intelligent Manufacturing*, 26(5):961–973, 2015.

[96] Jin Li, Chunjiang Zhao, Fuliang Jia, Shunyang Li, Shaohua Ma, and Jianguo Liang. Optimization of injection molding process parameters for the lining of iv hydrogen storage cylinder. *Scientific Reports*, 13(1):665, 2023.

[97] Shuangming Yang, Haowen Wang, Yanwei Pang, Yaochu Jin, and Bernabé Linares-Barranco. Integrating visual perception with decision making in neuromorphic fault-tolerant quadruplet-spike learning framework. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 54(3):1502–1514, 2023.

[98] Shuangming Yang, Haowen Wang, Yanwei Pang, Mostafa Rahimi Azghadi, and Bernabe Linares-Barranco. Nadol: Neuromorphic architecture for spike-driven online learning by dendrites. *IEEE Transactions on Biomedical Circuits and Systems*, 18(1):186–199, 2023.

[99] Sana Bouajaja and Najoua Dridi. A survey on human resource allocation problem and its applications. *Operational Research*, 17(2):339–369, 2017.

[100] Nicoletta Risi, Alessandro Aimar, Elisa Donati, Sergio Solinas, and Giacomo Indiveri. A spike-based neuromorphic architecture of stereo vision. *Frontiers in neurorobotics*, 14:568283, 2020.

[101] Marco Abarca, Giovanny Sanchez, Luis Garcia, Juan Gerardo Avalos, Thania Frias, Karina Toscano, and Hector Perez-Meana. A scalable neuromorphic architecture to efficiently compute spatial image filtering of high image resolution and size. *IEEE Latin America Transactions*, 18(02):327–335, 2020.

[102] Binxin Ru, Michael A Osborne, Mark McLeod, and Diego Granziol. Fast information-theoretic bayesian optimisation. In *International Conference on Machine Learning*, pages 4384–4392. PMLR, 2018.

[103] Erez Karpas and Daniele Magazzeni. Automated planning for robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):417–439, 2020.

[104] Luigi Biagiotti and Claudio Melchiorri. *Trajectory planning for automatic machines and robots*. Springer Science & Business Media, 2008.

[105] Pedro Tavares, Daniel Marques, Pedro Malaca, Germano Veiga, Pedro Costa, and António P Moreira. Optimal automatic path planner and design for high redundancy robotic systems. *Industrial Robot: the international journal of robotics research and application*, 47(1):131–139, 2020.

[106] Jarmo Söderman and Frank Pettersson. Structural and operational optimisation of distributed energy systems. *Applied thermal engineering*, 26(13):1400–1408, 2006.

[107] Gavish. Optimization models for configuring distributed computer systems. *IEEE transactions on computers*, 100(7):773–793, 1987.

[108] Seán Mc Loone and George Irwin. Improving neural network training solutions using regularisation. *Neurocomputing*, 37(1-4):71–90, 2001.

[109] Antanas Verikas and Adas Gelzinis. Training neural networks by stochastic optimisation. *Neurocomputing*, 30(1-4):153–172, 2000.

[110] P Patrick Van Der Smagt. Minimisation methods for training feedforward neural networks. *Neural networks*, 7(1):1–11, 1994.

[111] Ana C Lorena, Luís PF Garcia, Jens Lehmann, Marcilio CP Souto, and Tin Kam Ho. How complex is your classification problem? a survey on measuring classification complexity. *ACM Computing Surveys (CSUR)*, 52(5):1–34, 2019.

[112] Arnold F Sagonda and Komla A Folly. A comparative study between deterministic and two meta-heuristic algorithms for solar pv mppt control under partial shading conditions. *Systems and Soft Computing*, 4:200040, 2022.

[113] Sachin Desale, Akhtar Rasool, Sushil Andhale, and Priti Rane. Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey. *Int. J. Comput. Eng. Res. Trends*, 351(5):2349–7084, 2015.

[114] Sunith Bandaru and Kalyanmoy Deb. Metaheuristic techniques. In *Decision sciences*, pages 709–766. CRC Press, 2016.

[115] George B Dantzig. Linear programming. *Operations research*, 50(1):42–47, 2002.

[116] Narendra and Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on computers*, 100(9):917–922, 1977.

[117] James E Kelley, Jr. The cutting-plane method for solving convex programs. *Journal of the society for Industrial and Applied Mathematics*, 8(4):703–712, 1960.

[118] Hussain Alibrahim and Simone A Ludwig. Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization. In *2021 IEEE congress on evolutionary computation (CEC)*, pages 1551–1559. IEEE, 2021.

[119] Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[120] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[121] Rolando Somma, Sergio Boixo, and Howard Barnum. Quantum simulated annealing. *arXiv preprint arXiv:0712.1008*, 2007.

[122] SN Sivanandam and SN Deepa. Genetic algorithm optimization problems. In *Introduction to genetic algorithms*, pages 165–209. Springer, 2008.

[123] Ahmad Rezaee Jordehi. Particle swarm optimisation for dynamic optimisation problems: a review. *Neural Computing and Applications*, 25(7):1507–1516, 2014.

[124] Wu Deng, Junjie Xu, Yingjie Song, and Huimin Zhao. An effective improved co-evolution ant colony optimisation algorithm with multi-strategies and its application. *International Journal of Bio-Inspired Computation*, 16(3):158–170, 2020.

[125] Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019.

[126] Zeno Jonke, Stefan Habenschuss, and Wolfgang Maass. Solving constraint satisfaction problems with networks of spiking neurons. *Frontiers in neuroscience*, 10:118, 2016.

[127] Rainer Malaka and Sebastian Buck. Solving nonlinear optimization problems using networks of spiking neurons. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 6, pages 486–491. IEEE, 2000.

[128] Stefan Habenschuss, Zeno Jonke, and Wolfgang Maass. Stochastic computations in cortical microcircuit models. *PLoS computational biology*, 9(11):e1003311, 2013.

[129] Md Zahangir Alom, Brian Van Essen, Adam T Moody, David Peter Widemann, and Tarek M Taha. Quadratic unconstrained binary optimization (qubo) on neuromorphic computing system. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3922–3929. IEEE, 2017.

[130] Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using qubo models. *arXiv preprint arXiv:1811.11538*, 2018.

[131] Francisco Barahona. On the computational complexity of ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.

[132] Andrew Lucas. Ising formulations of many np problems. *Frontiers in physics*, 2:5, 2014.

[133] Naeimeh Mohseni, Peter L McMahon, and Tim Byrnes. Ising machines as hardware solvers of combinatorial optimization problems. *Nature Reviews Physics*, 4(6):363–379, 2022.

[134] Hiroshi Kagawa, Yasuaki Ito, Koji Nakano, Ryota Yasudo, Yuya Kawamata, Ryota Katsuki, Yusuke Tabata, Takashi Yazane, and Kenichiro Hamano. Fully-pipelined architecture for simulated annealing-based qubo solver on the fpga. In *2020 Eighth International Symposium on Computing and Networking (CANDAR)*, pages 39–48. IEEE, 2020.

[135] Ryan Hamerly, Takahiro Inagaki, Peter L McMahon, Davide Venturelli, Alireza Marandi, Tatsuhiro Onodera, Edwin Ng, Carsten Langrock, Kensuke Inaba, Toshimori Honjo, et al. Experimental investigation of performance differences between coherent ising machines and a quantum annealer. *Science advances*, 5(5):eaau0823, 2019.

[136] Andrew D King, Sei Suzuki, Jack Raymond, Alex Zucca, Trevor Lanting, Fabio Altomare, Andrew J Berkley, Sara Ejtemaee, Emile Hoskinson, Shuiyuan Huang, et al. Coherent quantum annealing in a programmable 2,000 qubit ising chain. *Nature Physics*, 18(11):1324–1328, 2022.

[137] Nickson Mwamsojo, Frederic Lehmann, Kamel Merghem, Badr-Eddine Benkelfat, and Yann Frignac. Optoelectronic coherent ising machine for combinatorial optimization problems. *Optics Letters*, 48(8):2150–2153, 2023.

[138] Toshimori Honjo, Tomohiro Sonobe, Kensuke Inaba, Takahiro Inagaki, Takuya Ikuta, Yasuhiro Yamada, Takushi Kazama, Koji Enbutsu, Takeshi Umeki, Ryoichi Kasahara, et al. 100,000-spin coherent ising machine. *Science advances*, 7(40):eabh0952, 2021.

[139] Markus Graber and Klaus Hofmann. An enhanced 1440 coupled cmos oscillator network to solve combinatorial optimization problems. In *2023 IEEE 36th International System-on-Chip Conference (SOCC)*, pages 1–6. IEEE, 2023.

[140] Olivier Maher, Manuel Jiménez, Corentin Delacour, Nele Harnack, Juan Núñez, María J Avedillo, Bernabé Linares-Barranco, Aida Todri-Sanial, Giacomo Indiveri, and Siegfried Karg. A cmos-compatible oscillation-based vo2 ising machine solver. *Nature Communications*, 15(1):3334, 2024.

[141] Fuxi Cai, Suhas Kumar, Thomas Van Vaerenbergh, Xia Sheng, Rui Liu, Can Li, Zhan Liu, Martin Foltin, Shimeng Yu, Qiangfei Xia, et al. Power-efficient combinatorial optimization using intrinsic noise in memristor hopfield neural networks. *Nature Electronics*, 3(7):409–418, 2020.

[142] Mingrui Jiang, Keyi Shan, Chengping He, and Can Li. Efficient combinatorial optimization by quantum-inspired parallel annealing in analogue memristor crossbar. *Nature communications*, 14(1):5927, 2023.

[143] Z Fahimi, MR Mahmoodi, H Nili, Valentin Polishchuk, and DB Strukov. Combinatorial optimization by weight annealing in memristive hopfield networks. *Scientific Reports*, 11(1):16383, 2021.

[144] Sergei V Isakov, Ilia N Zintchenko, Troels F Rønnow, and Matthias Troyer. Optimised simulated annealing for ising spin glasses. *Computer Physics Communications*, 192:265–271, 2015.

[145] Y Kihara, M Ito, T Saito, M Shiomura, S Sakai, and J Shirakashi. A new computing architecture using ising spin model implemented on fpga for solving combinatorial optimization problems. In *2017 IEEE 17th International Conference on Nanotechnology (IEEE-NANO)*, pages 256–258. IEEE, 2017.

[146] Masanao Yamaoka, Chihiro Yoshimura, Masato Hayashi, Takuya Okuyama, Hidetaka Aoki, and Hiroyuki Mizuno. A 20k-spin ising chip to solve combinatorial optimization problems with cmos annealing. *IEEE Journal of Solid-State Circuits*, 51(1):303–309, 2015.

[147] Takuya Okuyama, Chihiro Yoshimura, Masato Hayashi, and Masanao Yamaoka. Computing architecture to perform approximated simulated annealing for ising models. In *2016 IEEE international conference on rebooting computing (ICRC)*, pages 1–8. IEEE, 2016.

[148] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. *Physical Review E*, 58(5):5355, 1998.

[149] Arnab Das and Bikas K Chakrabarti. Colloquium: Quantum annealing and analog quantum computation. *Reviews of Modern Physics*, 80(3):1061–1081, 2008.

[150] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. *Science*, 292(5516):472–475, 2001.

[151] Tameem Albash and Daniel A Lidar. Adiabatic quantum computation. *Reviews of Modern Physics*, 90(1):015002, 2018.

[152] Hayato Goto, Kosuke Tatsumura, and Alexander R Dixon. Combinatorial optimization by simulating adiabatic bifurcations in nonlinear hamiltonian systems. *Science advances*, 5(4):eaav2372, 2019.

[153] Andrea Grimaldi, Luciano Mazza, Eleonora Raimondo, Pietro Tullo, Davi Rodrigues, Kerem Y Camsari, Vincenza Crupi, Mario Carpentieri, Vito Puliafito, and Giovanni Finocchio. Evaluating spintronics-compatible implementations of ising machines. *Physical Review Applied*, 20(2):024005, 2023.

[154] Nihal Sanjay Singh, Keito Kobayashi, Qixuan Cao, Kemal Selcuk, Tianrui Hu, Shaila Niazi, Navid Anjum Aadit, Shun Kanai, Hideo Ohno, Shunsuke Fukami, et al. Cmos plus stochastic nanomagnets enabling heterogeneous computers for probabilistic inference and learning. *Nature Communications*, 15(1):2685, 2024.

[155] Dmitry Ivanov, Aleksandr Chezhegov, Mikhail Kiselev, Andrey Grunin, and Denis Larionov. Neuromorphic artificial intelligence systems. *Frontiers in Neuroscience*, 16:959626, 2022.

[156] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[157] Jongkil Park, Theodore Yu, Siddharth Joshi, Christoph Maier, and Gert Cauwenberghs. Hierarchical address event routing for reconfigurable large-scale neuromorphic systems. *IEEE transactions on neural networks and learning systems*, 28(10):2408–2422, 2016.

[158] Sacha J Van Albada, Andrew G Rowley, Johanna Senk, Michael Hopkins, Maximilian Schmidt, Alan B Stokes, David R Lester, Markus Diesmann, and Steve B Furber. Performance comparison of the digital neuromorphic hardware spinnaker and the neural network simulation software nest for a full-scale cortical microcircuit model. *Frontiers in neuroscience*, 12:291, 2018.

[159] Oliver Rhodes, Luca Peres, Andrew GD Rowley, Andrew Gait, Luis A Plana, Christian Brenninkmeijer, and Steve B Furber. Real-time cortical simulation on neuromorphic hardware. *Philosophical Transactions of the Royal Society A*, 378(2164):20190160, 2020.

[160] Chen Liu, Guillaume Bellec, Bernhard Vogginger, David Kappel, Johannes Partzsch, Felix Neumärker, Sebastian Höppner, Wolfgang Maass, Steve B Furber, Robert Legenstein, et al. Memory-efficient deep learning on a spinnaker 2 prototype. *Frontiers in neuroscience*, 12:840, 2018.

[161] Chit-Kwan Lin, Andreas Wild, Gautham N Chinya, Yongqiang Cao, Mike Davies, Daniel M Lavery, and Hong Wang. Programming spiking neural networks on intel's loihi. *Computer*, 51(3):52–61, 2018.

[162] Sadasivan Shankar. Energy estimates across layers of computing: from devices to large-scale applications in machine learning for natural language processing, scientific computing, and cryptocurrency mining. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2023.

[163] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[164] Karl Friston. The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, 11(2):127–138, 2010.

[165] Dante R Chialvo. Emergent complex neural dynamics. *Nature physics*, 6(10):744–750, 2010.

[166] Haim Sompolinsky, Andrea Crisanti, and Hans-Jurgen Sommers. Chaos in random neural networks. *Physical review letters*, 61(3):259, 1988.

[167] Mark D McDonnell and Lawrence M Ward. The benefits of noise in neural systems: bridging theory and experiment. *Nature Reviews Neuroscience*, 12(7):415–425, 2011.

[168] Elad Schneidman, Barry Freedman, and Idan Segev. Ion channel stochasticity may be critical in determining the reliability and precision of spike timing. *Neural computation*, 10(7):1679–1703, 1998.

[169] Tiago Branco, Kevin Staras, Kevin J Darcy, and Yukiko Goda. Local dendritic activity sets release probability at hippocampal synapses. *Neuron*, 59(3):475–485, 2008.

[170] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

[171] Frank C Hoppensteadt and Eugene M Izhikevich. Oscillatory neurocomputers with dynamic connectivity. *Physical Review Letters*, 82(14):2983, 1999.

[172] Darshit Mehta, Mustafizur Rahman, Kenji Aono, and Shantanu Chakrabartty. An adaptive synaptic array using fowler–nordheim dynamic analog memory. *Nature communications*, 13(1):1670, 2022.

[173] J Darby Smith, Aaron J Hill, Leah E Reeder, Brian C Franke, Richard B Lehoucq, Ojas Parekh, William Severa, and James B Aimone. Neuromorphic scaling advantages for energy-efficient random walk computations. *Nature Electronics*, 5(2):102–112, 2022.

[174] Prasad Raghavendra. Optimal algorithms and inapproximability results for every csp? In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 245–254, 2008.

[175] Subhash Khot. On the unique games conjecture (invited survey). In *2010 IEEE 25th annual conference on computational complexity*, pages 99–121. IEEE Computer Society, 2010.

[176] Bradley H Theilman and James B Aimone. Goemans-williamson maxcut approximation algorithm on loihi. In *Proceedings of the 2023 Annual Neuro-Inspired Computational Elements Conference*, pages 1–5, 2023.

[177] Bradley H Theilman, Yipu Wang, Ojas Parekh, William Severa, J Darby Smith, and James B Aimone. Stochastic neuromorphic circuits for solving maxcut. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 779–787. IEEE, 2023.

[178] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions and the bayesian restoration of images. *Journal of applied statistics*, 20(5-6):25–62, 1993.

[179] Bruce Hajek. Cooling schedules for optimal annealing. *Mathematics of operations research*, 13(2):311–329, 1988.

[180] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[181] Ralph Howard Fowler and Lothar Nordheim. Electron emission in intense electric fields. *Proceedings of the royal society of London. Series A, containing papers of a mathematical and physical character*, 119(781):173–181, 1928.

[182] Liang Zhou and Shantanu Chakrabartty. Self-powered timekeeping and synchronization using fowler–nordheim tunneling-based floating-gate integrators. *IEEE Transactions on Electron Devices*, 64(3):1254–1260, 2017.

[183] Yoshiki Matsuda. Benchmarking the max-cut problem on the simulated bifurcation machine. *Medium*, 2019.

[184] Jason Yik, Korneel Van den Berghe, Douwe den Blanken, Younes Bouhadjar, Maxime Fabre, Paul Hueber, Weijie Ke, Mina A Khoei, Denis Kleyko, Noah Pacik-Nelson, et al. The neurobench framework for benchmarking neuromorphic computing algorithms and systems. *Nature communications*, 16(1):1545, 2025.

[185] Mark Stopfer, Vivek Jayaraman, and Gilles Laurent. Intensity versus identity coding in an olfactory system. *Neuron*, 39(6):991–1004, 2003.

[186] David Layden, Guglielmo Mazzola, Ryan V Mishmash, Mario Motta, Pawel Wocjan, Jin-Sung Kim, and Sarah Sheldon. Quantum-enhanced markov chain monte carlo. *Nature*, 619(7969):282–287, 2023.

[187] Y Ye. The gset dataset https://web. stanford. edu/yyye/yyye, 2003.

[188] Matthias Dehmer and Abbe Mowshowitz. A history of graph entropy measures. *Information Sciences*, 181(1):57–78, 2011.

[189] Paul W Holland and Samuel Leinhardt. Transitivity in structural models of small groups. *Comparative group studies*, 2(2):107–124, 1971.

[190] Tanguy Pierog, Iu Karpenko, Judith Maria Katzy, Elena Yatsenko, and Klaus Werner. Epos lhc: Test of collective hadronization with data measured at the cern large hadron collider. *Physical Review C*, 92(3):034906, 2015.

[191] Asier Ozaeta, Wim Van Dam, and Peter L McMahon. Expectation values from the single-layer quantum approximate optimization algorithm on ising problems. *Quantum Science and Technology*, 7(4):045036, 2022.

[192] Timo Mantere and Janne Koljonen. Solving, rating and generating sudoku puzzles with ga. In *2007 IEEE congress on evolutionary computation*, pages 1382–1389. IEEE, 2007.

[193] Jaime G Carbonell, Ryszard S Michalski, and Tom M Mitchell. An overview of machine learning. *Machine learning*, pages 3–23, 1983.

[194] Gary Weiss. WISDM Smartphone and Smartwatch Activity and Biometrics Dataset . UCI Machine Learning Repository, 2019. DOI: https://doi.org/10.24432/C5HK59.

[195] Free Spoken Digit Dataset. Available online: https://github. com. *Jakobovski/free-spoken-digit-dataset (accessed on 17 May 2021)*, 2021.

[196] Youdi Gong, Guangzhen Liu, Yunzhi Xue, Rui Li, and Lingzhong Meng. A survey on dataset quality in machine learning. *Information and Software Technology*, 162:107268, 2023.

[197] Arnold C Englander. Machine learning of visual recognition using genetic algorithms. In *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 197–201. Psychology Press, 2014.

[198] Devi Parikh and C Lawrence Zitnick. The role of features, algorithms and data in visual recognition. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2328–2335. IEEE, 2010.

[199] Juan Li, Wenkai Xu, Limiao Deng, Ying Xiao, Zhongzhi Han, and Haiyong Zheng. Deep learning for visual recognition and detection of aquatic animals: A review. *Reviews in Aquaculture*, 15(2):409–433, 2023.

[200] David MW Powers and Christopher CR Turk. *Machine learning of natural language*. Springer Science & Business Media, 2012.

[201] Tatwadarshi P Nagarhalli, Vinod Vaze, and NK Rana. Impact of machine learning in natural language processing: A review. In *2021 third international conference on intelligent communication technologies and virtual mobile networks (ICICV)*, pages 1529–1534. IEEE, 2021.

[202] Yang Tang, Jürgen Kurths, Wei Lin, Edward Ott, and Ljupco Kocarev. Introduction to focus issue: When machine learning meets complex systems: Networks, chaos, and nonlinear dynamics. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(6), 2020.

[203] Di Qi and Andrew J Majda. Using machine learning to predict extreme events in complex systems. *Proceedings of the National Academy of Sciences*, 117(1):52–59, 2020.

[204] Amit Joshi, M Khosravy, and N Gupta. Machine learning for predictive analysis. *Proceedings of ICTIS*, 2020.

[205] Monika Rybczak, Natalia Popowniak, and Agnieszka Lazarowska. A survey of machine learning approaches for mobile robot control. *Robotics*, 13(1):12, 2024.

[206] Shouyi Wang, Wanpracha Chaovalitwongse, and Robert Babuska. Machine learning algorithms in bipedal robot control. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(5):728–743, 2012.

[207] Arash Hashemi, Grzegorz Orzechowski, Aki Mikkola, and John McPhee. Multibody dynamics and control using machine learning. *Multibody System Dynamics*, 58(3):397–431, 2023.

[208] Vesna Antoska Knights, Olivera Petrovska, and Jasenka Gajdoš Kljusurić. Nonlinear dynamics and machine learning for robotic control systems in iot applications. *Future Internet*, 16(12):435, 2024.

[209] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[210] Fang Liu, Guoming Tang, Youhuizi Li, Zhiping Cai, Xingzhou Zhang, and Tongqing Zhou. A survey on edge computing systems and tools. *Proceedings of the IEEE*, 107(8):1537–1562, 2019.

[211] Haochen Hua, Yutong Li, Tonghe Wang, Nanqing Dong, Wei Li, and Junwei Cao. Edge computing with artificial intelligence: A machine learning perspective. *ACM Computing Surveys*, 55(9):1–35, 2023.

[212] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P Sheth. Machine learning for internet of things data analysis: A survey. *Digital Communications and Networks*, 4(3):161–175, 2018.

[213] Laizhong Cui, Shu Yang, Fei Chen, Zhong Ming, Nan Lu, and Jing Qin. A survey on application of machine learning for internet of things. *International Journal of Machine Learning and Cybernetics*, 9(8):1399–1417, 2018.

[214] 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.

[215] 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.

[216] 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.

[217] 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.

[218] 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.

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

[220] 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.

[221] 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.

[222] 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.

[223] 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.

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

[225] 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.

[226] 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.

[227] 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.

[228] 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.

[229] 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.

[230] 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.

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

[232] 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.

[233] 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.

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

[235] 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.

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

[237] 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.

[238] 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.

[239] 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.

[240] Simon F Müller-Cleve, Vittorio Fra, Lyes Khacef, Alejandro Pequeño-Zurro, Daniel Klepatsch, Evelina Forno, Diego G Ivanovich, Shavika Rastogi, Gianvito Urgese, Friedemann Zenke, et al. Braille letter reading: A benchmark for spatio-temporal pattern recognition on neuromorphic hardware. *Frontiers in Neuroscience*, 16:951164, 2022.

[241] 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 (ISCAS)*, pages 845–848. IEEE, 2007.

[242] 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.

[243] 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.

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

[245] 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.

[246] 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.

[247] 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.

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

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

[250] 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.

[251] 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.

[252] 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.

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

[254] 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.

[255] 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.

[256] Dylan George Peterson. A biologically inspired supervised learning rule for audio classification with spiking neural networks. *University of Calgary*, 2021.

[257] Gary M Weiss, Kenichi Yoneda, and Thaier Hayajneh. Smartphone and smartwatch-based biometrics using activities of daily living. *Ieee Access*, 7:133190–133202, 2019.

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

[259] 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.

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

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

[262] Andrew J Oxenham. How we hear: The perception and neural coding of sound. *Annual review of psychology*, 69(1):27–50, 2018.

[263] 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.

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

[265] 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):063685, 2007.

[266] Yushi Zhang and Waleed H Abdulla. Gammatone auditory filterbank and independent component analysis for speaker identification. In *Interspeech*, 2006.

[267] Elizabeth Elias and James T George. A 16-band reconfigurable hearing aid using variable bandwidth filters. *Global Journal of Research and Engineering-GJRE-F*, 14(1), 2014.

[268] 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.

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

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

[271] 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.

[272] 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.

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

[274] 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.

[275] 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.

[276] Qian Liu, Yunhua Chen, and Steve Furber. Noisy softplus: an activation function that enables snns to be trained as anns. *arXiv preprint arXiv:1706.03609*, 2017.

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

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

[279] Patrik O Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(Nov):1457–1469, 2004.

[280] 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.

[281] 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.

[282] 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.

[283] 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(13):7979–7996, 2021.

[284] 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.

[285] 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.

[286] 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*, 14:637, 2020.

[287] 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.

[288] 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.

[289] 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*, pages 383–402, 2021.

[290] 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.

[291] 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.

[292] Yuwen Chen, Kunhua Zhong, Ju Zhang, Qilong Sun, and Xueliang Zhao. Lstm networks for mobile human activity recognition. In *2016 International conference on artificial intelligence: technologies and applications*, pages 50–53. Atlantis Press, 2016.

[293] Abdu Gumaei, Mabrook Al-Rakhami, Hussain AlSalman, Sk Md Mizanur Rahman, and Atif Alamri. Dl-har: Deep learning-based human activity recognition framework for edge computing. *Computers, Materials & Continua*, 65(2), 2020.

[294] Preeti Agarwal and Mansaf Alam. A lightweight deep learning model for human activity recognition on edge devices. *Procedia Computer Science*, 167:2364–2373, 2020.

[295] Jessica Sena, Jesimon Barreto, Carlos Caetano, Guilherme Cramer, and William Robson Schwartz. Human activity recognition based on smartphone and wearable sensors using multiscale dcnn ensemble. *Neurocomputing*, 444:226–243, 2021.

[296] Nidhi Dua, Shiva Nand Singh, and Vijay Bhaskar Semwal. Multi-input cnn-gru based human activity recognition using wearable sensors. *Computing*, 103(7):1461–1478, 2021.

[297] Ria Kanjilal and Ismail Uysal. The future of human activity recognition: Deep learning or feature engineering? *Neural Processing Letters*, 53(1):561–579, 2021.

[298] Mohamed Abdel-Basset, Hossam Hawash, Ripon K Chakrabortty, Michael Ryan, Mohamed Elhoseny, and Houbing Song. St-deephar: Deep learning model for human activity recognition in ioht applications. *IEEE Internet of Things Journal*, 8(6):4969–4979, 2020.

[299] Henry Friday Nweke, Ying Wah Teh, Mohammed Ali Al-Garadi, and Uzoma Rita Alo. Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges. *Expert Systems with Applications*, 105:233–261, 2018.

[300] Salwa O Slim, Ayman Atia, Marwa MA Elfattah, and Mostafa-Sami M Mostafa. Survey on human activity recognition based on acceleration data. *International Journal of Advanced Computer Science and Applications*, 10(3), 2019.

[301] Florenc Demrozi, Graziano Pravadelli, Azra Bihorac, and Parisa Rashidi. Human activity recognition using inertial, physiological and environmental sensors: A comprehensive survey. *IEEE access*, 8:210816–210836, 2020.

[302] Nida Saddaf Khan and Muhammad Sayeed Ghani. A survey of deep learning based models for human activity recognition. *Wireless Personal Communications*, 120(2):1593–1635, 2021.

[303] Riku Immonen and Timo Hämäläinen. Tiny machine learning for resource-constrained microcontrollers. *Journal of Sensors*, 2022(1):7437023, 2022.

[304] 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.

[305] 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.

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

[307] 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, et al. A historical survey of algorithms and hardware architectures for neural-inspired and neuromorphic

computing applications. *Biologically Inspired Cognitive Architectures*, 19:49–64, 2017.

[308] 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, 2018.

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

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

[311] Wolfgang Maass. To spike or not to spike: that is the question. *Proceedings of the IEEE*, 103(12):2219–2224, 2015.

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

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

[314] Giacomo Indiveri. Neuromorphic engineering. In *Springer Handbook of Computational Intelligence*, pages 715–725. Springer, 2015.

[315] 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.

[316] 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.

[317] 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.

[318] 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.

[319] Charlotte Frenkel, Jean-Didier Legat, and David Bol. A 65-nm 738k-synapse/mm 2 quad-core binary-weight digital neuromorphic processor with stochastic spike-driven online learning. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

[320] Youhui Zhang, Peng Qu, and Weimin Zheng. Towards" general purpose" brain-inspired computing system. *Tsinghua Science and Technology*, 26(5):664–673, 2021.

[321] Jan Stuijt, Manolis Sifalakis, Amirreza Yousefzadeh, and Federico Corradi. μbrain: An event-driven and fully synthesizable architecture for spiking neural networks. *Frontiers in neuroscience*, 15:664208, 2021.

[322] Shruti R Kulkarni, Maryam Parsa, J Parker Mitchell, and Catherine D Schuman. Benchmarking the performance of neuromorphic and spiking neural network simulators. *Neurocomputing*, 447:145–160, 2021.

[323] 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.

[324] 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.

[325] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, pages 3–4, 2013.

[326] 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.

[327] 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, 2021.

[328] 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, 2021.

[329] 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, 2020.

[330] 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, 2019.

[331] Hongyu An, Dong Sam Ha, and Yang 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, 2020.

[332] 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, 2021.

[333] S Mekruksavanich and A Jitpattanakul. Deep convolutional neural network with rnns for complex activity recognition using wrist-worn wearable sensor data. electronics 2021, 10, 1685, 2021.

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

[335] 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.

[336] 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.

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

[338] 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(2):743–755, 2020.

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

[340] 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.

[341] 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.

[342] Travis DeWolf, Pawel Jaworski, and Chris Eliasmith. Nengo and low-power ai hardware for robust, embedded neurorobotics. *Frontiers in Neurorobotics*, 14:568359, 2020.

[343] Daniel Rasmussen. Nengodl: Combining deep learning and neuromorphic modelling methods. *Neuroinformatics*, 17(4):611–628, 2019.

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

[345] 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.

[346] 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.

[347] Aaron R Voelker and Chris Eliasmith. Programming neuromorphics using the neural engineering framework. In *Handbook of Neuroengineering*, pages 1–43. Springer, 2021.

[348] Jozsef Suto. The effect of hyperparameter search on artificial neural network in human activity recognition. *Open Computer Science*, 11(1):411–422, 2021.

[349] Microsoft. Neural Network Intelligence, 1 2021.

[350] 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.

[351] Giacomo Indiveri, Bernabé Linares-Barranco, Tara Julia Hamilton, André van Schaik, Ralph Etienne-Cummings, Tobi Delbruck, Shih-Chii Liu, Piotr Dudek, Philipp Häfliger, Sylvie Renaud, et al. Neuromorphic silicon neuron circuits. *Frontiers in neuroscience*, 5:73, 2011.

[352] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE transactions on software engineering*, 34(4):485–496, 2008.

[353] Ling Liu, Yanzhao Wu, Wenqi Wei, Wenqi Cao, Semih Sahin, and Qi Zhang. Benchmarking deep learning frameworks: Design considerations, metrics and beyond. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1258–1269. IEEE, 2018.

[354] Carel P Kruger and Gerhard P Hancke. Benchmarking internet of things devices. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 611–616. IEEE, 2014.

[355] Juan Gómez-Luna, Izzat El Hajj, Ivan Fernandez, Christina Giannoula, Geraldo F Oliveira, and Onur Mutlu. Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware. In *2021 12th International Green and Sustainable Computing Conference (IGSC)*, pages 1–7. IEEE, 2021.

[356] J Ignacio Torrens, Marcus Keane, Andrea Costa, and James O'Donnell. Multicriteria optimisation using past, real time and predictive performance benchmarks. *Simulation Modelling Practice and Theory*, 19(4):1258–1265, 2011.

[357] Duc Truong Pham and Marco Castellani. Benchmarking and comparison of nature-inspired population-based continuous optimisation algorithms. *Soft Computing*, 18(5):871–903, 2014.

[358] Roger W Hockney. *The science of computer benchmarking*. SIAM, 1996.

[359] Changchang Zeng, Shaobo Li, Qin Li, Jie Hu, and Jianjun Hu. A survey on machine reading comprehension—tasks, evaluation metrics and benchmark datasets. *Applied Sciences*, 10(21):7640, 2020.

[360] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *International conference on machine learning*, pages 5338–5348. PMLR, 2020.

[361] Ramon Bertran, Alper Buyuktosunoglu, Meeta S Gupta, Marc Gonzalez, and Pradip Bose. Systematic energy characterization of cmp/smt processor systems via automated micro-benchmarks. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 199–211. IEEE, 2012.

[362] Alif Ahmed and Kevin Skadron. Hopscotch: a micro-benchmark suite for memory performance evaluation. In *Proceedings of the International Symposium on Memory Systems*, pages 167–172, 2019.

[363] BK Bershad, Richard P Draves, and Alessandro Forin. Using microbenchmarks to evaluate system performance. In *[1992] Proceedings Third Workshop on Workstation Operating Systems*, pages 148–153. IEEE, 1992.

[364] Nor Asilah Wati Abdul Hamid and Paul Coddington. Comparison of mpi benchmark programs on shared memory and distributed memory machines (point-to-point communication). *The International Journal of High Performance Computing Applications*, 24(4):469–483, 2010.

[365] Shihao Song, M Lakshmi Varshika, Anup Das, and Nagarajan Kandasamy. A design flow for mapping spiking neural networks to many-core neuromorphic hardware. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[366] Adarsha Balaji, Anup Das, Yuefeng Wu, Khanh Huynh, Francesco G Dell'Anna, Giacomo Indiveri, Jeffrey L Krichmar, Nikil D Dutt, Siebren

Schaafsma, and Francky Catthoor. Mapping spiking neural networks to neuromorphic hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(1):76–86, 2019.

[367] Siqi Wang, Tee Hiang Cheng, and Meng Hiot Lim. A hierarchical taxonomic survey of spiking neural networks. *Memetic Computing*, 14(3):335–354, 2022.

[368] Mike Davies et al. Taking neuromorphic computing to the next level with loihi2. *Intel Labs' Loihi*, 2(1), 2021.

[369] Gianvito Urgese, Antonio Rios-Navarro, Alejandro Linares-Barranco, Terrence C Stewart, and Konstantinos Michmizos. Powering the next-generation iot applications: new tools and emerging technologies for the development of neuromorphic system of systems. *Frontiers in Neuroscience*, 17:1197918, 2023.

[370] Intel. Benchmarks: Seeding a Collaborative Effort + An Audio Challenge!, 2022.

[371] Mike Davies. Benchmarks for progress in neuromorphic computing. *Nature Machine Intelligence*, 1(9):386–388, 2019.

[372] Intel. Intel MPI Benchmarks, 2021.

[373] Ohio State University. OSU Micro-Benchmarks, 2023.

[374] Ryan Taylor and Xiaoming Li. A micro-benchmark suite for amd gpus. In *2010 39th International Conference on Parallel Processing Workshops*, pages 387–396. IEEE, 2010.

[375] Jiuxing Liu, Balasubramanian Chandrasekaran, Weikuan Yu, Jiesheng Wu, Darius Buntinas, Sushmitha Kini, Pete Wyckoff, and Dhabaleswar K Panda. Micro-benchmark level performance comparison of high-speed cluster interconnects. In *11th Symposium on High Performance Interconnects, 2003. Proceedings.*, pages 60–65. IEEE, 2003.

[376] Joel Scheuner and Philipp Leitner. Estimating cloud application performance based on micro-benchmark profiling. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 90–97. IEEE, 2018.

[377] 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.

[378] 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.

[379] Jonathan Timcheck, Sumit Bam Shrestha, Daniel Ben Dayan Rubin, Adam Kupryjanow, Garrick Orchard, Lukasz Pindor, Timothy Shea, and Mike Davies. The intel neuromorphic dns challenge. *Neuromorphic Computing and Engineering*, 3(3):034005, 2023.