FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Neuromorphic Computing and Event-Driven Algorithms for Real-Time Analysis of Neuronal Activity Data

João Gil Marinho Mesquita

Master in Informatics and Computing Engineering

Supervisor: Paulo de Castro Aguiar

July 28, 2024

Neuromorphic Computing and Event-Driven Algorithms for Real-Time Analysis of Neuronal Activity Data

João Gil Marinho Mesquita

Master in Informatics and Computing Engineering

July 28, 2024

Abstract

The analysis of neuronal activity is a task that has intrigued humans for decades, providing insights into the inner workings of the brain and catalyzing the progress of both fundamental and clinical neuroscience. With the increasing trend of recording electrodes to register brain activity and the need for real-time analysis of neuronal activity, the computational systems responsible for handling this data require better computational efficiency and capability to analyze firing patterns.

Current solutions rely on conventional algorithms based on the *von Neumann* clock-driven architecture. However, this architecture is not optimal for neuronal activity analysis for several reasons. First, the clock-driven design is inefficient in processing neuronal spike events, which are asynchronous sparse events by nature. Moreover, conventional computers are limited by the *von Neumann* bottleneck, a limitation where the throughput between the CPU and memory is constrained by the rate of data transfer. Lastly, the spiking patterns must be converted to a digital format understood by traditional CPUs before processing the data, which adds significant latency to the data processing and compromises real-time operation.

To tackle these issues, the use of neuromorphic computing is explored, along with the implementation of event-driven algorithms dedicated to these computing chips. This computing paradigm attempts to mimic the structure and function of the brain using physical artificial neurons and synapses. Using Intel's Loihi neuromorphic research chip, this study aims to overcome the limitations of conventional computing methods and develop low-power devices with notable predictive capabilities in real-time detection of neuronal activity. This work focuses on detecting three biomarkers: channel bursts, network bursts, and high-frequency oscillations, detailing the data pre-processing steps and the underlying SNN architectures.

Both *in vitro* and *in vivo* practical studies yielded promising results. The benchmarking framework associated with Loihi allowed us to analyze the event-driven algorithms' performance in terms of running time, energy consumption, and memory usage. The neuromorphic solution registered notable improvements in power consumption and latency, two critical features of real-time monitoring systems. On the other hand, the classification metrics of the detectors are on par with state-of-the-art solutions, being able to detect the majority of relevant events. The HFO detector, in particular, proved to better adapt to data from new datasets compared to deep learning alternatives.

The outcomes of this study support the application of neuromorphic computing for online analysis of neuronal activity. The implemented algorithms achieved comparable detection capabilities while revealing significant improvements in energy consumption and latency. This work presents a reproducible framework for developing bioelectronics using Loihi to capture real-time activity patterns.

Keywords: Neuronal Activity Monitoring, Channel Bursts, Network Bursts, High-Frequency Oscillations, Neuromorphic Algorithms, Neuronal Pattern Detection, Intel Loihi, LAVA Software Framework, Computational Neuroscience, Neuromorphic Benchmarks

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Professor Paulo Aguiar, for being actively interested in this project and always providing invaluable feedback. I am very thankful for the opportunity you granted me and for introducing me to the area of computational neuroscience. Likewise, I would like to thank the whole NCN team for being such a welcoming group, full of bright people, and conducting exciting research. The working environment at the research group made this work a lot easier!

A special thanks to *Intel Labs* for acknowledging the potential of this project and granting access to Intel's virtual machines to run and benchmark algorithms on Loihi.

My heartfelt thank you to all my friends who follow this journey we call life by my side and are always there to share the good and bad moments. Thank you to Rui Alves and Bruno Rosendo, my "partners in crime" during these 5 years of the master's program. We've been through a lot together, but never failed to crack a laugh once in a while. Thank you to *Gunas Lda.*, the squad that accompanies me many nights to decompress from the stress of life. Thank you *Leo*, *Yuri*, and *Forever* for following my crazy ideas once in a while.

Last but not least, a big thank you to my family for always being supportive, loving, and letting me choose my own path. No matter where I go, nothing beats home.

João Gil Marinho Mesquita

“ When do you think people die? When they are shot with a bullet? No.
When they eat a soup made from a poisonous mushroom? No.
People die... when they are forgotten. ”

Dr. Hiriluk

Contents

List of Figures

List of Tables

Abbreviations and Symbols

AdEx	Adaptive Exponential Integrate-and-Fire
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASIC	Application-specific integrated circuit
BNN	Biological Neural Network
CB Detector	Channel Burst Detector
DBS	Deep Brain Stimulation
EoI	Event of Interest
EZ	Epileptogenic Zone
FET	Field-effect Transistor
FPGA	Field-programmable Gate Array
FR	Fast Ripple
GUI	Graphical User Interface
HDMEA	High-density Microelectrode Array
HFO	High Frequency Oscillation
IAP	Intracellular Action Potential
IBI	Inter-Burst Interval
iEEG	Intracranial Electroencephalography
IF	Integrate and Fire
IQR	Interquartile Range
ISI	Inter-Spike Interval
LFP	Local Field Potential
LIF	Leaky-Integrate-and-Fire
LTP	Long-Term Plasticity
MEA	Microelectrode Array
NB Detector	Network Burst Detector
NCN	Neuroengineering and Computational Neuroscience
NI	Neural Interface
sEEG	Stereo-Electroencephalography
SNN	Spiking Neural Network
SNR	Signal-to-Noise Ratio
STDP	Spike-Timing-Dependent-Plasticity
STP	Short-Term Plasticity
VM	Virtual Machine

Chapter 1

Introduction

1.1 Context

The brain has been a target of human research since 3200 BC, during the ancient Egyptian times [63]. Understanding the inner workings of the brain and the system around it, using the brain seems paradoxical. Yet, it is an intriguing activity that may bring countless advantages to humanity, such as treating neurological disorders ([7], [2], [29]) and reaching *artificial general intelligence* [8].

Nowadays, the high complexity of this organ and the system around it is widely acknowledged. However, that is the outcome of years of accumulated research, which is still incomplete. The ancient Greeks believed that the heart was the seat of intelligence, while the Egyptians believed that the brain was a useless organ that should be removed during mummification [58]. Around 170 BC, the brain started to gain the interest of the general scientific community after the Roman physicist Galen suggested that the brain was where our memory, personality, and thinking resided [35].

In recent years, new areas of research have arisen that are attempting to know more about how the nervous system works through different methods. One such area is *neuromorphic computing*, which is inspired by the structure and function of the human brain. Neuromorphic systems are designed to emulate the behavior of neurons and synapses using physical artificial neural networks [81], thus providing high performance, low power consumption, and the ability to mimic the stochastic, non-linear behavior of neuronal signals.

Being first proposed in the 1980s by Professor Carver Mead [14], neuromorphic computing is an active area of research with a lot of growth potential. The widespread adoption of artificial intelligence technology and the perceived end of the *Moore's Law* [93] are some of the factors that direct much attention towards neuromorphic computing research and its potential. It has gained attention from major chip manufacturers, such as *IBM*, with its *TrueNorth* chip, and *Intel*, with *Intel Lab's Loihi*, as well as the United States military.

1.2 Problem and Motivation

A wide range of diseases can be treated through electrical stimulation. Epilepsy [7] is one of the medical conditions where miniaturized bioelectronic implants can monitor, process, and stimulate neural responses from the nervous system to the body to trigger desired responses, offering an alternative to conventional pharmacological methods. This emerging life science field, bioelectronic medicine, is not only presenting new treatments for numerous neurological disorders but also being tested in treatments for restoring lost function to paralyzed patients and victims of spinal cord injury (SCI) [27].

Adopting this treatment has some nuances that must be solved before deploying it to the public. Neural interfaces (NIs) are the machinery responsible for connecting bioelectronic medicine to the human body. These interfaces must be attached to the body to detect and generate electrical signals, so they have to obey strict requirements regarding power usage, performance, dimensions, and biocompatibility. Due to the nature of the brain signals, the computing system responsible for processing them must also be able to handle highly parallel and asynchronous data.

The aforementioned requirements showcase why the conventional *von Neumann* architecture is not ideal for implementing neural interfaces. The clock-driven architecture makes the processing unit inefficient in terms of power consumption since data arrives through events. Furthermore, the densely packed network of connections brings performance issues, and the computation speed is limited by the *von Neumann bottleneck*, which is the gap between the speed of computation and memory access. Besides, there is an abstraction layer between the spiking events and the microelectrode array recordings since the events must be converted and processed to be handled by traditional computing systems, which may hamper the processing of such events and cause the loss of non-linear behaviors.

Additionally, a missing crucial property of neural interfaces is their ability to process, learn, and answer in real-time to data. Once again, the structure of traditional systems is not built for these scenarios, which raises the question: "Is there an alternative computing paradigm tailored for parallel, real-time operation that would enable building more efficient neural interfaces?".

The goal of this dissertation is to answer the question above. More specifically, to validate neuromorphic computing as an alternative computing paradigm to implement the NIs that drive bioelectronic medicine and showcase practical examples of its usage, including algorithms tailored for the event-driven chip and comparing them to the traditional computing counterpart. Most existing bioelectronic devices cannot adapt their behavior after being installed. Harnessing the properties mentioned above, neuromorphic computing is a promising solution for implementing an adaptive closed-loop bioelectronic device capable of continuously monitoring physiological signals and adapting its response in real-time to internal and external changes, resulting in better control of symptoms and side effects [27].

1.3 Objectives

The main goal of this thesis is to **develop event-driven algorithms for characterizing neuronal signals in real-time, running on neuromorphic chips**. The work is conducted from a software engineering perspective, so the biological experiment and the collection of neuronal activity were managed by the Neuroengineering and Computational Neuroscience (NCN) group from the Institute for Research and Innovation in Health (i3s).

To achieve this, the following objectives were defined:

1. Review the existing literature on neuromorphic computing, neural signal processing, microelectrode array recordings, and bioelectronic medicine.

2. Seek relevant applications for monitoring the activity of neuronal populations in real-time in a clinical setting and select the activity patterns the neuromorphic system will detect.

3. Define electrophysiological biomarkers to detect the selected activity patterns, such as network bursts and high-frequency oscillations.

4. Assess the different neuromorphic systems available and determine the optimal hardware and software for this research. This involves learning how to work with the chosen technology stack and take advantage of its neuromorphic architecture.

5. Understand the different data sources that will be available to conduct this work and how to feed the data to the event-driven algorithms. In particular, study the available experimental setup that captures the dynamics of neuron populations in response to configurable stimulation.

6. Explore and develop algorithms for monitoring and characterization, in real-time, of the activity of neuronal populations. The focus is on identifying relevant activity patterns proven in the literature and creating a neuromorphic version of such algorithms.

7. Evaluate the performance of the neuromorphic system and compare it with traditional computing systems.

1.4 Document Structure

The document splits into seven chapters. After the Introduction where the context, problem, motivation, and objectives of this report are elucidated, Chapter 2 presents the state of the art about neural interfaces, spiking neural networks, neuromorphic computing, and the *lava* software framework, as well as a section for related work. Chapter 3 presents the adopted methodology for this investigative work.

From here, Chapter 4 and Chapter 5 are two independent practical studies. The former is an exploratory work that detects different electrophysiological biomarkers on an *in-vitro* population

of cortical rat neurons. The latter is an in-depth high-frequency oscillation detector on stereo-encephalography data from human recordings. Each of these chapters describes the experiment's methodology, the proposed solution, and the associated results.

Finally, Chapter 6 highlights the findings of the thesis, mentioning the results, whether the objectives were met, and perspectives of future work.

Chapter 2

State of the Art

This chapter provides a foundation of knowledge required to follow the remaining report. The current developments in *neuromorphic computing* and associated research areas are mentioned. Moreover, works of investigation related to the topic are presented and compared to understand what knowledge can be transferred from previous research.

2.1 Neural Interfaces

Neural interfaces are specialized devices that link bioelectronic medicine to the body. These electronics can both record electric signals and stimulate desired populations of neurons. Due to their reduced dimensions, among other restrictions, they are connected to the human body and perform continuous sensing, processing, and modulation of electronic signals produced in the nervous system to treat medical conditions [38].

NIs can be classified according to the interface level with the nervous system into three main classes: cortical, spinal cord, and peripheral. In any case, they usually follow a similar structure, composed of four key modules: a tissue interface, an electronic sensing interface, a neural signal processing unit, and a stimulating device, as depicted in figure 2.1.

The tissue interface represents the interface to the human body from which neuronal signals will be first detected and later stimulated. Several techniques can be used to create a two-way exchange of information with the nervous system, such as MEAs [62] and microprobes [1]. The following subsections dive deeper into the three remaining modules of a NI.

2.1.1 Sensing Neural Activity

The first step to connect bioelectronics to the body is detecting neural activity. NIs require the acquisition of electrophysiological data via accurate, flexible, and biocompatible sensors able to transduce neural activity into a signal that can be processed to produce an output that modulates internal function.

The sensors may be of two types: intracellular and extracellular. Intracellular recordings measure the voltage across neuronal membranes, while in extracellular ones, the electrodes are placed

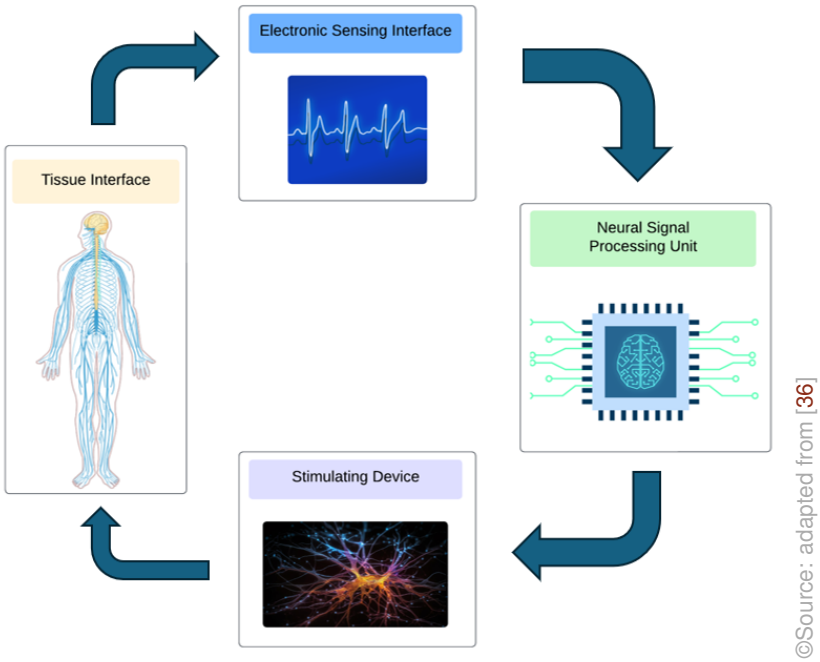


FIGURE 2.1 – Structure of a Neural Interface. The cycle represents the intended closed-loop system of recording electrophysiological data, processing, and respective stimulation of the nervous system.

in the extracellular fluid, reducing the signal amplitude but covering a larger area. Hence, intracellular recordings are involved in studies that require precise measurements of the electrical activity at the single-cell level [92]. This type of recording is ideal for understanding how to simulate a specific behavior to derive a particular output. For example, modulation of the vagus nerve to regulate the heart chamber activation requires a precise understanding of the behavior of specific cells in the basal ganglia responsible for heart respiration [26]. On the contrary, extracellular recordings are not focused on a single cell but rather on the electrical behavior of a region. Depending on the nature of the electrodes, they record single/multiple unit recordings, whole nerve recordings, or field potentials.

The neural activity in the brain includes complex spatial interactions between cells at varying time scales, which makes these networks particularly difficult to study [27]. This complexity needs to be accounted for during the design phase of the NI. Depending on the neuronal activity that needs to be recorded, the need to record activities that span different time scales (from ms to years) and space scales (from single cell types to large brain areas) varies. Generally speaking, the quality of a sensor is evaluated in terms of its *selectivity* and *sensitivity*.

Selectivity is linked to spatial resolution and represents the ability to detect a specific type of signal while ignoring the noise. Floating microelectrode arrays are an example of sensors with high selectivity. However, they are designed with a single recording electrode, limiting the spatial resolution of the recording sites. To tackle this issue, MEAs neural probes can be used [62]. Conversely, sensitivity reflects the ability to detect slight changes in the electrical signal it is designed to detect. Both properties must be maximized for optimum signal receptivity [38], which

is crucial to delivering a reliable and faithful signal to the following modules.

2.1.2 Signal Processing and Analysis

This module is where the neural signal processing occurs. It is responsible for interpreting the electrical activity received from the sensing module and creating knowledge from this data. Traditional implementations of this layer relied on conventional computing to process biosignals, such as digital signal processors (DSPs) or even full-custom synchronous, clocked, digital application-specific integrated circuits (ASICs). Even though they provide more flexibility in terms of operations and programmability, traditional implementations lack power efficiency and scalability.

This dissertation focuses on neuromorphic processing units that attempt to faithfully emulate the properties of real neural circuits. It is an *in-memory computing* processing architecture, driven directly by its input signals in real-time, without relying on an external storage mechanism. To minimize the power consumption and optimize the performance of the neuromorphic chip, it is vital to tune their time constants and processing rates according to the dynamics and time constants of the signals they need to process. In the case of neural signals, these time constants can range from milliseconds to minutes.

The major challenge in building NIs with neuromorphic processing units and applying them to existing bioelectronic medicine tasks is determining the ideal way to program them. Unlike DSPs, there is no well-established programming framework for such devices, and they follow a different computing paradigm that we are not used to. Thus, there are two possible ways to configure such systems:

1. Train them through a learning procedure
2. Rely on high-level abstractions, using instances of *computational primitives*, similar to how software engineers use abstract high-level programming language routines instead of low-level code [27].

There are several successful examples of using learning in conventional AI approaches, where closed-loop interactions are not required and large datasets are available for training. However, the translation to the neuromorphic computing implementation is not straightforward, as most bioelectronic medicine applications have demanding power consumption requirements. Not only that, the data in most applications is strongly subject-dependent and can change significantly according to different subjects, environments, or other changes during operation. For this reason, the neuromorphic devices require always-on spike-based learning and plasticity mechanisms to adapt to the peculiarities of specific individuals and any eventual changes.

An alternative approach to program neuromorphic chips is to find the best set of computational primitives to use and combine them most appropriately on a case-by-case basis [56]. By simulating the behavior of processing operations observed in animal nervous systems, the construction of computational primitives presents an approach to carry out linear, non-linear, adaptive filtering operations and other types of operations that enable the implementation of SNNs with on-chip and online spike-based learning properties to solve classification or regression tasks [88].

2.1.3 Neurostimulation

To close the loop, neurostimulation is the final step in handling neurological diseases, where the nervous system can be regulated through electrical stimulation. Electrophysiological stimulation is already used for deep brain stimulation and treating several neurological conditions. Since it targets specific parts of the nervous system, it has fewer side effects and less addictive potential when compared to drug therapies. Nevertheless, current stimulation protocols are far from replicating the richness of properties of natural neural activity, characterized by their stochastic and asynchronous nature.

Although DBS patterns are usually delivered with regular stimulation settings, several studies demonstrate irregular patterns of stimulation are more effective [23]. In the case of Parkison's disease, non-periodic stimulation induces irregular firing patterns that help to desynchronize highly clustered neurons, making their firing less burstlike. Similar results display how seizures can be suppressed using asynchronous non-periodic electrical stimulation to synchronize the nucleus accumbens [46].

Thanks to the intrinsic nature of neuromorphic systems, they can generate Poissonian activation patterns that have demonstrated significant advantages compared to traditional protocols based on uniform periodic square pulses. That, along with the capability to produce outputs driven by the changes in the inputs and adapted in real-time, allows neuromorphic systems to generate spike trains that change over time with realistic rates and variability.

2.2 Spiking Neural Networks

SNNs are a type of artificial neural network that mimics the behavior of biological neurons and synapses with higher fidelity than traditional neural networks. Unlike traditional ANNs, which use continuous-valued signals to represent information, SNNs rely on discrete events called spikes, resembling the action potentials in biological neurons.

These spike-based networks are based on the only system we are aware of for general intelligence: the nervous system. Accordingly, their fundamental computation unit is the spiking neuron, accumulating input signals over time and generating output spikes when a certain threshold is reached [33].

The interconnected neurons of SNNs receive input spikes from their predecessors and propagate their output spikes to the neurons in the following layer. Each connection between neurons, representing an artificial synapse, has an associated weight that controls the strength of the connection between the pair. The output spike strength is determined by the neuron's activation function, which can be derived from several factors, such as the current input, the neuron's membrane potential, or a combination of multiple factors. These dynamics are described in figure 2.2.

Figure 2.3 illustrates the behavior of a biological neural network. In the simplest form, a synapse is composed of a *pre-synaptic neuron* (sending) and a *post-synaptic neuron* (receiving). The synapse travels from the cell body (*soma*) of the sending cell through the axon until it reaches

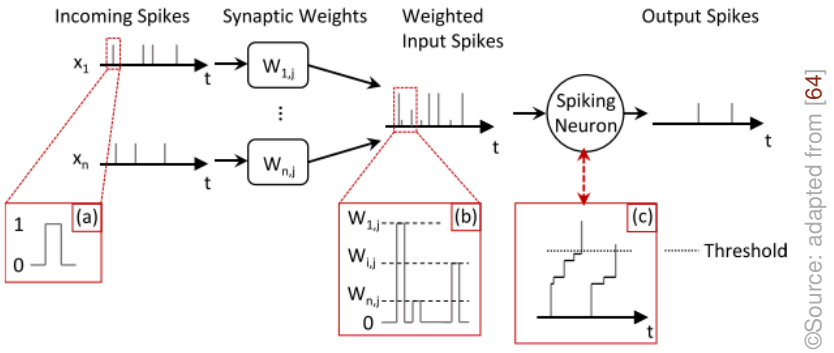


FIGURE 2.2 – Schematic representation of a spiking neuron in an SNN.

the receiving cell via its dendrites. In reality, neurons are heavily interconnected, meaning each neuron may receive inputs from many neurons and output to several other neurons. The accumulation of electrical signals during a short period is what causes a neuron to fire. The *resting membrane potential* of each neuron is approximately -70 mV, and it can increase up to around +40 mV during the process we call an *action potential* [19]. Table 2.1 presents a comparison between BNNs, ANNs, and SNNs.

2.2.1 Data Encoding in SNNs

When referring to SNNs, it is essential to distinguish the two major encoding schemes: *rate coding* and *temporal coding* (figure 2.4). Encoding analog data into binary spike trains resembling the biological neurons' encoding style is the first step to building artificial SNNs. In *rate-coding*, the strength of the input signals is represented by the firing rate of the spike trains. Neurons exposed to higher-intensity stimuli fire high-frequency pulses, while lower-frequency spikes are emitted in response to low stimulation [48]. This coding technique can be further divided into three subcategories: *count*, *density*, and *population rate* coding. For a detailed explanation of these techniques, refer to [21].

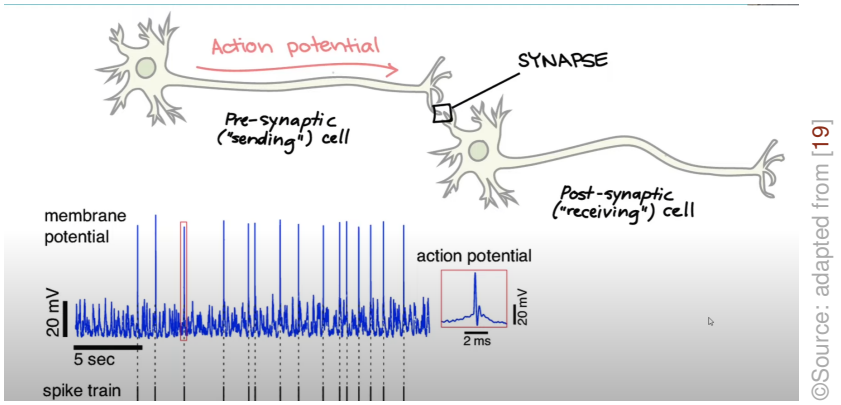


FIGURE 2.3 – Illustration of the neuronal behavior SNNs replicate.

Properties	**Biological NNs**	**ANNs**	**SNNs**
Information Representation	Spikes	Scalars	Spikes
Learning Paradigm	Synaptic Plasticity	Backpropagation (BP)	Plasticity / BP
Platform	Brain	von Neumann-based CPU	Neuromorphic System

TABLE 2.1 – A comparison of properties between BNNs, ANNs, and SNNs

Several years after Thorpe proposed the idea of exact spike timings as a coding scheme [87], evidence showed that the precise timing and order of spikes are crucial information for biological neural networks. That is when *spatiotemporal-coding* schemes arrived. In this case, data is encoded into the spatial and temporal firings of the spikes according to the stimuli's intensity. Hence, neurons that receive higher-intensity stimulation spike first, followed by those stimulated at a slower pace [48].

From an information processing perspective, a spatiotemporal coding scheme is more powerful because it can encode the same information with fewer pulses, improving information density and energy efficiency. In this paper, the focus will be on temporal-coding spiking neural networks to represent and process temporal data naturally. This property is beneficial for modeling time-varying and event-driven data, such as neuronal activity [82]. For this reason, future mentions of spiking neural networks will refer to networks utilizing temporal coding.

The following subsection presents different models that aim to replicate the dynamics of biological neural networks with varying levels of complexity.

2.2.2 SNN Neuron Models

When developing large-scale brain models of spiking neurons, we must find compromises between two seemingly mutually exclusive requirements [30]:

1. Model's complexity. This will dictate its computational requirements.

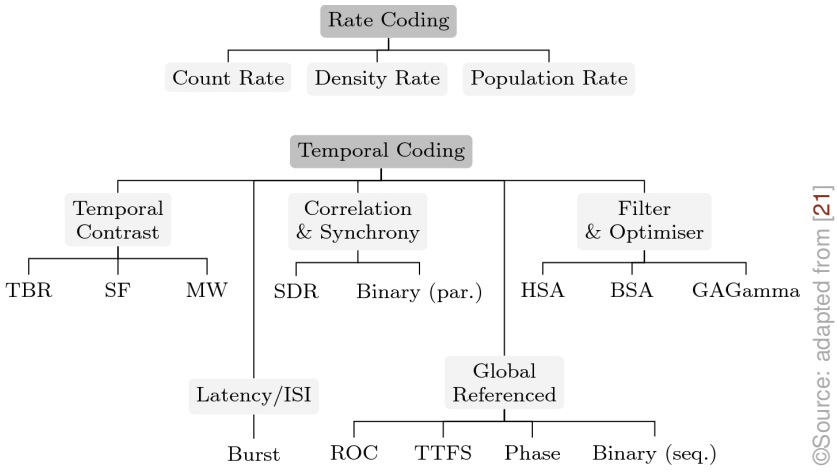


FIGURE 2.4 – Taxonomy of rate and temporal coding techniques.

2. Capability of producing rich firing patterns exhibited by real biological neurons.

Intuitively, models representing complex dynamics of biological neurons require more computational power, making them unsuitable for large simulations. That is why several models with different fidelity to the nervous system have been proposed over the years, and they can be broadly organized into three categories:

1. Models where the spike generation is due to a system of differential equations, including an equation for the membrane potential and equations for voltage-dependent conductances. By reducing the parameter space compared to complex models of neurons, these models become amenable to insightful mathematical analysis while maintaining a high fidelity to their biological counterparts.

2. Models where action potentials occur when the voltage crosses a threshold. These are less complex than the previous type, making them easier to include in a dense network and also mathematically analyzable.

3. Models where action potentials are negligible. Instead, the firing rate of the neuron is the critical variable. Whilst many details are lost, it offers a simple model that can perform complex computations with the right architecture.

There are several factors to take into account when choosing a neuron model. For instance, the level of explanation, the available data, and the processing demand. Anyhow, an intuitive approach is to start using simple models to gain some knowledge of the problem and use the outcomes to develop a more complex solution [22].

2.2.2.1 The Leaky-Integrate-and-Fire Model

The Leaky-Integrate-and-Fire (LIF) is a simple SNN model that is widely used in the field of computational neuroscience. The three main rules that govern this model are the following [19]:

1. The membrane potential, V, evolves according to the following differential equation:

$$\tau \frac{dv}{dt} = -V(Leak) \tag{2.1}$$

2. When a neuron receives a spike, V increases by synaptic weight w:

$$V \leftarrow V + W(Integrate) \tag{2.2}$$

3. When $V > V_t$, the neuron fires a spike and resets the voltage:

$$V \leftarrow 0 \ (Discontinuous\ dynamics) \tag{2.3}$$

According to the equation 2.1, the voltage (V) exponentially decays when it receives no input, and the rate of decay is controlled by a *time constant* named τ, that corresponds to the time it takes

for the membrane potential to go from V to $\frac{1}{e}V$. A lower τ value will make the membrane potential decay to the resting potential faster. Conversely, a higher time constant slows down the leaking process. A compelling property of this model is that it can be modeled as a circuit composed of a resistor and a capacitor in parallel, which, respectively, represent the leakage and capacitance of the membrane [49].

2.2.2.2 Dynamic Threshold Leaky-Integrate-and-Fire

This model, also called a *2D LIF* model, extends the LIF model by turning the $V_{threshold}$ dynamic. The voltage threshold has an initial value but increases as spikes occur. Without action potentials, the voltage threshold decays back to the initial value. The additional behavior is explained via the following rules [19]:

1. Voltage Threshold Leak:
$$\tau\frac{dV_t}{dt} = 1 - V_t \tag{2.4}$$
2. After a spike:
$$V \leftarrow 0 \tag{2.5}$$
$$V_t \leftarrow V_t + \delta V_t \tag{2.6}$$

While unable to model the behavior of neurons precisely, the *dynamic threshold LIF* presents more flexibility than the 1D LIF and adapts to changes in the input stimuli, making it another attractive simple neuron model. The adaptive threshold and voltage dynamics of this 2D LIF can be observed in figure 2.5.

2.2.2.3 Izhikevich Simple Neuron

Compared to the previously mentioned approaches, the *Izhikevich Simple Neuron* model produces more realistic behaviors while being far more efficient than biophysical accurate *Hodgkin-Huxley-type* neuronal models. It can model different types of neuron dynamics, including excitatory cortical cells (*regular spiking*, *instrinsically bursting* and *chattering* neurons), inhibitory cortical cells (*fast spiking* and *low threshold spiking* neurons), and other dynamics such as *thalamo-cortical* and *resonator* neurons [30].

The following three equations define the *Izhikevich* model, where v and u are dimensionless variables and a, b, c, and d are dimensionless parameters [32]:

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{2.7}$$

$$u' = a(bv - u) \tag{2.8}$$

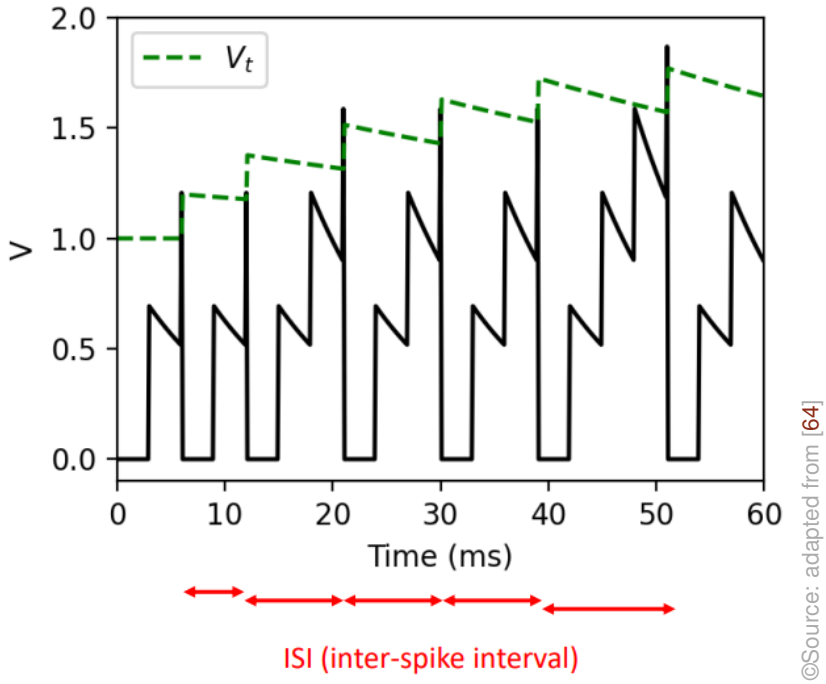


FIGURE 2.5 – Neuron dynamics of a Dynamic Threshold LIF model.

$$if\, v \geq 30\, mV,\, then: \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{2.9}$$

In practice, the set of equations (2.7, 2.8, 2.9) represent a "one-fits-all" choice to be used when large-scale networks are simulated. Regardless, more accurate alternatives are available to analyze the behavior of a single neuron. To turn the neural network heterogenous, the tunable parameters (a, b, c, and d) can vary for each neuron according to a uniformly distributed random variable, enabling the existence of diverse neural dynamics such as regular spiking, chattering, or fast spiking behaviors [30].

Despite that, this model has a critical flaw when it comes to neuromorphic computing. The tunable parameters do not have a biophysical-equivalent component, such as resistance or capacity, making it impossible to model using a purely neuromorphic system.

2.2.2.4 Hodgkin-Huxley

The *Hodgkin-Huxley* neuron model was first used to calculate the action potentials in the squid giant axon. It is one of the most popular techniques to describe the electrical behavior of neurons using a set of non-linear differential equations. The work done by Hodgkin and Huxley was the starting point for the biophysical understanding of the structures now known as ion channels [22]. It not only describes the structural and functional properties of ion channels, including the mechanisms of ion permeating, selectivity, and gating, but also tackles the neuron dynamics at a cellular level by predicting action potentials and the conditions that control their timing [98].

Nonetheless, the high accuracy of this model also makes it computationally expensive, more than any other model mentioned above. On top of that, similar to the *Izhikevich* model, it has many configurable parameters that cannot be mapped to biophysical components. Therefore, it is not a promising option for this dissertation.

2.2.2.5 Adaptive Exponential Integrate-and-Fire Model

The *AdEx* model is a spiking neuron model with two variables. The first equation (2.10) describes the membrane potential dynamics and includes an activation term with an exponential voltage dependence. Voltage is coupled to a second equation (2.11) representing the adaptation. Both variables are reset when triggering an action potential (2.12). The model can describe known neuronal firing patterns, e.g., adapting, bursting, delayed spike initiation, fast-spiking, and regular spiking [39].

$$C\frac{dV}{dt} = -g_L(V - E_L) + g_L\Delta_T \exp\frac{V - V_T}{\Delta_T} - w + I \tag{2.10}$$

$$\tau_w\frac{dw}{dt} = a(V - E_L) - w \tag{2.11}$$

$$if\ spike\ occurs,\ then: \begin{cases} v \leftarrow V_r \\ w \leftarrow w + b \end{cases} \tag{2.12}$$

Presented by Brette and Gerstner in 2005, the *AdEx* model takes inspiration from the exponential integrate-and-fire model [34] and the 2-variable model of Izhikevich [30]. As portrayed in figure 2.6, this model is one of the most accurate representations of biological neurons belonging to the integrate-and-fire family of neuron models [86]. With the appropriate choice of parameters, the *AdEx* model can reproduce up to 96% of the regular-spiking behavior of a Hodgkin–Huxley-type model [83]. Moreover, exciting spike timing predictions were also found for real cortical neurons using genetic algorithms for parameter optimization [51].

2.2.2.6 CUBA Leaky-Integrate-and-Fire Model

The Current-based (CUBA) Leaky-Integrate-and-Fire Model is a model belonging to the integrate-and-fire family that is utilized in many networks due to its simplicity. Here, the neurons are connected through synapses that transmit current. Popular tools such as Slayer [89] and LAVA [68] support this model.

The CUBA LIF model describes the membrane potential of a neuron as it integrates incoming synaptic currents and leaks over time. When the membrane potential reaches a certain threshold, the neuron fires a spike, resetting the potential [94]. The core equations governing the CUBA LIF model are as follows:

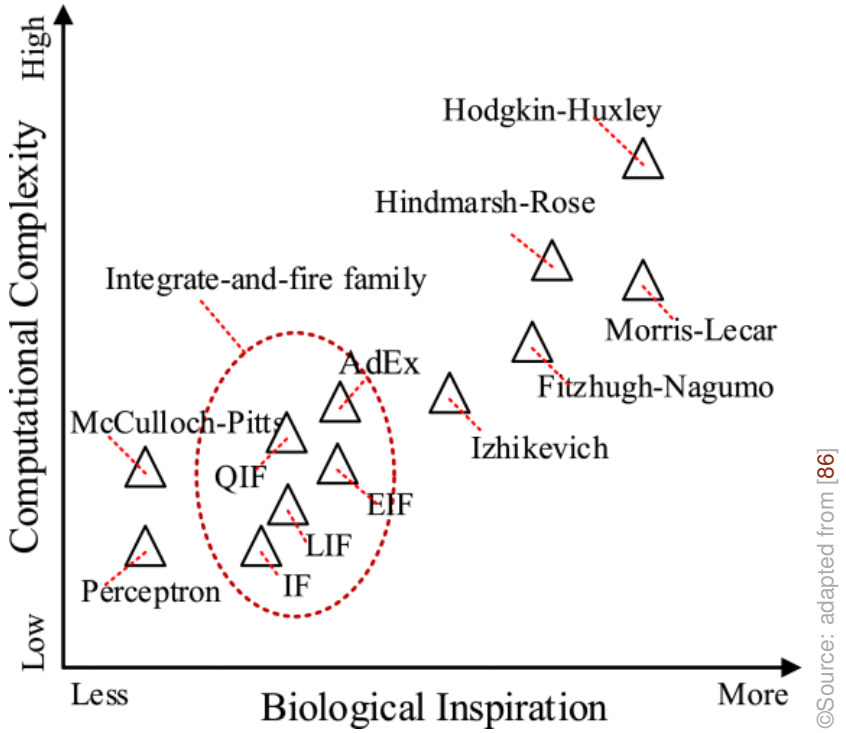


FIGURE 2.6 – Comparison of popular neuron models regarding their computational complexity and biological inspiration.

1. The synaptic current, U_t, evolves according to the following equation:

$$U_t = U_{t-1}\,(1 - d_U) + I_t \tag{2.13}$$

2. The membrane potential, V_t, is updated according to the synaptic current of that time-step:

$$V_t = V_{t-1}\,(1 - d_V) + U_t + bias \tag{2.14}$$

3. When $V > V_t$, the neuron fires a spike and resets the voltage:

$$V_t \leftarrow 0\ (Discontinuous\ dynamics) \tag{2.15}$$

In this set of equations, I_t corresponds to the synaptic current the neuron receives at timestep t. Whilst the *bias* represents a constant increment of the membrane potential for specific scenarios.

2.2.3 Architecture of SNNs

As a computational model that takes inspiration from the nervous system, it may seem intuitive that the architecture of spiking neural networks should be based on the architecture of biological neural networks. In spite of that, there are two main obstacles preventing the creation of SNNs in this fashion. Firstly, the amount of neurons and synapses in the neural system makes the creation

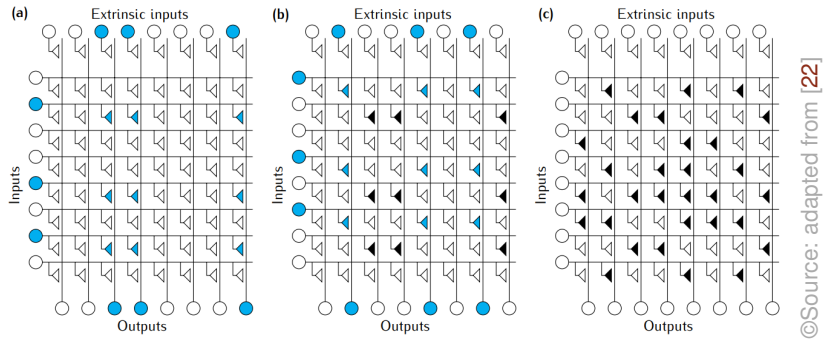


FIGURE 2.7 – Storage of associations on a Feed Forward Associative Network with binary weights. a) Storage of the first association in the network. The input and output patterns are shown as sequences of empty and filled circles and the potentiated synapses are filled. b) Storage of a second association. The original set of potentiated synapses remains (colored black). c) After several associations are stored in the network.

of a full-scale model computationally infeasible. The second restriction is the lack of knowledge of the neurons' functional and structural properties, their arrangement in space, and how they interconnect. Thus, many design questions are raised concerning the communication between neurons, how to model axons and the propagation of action potentials along them, how to scale the number of neurons of different classes without modifying the nature of their interactions, whether the location of neurons in space is relevant, to name a few.

The most commonly investigated properties in these models are the patterns of firing within the array of neurons and how the network can learn and perform valuable computations. The following subsections present some spiking architecture examples and their practical use.

2.2.3.1 The Associative Memory

Associative memory models are a fundamental type of network scheme within computational neuroscience. These models are primarily concerned with storing and recalling patterns, which can represent stimuli or events, and their simplicity makes them ideal for understanding how SNNs store information and retrieve it when necessary. They come in two main forms: feedforward and recurrent networks.

In a feedforward associative network, the objective is heteroassociation, that is, associating two different patterns of activity. For instance, an animal might learn to associate the visual pattern of a rotten branch with the experience of the branch breaking when it tries to swing on it. Contrarily, recurrent associative networks focus on autoassociation, where a pattern is associated with itself. This is akin to recalling a memory from a fragmented input, such as remembering a person's face from just a glimpse of it [22].

Associative networks usually rely on binary synapses and binary input/output patterns, as seen in figure 2.7. Regardless of their simplicity, these models serve as foundational tools for building more complex neural networks and detecting complex activity patterns.

2.2.3.2 Networks of Simplified Spiking Neurons

Integrate-and-fire models are a cornerstone in the study of SNNs, providing a simplified yet reasonably accurate approach to understanding how neurons communicate through spikes. That is why this family of neuron models is commonly chosen to build spiking networks of bigger dimensions. Regardless, the field of SNNs does not seem to gravitate toward a unified solution or architecture that fits most needs, making the network design choice a challenging task [54].

While there is no one-fit-all solution, by analyzing previous investigative works, it is possible to outline some characteristics that help to choose a valid architecture:

1. **Neuron Models**: A plethora of spiking neuron models are currently available throughout the trade-off spectrum between biological accuracy and computational feasibility. The electrophysiological dynamics present in the chosen model may influence the architecture of the resulting network. Section 2.2.2 describes some of the most popular neuron models in greater detail.

2. **Learning Mechanism**: An SNN can be configured to perform a task in several ways. The most obvious approach is through learning, where the literature points to three native training methods: supervised learning with surrogate gradient descent, unsupervised learning with local learning rules, and reinforcement learning. On top of that, it is also possible to train an ANN first and then convert it into a spiking network [101]. A final approach is to fine-tune the network manually according to the dataset, relying on a preliminary study of the input data to optimize the network to the task at hand. The adopted learning mechanisms help to decide how neurons will be connected to take advantage of the learning strategy.

3. **Nature of the Problem**: Another crucial variable is the nature of the problem. Whether the SNN is designed to classify images or to act and mimic the behaviors found in nature will impact the underlying system architecture. Take the N-MNIST [37] dataset as an example, the neuromorphic version of the well-known MNIST [61] dataset. Since the goal is to classify each image as a digit ranging from 0 to 9, it would make sense that the final layer of the SNN conducting this task contains 10 neurons, each representing a digit.

2.2.4 Learning in SNNs

The concept of online learning is a critical advantage of SNNs, allowing them to adapt to new data in real-time. Analogous to biological learning, SNNs learn by strengthening or weakening the artificial synapses, represented by a weight that controls the strength of the connection between the pair. In some cases, learning may also occur by removing synapses or neurons.

Not all SNN implementations require online learning. Some tasks, due to their problem formulation and predictability, can be solved using an SNN architecture that is optimized for that same problem. This requires adjusting the neuron's time constants and the network weights as an optimization step after conducting a knowledge discovery of the input data. While this is a valid solution, as problems grow in complexity, the need for network learning starts to appear.

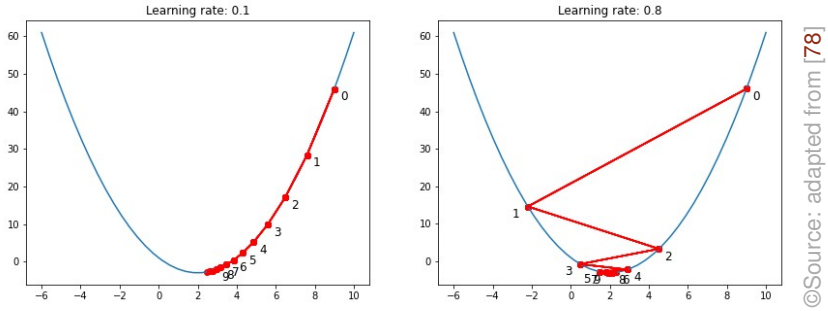


FIGURE 2.8 – First 12 steps taken by Gradient Descent for small and big learning rates.

The following subsections introduce popular learning techniques employed in Spiking Neural Networks. The first techniques are native training methods, that is, the learning occurs in a spiking-natured network. The last method exhibits an alternative by training an ANN and later converting it to an SNN.

2.2.4.1 Supervised Learning with Surrogate Gradient Descent

Surrogate Gradient Descent is a technique that trains SNNs based on simple biological models, such as the LIF. Before explaining how this algorithm works, it is essential to understand how *Gradient Descent* works and why it cannot train SNNs.

Gradient Descent is an optimization algorithm used to minimize the cost of a function using its gradient. The gradient is represented by a vector of derivatives in each axis direction, also called *partial derivatives* (equation 2.16).

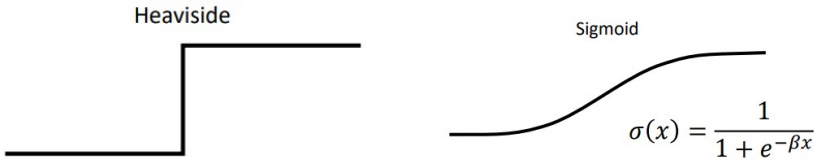
$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \dots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{2.16}$$

The idea is to take iterative steps opposite to the gradient. At each timestep, the following point is calculated by taking the gradient at the current position, scaling it according to a set *learning rate* (η), and subtracting the obtained gradient from the current position, as shown in equation 2.17.

$$p_{n+1} = p_n - \eta \nabla f(p_n) \tag{2.17}$$

The *learning rate* has a strong influence on the performance of the algorithm. If the chosen value is too small, *gradient descent* may take too long to converge. However, if it is too big, the value may not converge to the optimal point or even diverge completely [78]. Figure 2.8 showcases the influence of different learning rates on the learning process.

Hence, gradient descent requires that the loss function is differentiable for every point in its domain. This is the condition that the mentioned SNNs fail to comply with. The LIF model behavior depends on whether the voltage of a neuron exceeds the threshold voltage, such that it



(A) Heaviside function (B) Sigmoid function

FIGURE 2.9 – Heaviside and Sigmoid function plots [19]

can be represented by a *heaviside* function, as the one in figure 2.9. The gradient of this step function is zero for all data points except where it is not defined, so gradient descent is not a viable option for learning in the network. *Surrogate gradient descent* was proposed to solve this problem. By defining a smooth version of the threshold function, similar to the sigmoid presented in figure 2.9, the learning process uses the smoothened function to compute the gradients and perform the backward propagation step, minimizing the loss function. The network's forward pass still uses the *heaviside* function to calculate the state of each neuron, so the surrogate function is defined solely to enable learning through back-propagation.

SpikeProp [12] is one of the first working examples of supervised learning in SNNs using surrogate gradient descent, built to calculate the set of the desired firing timings of output neurons for a given input firing activity. Normally, the derivative term responsible for directing learning depends only on the firing time, but Sporea et al. [44] proposed an extended version named SuperSpike that relies on the derivative of the membrane potential instead of the spike, which allows training a model with an absence of spikes.

A novel tool, Slayer [89], designed to configure deep SNNs with learning capabilities, not only distributes the credit of error (gradient) through the network layers, but also back in time in order to solve the drawback of event-based methods. The Spike LAYer Error Reassignment platform assumes a stochastic spiking neuron approximation for the IF model with a refractory response and can simultaneously learn both synaptic weights and axonal delays [54].

2.2.4.2 Unsupervised Learning with Synaptic Local Learning Rules

The biological process by which certain patterns of synaptic activity lead to variations in synaptic strength is known as synaptic plasticity. Despite being considered the main learning mechanism on biological neural networks, the learning performance of this approach is usually lower than that of supervised learning on in-silico models [85].

One of the primary models for biological learning is *spike-timing-dependent-plasticity (STDP)*. This local learning strategy follows the *Hebb's rule*: "Cells that fire together, wire together.". As such, the synapses of neurons that fire together are strengthened, while the synapses of neurons that fire out of sync are weakened. This is accomplished by comparing the firing timing of pre (t_{pre}) and post-synaptic (t_{post}) neurons. When the pre-synaptic neuron fires before the post-synaptic neuron, the synaptic weight of the connection increases (*Long-Term Potentiation*). Instead, when

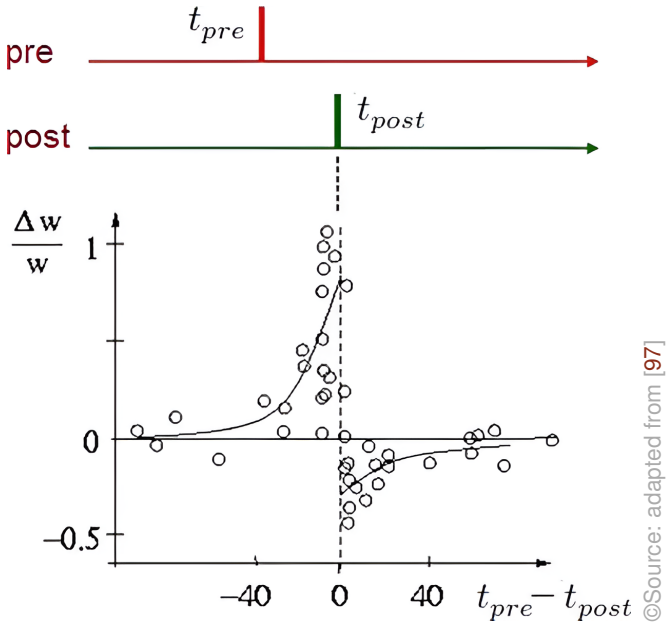


FIGURE 2.10 – Schematic of spike-timing-dependent plasticity. The STDP function expresses the synaptic weight change as a function of the relative timing of pre and post-synaptic spikes. The circles are experimental recordings, and the solid curve is a fitting STDP function

$t_{post} < t_{pre}$, the synaptic weight of the connection weakens (*Long-Term Depression*) [97]. This behavior is depicted in figure 2.10.

The main issue with this solution is that it does not work well in solving demanding tasks. Many alternative hypotheses were proposed [100] and are the target of current research. Another type of plasticity is *STP* [79], that is, a type of plasticity with strong biological evidence whose effect is constrained in time (milliseconds to seconds). Conversely, if the change is persistent, it is a case of long-term plasticity (LTP).

In biology, synaptic efficacy can be modulated by the interactions between Ca^{2+} and neurotransmitters, kinases, and so forth. Contrastingly, the conductance in artificial synapses can be controlled by tuning the movement of ions or other physics [16]. Another paper proposes modeling the short-term learning aspect by splitting the synaptic efficacy G that weights a presynaptic input into a long-term weight W and an additive or multiplicative short-term component F ($G = W + F$) [42]. Subsequently, an update rule depending on local variables increments F, which otherwise decays exponentially with time, indicating a type of learning followed by forgetting.

2.2.4.3 Reinforcement Learning using Reward-modulated Plasticity

Also known as Reward-modulated STDP (R-STDP), this strategy introduces a reward signal to modulate the synaptic plasticity through reinforcement learning. If the reward is positive, the associated synapse is potentiated. Otherwise, the synapse is depressed. Following Izhikevich's work [31], dopaminergic neurons have two distinct firing patterns: a slow (1-8Hz) firing rate in the absence of stimulus, termed background firing, and a period of rapid firing activity, followed

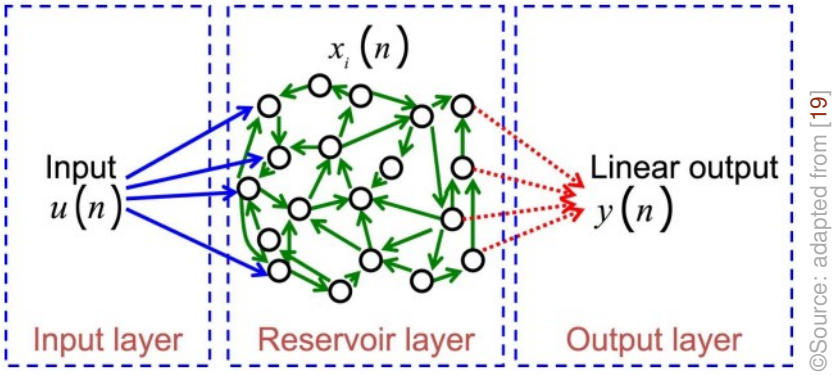


FIGURE 2.11 – Reservoir Computing Network

by a period of inactivity, named burst firing. In this case, the modulation is conducted by including an eligibility trace z for pre and post-synaptic spike occurrence as follows [54]:

$$\frac{dw}{dt} = \eta r(t) z_{i,j}(t) \tag{2.18}$$

$$\frac{dz_{i,j}}{dt} = -\frac{z_{i,j}(t)}{\tau} + STDP(t) \tag{2.19}$$

$r(t)$ is the reward given at time t and z is the eligibility trace.

2.2.4.4 Reservoir computing

Reservoir computing, also known as *Echo State network*, is another interesting model for biological learning. It divides the SNN into three layers: Input, Reservoir, and Output. Because of the random structure of the Reservoir layer, it performs different kinds of computations. While many of these calculations are useless, this method aims to extract the useful computations from the middle layer to compute the output. That is why the training only occurs in the connections between the Reservoir layer and the Output layer, to provide bigger weights to meaningful computations and smaller weights to irrelevant ones [19]. The architecture of such a network is shown in figure 2.11.

Regarding the employed learning strategy, the SNN can update the final layer's weights through supervised or unsupervised strategies. The weights act as a selector of relevant computations from the pool of feature neurons in the middle layer. Given the right circumstances, the reservoir layer can model the neurons tailored to the task that needs to be solved. Thus, by performing a preliminary analysis of the type of data fed into the model, it is possible to create a heterogeneous combination of feature neurons in the Reservoir Layer that captures relevant patterns more efficiently, for example, by varying the membrane potential time constants, synaptic decays, or the connectivity between neurons.

2.2.4.5 ANN Training followed by ANN-to-SNN conversion

As a last method, training an artificial neural network followed by the subsequent conversion to a spiking neural network has achieved comparable results to original ANNs (e.g. VGG and ResNet) [54]. Most methods have focused on converting ReLU layers to IF neurons ([101], [73] and [11]) since the conversion can be verified mathematically. Leveraging the event-driven nature of the algorithms on a neuromorphic design may decrease the costs of signal transmission and computation while holding a similar performance. The conversion is done by sharing pre-trained parameters from an ANN to an SNN, and it is an energy-efficient solution to the deployment time problem of ANNs. Yet, the main drawback of this learning strategy is that it is limited to learning patterns identified by the source ANN, so it does not learn based on the discrete timing of events like the previously mentioned approaches.

2.2.5 Future of SNNs

SNNs are an exciting architecture for neuromorphic computing due to their ability to process information in an event-driven and asynchronous manner that is closer to how biological systems operate. They are gaining traction in several fields concerning real-time and temporal data processing, such as sensory processing, time-series analysis, and pattern recognition. Exploring the dynamics of spiking neural networks may provide insights into crucial biological questions, such as the role of spikes, local learning rules, and unknown interactions with synapse/neuron dynamics.

2.3 Neuromorphic Computing

Neuromorphic computing is an emerging field of computer science and engineering that aims to mimic the structure and function of the human brain using hardware and software systems inspired by biological neural networks. The field is inspired by the notion that the brain is a highly efficient computing system that can outperform current computational machines in many tasks with minimal energy consumption.

Here, traditional chips running clock-driven algorithms are replaced by asynchronous, event-driven algorithms that run on neuromorphic chips. These neuromorphic chips contain artificial neurons and synapses to mimic the complex behavior of the human nervous system. They follow a Spiking Neural Network (SNN) computational model that records the timing and rate of the spikes to process information. Instead of binary values, the fundamental communication unit is the spiking event. Instead of relying on an external memory unit, the circuit has built-in memory, for instance, in the network connections through the use of *memristors* [81].

Memristors are electronic devices that can change the resistance of the connections according to the applied voltage. They simulate artificial synapses in analog neuromorphic circuits by varying the resistance state of the connection between neurons. The resistance state can be changed by

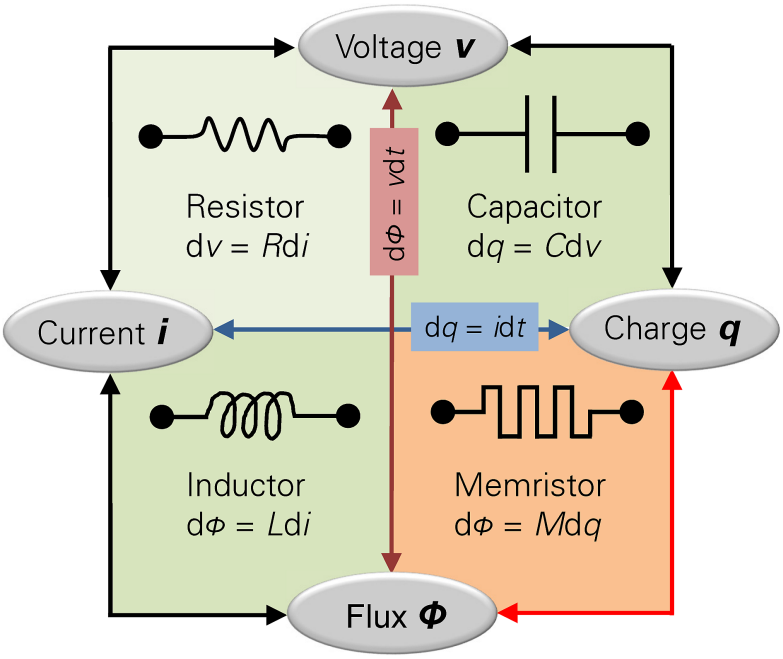


FIGURE 2.12 – Conceptual symmetries of a resistor, capacitor, inductor, and memristor. Adapted from Wikipedia

applying a voltage to the memristor, which alters its resistance, consequently changing the synaptic strength. In addition, neuromorphic systems can also use spike-timing-dependent plasticity (STDP) to modify the strength of the synapses based on the timing of the spikes generated by the neurons. Figure 2.12 clarifies the electrical properties influenced by a memristor and compares it with well-known circuit components.

The event-driven approach enables highly efficient computation, as power is only drawn when relevant events occur. Moreover, the parallel nature of neuromorphic chips is ideal for representing and processing the firing patterns of neuronal data more efficiently. An added benefit of this architecture is the direct communication between the biological system and the neuromorphic processor, which allows it to capture stochastic, non-linear behaviors of neuronal signals.

Recently, the rise of AI large language models (LLMs) and their widespread application have raised awareness of their energy and computational demands. One of the primary causes for these models' computational complexity is the high precision of the parameters connecting artificial neurons, as the number of parameters in Transformer ANNs is in the order of billions and rising. Hence, researchers are attempting to lessen the precision of their parameters through a process called quantization. Xu et al. [103] reduced the parameter's precision and created a 1-bit model occupying only 10% of the original network's memory, retaining a similar performance. Quantized models show the possibility of encoding significant data into 1-bit models, identical to SNNs, and prove that by using a hardware system tailored to them, the efficiency gains are evident.

Figure 2.13 represents the four essential parts of a neuromorphic computing device: receiving and processing spikes, emulating neurons, learning, and sending spikes.

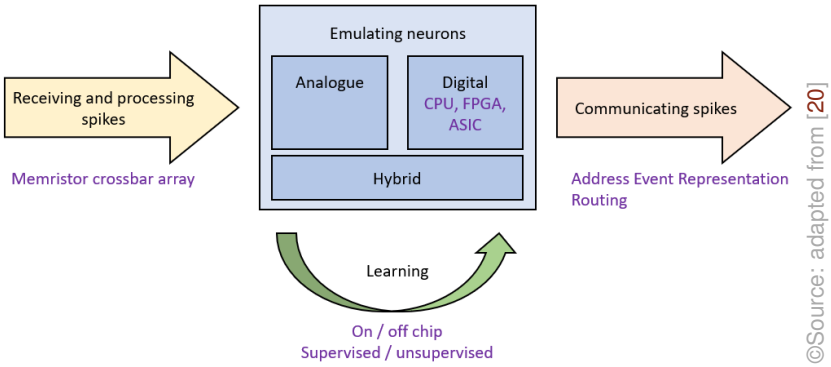


FIGURE 2.13 – Essential parts of a neuromorphic computing chip

2.3.1 Neuron Emulation

Neuron emulation is concerned with replicating the behavior of biological neurons. The several solutions on the market can be split into analog, digital, and hybrid approaches, each having its advantages. Analog approaches [50], such as *subthreshold analog* and *above threshold analog*, exceed the others in efficiency.

On the other hand, digital approaches offer unrivaled flexibility at the expense of efficiency. This flexibility-efficiency trade-off justified the emergence of different digital systems: CPU-based [74], FPGA-based [90], and ASIC-based, with an increasing focus on efficiency. *ASIC* systems are fully customized chips optimized for specific tasks. Finally, hybrid approaches combine both aforementioned systems. Figure 2.14 presents an overview of products implementing different neuron emulation strategies. The adopted chip for this dissertation, *Loihi*, consists of a digital application-specific integrated circuit.

2.3.2 Intel Loihi

Loihi, developed by Intel Labs, is a research chip that embodies the principles of neuromorphic engineering, offering a system that emulates the neural structures and processes of biological brains. Such systems' design is fundamentally different, relying on asynchronous event-based processing rather than the synchronous clock-based systems seen in traditional CPUs and GPUs.

In 2018, Intel launched the first Loihi chip [68], marking the company's entry into this industry. The processor intended to facilitate self-learning capabilities and included features such as support for SNNs, biological neuron models, and an asynchronous network-on-chip (NoC) that carried all communication between neuromorphic cores. Due to the effort to build a community around these chips, the Intel Neuromorphic Research Community (Intel NRC), the first-generation chip was evaluated in a wide range of applications, such as adaptive robot arm control [95], visual-tactile sensory perception [91], and solving hard optimization problems such as railway scheduling.

Later, in 2021, Intel introduced Loihi 2, offering significant improvements in both hardware and software. The second-generation chip features a more advanced neuron model and learning rules, supporting a broader range of neural computations. It also includes improvements in

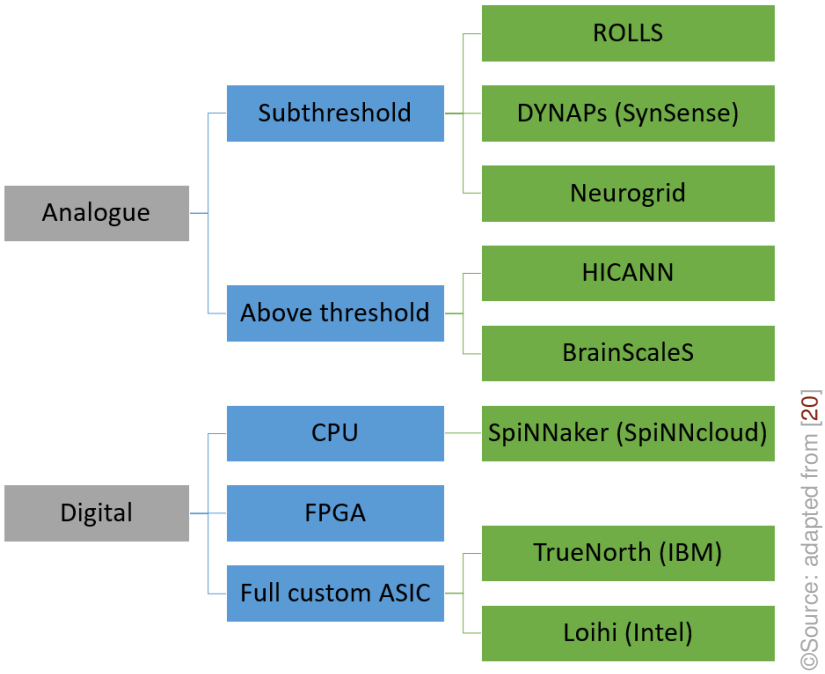


FIGURE 2.14 – Neuron emulation approach of different systems in the market

chip-to-chip communication and scalability, paving the way for more complicated neuromorphic systems. Additionally, the new processor permits spikes to carry integer-valued payloads rather than binary information to provide greater numerical precision. Table 2.2 showcases some of the main differences between the two chips. Meanwhile, figure 2.15 presents an overview of the most recent CPU architecture.

Last but not least, the existence of Lava, a community-driven, open-source neuromorphic computing framework developed by Intel, was a deciding factor in choosing Loihi as the processor for this dissertation. The following section introduces this tool and explains its relevancy.

Features	**Loihi**	**Loihi 2**
Die Area	$60mm^2$	$31mm^2$
Core Area	$0.41mm^2$	$0.21mm^2$
Transistors	2.1 billion	2.3 billion
Max # Neurons Cores / Chip	128	128
Max # Processors / Chip	3	6
Max # Neurons / Chip	128,000	1 million
Memory / Neuron Core	208KB, fixed allocation	192KB, flexible allocation
Neuron Models	Generalized LIF	Fully Programmable
Information Coding	Binary Spike Events	Graded Spike Events (up to 32-bit payload)

TABLE 2.2 – Feature Comparison between Loihi and Loihi 2. Adapted from [43]

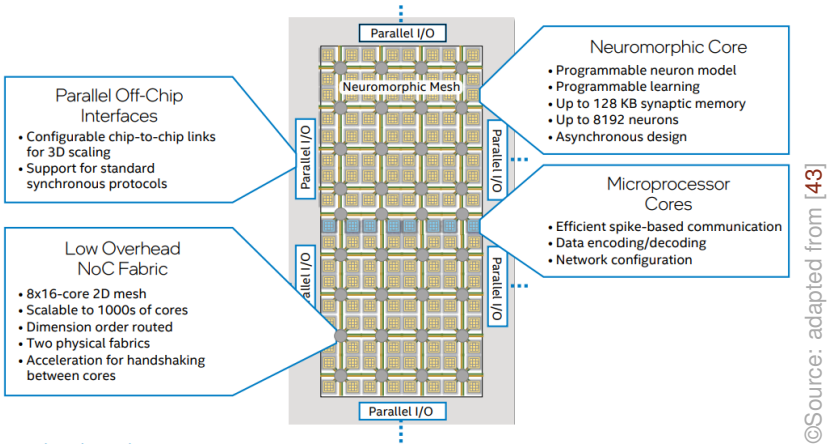


FIGURE 2.15 – Intel Loihi 2 processor architecture

2.4 Lava Software Framework

Lava is an open-source software framework for developing neuro-inspired applications and deploying them to neuromorphic hardware. The tool allows developers to build applications that exploit the principles of neural computation, allowing neuromorphic systems to intelligently process, learn from, and respond to real-world data with significant benefits in power efficiency and speed compared to conventional computer architectures.

This tool created by *Intel* aims to create an open, community-developed code base that unites the diverse approaches pursued by the neuromorphic computing community. Lava is platform-agnostic so that applications can be prototyped on conventional processors and deployed to heterogeneous system architectures spanning both traditional CPUs as well as a growing range of neuromorphic chips, including Intel's Loihi [60].

Note that the lowest-level components necessary for developing and deploying applications to Loihi 2 hardware systems are restricted to Intel NRC members. Accordingly, the research group I am working with obtained access to the proprietary materials after applying to the INRC with this dissertation topic in mind.

2.4.1 Lava's Foundational Concepts

This section will introduce the four main elements that govern this framework. First, *Processes* are the fundamental building block in the Lava architecture from which all algorithms and applications are built. Every computational block in Lava is a *Process* that has its own private memory and communicates with its environment solely via messages. As such, a *Process* can be as simple as a single neuron or as complex as an entire neural network or a general program like a search algorithm.

However, *Process* classes only define the interface of a process in terms of internal variables, ports, and other class methods. To specify the behavioral implementation of a *Process* and make it executable on a particular hardware, *Process Models* are used. A *Process* can have multiple

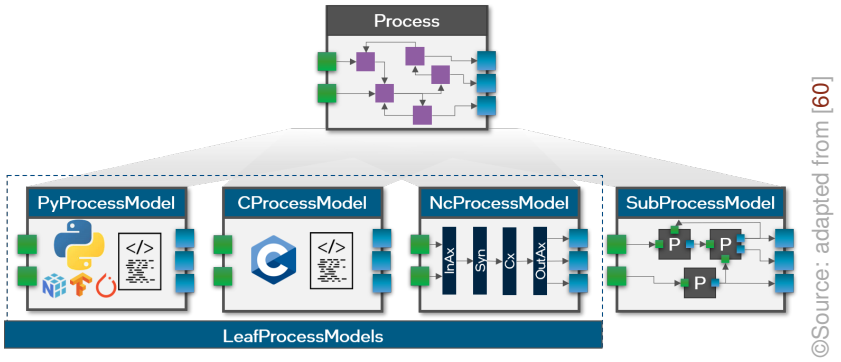


FIGURE 2.16 – Different types of Process Models in the Lava framework

ProcessModel implementations, each representing a different behavior, similar to how an interface can have various implementations in the conventional programming paradigm. This abstraction allows quick algorithm prototyping on a traditional CPU-based system and later implementation on lower-level platforms, such as embedded systems or neuromorphic hardware. Figure 2.16 showcases different types of process models.

A missing fundamental step is the connectivity between computational units. *Processes* are connected via ports for message-based communication over channels. There are four main types of ports that can be grouped in two different ways:

- *OutPorts* connect and send messages to *InPorts* by value;
- *RefPorts* connect to *VarPorts*, enabling the access to a variable by reference;

Besides the ones mentioned above, additional port types exist for specific scenarios. For instance, *ReshapePorts* to change the shape of a port and *ConcatPorts* to concatenate multiple ports into one.

As a final point, Lava supports cross-platform execution of processes on a distributed set of compute nodes. Each node has associated computing resources, such as CPUs or neuro cores, and peripheral resources, such as sensors or actuators. Given a graph of *Processes*, their *ProcessModels*, and a run configuration, the compiler can map processes to the designated nodes, distributing the computation between them.

2.4.2 Lava Software Stack

The core components of the Lava software stack, the *Magma layer*, are comprised of the *Communicating Sequential Process API*, a powerful compiler, and a runtime. Magma is the foundation on which new processes are built. As presented in figure 2.17, the *Process Library* lays on top of *Magma* and offers a growing collection of generic, low-level, and reusable components from which higher-level applications are created.

Some libraries are under active development and released as part of the Lava software framework. At the time of writing, these are for deep learning, optimization, and dynamic neural fields.

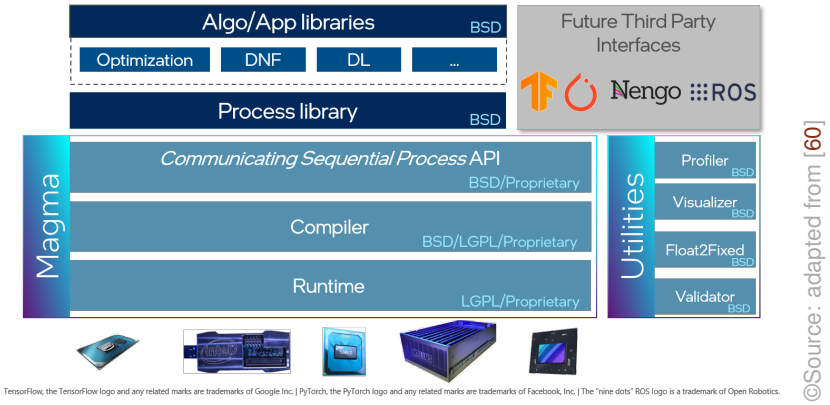


FIGURE 2.17 – Software stack of the Lava framework

Efforts are being made to support vector symbolic architectures and evolutionary optimization soon. Aside from these components, the framework has utilities for application profiling, automatic float-to-fixed-point model conversion, and network visualization.

2.4.3 Development and Deployment

As previously stated, *Lava* is a multi-platform Python framework. This means that the developed Python code can be translated and deployed to different hardware systems, including conventional CPUs and Intel's *Loihi*, with the help of dedicated compilers.

This crucial feature helped to choose Lava as the adopted programming framework since it is possible to rely on a regular computer for the beginning development stage. Thanks to the abstraction layers built onto the framework, through the definition of interfaces, also known as *AbstractProcesses*, switching between platforms simply requires defining a *ProcessModel* for each architecture. It is also possible to create various process implementations for the same architecture using tags.

Therefore, Lava encourages developing algorithms that run on a *von Neumann*-based system relying on a Loihi simulation and later translating them into a format natively supported by the Loihi hardware. This aligns well with this assignment since access to Intel's specialized hardware is unavailable at the start of the work.

2.4.4 Fixed-Point Precision Architecture

The Loihi hardware follows a fixed-point precision architecture, contrasting with the popular floating-point precision model. At the expense of sacrificing dynamic range, computations get faster and consume less power. The fixed-precision model, also known as bit-accurate, only supports integer computations. Thus, every float is converted into an integer through parameter space mapping.

In the case of Loihi, the float-to-integer mapping is explained in the in-vitro practical experiment 4.4.4. As a result of this conversion, the precision and range of the process parameters that

follow this architecture are reduced according to the number of allocated bits for each. Anyhow, the neuromorphic chip's specialized digital circuits take advantage of the constrained parameter space to perform efficient computations. Each neuron and synapse uses fixed-precision registers to store values such as membrane potentials, synaptic weights, and voltage thresholds, ensuring rapid updates and real-time capabilities.

Lava developers recommend initially building models for a simulated environment that runs on floating-point precision. Then, translate the algorithm with decimal numbers to a bit-accurate version that also runs on a traditional CPU through the Loihi simulator. This way, the transition from the fixed-precision process running on the Loihi simulator to the fixed-precision program running on Loihi will be smoother and less prone to errors.

2.5 High-Frequency Oscillations and Epilepsy Treatment

2.5.1 Epilepsy and Existing Treatment Options

Epilepsy is the most common severe neurological disease, where patients suffer from recurrent seizures, which can have a variety of repercussions, including a threefold increased chance of premature death when compared to healthy individuals [99]. The standard initial treatment for epilepsy is anti-seizure medication (ASM), which results in seizure freedom in about 60% of patients. For the remaining patients, considered to have drug-resistant or refractory epilepsy, other treatment options must be explored.

In patients with drug-resistant focal epilepsy, seizure freedom may be achieved after epilepsy surgery. The primary goal of epilepsy surgery is to disconnect the epileptogenic zone (EZ), which is responsible for causing seizures. The EZ is defined as "the minimum amount of cortex that must be resected to produce seizure freedom" [47]. Current implementations use the seizure onset zone (SOZ), the area of the brain where seizures start, as an approximator of the EZ. However, studies suggest that this approximation may be hindering the effectiveness of this treatment. For this reason, it is important to find new EZ biomarkers that allow delineating the area of interest in greater detail [3]. A variety of eletrophysiological and imaging methods were also tested, for example, by using intracranial EEG (iEEG) recordings, the area of interest can be outlined through epileptiform potentials (i.e., epileptic spikes or spike waves). However, these traditional biomarkers have been shown to lack a stable correlation with the disease activity [84].

2.5.1.1 Stereo-Electroencephalography (sEEG)

sEEG is a technique developed during the 1950s in Europe that allows recording electrical activity in the brain both cortically and subcortically [5]. The technique involves the surgical implantation of intracerebral electrodes with 4-18 contacts with a diameter of 1 mm or less and placed 2-10 mm apart. Each of these contacts will record the electrical activity of nearby neurons [57] as represented in figure 2.18. SEEG recordings are characterized by a high signal-to-noise ratio, higher spatial specificity, and the possibility of recording deep brain areas. With that stated, since only a

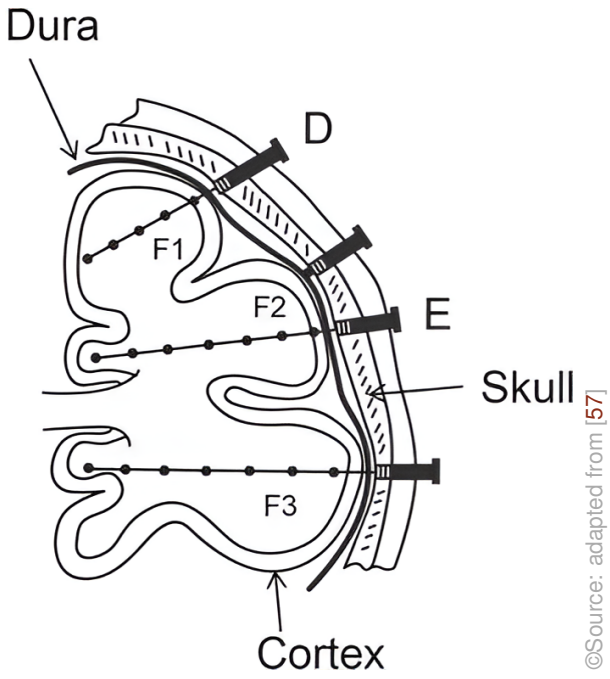


FIGURE 2.18 – A diagram depicting an sEEG implant, including the coverage across the frontal lobe, encompassing both surface and deep regions.

few electrodes are implanted, it has poor spatial sampling. Given the low amplitude characteristics of HFOs, the high quality of sEEG signals is critical for HFO detection [3].

2.5.2 High-Frequency Oscillations as a Relevant Biomarker

Recent research suggests that high-frequency oscillations (HFOs) can serve as a useful biomarker for epileptogenic tissue [10]. In general, these events are defined as spontaneous EEG patterns that consist of at least four oscillations that stand out from the background noise of the signal. HFOs are characterized by low amplitude and a high frequency that can be further divided into two categories: ripples (80-250Hz) and fast ripples (250-500Hz). Manual HFO detection is prone to errors due to the variability in the duration and frequency of these events. In fact, these properties can differ for each patient and even within different electrode locations of the same patient. Nonetheless, table 2.3 presents the duration and event rate distribution for ripples and fast ripples in the epileptic human brain [4]. Figure 5.3 showcases some recorded HFOs in a clinical setting.

		Ripple	**Fast Ripple**
Duration (ms)	Average	60	23
	Range	[50, 120]	[10-40]
Rate (per min)	Average	5	2.1
	Range	[0.1, 20]	[0.5, 6]

TABLE 2.3 – Average and Range Distribution of the duration and frequency of ripples and fast ripples in the epileptic human brain [4]

The delineation of the EZ using HFO has proven to be highly predictive of seizure outcome [96]. Ergo, it has the potential to improve the outcome of epilepsy surgery. Recent data suggests that HFOs are also measurable by non-invasive scalp EEG, which opens the opportunity to monitor disease severity and explore treatment responses via electrical stimulation [52, 102]. The latter use cases of HFO detection require the development of systems capable of identifying relevant events and actuating accordingly in real-time, which is why this study presents a neuromorphic solution for this problem.

2.5.3 Existing Methods for HFO Detection

The Morphology detector and the Spectrum detector are two of the most popular conventional algorithms to detect high-frequency oscillations [55]. Both strategies are split into two stages. First, the detectors establish a baseline by identifying high entropy segments with low oscillatory activity when the signal is transformed in the time-frequency domain [76] and tag the events that surpass this baseline threshold as Events of Interest (EoI). The second stage is where the classifiers deem the EoI as HFO or not. The Morphology detector marks the EoI as an HFO based on a predefined template for the morphology of an HFO. The Spectrum detector, on the other hand, evaluates the time-frequency signature of the EoI and tags the event as an HFO if it exhibited an isolated high-frequency peak [53].

Moreira [3] explored a new approach for HFO detection using deep learning predictive models. The work presents a Convolutional neural network (CNN) and a Long short-term memory (LSTM) network, both able to outperform the traditional detectors when the signal-to-noise ratio is at least 10dB. The data goes through a pre-processing stage where it is filtered to a desired frequency and windowed before being fed into the model.

The solutions mentioned above have a crucial limitation, as they require the data to be pre-recorded before performing the detection – offline detectors. Offline detection is not suitable for many scenarios, such as monitoring disease severity, raising a need for alternative implementations that do not rely on conventional computers. In 2022, Burelo [53] et al. developed a spiking neural network able to detect HFOs running on the neuromorphic hardware DYNAP-SE2. Instead of following a traditional machine learning approach, the team optimized a range of values for the synaptic parameters based on the analysis of HFOs and noise samples in the training data. Therefore, the network weights are not learned through backpropagation or local learning rules.

2.6 Related Work

This dissertation represents the first usage of digital neuromorphic ASICs and Intel's Loihi, in particular, from the NCN research group in i3s, to which I am affiliated for the execution of this investigation work. Despite that, the lab conducted several experiments where they analyzed neural activity using traditional CPUs, namely detecting relevant biomarkers included in this dissertation.

Similar to chapter 5, the work conducted by Moreira [3] at NCN attempts to detect HFOs using deep learning models instead of SNNs. The same datasets were available to train the models, making the data processing stage a relevant starting point for the real-time implementation.

From another group, Sharifshazileh et al. [69] developed an electronic neuromorphic system for real-time detection of HFOs in iEEG. Although the neuromorphic hardware utilized is different (DYNAP-SE), the objectives of the investigation are aligned with this report in that they both seek to validate neuromorphic computing as an advantageous solution to detect patterns in neural activity. Moreover, despite having different hardware, the underlying spiking neural network can be inspired by the one presented.

Lastly, Dias et al. [15] proposed a memristor-based neuromodulation device for the real-time monitoring and control of neuronal populations. The work was conducted at the NCN lab using in-vivo populations with an identical setup to chapter 4.

Chapter 3

Methodology

This chapter presents the adopted methodology and design decisions applied to both practical studies. First, the available data sources are introduced, followed by an overview of the different hardware backends, the technology stack, and the evaluation approach and metrics.

3.1 Data Sources

As multiple characteristics of neuronal populations are analyzed, the underlying data sources are also different. Thus, the following datasets are utilized:

1. **Spiking activity of in-vitro cortical rat neurons**: For detecting BNN characteristics derived from the spiking activity of the network, such as network bursts. This electrophysiology data is produced at the *NCN research lab* and is further explained in section 4.2.

2. **Synthetic sEEG data from epileptic patients**: Validation dataset of synthesized elements to train and test HFO detectors [71]. This publicly available simulated data was created using sEEG recordings of non-REM slow-wave sleep of drug-resistant epileptic patients. Sub-section 5.2.1 clarifies the data in further detail.

3. **Clinical sEEG data from epileptic patients**: Restricted dataset acquired in collaboration with Hospital de Santo António to test the detection of high-frequency oscillations in a clinical setting. Sub-section 5.2.2 dives into the details of the recordings conducted through this collaboration.

3.2 Available Hardware

Acknowledging the potential of the experiment presented in this dissertation, Intel granted the NCN lab group access to Intel's Loihi research chip to support this project. Therefore, the developed event-driven algorithms were tested in the following running configurations:

1. Loihi simulator running on a traditional CPU.

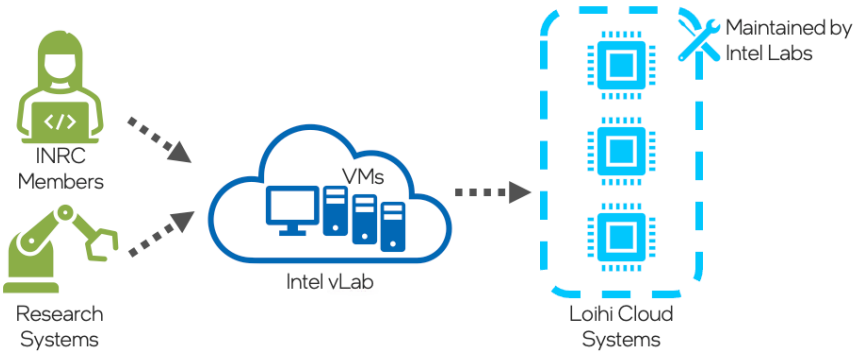


FIGURE 3.1 – Loihi Cloud Systems access through Intel vLab Virtual Machines.

2. Neuromorphic Loihi device accessed through SSH.

Intel does not initially grant access to the physical neuromorphic chip, preferring to first implement a proof of concept. However, Intel granted the credentials to access their VMs and run the event-driven algorithms in a neuromorphic environment, as depicted in figure 3.1.

Thus, the methodology recommended in the LAVA documentation was followed, starting by implementing the algorithms to run on Python in conventional CPUs and later adding the Process-Model versions using the fixed-precision architecture, explained in sub-section 2.4.4, that runs on Loihi.

3.3 Feeding Data in Real-Time

In the ideal scenario, the computational setup will receive the input directly from the experimental environment, process it, and create the output in real-time, as portrayed in figure 3.2.

However, for the purpose of iterative development and building a simple working solution, the neuronal activity is stored in binary files, which is adequate for storing large amounts of data. Then, the input data is loaded by the computer and fed into the developed SNN models. However, there is a missing component in the system: the bridge between the binary file containing the spiking activity and the *SNN*.

Since we intend to create a system able to receive spikes in real-time, the continuous arrival of spikes must be emulated when using spiking activity from a binary file. For this reason, the firing activity has to be sorted by ascending order of arrival, where each spike indicates the electrode from which it originated. To complete the emulation of online arrival, we define a *Lava Process* named *SpikeEventGenerator* that is going to generate the input feeding the spiking neural network, thus replacing the *MEA* component in figure 3.2. This Lava Process is synchronized with the network and feeds the input to the SNN according to the time steps and respective mapping between the input electrodes and the network's neurons. Essentially, It replaces the outside stimuli component during development, and the implementation can be found in Appendix C.2.

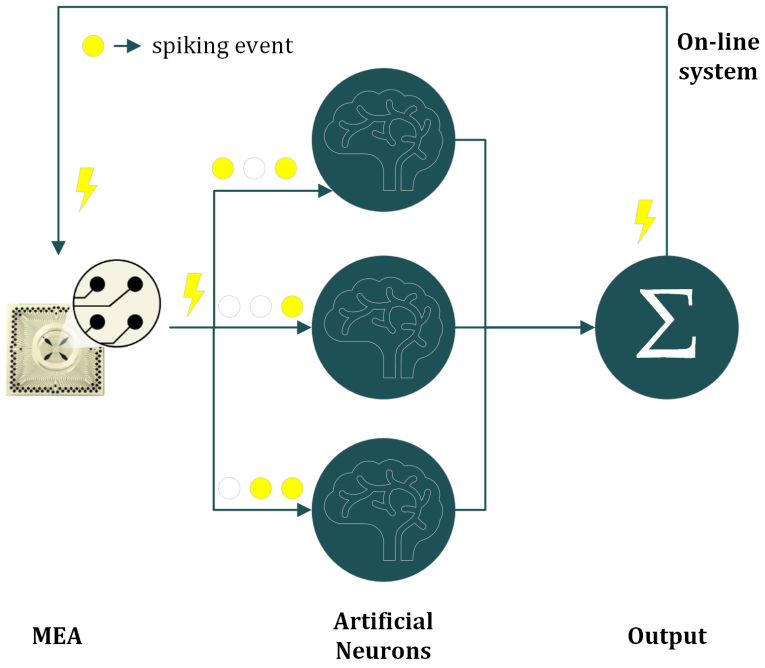


FIGURE 3.2 – Representation of the ideal *online* neuromorphic system, receiving direct electrical input from the sensors and delivering output in the same fashion, in real-time.

To leverage the benefits of neuromorphic computing and optimize the system's efficiency, a neuromorphic system should avoid processing the data on a traditional von Neumann-based machine since it is inadequate for treating asynchronous data. Hence, by reproducing the real-time feeding of data to the SNN, the transition from running the neuromorphic algorithms in a conventional CPU using a simulator to the specialized neuromorphic hardware (Loihi) connected to an external analog system that feeds the stimuli is guaranteed to involve less design changes.

3.4 Software Working Environment

We use the Lava framework to model neuro-inspired neural networks and deploy them to different hardware backends. This open-source tool created by Intel fosters a community-developed environment open to extension. To compile and execute programs built on Lava for different backends, the framework counts on a low-level interface called Magma with a powerful compiler and runtime library.

Since Lava is a Python framework, Python is the primary programming language in the project. Its extensive ecosystem, ease of use, and availability of numerous libraries for scientific research make it a great option. This programming language has many ready-to-use packages for data processing and neural networks, allowing us to load, process, and visualize data using the same coding language.

We used both Lava and Slayer 2.0 to configure the SNNs. The Spike Error Layer Reassignment Platform (Slayer 2.0) comes built into the Lava Framework [60] and is designed to add supervised

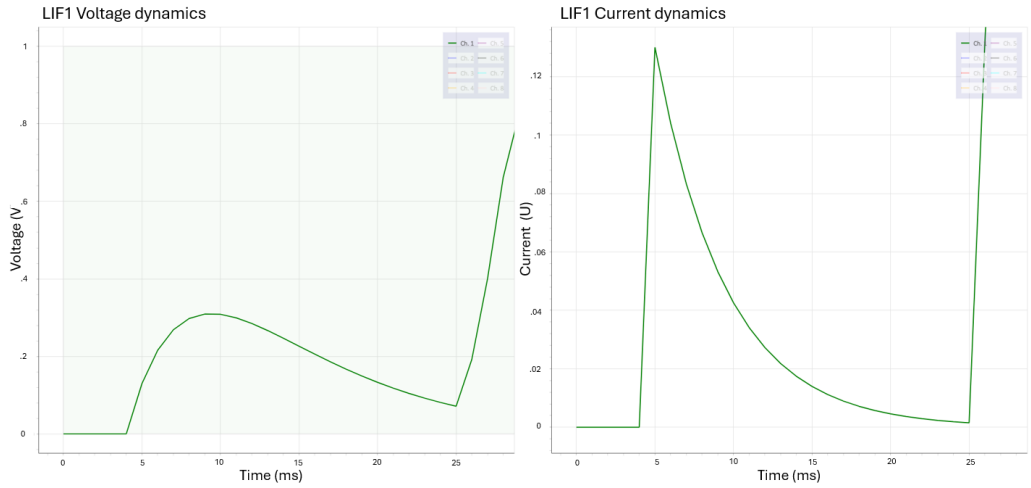


FIGURE 3.3 – Voltage and Current dynamics of a LIF neuron at a millisecond time scale.

learning capabilities to deep SNNs. The Network Exchange Library (NetX) is included in the Lava Toolbox to translate Slayer models to Lava Processes.

Regarding data processing, we relied on the NumPy and Pandas libraries for efficient data manipulation. Due to the nature of the problems being solved, interactive visualization tools are essential. So, we included Bokeh [13] in our tech stack due to its capability to handle large datasets and render high-quality interactive visualizations. The output graphs are also in HTML format, making them universally accessible.

We rely on Jupyter Notebook to combine executable code, visualizations, and documentation into a single document. It also offers a development environment when accessing the Intel Labs virtual machines through SSH for the fixed-point precision version of the algorithms.

At last, all the dissertation coding repositories are stored on GitHub [66, 65]. The repository using the proprietary libraries from Intel targeted toward the Loihi hardware cannot be shared publicly, as only members of the INRC have access to the tools.

3.5 Data Visualization

Analyzing the behavior of spiking neural networks at different time scales offers helpful insights into the system's dynamics. This behavior can be shown through the voltage and current evolution across time, also referred to as the *inner state*.

On the one hand, visualizing the inner state at a millisecond time scale shows the behavior of the modeled neurons in response to individual stimulations. The increased resolution facilitates viewing meaningful properties such as the voltage and current decay time constants. Figure 3.3 is an example of such a case, where it is possible to see how the voltage (left picture) reacts to a single impulse, represented by the sudden current increase (right picture).

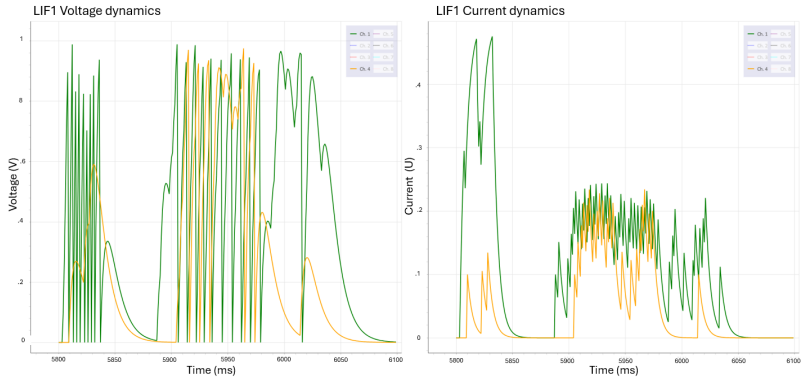


FIGURE 3.4 – Voltage and Current dynamics of 2 LIF neurons at a larger time scale.

On the contrary, looking at a larger window of the network dynamics provides other kinds of valuable information, such as the detection of bursting periods, intervals with no activity, refractory states, and causality between neurons, to name a few. Figure 3.4 presents the internal state of a *LIF* layer with two neurons at a larger time scale. In this case, it is possible to detect periods of high-frequency firing and analyze whether the activity of both channels is synchronized.

Ergo, a set of utility functions was developed leveraging the existing *bokeh* [13] python library to create interactive plots that permit adjusting the time window of visualization, showing or hiding the activity of selected neurons, and other functionalities that add flexibility to the plots. Besides, the plots are exported as *HTML* files, making them easily shareable.

3.6 Evaluation Approach and Metrics

Throughout the dissertation, an effort was made to standardize the classification metrics. By presenting the results using the same methodology, it becomes easier to understand the project's outcomes and relate the different practical studies. Accordingly, the result presentation will follow a set structure:

1. Show the dynamics of the modeled neurons' state at different time scales.

2. Present the classification metrics: accuracy, precision, recall, f1-score, and specificity, and explain which ones are most relevant for each use case.

3. For the algorithms with an equivalent fixed-precision implementation, compare the results of the previous two stages with the floating-point procedure.

4. Provide a profiling analysis of the processes running on the Loihi hardware, focusing on the running time, energy consumption, activity, and memory usage. Intel Labs recommends following a benchmarking structure designed by the team to make results universally comprehensible. Each workload was repeated five times and averaged out to achieve accurate measurements. Additionally, it is essential to select a specific Loihi board to execute all

programs in the same conditions, which is done by setting the environment variables of the connected Intel vLab machine through SSH.

5. Discussion and interpretation of the studies' outputs.

Chapter 4

Practical Study on an *in vitro* Neuronal Population

This chapter introduces the practical experiment conducted to analyze neuronal signals in real-time from an in vitro population of neurons, that is, a neuronal population that is cultivated in a closed and controlled laboratory. The experiment was executed at the Neuroengineering and Computational Neuroscience Lab in i3s, and the outputs were used to test and validate the developed neuromorphic algorithms.

4.1 Problem Formulation

Given an *in vitro* setup of hippocampal neuronal cultures, the experiment aims to develop event-driven algorithms based on neuromorphic computing to monitor the activity of the neuronal populations. By relying on electrophysiological biomarkers, these algorithms aim to characterize the biological network's properties, such as detecting network bursts, high-frequency oscillations, synchronization, and other pertinent events.

The detection of specific patterns of neural activity is a vital challenge in neuro-engineering, as it enables the development of new treatments for chronic diseases and deepens our understanding of the biological system the neuromorphic device pretends to emulate, the nervous system. More specifically, the online processing of neuronal activity simultaneously with data acquisition is an open challenge in the field that this experiment attempts to address and is a fundamental step to building efficient closed-loop systems [24]. In this dissertation, the focus is on the online detection of relevant events with a neuromorphic design to validate the benefits of the architecture. Hence, there is no network feedback control as in a closed-loop neuronal stimulation system.

The network feature that this study focuses on is Activity Bursts. Channel and Network Bursts, to be more precise, because they reflect the collective dynamic behavior of neural circuits. Channel bursts, which represent the synchronized firing of individual neurons, provide insights into the excitability and responsiveness of neurons. Conversely, network bursts measure the coordinated activity across multiple neurons, capturing the intricate patterns of interaction within the neural

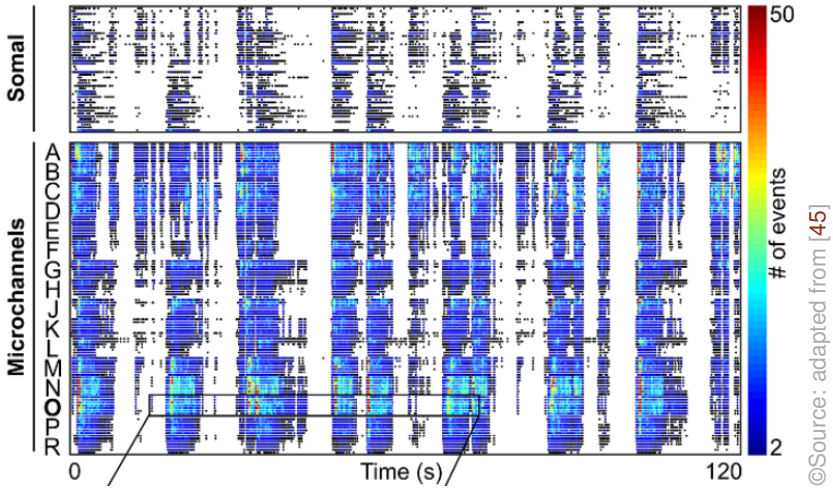


FIGURE 4.1 – Color-coded raster plots of 2 minutes of activity of all the active microelectrodes of a representative hippocampal culture.

network. Bursting refers to a series of rapid spiking, followed by a quiescent period with no or few spikes. By detecting these two biomarkers, it is possible to assess the algorithm's performance in replicating localized and distributed neural activities, ensuring that the computing device can faithfully emulate the complex, real-time dynamics observed in BNNs.

With this practical study, we intend to validate the technology with the detection of a straightforward biomarker and set the stage for building more complicated examples, like Chapter 5.

4.2 Dataset

The training and testing data to develop the Activity Burst detectors was gathered at the NCN lab. The i3s group handled the biology experiment and assisted in generating said data. To gather reliable performance metrics of the model, several recordings were conducted.

The outputs of the experiment are stored in files following the hierarchical data format (*HDF5*) [59], specialized in storing large amounts of data. Then, the input data is loaded by the computer and fed into the developed SNN models. The plot in figure 4.1 exemplifies the information encoded in the input file, where each dot represents a spiking event on a specific microchannel.

Each recording is roughly 5 minutes, with an estimated spike frequency of 133.4 spikes/second over the 251 channels. Consequently, each input file contains around 40000 spiking events. Section 4.3 explains how the data is loaded and handled to integrate with the remaining solution pipeline.

4.2.1 Ground Truth

As shown in figure 4.1, the input data represents the activity of neuronal cultures, so there is no pre-defined labeling of the ground truth events, in this case, the timestamps of the channel and network bursts. Given that there is no universal definition of a channel or network burst, the

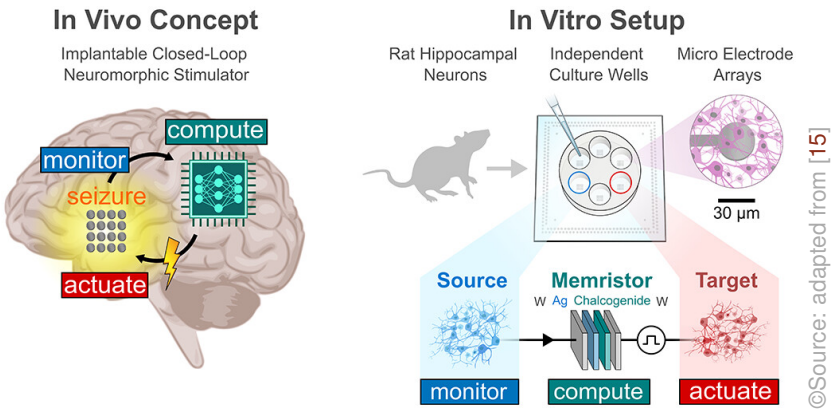


FIGURE 4.2 – Setup of a memristor-based neural interface in both *in vivo* and *in vitro* settings.

ground truth events will be generated using a computational utility for each definition of activity bursts employed. A timestamp will be associated with each activity burst, representing the time step in which the burst condition was satisfied. However, there is a causality time window where the SNN can spike to consider the spike related to the annotated event.

4.2.2 NCN Lab Methods

4.2.2.1 Cell Cultures

The experiments followed the European legislation regarding the use of animals for scientific purposes and the protocols approved by the ethical committee of i3S. Embryonic (E18) rat hippocampal neurons were cultured on a six-well chamber MEA (256-6well MEA200/30iR-ITO-rcr) (MCS, Germany) with a density of 5×104 cells/well. Each well of the six-well MEA has 42 TiN circular electrodes (array of 7×6) with a diameter of 30 µm [24].

4.2.2.2 Lab Experimental Protocol

Regarding the experimental protocol, the neuronal cultures of cortical rat neurons were analyzed in real-time using MEAs, with a total of 252 electrodes split into a 6-well configuration. There was no outside stimulus influencing the population. As a result, the recorded electrical activity corresponded to the spontaneous firing of the cells. The cell cultures were maintained at 37ºC and 5% CO_2 using a stage-top incubator. The electrophysiology signals were sampled at 10kHz and filtered with a second-order high-pass Butterworth filter at 200Hz. Using the Experiment software from MCS, any peaks that were 6 to 8 standard deviations below or above the baseline were identified as spikes. The 3ms after each detection were ignored to prevent biphasic spikes from being identified twice. Finally, the spike data was exported to *HDF5* format for posterior analysis. Figure 4.2 showcases the *in vitro* protocol from a similar experiment conducted at the NCN lab by Dias et al. [15]. Note that this work does not have an actuator since we only monitor the population activity.

Channel Index	Spike Times (ms)		
0	69486.8	173984.7	193738.7
1	1427.1	1430.4	
2	203210.7		

TABLE 4.1 – Structure of the original *HDF5* file containing the spike events

4.3 Input Handling

The first step in tackling the problem of Activity Burst detection is to understand the input data derived from the experimental setup. The next three segments depict the data understanding process.

4.3.1 Data Acquisition

The input data is stored in an accessible file location using Matlab binary files. To use it, the *loadmat* utility function from SciPy allows importing Matlab-formatted files into a Python dictionary. The relevant fields for this study store the total recording time and the spike timings for every channel.

4.3.2 Data Processing

The original data after loading follows the format shown in table 4.1. Yet, the system is built to receive input in real-time, so the continuous arrival of spikes must be emulated when using spiking activity from a file, as mentioned in section 3.3. For this reason, the original format must be transformed into a structure that fits the online arrival of action potentials. The resulting structure is shown in table 4.2, where the spikes are in ascending order of arrival, and each spike references the channel it originated from.

As an optional step, it is possible to indicate a set of relevant channels from which the spikes are included in the processed input file. Lastly, the information is exported into a comma-separated value (CSV) file to be used in the following stages.

4.3.3 Data Visualization

To gain an intuition of the data that will be fed to the SNN, the processed input is shown in figure 4.3 at different scales. As mentioned before, a set of utility functions for visualization was implemented based on the Bokeh Python library, supporting raster plots, line plots, box plots, and

Spike Time (ms)	Channel Index
15.1	1
17.9	0
22.3	2

TABLE 4.2 – Structure of the processed file containing the ordered spike events

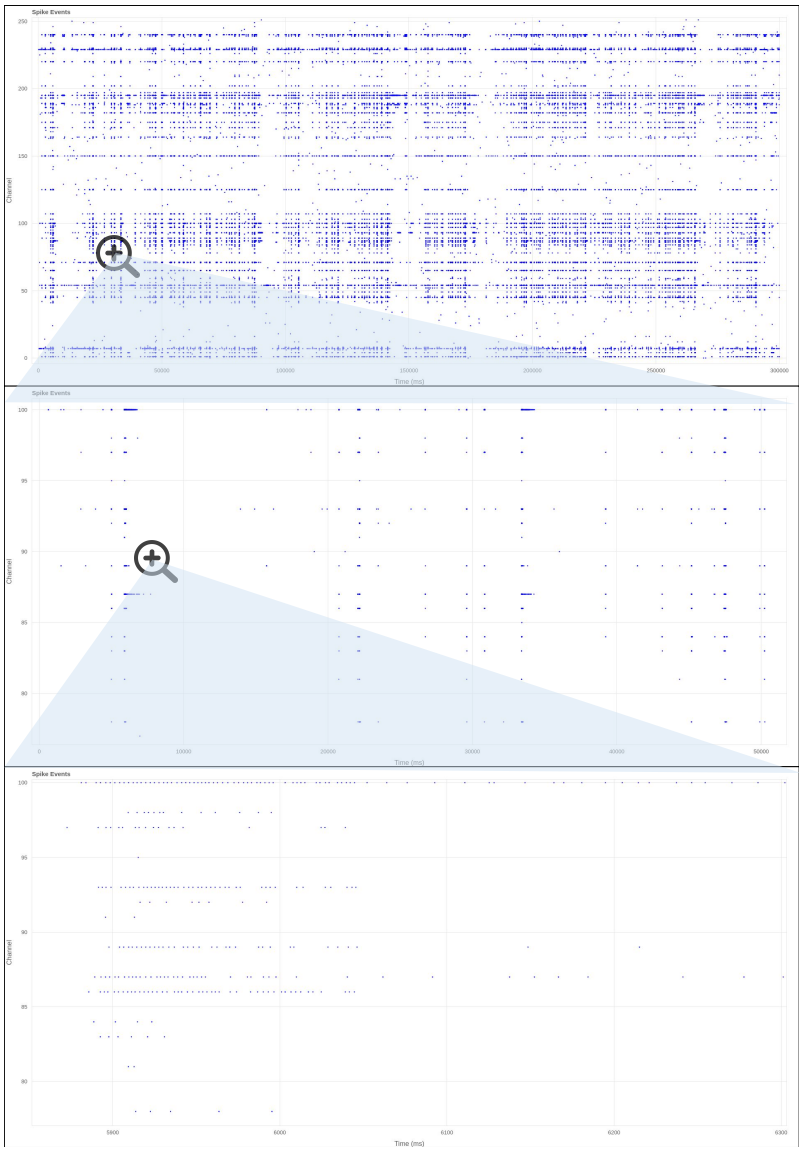


FIGURE 4.3 – Raster plot of the spiking activity of a neuronal population at increasing levels of resolution.

histograms. These interactive graphs support panning, zooming, and other tools that promote a refined visualization experience. In this case, the output chart permitted analyzing the input and understanding the population dynamics at different time scales.

4.4 Proposed Solution

Since the definition of activity bursts is not agreed upon in the literature, and to show the flexibility of these networks, several burst detectors are presented in this chapter for different definitions of channel and network bursts. The following sub-sections introduce each interpretation and the respective solution.

As a common step for all detectors in this chapter, the ideal SNN parameters of the network were found through a heuristic search over the tunable parameters. Due to the nature of the problem and the chosen LIF neurons (CUBA LIF), which are mathematically analyzable, it is possible to find the ideal network parameters using the algorithm mentioned below:

4.4.1 Parameter Search

The LAVA LIF neurons update their inner state according to the system of equations 2.13 and 2.14. The dynamics of the current and membrane potential of the modeled neurons over time can be seen as first-order linear difference equations, which can be recursively solved and simplified into the following expressions:

$$\begin{cases} u(t) = u(0)(1-d_u)^t + \sum_{k=0}^{t} a_{in}(k)(1-d_u)^{t-k} \\ v(t) = v(0)(1-d_v)^t + \sum_{k=0}^{t} u(k)(1-d_v)^{t-k} \end{cases} \tag{4.1}$$

In this project, Activity Bursts are defined through the following template:

1. A Channel Burst corresponds to any sequence of N spikes that occurs within T_c ms on a single channel.

2. A Network Burst corresponds to any neuronal activity where C channels burst within T_n ms.

Where a channel burst referred to in 2) depends on the definition in 1).

Hence, to detect these characteristic features given a real-time feed of spiking events, the underlying neuron dynamics must be fine-tuned to fire when, and only when, the above-mentioned conditions are met. In practice, this means that the neurons must gradually increase the voltage and, upon receiving a defined number of spikes (n_{spikes}) within a set time window, fire an action potential. Reversely, if the stimulus arrives at a lower frequency than the designated limit, the natural decay of the inner variables shall prevent spiking.

To find the parameter combination that better captures channel or network bursts, it is necessary to calculate the values of $u(t)$ and $v(t)$ over time, given a specific activation pattern, to ensure that the membrane potential reaches the threshold voltage if enough spikes are fed in the window of time. Although it is possible to describe the sum of $u(t)$ or $v(t)$ mathematically until a given time step, we opted for a numerical modeling approach given the complexity of the former solution. So, using a Python simulation of the LIF neuron state, given the values for d_u, d_v, and a_{in}, the utility method returns the final current and voltage after a specified number of time steps. The implementation can be found in figure C.1.

The explorable parameters in this procedure are the current decay, the voltage decay, and the weights of the dense layer that define the intensity of a single spike in the state of the neurons. Therefore, the best configuration is found by testing several combinations of these parameters around the upper limit of detection. The upper limit of detection corresponds to the lowest spike frequency that still generates an activity burst, or in other words, considering the case of Channel

Burst detection, the occurrence of N spikes, where the first and last spikes are exactly T_c ms apart. For simplicity's sake, let us refer to the time the last stimuli is fed as $t_{stimuli}$. Note that the instant when the voltage reaches its peak value is not necessarily $t_{stimuli}$. In fact, the most likely behavior is that the processing unit reaches the maximum voltage some steps afterward, as the voltage depends not only on the previous value but also on the current. This behavior is seen in figure 3.3, where the voltage reaches its peak at t=9 ms (t_{max}), even though the input is given at t=5 ms.

The heuristic used to find the ideal parameters required predicting the membrane potential in three scenarios: 1) applying evenly-spaced stimulation with the minimum frequency for bursting (v_{limit}), 2) applying the same input, but delaying the last stimuli by one time-step ($v_{delayed}$), and 3) feeding consecutive $n_{spikes} - 1$ action potentials spaced by the minimum time step that yielded results, i.e., where the three conditions are met ($v_{consecutive}$). For each scenario, the voltage was measured one step before the last stimulation $v(t_{stimuli} - 1)$, at the stimulation step $v(t_{stimuli})$, and at the time it reached its peak value $v(t_{max})$. The relevant combinations of parameters are the ones where:

$$\begin{cases} v_{limit}(t_{stimuli} - 1) < v_{threshold} \\ v_{limit}(t_{max}) >= v_{threshold} \\ v_{delayed}(t_{max}) < v_{threshold} \\ v_{consecutive}(t_{max}) < v_{threshold} \end{cases} \tag{4.2}$$

In principle, all the parameter combinations that agree with these conditions are good candidates for the SNN detector. If a tiebreaker is necessary, the blend of values including the lowest $v_{limit}(t_{max})$ is chosen by default.

We could insert learning capabilities into the network as an alternative to the parameter search, but it brings some shortcomings. For example, it adds unnecessary complexity to the model, making it less understandable and prone to overfitting. Additionally, only the synaptic weights could be updated through learning since the neuron time constants are not adaptable.

4.4.2 Channel Burst Detection

The first implemented type of activity burst detector was a single-channel burst detector. Here, a Channel Burst is defined as *Any sequence of 4 or more spikes that occur within 20 ms in a single channel*.

Having the definition set, the first step is to find the ground truth using the method introduced in sub-section 4.2.1, which will be used to evaluate the model's performance. The next stage is the definition of the SNN architecture. As there is no right or wrong architecture, especially in temporal networks, the adopted strategy was to experiment with different layer compositions to gain an intuition of what would be a good structure. Starting with channel burst detection proved to be the right choice as it is a straightforward biomarker compared to the other solutions presented in this paper.

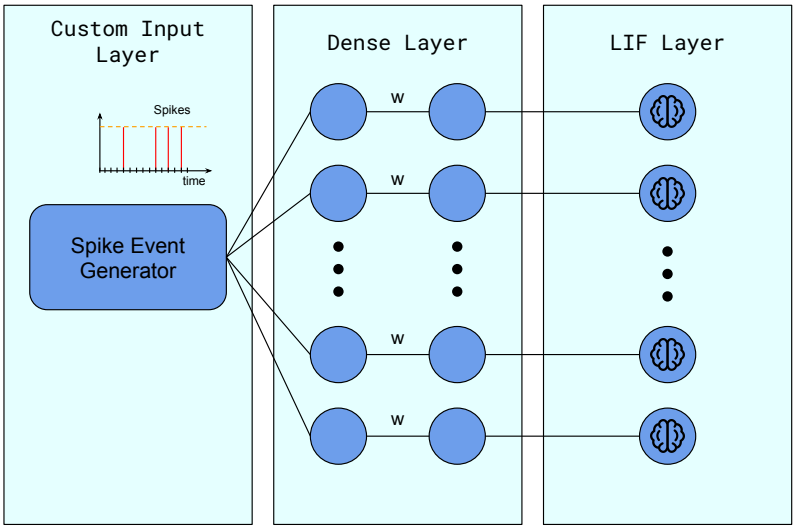


FIGURE 4.4 – SNN Architecture for Channel Burst Detection.

4.4.2.1 SNN Architecture

Figure 4.4 presents the chosen architecture, comprised of three layers:

1. **Custom Input Layer**: Lava Process that sends, for every time step, the input spikes to the remaining network. The spiking dynamics of this layer reflect the dynamics of the in-vitro neuronal population it originates from. Thus, the number of edges, or artificial synapses, branching out of the *Spike Event Generator* to the Dense layer is equal to the number of recorded channels in the experiment.

2. **Dense layer**: There is a one-to-one mapping between recorded channels ($num_{channels}$) and the input nodes of the Dense layer, meaning that the previous layer acts as a replacement for the real-time arrival of events as explained in section 3.3. Meanwhile, this layer controls the intensity of individual spikes before handing them to the in-silico neurons of the LIF layer. Notice that the weight of each connection, w, is the same for all the connected nodes. The non-represented weights are equal to 0.

3. **LIF Layer**: Final layer composed of $num_{channels}$ refractory leaky integrate-and-fire neurons. Each node is responsible for detecting the bursting activity of the channel it is connected to. The optimal d_u and d_v values found through the parameter search dictate the synaptic and membrane potential time constants of these modeled neurons. Any spike of this final layer can be interpreted as detecting a channel burst.

4.4.2.2 Adding Refractory Capabilities

Until now, we have omitted a key feature of burst detection, the inter-burst interval (IBI). It is defined as the minimum time spacing between the end of one burst and the start of another, representing the duration of the silent phase between consecutive bursts.

So that the presented SNN can take the IBI into account, the LIF neurons were adapted into Refractory Leaky Integrate-and-Fire neurons, meaning that the voltage of the nodes sticks to the baseline voltage after an action potential for a selected number of time steps, known as the refractory period, thus avoiding consecutive spikes at a distance inferior to the IBI.

Lava already had a refractory version of the LIF Process, which facilitated the switch from the original LIF. However, after examining the neuron dynamics through interactive plots, it was found that the subthreshold behavior of the neurons, i.e., the state dynamics when the voltage is below the voltage threshold, was not implemented correctly. Thankfully, Lava encourages open-source contributions following the developer guidelines indicated in the documentation. As such, the first contribution made as an outcome of the dissertation was fixing the floating-point version of the refractory LIF model. The Lava maintainers welcomed this change, which is now included in the library's main branch.

4.4.2.3 Network Parameters

The process to obtain most parameters was previously explained. For the Dense layer weights and the LIF time constants, the Parameter Search procedure 4.4.1 returns those values ($du = 0.45$, $dv = 0.05$, and $weights = 0.2$). The baseline and threshold voltage are fixed for all SNN models, set to 0 and 1, respectively. Last, the refractory period is also fixed according to the chosen definition of activity burst. Ergo, in this case, the refractory period is 20 ms since the maximum duration from the first to last spike of a channel burst is 20 ms.

Remember that these values correspond to the parameters for the floating-point precision model that runs on the Loihi simulation. In order to run the fixed-precision models, the parameters must be converted from floats to integers, as demonstrated in sub-section 4.4.4.

4.4.3 Network Burst Detection

Moving onto a more complicated electrophysiological pattern, the succeeding algorithm attempts to detect network bursts (**NB Detector**). We characterize a Network Burst as "Any Sequence of 10 or more Channel Bursts from different channels that occur within 20 ms". As for the Channel Burst, we kept the definition from the Channel Burst Detector (**CB Detector**) 4.4.2.

In both channel and network burst definitions, the number of spikes and the time window they must belong to are chosen analytically by looking at raster plots of the input data similar to the one presented in figure 4.3. Again, due to the lack of a universal ground truth, the activity burst definitions are derived from the features of the input data spike trains via visual identification. While not ideal, this solution was enough to accomplish the aim of this study, which is to validate and benchmark neuromorphic algorithms to analyze neuronal activity patterns.

Akin to Channel Burst detection, the first step involves calculating the ground truth using the method introduced in sub-section 4.2.1. The ground truth contains the exact timestamps when the network burst conditions are met and will be used to assess the model's performance. Concerning the SNN architecture, the problem can be split into two stages:

1. Detect the Channel Bursts for each channel.
2. Given the Channel Burst timestamps for each channel, detect the Network Bursts.

4.4.3.1 SNN Architecture

The first phase of the algorithm is already described in the previous sub-section. Thus, we can take advantage of the architecture 4.4.2.1 to model the initial layers of this SNN. The second phase can be interpreted as a Coincidence Detector since each Channel Burst is represented by a spike by the end of the first stage (Middle LIF Layer) in figure 4.5. As the name suggests, coincidence detection refers to a system that detects the occurrence of temporally close but spatially distributed input signals [17]. Likewise, the detection of Network Bursts using the timestamps of Channel Bursts from separate channels follows a similar principle. Therefore, the final two layers in figure 4.5 implement the second stage, where the output is generated by a single neuron, indicating if there is a network burst or not in real-time.

Figure 4.5 presents the chosen architecture, comprised of five layers:

1. **Custom Input Layer**: Lava Process that sends, for every time step, the input spikes to the remaining network. The spiking dynamics of this layer reflect the dynamics of the in-vitro neuronal population it originates from. Thus, the number of edges, or artificial synapses, branching out of the *Spike Event Generator* to the Dense layer is equal to the number of recorded channels in the experiment.
2. **Dense layer**: There is a one-to-one mapping between recorded channels ($num_{channels}$) and the input nodes of the Dense layer, meaning that the previous layer acts as a replacement for the real-time arrival of events as explained in section 3.3. Meanwhile, this layer controls the intensity of individual spikes before handing them to the in-silico neurons of the LIF layer. Notice that the weight of each connection, $w1$, is the same for all the connected nodes. The non-represented weights are equal to 0.
3. **Middle LIF Layer**: First LIF layer composed of $num_{channels}$ refractory leaky integrate-and-fire neurons. Each node is responsible for detecting the bursting activity of the channel it is connected to. The optimal d_u and d_v values found through the parameter search dictate the synaptic and membrane potential time constants of these modeled neurons. Any spike of this layer can be interpreted as detecting a channel burst.
4. **Middle Dense Layer**: This layer aggregates the activations from $num_{channels}$ input nodes into a single output node by adding them together after multiplying by a weight $w2$. This

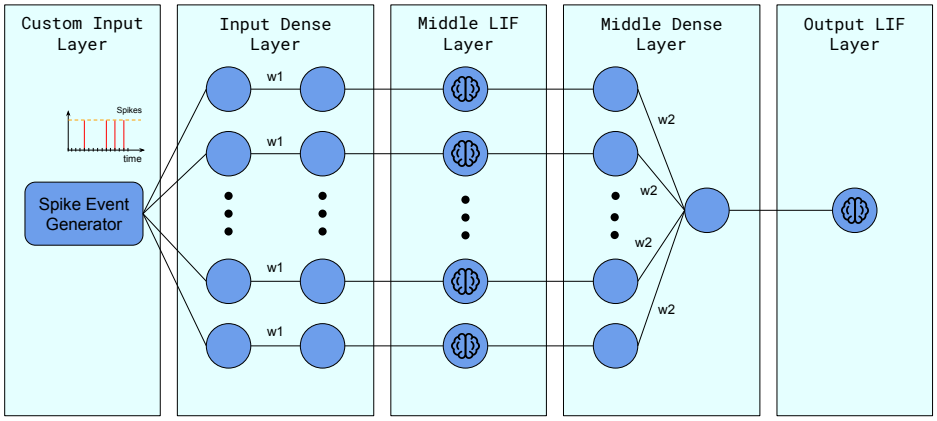


FIGURE 4.5 – SNN Architecture for Network Burst Detection.

process, also known as weighted sum, is typical in ANNs and can be expressed mathematically: $\sum_{i=1}^{n}(w2_i \cdot a_i) + b$. In this case, the bias (b) is equal to 0.

5. **Output LIF Layer:**: Last LIF layer composed of a single refractory leaky integrate-and-fire neuron. The output of this node indicates the presence or absence of a Network Burst. An extra parameter search is conducted to discover this neuron's finest du and dv values. The internal dynamics of the Middle LIF neurons are independent of the dynamics of the Output LIF neuron.

Following the footsteps of the Channel Burst detection, all in-silico neurons have refractory capabilities with a configurable refractory period for each LIF layer.

4.4.3.2 Network Parameters

In addition to the parameters involved in the first stage, the Network Burst detection includes a new set of synaptic and membrane potential time constants for the Output LIF neuron and w_2, the weight shared by all nodes in the Middle Dense layer. These three tunable values were calculated by applying another parameter search following the procedure in sub-section 4.4.1 ($du_2 = 0.9$, $dv_2 = 0.05$, and $w2 = 0.15$). At last, the refractory period of the output neuron is fixed and derived from the definition of the network burst. Hence, the neuron stays in the refractory state for 20 ms after a spike because the maximum duration between the first and last channel burst is 20 ms.

Once again, these values correspond to the parameters for the floating-point precision model that runs on the Loihi simulation. In order to run the fixed-precision models, the parameters must be converted from floats to integers, as demonstrated next.

4.4.4 Fixed-Point Precision Version

The importance of implementing a bit-accurate version of Lava processes is explained in subsection 2.4.4. From a development perspective, the floating-point precision and bit-accurate versions derive from a common Lava Process. They are distinguished, however, by attributing a unique tag to each Lava Model that implements the respective Process. The conventional tag used for the algorithm using floats is "floating_pt", while the tags "bit_accurate_loihi" and "fixed_pt" are attributed to the procedure relying solely on integers.

To run code in Lava, the desired running configuration must be selected. For instance, Loihi2 simulator or Loihi2 hardware. Attached to the running configuration, a tag can be selected to specify the implementation of the Lava Process (Lava Model) you wish to run.

4.4.4.1 Bit-Fixed Models

All Lava Processes have to support a bit-fixed version to run on Loihi's neuromorphic hardware. Hence, looking at the Processes used during this practical experiment, we require a fixed-precision version of the Dense and Refractory LIF. The Spike Event Generator does not need a fixed-precision version because it will run in the neuromorphic chip's embedded processors that run Python code and communicate in real-time with the neuromorphic cores. Looking at the Dense Process first, the integer-only version of this layer is included in the framework, which saves us development time.

On the other hand, the leaky integrate-and-fire model with refractory capabilities was not implemented following the bit-accurate structure. Because of this, we had to develop this version, which was named "PyLifRefractoryModelBitAcc". Due to Intel's efforts to keep the set of tools that allow running code in Loihi private and restricted to members of the INRC, the documentation is limited, increasing the challenge of extending the libraries' capabilities. Anyhow, by looking at other examples, we were able to develop the model and contribute once again to extend the capabilities of the Lava framework. The Results section compares the output of both versions, validating our design.

4.4.4.2 Parameter Conversion

All that is left to do to transition to fixed-point precision is to find a proper mapping between float parameters and integers. Because our spiking layers do not rely on a voltage bias and the baseline voltage is 0, fewer parameters have to be altered. Effectively, only four parameters shall change: the synaptic time constant (du), the membrane potential time constant (dv), the threshold voltage (v_{th}), and the Dense layer's weights (w).

Starting with the time constants, du and dv, they are stored in 16-bit variables ($uint16$) and contain 12 bits of precision, meaning that their values range from 0 to $2^{12} - 1$. In the floating-point model, the time constants range from 0 to 1. Hence, they must be scaled as follows:

$$\begin{cases} du_{integer} = du_{float} \cdot 4095 \\ dv_{integer} = dv_{float} \cdot 4095 \end{cases} \tag{4.3}$$

The threshold voltage and the weights have to be scaled by the same amount. The bit-accurate Dense Model defines a precision of 8 bits for each weight. Following the same principle, the maximum scalar we can use is $2^8 - 1$. However, the only fixed-precision example from Intel Labs opted to multiply by 240 instead, which is smaller than 255, so we chose this value for consistency sake:

$$\begin{cases} v_{th-integer} = v_{th-float} \cdot 240 \\ w_{integer} = w_{float} \cdot 240 \end{cases} \tag{4.4}$$

4.5 Results

This section exhibits the results of the *in vitro* practical experiment. First, the outcomes of the Channel Burst Detection are shown, followed by the Network Burst Detection. For each detector, a comparison between floating-point and fixed-point precision is included. The results will be presented according to the approach introduced in the methodology 3.6.

4.5.1 Channel Burst Detector

4.5.1.1 Dynamics of the Modeled Neurons

The 252 output neurons of the detector signal the bursts of the corresponding 252 channels. It is, therefore, interesting to visualize how the voltage and current of these neurons oscillate over time in response to different input patterns, namely spontaneous activity or rapid sequences of spikes.

Figure 4.6 plots the value of these two variables over time in a line chart. Since it would not be readable to include 252 lines, 8 *in silico* neurons were selected to represent the population, each with a different color. The figure displays three rows with an increasing time resolution. The first column corresponds to the membrane potential, and the second to the synaptic current.

At the higher time scale, we can observe the population's behavior at a macro level. For example, neuron 6 (orange line) is the most active but rarely triggers an action potential. This indicates that the underlying channel spikes frequently but not in the bursting range. In contrast, neuron 0 (green line) often receives consecutive activations and is prone to channel bursts. As we zoom into the chart, the subthreshold dynamics of the neurons become apparent. Notice that the current decay is much steeper than the voltage decay due to the underlying time constants ($du = 0.45$ and $dv = 0.05$). The last row demonstrates a neuron's spiking behavior. As it reaches the voltage threshold, the potential shoots back to 0 and remains at the baseline level for as long as the refractory period takes, even though it receives a positive current. In comparison, the red neuron receives a single activation, and the potential decays naturally.

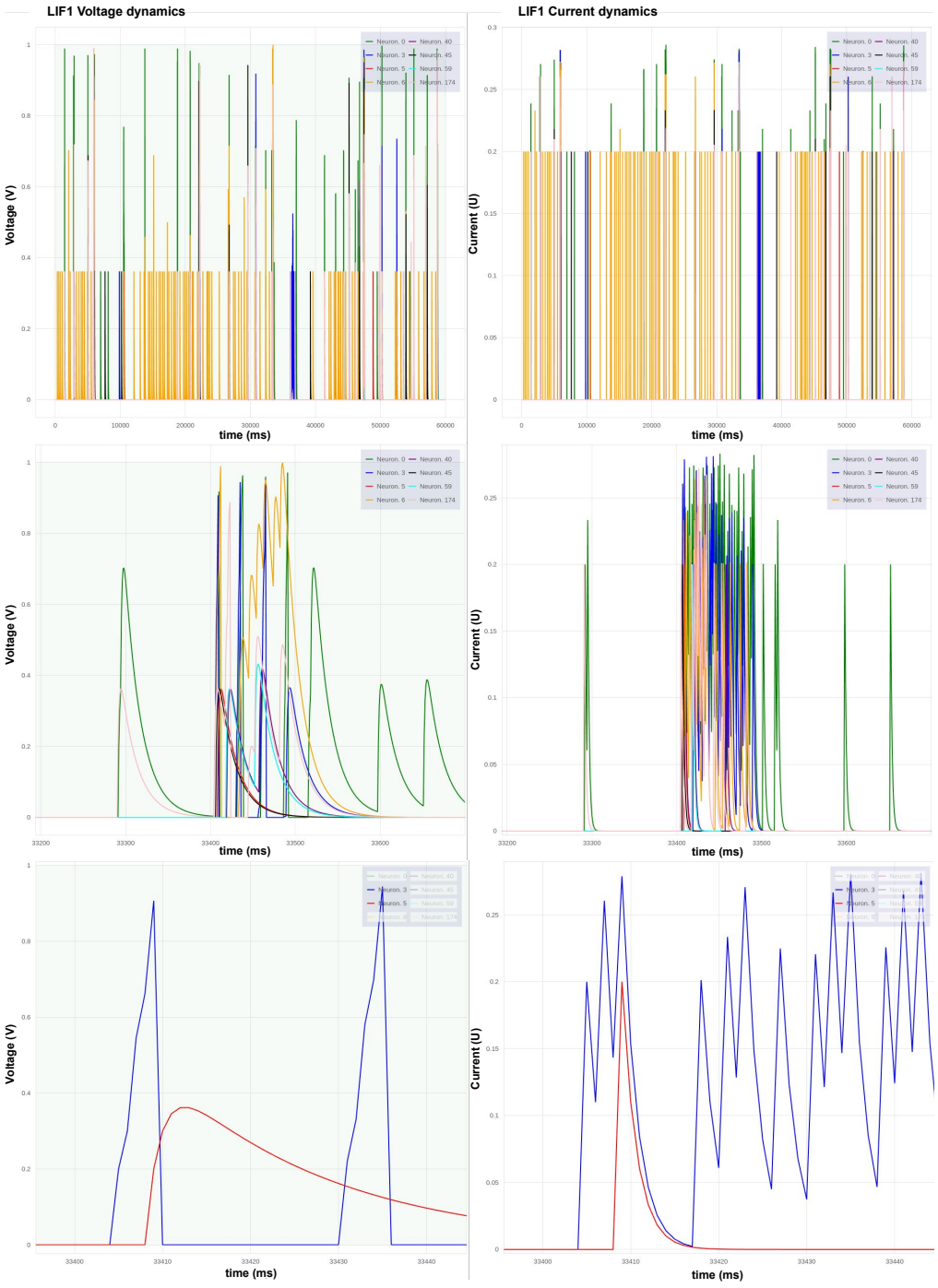


FIGURE 4.6 – Voltage and Current Dynamics of the Channel Burst Detection at different time scales (floating-point precision).

		Actual Values	
		Positive	Negative
Predicted Values	Positive	730	80
	Negative	23	59167

TABLE 4.3 – Confusion Matrix of the Channel Burst Detection (floating-point precision).

4.5.1.2 Classification Metrics

The predictions of the model are summarized in the confusion matrix 4.3. The True Negative (TN) count is very high since the ground truth represents a channel burst in a single time instant, the moment the burst condition is satisfied, and to validate whether a predicted channel burst succeeds an actual channel burst, we defined a causality window (c_{window}). Ergo, considering a real channel burst occurs at time t_{burst}, if the associated neuron spikes inside the interval $[t_{burst}, t_{burst} + c_{window}]$, the prediction is True Positive (TP). Otherwise, if the SNN spikes outside the causality window, the event corresponds to a False Positive (FP). Following the same logic, if the respective SNN neuron does not spike when an actual channel burst occurs, the event is a False Negative (FN). So, the TN statistic and associated metrics are not adequate for measuring the model's performance.

The precision, recall, and f1-score are the metrics that reflect the model's performance more rigorously since they do not consider the TNs. Table 4.4 contains these metrics and substantiates the point referenced above as the accuracy and specificity are over-estimated and do not exhibit the predictive capabilities of the SNN. Even then, using a small spiking neural network and leveraging individual neurons' capabilities, we could detect most channel bursts. We can see that approximately 90% of the predicted bursts correspond to actual channel bursts, and about 97% of the total channel bursts were detected by the network. In the discussion section, we interpret these results and indicate areas for improvement.

4.5.1.3 Fixed-Point Precision Results

The performance of the CB Detector with bit-accurate architecture is very similar to the floating-point version, showing that the variables' resolution under the optimized version is sufficient to capture the patterns of interest of this problem. Interestingly, as shown in tables 4.4 and 4.5, the model predicts channel bursts slightly better, resulting in a higher precision, recall, and f1-score.

	Accuracy	Precision	Recall	F1-Score	Specificity
Channel Burst Detector (float)	99.83%	90.12%	96.95%	93.41%	99.87%
Channel Burst Detector (bit-fixed)	99.83%	90.25%	97.08%	93.54%	99.87%
Channel Burst Detector (non-refractory)	99.82%	87.81%	99.47%	93.28%	99.82%
Network Burst Detector (float)	99.99%	77.78%	100%	87.50%	99.99%
Network Burst Detector (bit-fixed)	99.99%	77.78%	100%	87.50%	99.99%
Network Burst Detector (non-refractory)	99.91%	22.76%	100%	37.09%	99.90%

TABLE 4.4 – Classification Metrics of the Detectors of the *In Vitro* Study.

		Actual Values	
		Positive	Negative
Predicted Values	Positive	731	79
	Negative	22	59168

TABLE 4.5 – Confusion Matrix of the Channel Burst Detection (fixed-point precision).

Comparing this version's voltage and current dynamics (figure A.1) with the high-precision one, they are practically indistinguishable, with no visible discretization. The only perceptible difference is the change in scale, as the neuron's membrane potential and synaptic current reach the order of thousands or tens of thousands in the integer-only version.

4.5.1.4 Profiling Analysis

We ran the bit-accurate solution on Intel's neuromorphic chip. On top of proving that the developed programs could transition from the simulator to the neuromorphic processor, we present a performance report of the event-driven algorithms running on the specialized hardware. Adequately presented results should be easy to understand and allow comparisons between solutions. For this reason, the following reporting structure recommended by Lava is included:

"The CB Detector model occupies **1** neuromorphic core. A single time step takes **30.2** μs on the neuromorphic cores and consumes **13.17** μJ of energy ($6.41 \cdot 10^{-2}$ μJ of which is dynamic energy), resulting in an energy-delay product of $3.97 \cdot 10^{-4}$ μJs. All measurements were obtained using Lava on Loihi version v.0.6.0 on **Oheogulch** board **ncl-ext-og-02**. Tables 4.6 and 4.7 show a breakdown of energy consumption and further relevant metrics."

To clarify the benchmarked features, the static and dynamic measures of power and energy represent the consumption when the system is not running workload and the additional contribution from the workload, respectively. Concerning the Energy per Inference, it is equal to $\frac{Power}{Throughput}$, where Throughput = $\frac{NumberSteps}{ExecutionTime}$. The Execution Time corresponds to the total running time of the program and depends on the specified number of steps. Latency, instead, reflects the time between sending input and receiving a corresponding output, providing a metric of time independent of the number of iterations. Finally, the Energy Delay Product is calculated by multiplying the Energy per Inference by the Latency, and the number of reserved neuromorphic cores is given by the Profiler class from Lava.

In addition to the presented metrics, the hardware sensors connected to the Loihi 2 setup allow visualizing the execution time per time step, the distribution of neuromorphic operations

Algorithm	Power (W)			Energy (J)			Energy per Inference (μJ)
	Static	Dynamic	Total	Static	Dynamic	Total	
CB Detector (bit-fixed)	0.4340	0.0022	0.4362	1.5723	7.687e-3	1.58	13.1367
NB Detector (bit-fixed)	0.4539	0.0056	0.4595	2.8273	3.43e-2	2.8616	15.897

TABLE 4.6 – Performance Benchmarking Comparison of different Activity Burst Algorithms - Part 1

Algorithm	Execution Time (s)	Number of Steps	Latency (μs)	Throughput (steps / s)	Nº Neurocores
CB Detector (bit-fixed)	3.6224	120000	30.2	33127	1
NB Detector (bit-fixed)	6.228	180000	34.6	28902	2

TABLE 4.7 – Performance Benchmarking Comparison of different Activity Burst Algorithms - Part 2

(synaptic operations, neuron updates, output spikes, or input spikes), and per-core SRAM memory utilization. Figure 4.7 compiles the mentioned features for the CB Detector.

4.5.2 Network Burst Detector

4.5.2.1 Dynamics of the Modeled Neurons

In the case of the NB Detector, there are two layers of neurons with different pattern recognition responsibilities. Consequently, figure 4.8 contains four charts corresponding to the voltage and current state along time for the two LIF layers.

As we have already presented the dynamics of the CB Detector (LIF1), we will focus on the Output LIF behavior (LIF2) and analyze it in greater detail. The final layer is composed of a single leaky integrate-and-fire neuron that spikes whenever the SNN predicts a network burst. The dependability of LIF2's input on LIF1 is observable through image a) since the synaptic current of LIF2 increases after an action potential is triggered on LIF1.

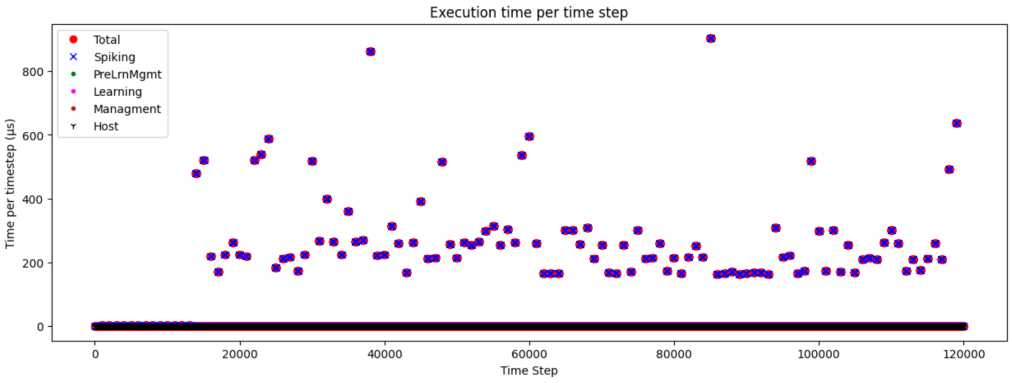
Image b) covers an example where a Network Burst was detected at t=65.668s. After signaling the event, the network remains silent for a fixed interval controlled by the refractory period, after which it recovers the ability to hyperpolarize. The second peak in voltage did not end in a network burst, and the membrane potential slowly decreased toward the baseline.

4.5.2.2 Classification Metrics

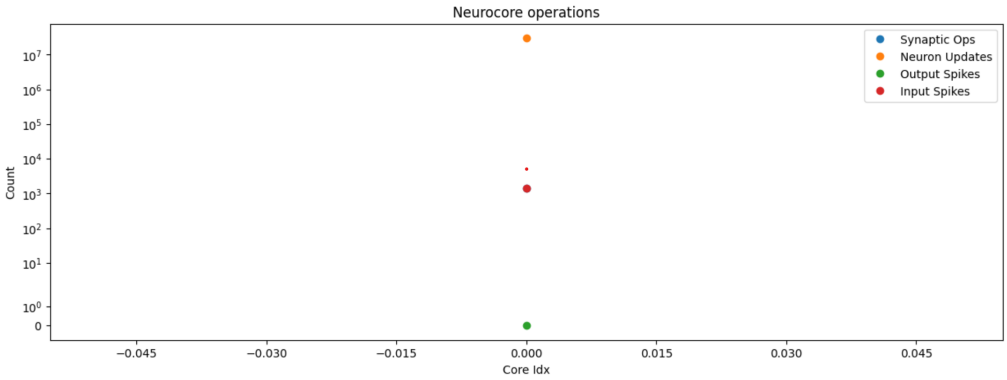
The model's predictions are summarized in the confusion matrix 4.8. For the reasons explained in the CB Detector results, the TN statistic and associated metrics are inadequate for measuring the model's performance. Rather, the precision, recall, and f1-score should be the target of attention. Table 4.4 exhibits these metrics. The results indicate that the developed SNN does a perfect job of detecting the existing network bursts, capturing 100% of the annotated events. But despite that, it also labels events as relevant, which, according to our definition, are not. This means that the spiking network is extra sensitive to bursting activity, thus overestimating the occurrence of population bursting in about 22.22% of the cases. We delve deeper into this phenomenon in the discussion section 4.6.

4.5.2.3 Fixed-Point Precision Results

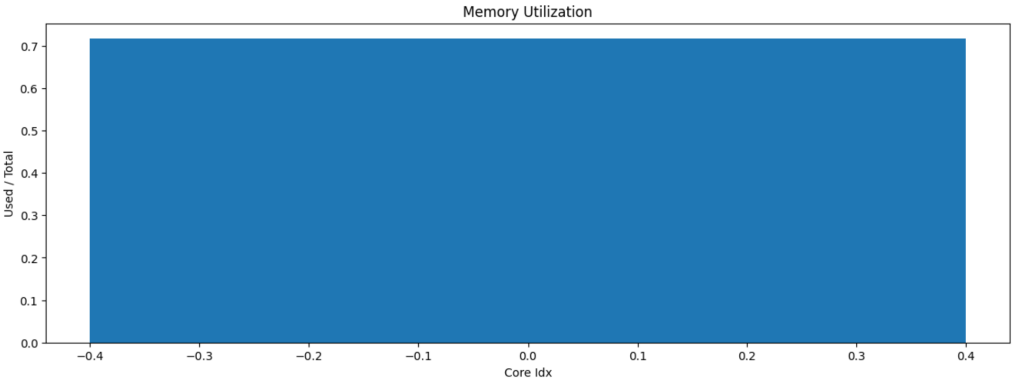
As expected, the bit-accurate version yields equivalent results, ensuring the performance will be maintained when deployed in the neuromorphic processor (tables 4.4 and 4.9). Once again, the



(A) Evolution of the Execution Time per time step of the CB Detector.



(B) Distribution of different operations occurring in the Neuromorphic Mesh of the CB Detector.



(C) Memory Utilization per core of the CB Detector.

FIGURE 4.7 – Profiling Results of the CB Detector.

		Actual Values	
		Positive	Negative
Predicted Values	Positive	28	8
	Negative	0	99964

TABLE 4.8 – Confusion Matrix of the Network Burst Detection (floating-point precision).

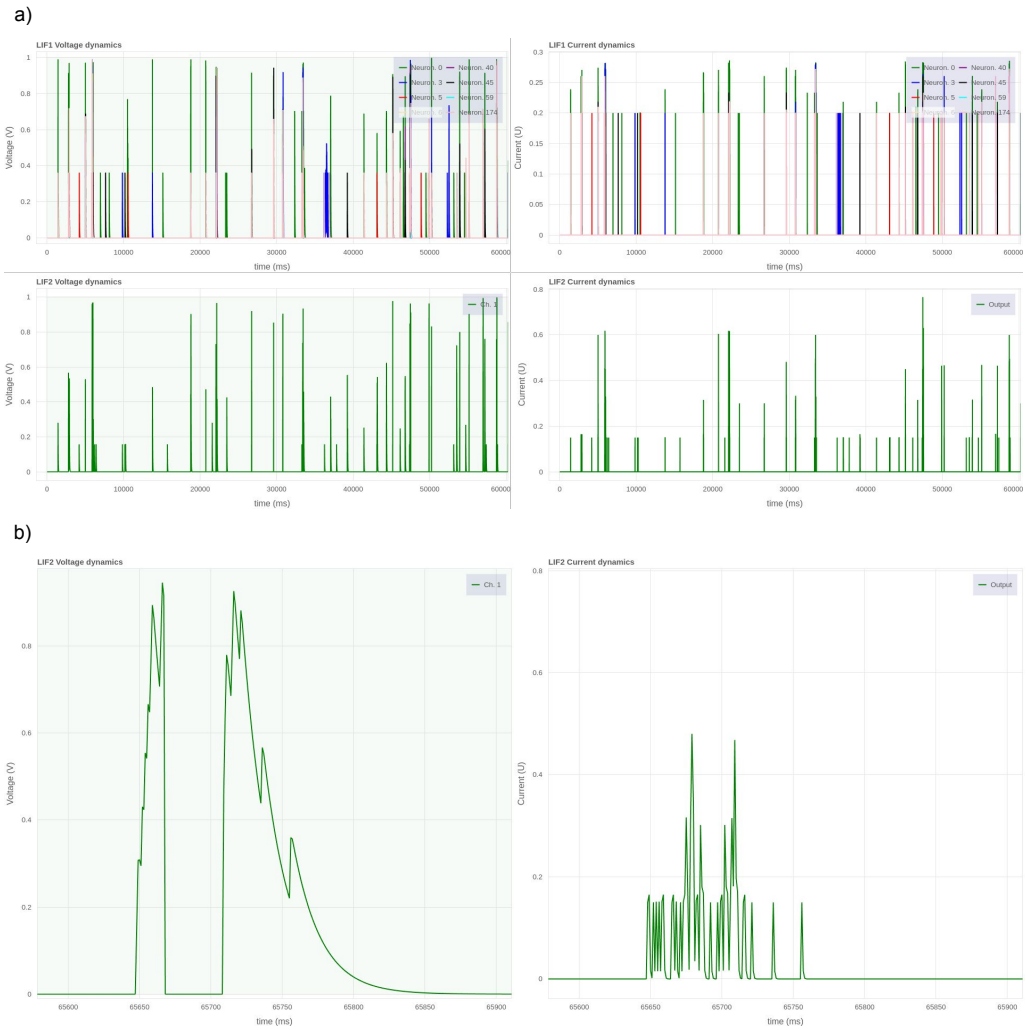


FIGURE 4.8 – Voltage and Current Dynamics of the Network Burst Detection at different time scales (floating-point precision).
a) Dynamics of the Middle and Output LIF layers in a big time window.
b) Detailed look into the membrane potential and current of the Output LIF neuron.

		Actual Values	
		Positive	Negative
Predicted Values	Positive	28	8
	Negative	0	99964

TABLE 4.9 – Confusion Matrix of the Network Burst Detection (fixed-point precision).

neuron's state fluctuations were compared with the high-precision version, as portrayed in figure A.2. Looking closely at Figure b) in both examples, you can notice some slight changes to the dynamics of the output neuron, the product of the lost resolution. Nevertheless, the disparities are negligible according to the results.

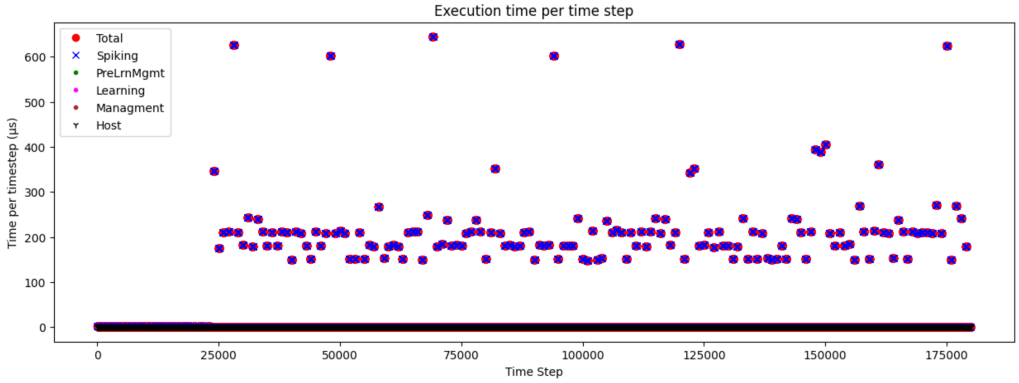
4.5.2.4 Profiling Analysis

The results of appraising the NB Detector on Loihi are presented following the previously defined reporting structure. Starting with the performance summary:

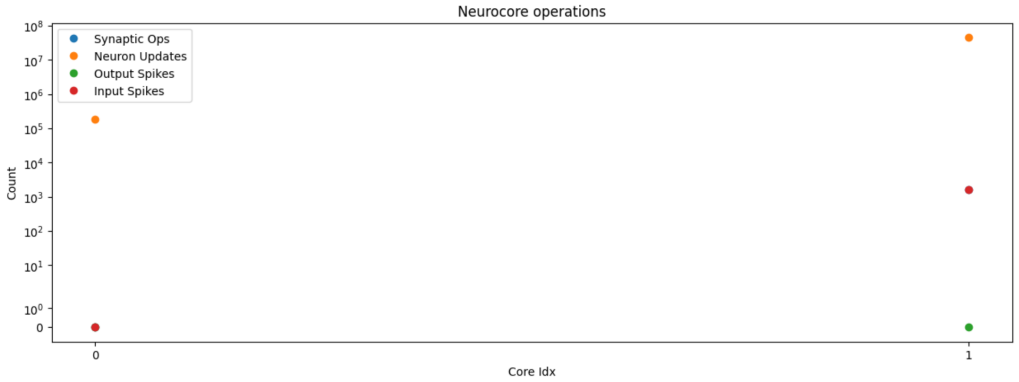
"The NB Detector model occupies **2** neuromorphic cores. A single time step takes **34.6** μs on the neuromorphic cores and consumes **15.897** μJ of energy (**0.285** μJ of which is dynamic energy), resulting in an energy-delay product of $5.51 \cdot 10^{-4}$ μJs. All measurements were obtained using Lava on Loihi version v.0.6.0 on **Oheogulch** board **ncl-ext-og-02**. Tables 4.6 and 4.7 show a breakdown of energy consumption and further relevant metrics."

The CB Detector profiling portion 4.5.1.4 contains an explanation of the benchmarked criteria. In addition, consult figure 4.9 for further details of execution time, allocation of neuromorphic-related operations, and memory utilization. To evaluate the performance of this model, we ran it five times for 180000 steps, corresponding to 3 minutes of input data (each step is mapped to 1 ms of elapsed time). In turn, the total execution time was 6.23 seconds, which is almost two orders of magnitude lower than the processed real-time (180 seconds). Hence, it is safe to assume the latency of the system is low enough to support real-time operation.

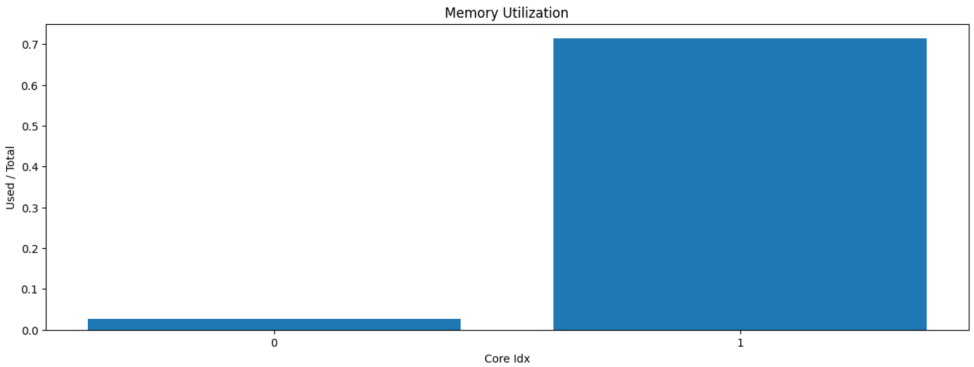
The communication between the embedded CPUs and the neuromorphic cores is the biggest running time bottleneck in Loihi. Because we are feeding a spiking input derived from a file, we need a Python Process that reads this file and feeds the data at every time step, making this bottleneck unavoidable. In a future scenario, however, we plan to connect the input source directly to the neuromorphic chip, thus removing the delay added by the Python Process. To better represent this scenario, the microprocessors and the neuromorphic ASICs communicate after a set number of time steps (1000) rather than communicating at every instant. This change was only applied in the profiling step, so even though we are capping the communication between the Spike Generator and the remaining SNN to once every 1000 steps, which in turn affects the behavior of the model, this is not an issue because it still returns representative results of the model's performance.



(A) Evolution of the Execution Time per time step of the NB Detector.



(B) Distribution of different operations occurring in the Neuromorphic Mesh of the NB Detector.



(C) Memory Utilization per core of the NB Detector.

FIGURE 4.9 – Profiling Results of the NB Detector.

4.6 Discussion

Looking at the results, the application of neuromorphic technologies to characterize the behavior of *in vitro* neural populations in real-time seems to be a promising prospect. In this study, we focused on catching activity bursts to test the feasibility and the advantages brought by an event-driven implementation. The main objective was not to achieve a certain model performance, especially since the nature of this problem makes it difficult to compare to existing solutions, as the definition of channel and network bursts is volatile. Instead, we focused on highlighting the gains of having a neuromorphic design while guaranteeing interesting detection capabilities.

4.6.1 Predictive Capabilities

Anyhow, looking at the model's performance first (table 4.4), in both channel and network burst detection, we were able to build a design that captures the majority of windows of interest, offering a valuable tool to capture bursts in real-time. Overall, the CB detector was the best performant with an f1-score of 93.54%, which was expected due to being a more straightforward recognition pattern. An alternative CB detector without refractory capabilities was also tested to understand the influence of forcing a quiescent period after spiking. Interestingly, the non-refractory detector returns a similar f1-score, but looking at the precision and recall, we can conclude that this version captures more channel bursts at the expense of firing more frequently in response to spontaneous activity. Therefore, it is not clear which model is better. Depending on the problem, having a higher TP count might justify increasing the FP count. Contrastingly, when we remove the refractory period from the NB Detector, the SNN's performance drops drastically (precision=22.76%) due to a surge in false positives, so the implemented refractory model was a critical part of the success of the NB Detector.

According to our definition, the NB Detector can detect all the existing network bursts. However, the classifier's precision is significantly lower than the CB classifier, around 77.78%. This implies that the SNN is over-sensitive to spikes, causing it to assume the existence of relevant events more frequently. Again, depending on the task's priorities, it might be better to capture all the network bursts even if 22.22% of the detections are FPs. If that is not the case, the SNN should be re-configured to decrease its sensitivity. Before indicating some ways to enhance precision, let us visualize some occasions where the network predicted a network burst when, in reality, there is none to understand what is happening.

Figure 4.10 demonstrates an example of the network response during a False Positive. For each sub-figure, the top images illustrate the dynamics of the Middle LIF neurons, while the bottom pair represents the voltage and current of the Output LIF neuron. Notice that several channels of the middle layer are bursting in a short amount of time, so even though the condition of a network burst: "Any Sequence of 10 or more Channel Bursts from different channels that occur within 20 ms," is not satisfied, there is synchronized neuronal activity in the predicted window, just not enough to consider it a network burst according to the pre-conditions. So, the wrongly detected network bursts correspond to synchronized network activity but with lower intensity than

FIGURE 4.10 – Voltage and Current Dynamics of the Network Burst Detection for a False Positive, Picture a), and for a True Native, Picture b).

the one required, pointing out the network's extra sensitivity. Understanding the origin of the FPs permits optimizing the network in the desired direction. For instance, if we want to generate an SNN focused on minimizing the number of FPs, we could perform the parameter search 4.4.1 with stricter network burst conditions by increasing the number of necessary channels to trigger a network burst or reducing the burst time window. Further, the definition of a network burst also affects the performance. If we adopt a stricter definition, there will be less variability, making it easier to differentiate baseline activity from bursting activity.

Another crucial result is the performance of the bit-fixed detectors compared to the floating-precision ones. The granularity of the integer-based design proved to be more than sufficient to perform this task since the classification metrics are identical in both scenarios. Hence, the highly optimized version that runs on Loihi does not deteriorate the detection capabilities of the system. Meanwhile, the energy efficiency of the neuromorphic solution is significantly better than traditional CPU-based solutions, as shown in the next section.

4.6.2 System Performance

Traditional CPUs and GPUs, the current go-to hardware to build neural networks, consume tens to hundreds of watts during predictive tasks [18]. Even for power-efficient solutions, such as lower-power GPUs and mobile processors, the power consumption ranges from 5-10 watts and 1-6 watts, respectively [41]. In contrast, neuromorphic chips offer an alternative design specialized for real-time operation with noticeable power efficiency gains. The profiling results of both detectors running on Loihi substantiate this claim.

In terms of power consumption, the heaviest model (NB Detector) depletes 0.4595 J/s, where solely 1.2% is caused by dynamic operation (table 4.6). This is a testament to the advantages of an event-driven design since the system only expends extra energy upon arrival of relevant data, in this case, spiking activity. Hence, the total energy operational cost of the model (1.58J) is close to the energy expenditure in the idle state (1.5723J). Comparing it with *von Neumann*-based systems, we see a power reduction of 1 order of magnitude compared to dedicated low-power systems, and the decrease becomes even more apparent in comparison to regular CPUs/GPUs. Note that the sensors tracking power and energy consumption do not return precise results, which is why the results correspond to the average of 5 runs.

Latency is another aspect where the neuromorphic chip stands out. This property measures the time between sending an input and receiving a corresponding output. Loihi takes tens of microseconds since receiving an input and sending a corresponding response (table 4.7), whereas a conventional medical embedded device requires at least hundreds of microseconds [70]. Most existing Activity Burst detectors are designed for offline use and do not support real-time operation, but the analog-to-digital signal conversion bottleneck combined with the CPU processing time would result in a millisecond-range latency at best. Thus, a Loihi system paired with analog signal processing can respond to a real-time feed of data faster than existing non-neuromorphic solutions.

For burst detection, the system received data acquired with a sampling rate of 1000 Hz, which means the processing unit must be able to process 1000 samples per second (1 ms latency). Meanwhile, the actual latency of our system, analyzing the slowest detector, is 29 times lower than required. Therefore, the neuromorphic design provides more than enough time resolution, showing the promise of using this architecture to develop higher complexity detectors with sub-millisecond latency.

These comparisons underscore the transformative potential of neuromorphic computing, offering not only drastic reductions in power consumption but also improvements in real-time processing speed, making it a compelling alternative to traditional computing architectures.

Chapter 5

Practical Study on human's *in vivo* recordings

This chapter further explores event-driven algorithms' capabilities and the efficiency of such programs when compared to traditional approaches. Hence, we will attempt to detect high-frequency oscillations in sEEG recordings of humans following the work conducted by Moreira et al. [3] from the NCN i3s group.

5.1 Problem Formulation

Given a dataset of sEEG human recordings representing the voltage signal amplitude in different brain regions over time, the experiment aims to develop event-driven algorithms based on neuromorphic computing to monitor neural activity and detect high-frequency oscillation events. These events can be either Ripples or Fast Ripples, the difference being the frequency range they actuate on.

Detecting such events is a relevant task since solid evidence suggests that HFO detection in different brain regions makes it possible to outline the epileptogenic zone (EZ) with greater precision, thus improving the surgical results of patients undergoing surgery to cure epilepsy. Moreover, recent studies propose tracking HFOs to monitor disease severity and explore treatment responses via electrical stimulation. These scenarios require a computing system capable of receiving and processing electrophysiology data from the brain in real-time. Thus, this practical study intends to create a neuromorphic HFO Detector and compare it with existing solutions. As an alternative to the deep-learning detector implemented by our research group [3], we intend to create a real-time system that can capture these high-frequency events with a more power-efficient design and validate it using Loihi.

Transitioning from the Activity Burst Detector (chapter 4), we plan to demonstrate the possibility of developing SNNs to capture more complex neural activity patterns. The sEEG data serving as input is incompatible with spiking neural networks, as they only receive spikes, so we demonstrate how it is possible to extract relevant information from a continuous signal and create

a temporal-coded neural network. Hereby, while the previous chapter focused on measuring the performance gains of a neuromorphic design and comparing it with a *von Neumann* system, this chapter will focus on the software side of a neuromorphic solution, building a more complex data pipeline and designing novel neuron models in Lava that fit our problem.

The main objectives of this experiment are the following:

1. Prove it is possible to design SNNs able to detect complex neuronal patterns by detecting high-frequency oscillatory activity.
2. Illustrate the necessary steps to convert a continuously recorded signal to a compatible SNN format, i.e., activation spikes, without losing the inherent information.
3. Explore the programmability of Lava and the Loihi chip, for example, through designing custom Processes or neuron models.
4. Compare the results with existing detectors and ascertain the potential of the technology for this particular use case.

5.2 Datasets

In order to develop an AI HFO detector, we must gather enough annotated sEEG data to dictate the learning patterns of the model. To accomplish this, two types of data were used:

- Clinically Relevant Synthetic Data: This data was derived from a publicly available dataset where the position of each HFO was known [71].
- Clinical Data from Epilepsy Patients: This data was acquired as a byproduct of a collaboration with *Hospital de Santo António* from a restricted pool of epilepsy patients.

5.2.1 Synthetic Dataset

The synthetic dataset was first developed to evaluate the performance of traditional HFO detectors in datasets with data from epilepsy patients. Alas, to make the artificial dataset as close to the real data as possible, the first step of the original authors was to study recordings from drug-resistant epilepsy patients who underwent surgical electrode implantation as part of their treatment. All patients were recorded in the same conditions, and the channels that contained relevant events on a bipolar montage were selected. The events of interest (EoIs) in the original study were spikes, ripples, and fast ripples. From these clinical recordings, the objective was to acquire HFO events and a baseline estimation to later build a combined artificial background signal with biological events.

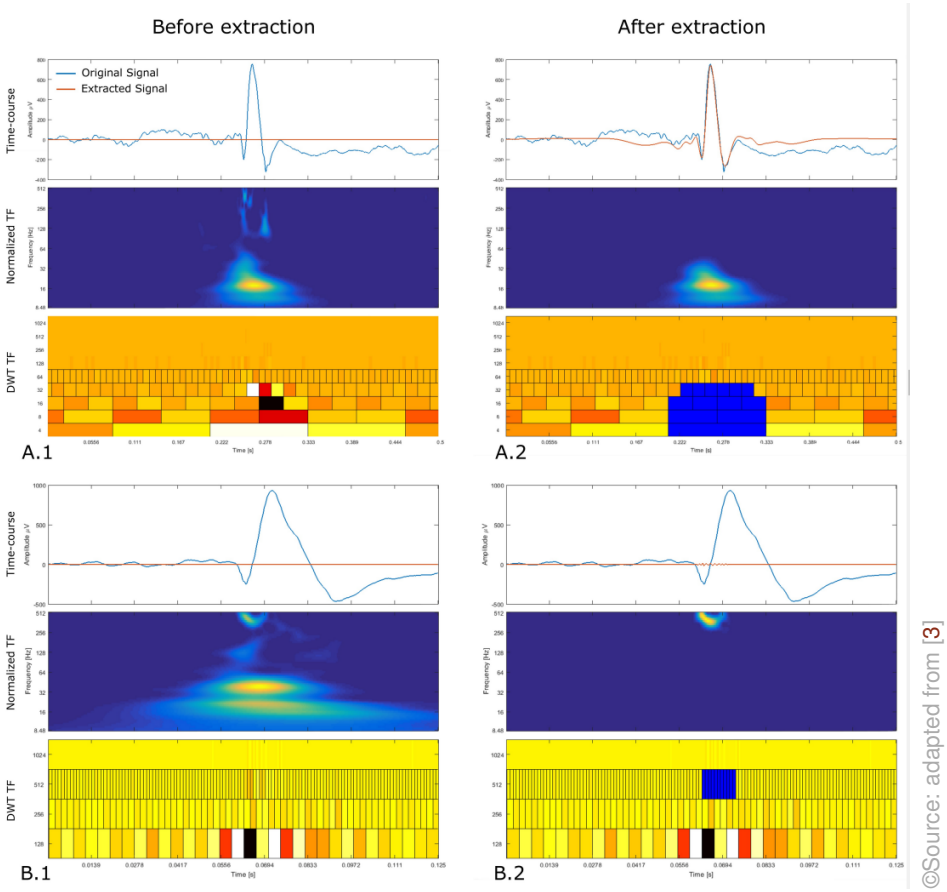


FIGURE 5.1 – Representation of the event extraction GUI. The three top panels (A.1 and A.2) show the extraction of an epileptic spike, while sub-figures B include a fast ripple on a spike. Both sub-figures are represented at the beginning (.1) and at the end (.2), showcasing the original and synthesized signals, as well as the continuous and discrete time-frequency images. On the bottom panel, the user can select which discrete time-frequency coefficients they desire to include, turning the tiles to blue, which affects the extracted signal in the top panel and the continuous time-frequency image in the middle panel. (adapted from [3] and [71])

5.2.1.1 Event Extraction

The event extraction starts with experts manually selecting the channels containing relevant events through visual inspection of a sEEG bipolar signal. Next, the isolated events had to be separated from the signal. This was accomplished by calculating the time-frequency map of detected events using the discrete wavelet transform and then, from its coefficients, selecting the pertinent ones to reconstruct the isolated EoIs. Using these coefficients, the spikes, ripples, or FRs could be modeled without the background activity. The original authors relied on a GUI to perform the extraction, shown in figure 5.1.

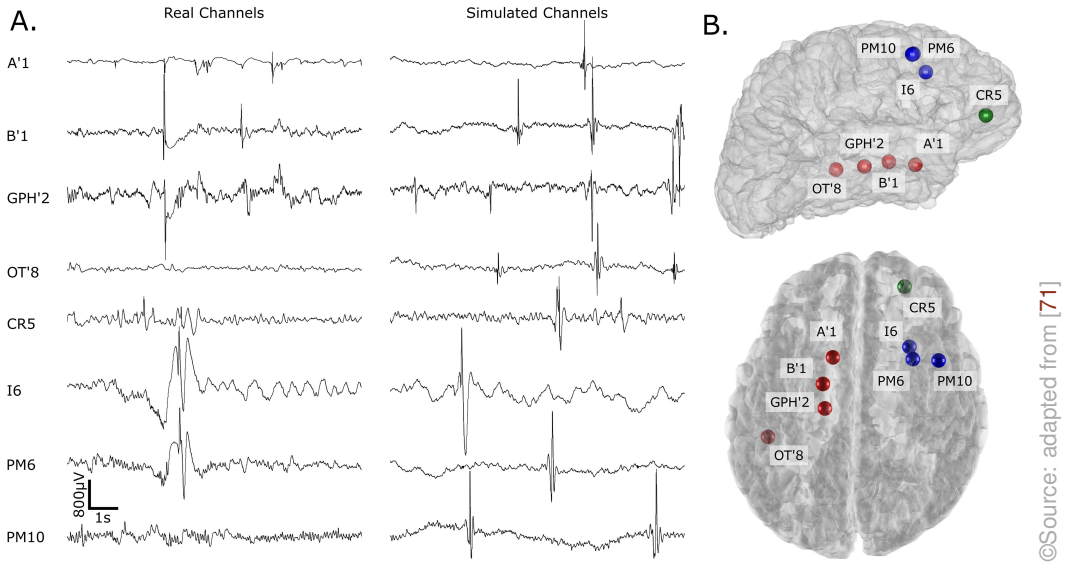


FIGURE 5.2 – Example of sEEG data and their corresponding simulations for different electrode positions in the brain.

5.2.1.2 Background Activity

Similar to the event extraction, the first step to model the baseline activity was to annotate sections of the signal containing only background activity. Reviewers were asked to mark several sections of the recording for each channel to capture a more accurate representation of the background signal throughout the recording. Then, an autoregressive (AR) model was fitted to each annotated section, and the coefficients were averaged over the segments of each channel. At last, a Gaussian white noise distribution was filtered for each channel with the averaged AR coefficients [71].

5.2.1.3 Synthetic Signal

The combined artificial signal proposed by Roehri et al. [71] aims at simulating seven classes of events: spike (S), ripple on spike (S-R), fast ripple on spike (S-FR), ripple and fast ripple on spike (S-R-FR), ripple (R), fast ripple (FR), and co-occurring ripple with fast ripple (R-FR). Hence, random EoIs from the extracted pool were added with a rate of 3 events/minute for each class to the simulated baseline activity of 8 different bipolar channels (channel positioning illustrated in figure 5.2). The events were added with varying signal-to-noise ratios (0, 5, 10, and 15 dB), impacting how much the EoIs stand out from the background activity.

Each channel had 30 realizations of 2 minutes for each SNR, amounting to 180 events of each class for each SNR value. This translates into a total of 5760 ripple events (including all S-R, S-R-FR, R, and R-FR) and 5760 FR events (including all S-FR, S-R-FR, FR, and R-FR) per SNR value. Presenting from another perspective, the synthetic dataset includes 960 channels ($8\ regions \times 30\ realizations \times\ 4\ SNRs$). Since we are not interested in the detection of epileptic spikes, each channel contains 18 relevant events/minute ($3\ events/minute \times 6$ relevant classes),

resulting in a total of 17280 relevant events/minute. The Data Visualization segment 5.3.3 displays snippets of the dataset in the presence of various events.

It is important to note that each realization of a channel is independent. Not only that, but the realizations of a channel for different signal-to-noise ratios are also disconnected, and the same applies to realizations from distinct brain regions. This means that our detector cannot attempt to capture spatial dynamics of the network involving multiple channels because the underlying input data is not synchronized per channel. Therefore, the identification of HFOs must be done on a per-channel basis.

5.2.1.4 Ground Truth

The ground truth of this dataset corresponds to a single point in time for each event, the insertion timing. Following the authors' suggestion, we assume a detection is correct if it falls within a confidence range of the instant of insertion. Based on the average duration of ripples and fast ripples in theory (table 2.3) and in our data, the confidence range to detect fast ripples using our models is 80 ms. Meanwhile, the 180 ms after an event are considered to be inside the HFO's causality window for ripple detection and detection on both bands (80-500Hz).

Additionally, the input data provides the annotations of five traditional HFO detectors: STE [77], SLL [6], HIL [9], MNI [75], and Delphos [72]. The first four predict the window of the HFO, while the latter outputs a single point like our SNN. The traditional HFO detectors, as well as the deep-learning one [3], consider the 200 ms around the insertion event as ground truth for any type of oscillatory event. Since the window is larger than the ones we use, we can ensure our SNN's results are not inflated due to a large confidence window. In the Results and Discussion section, the predictive capabilities of our model are compared with existing approaches.

5.2.2 Clinical Dataset

Through a collaboration with *Hospital de Santo António*, we were able to test our neuromorphic HFO detector using real data from epilepsy patients. As a result of this partnership, we acquired sEEG recordings from 4 patients undergoing epilepsy surgery preparation. While the sample size is insufficient to train our model, it is a valuable asset to test the performance of the spiking neural network in a clinical setting when trained using synthetic data. Creating a clinical dataset for HFO detection is a complicated task since it implies having clinicians manually annotate the ripples and fast ripples, which is a time-consuming process. Thus, this is one of the first works testing neuromorphic technologies to detect HFOs, especially including trials with clinical data and, to the best of our knowledge, the first implementation using Intel's Loihi.

5.2.2.1 SEEG Recording

The Intracerebral recordings were performed using depth electrodes implanted in orthogonal and oblique trajectories. Each patient had between 12 and 16 electrodes, each containing 8 to 18 contacts. The electrodes had a diameter of 0.8mm and a length of 2mm and were separated by

	Patient 1	Patient 2	Patient 3	Patient 4	Total
Ripples	718	329	923	517	2487
Fast Ripples	559	194	410	376	1539

TABLE 5.1 – Marked Ripple and FR count for each patient.

1.5mm. The iEEG recordings were acquired on a digital system with a sampling rate of 2048 Hz with 16-bit resolution, a high-pass filter (cutoff = 0.0016 Hz), and an antialiasing low-pass filter (cutoff = 680Hz) [3].

5.2.2.2 HFO Annotation

Following Gotman's group methodology [25], 5 minutes of NREM sleep were selected from the third night after sEEG implantation. Special care was taken to ensure the trace segment was free of artifacts. HFOs were manually identified and annotated during the selected time. The oscillatory activity was only considered when it clearly stood out from the background signal, with more than four up-down motions, and was classified as ripple if its frequency was in the 80-250Hz range or fast ripple in the 250-500Hz range. The Raw sEEG was later anonymized and exported to feed it to the data pipeline.

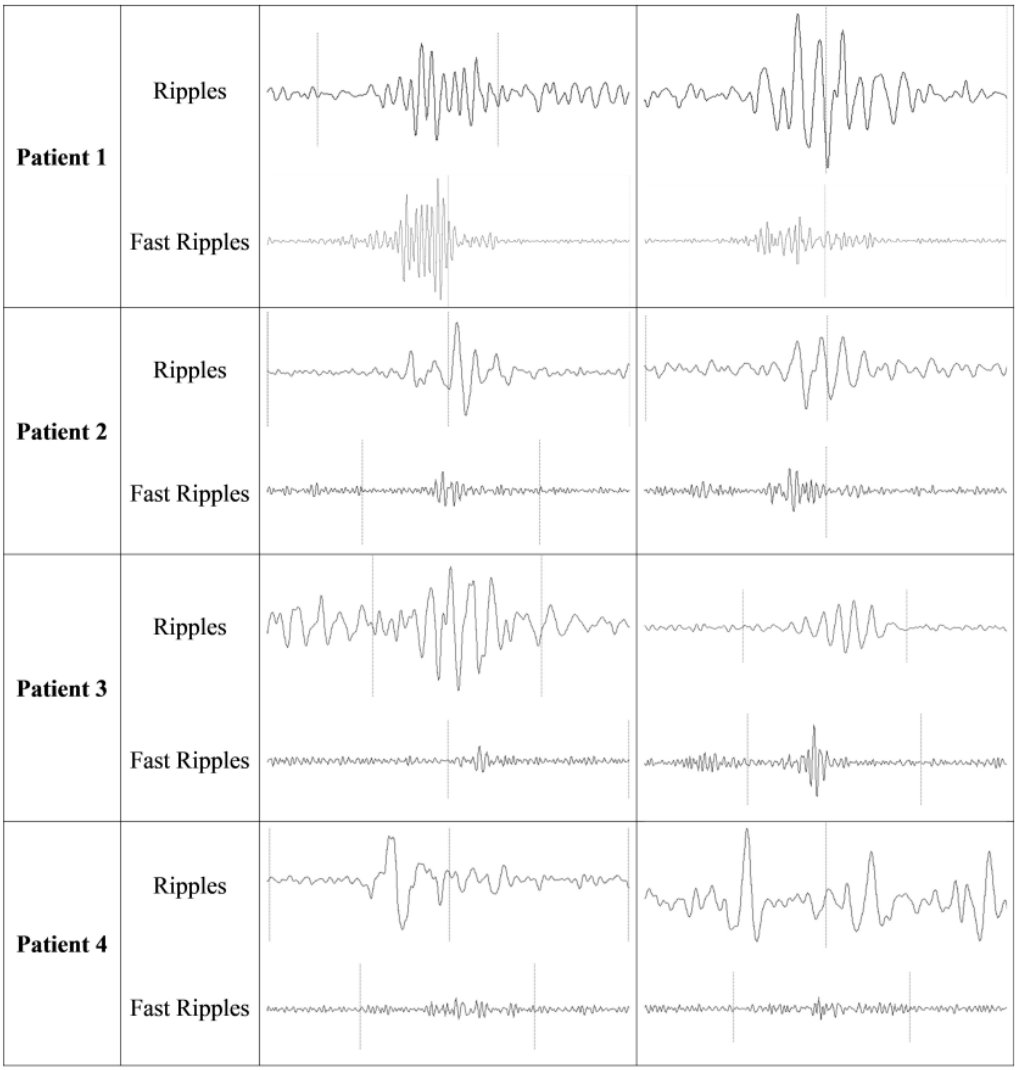
Figure 5.3 depicts the type of clinical signals collected in this dataset for each patient. The ripple and fast ripple shown correspond to the signal after filtering in their corresponding range. It is clear that the signal's quality varies significantly with the patient, with patient 4 standing out as the worst quality. Furthermore, table 5.1 includes the number of annotated ripples and FR for each patient. Due to software limitations from our clinical partners, we were only able to obtain an instantaneous time step as ground truth for the HFO location. This is similar to the ground truth in the synthetic dataset, yet, in this case, the time point does not represent the insertion timing but rather when the expert visually detected the event. Consequently, we will consider a causality window (*causality_window*) around the annotated time to classify predictions as correct or incorrect.

5.3 Input Handling

The synthetic and clinical datasets share a similar format, which simplifies the input handling phase as we can reutilize portions of the code for both applications.

5.3.1 Data Acquisition

We took advantage of the previous work conducted by Sofia [3] in our lab to replicate the acquisition step. The data for the *in vivo* experiment is stored in Matlab binary files. Thus, we can import the data into a Python dictionary using the *loadmat* utility function from SciPy. The relevant fields for this study store the sampling rate, recording duration, total number of samples per electrode, channel information, annotated markers (position and label), and the sEEG data. In addition to



©Source: adapted from [3]

FIGURE 5.3 – Examples of Ripple and FR events for the four patients in the clinical dataset filtered in the respective ranges.

Channel Index	Annotated Events		
0	('Fast-Ripple', 1000)	('Ripple', 7520)	('Ripple', 12000)
1	('Ripple', 2573)	('Fast-Ripple', 3060)	
2	('Spike+Ripple+Fast-Ripple', 530)		

TABLE 5.2 – Structure of the processed HFO file containing the ground truth.

the binary files, CSV files contained the ground truth events and the annotations using traditional detectors.

5.3.2 Data Processing

Before handing the data to the real-time algorithms, the sEEG and annotated markers' data need to go through feature engineering. Tackling the sEEG data first, this field has a similar structure to the spiking data of the *in vitro* practical study. As we mentioned in the data processing sub-section 4.3.2, the input should be formatted to a style that fits the online arrival of events. Hence, we applied the same transformation here to the raw sEEG signal, including the optional selection of relevant channels.

As for the annotated events, we utilized the Pandas library to load the CSV into a Python Dataframe. Since the neuromorphic algorithm runs at the millisecond precision, we started by converting the *Position* field to milliseconds. Then, the ground truth information was extracted from the Dataframe into a NumPy structured datatype with shape $(\text{number of channels}, \text{number of events})$ to utilize the same data structure as the input data. Each event in the array had two fields: *label* and *position*, represented in table 5.2 by a 2D tuple. To perform this conversion, we had to map each relevant event to the associated channel index since the raw markers made this association through the *Target* field, a string indicating the name of the electrode, and respective contact point. Thus, we created a column *channel_idx* indicating the channel index associated with each marker.

The outputs of this stage are two NumPy files — one for the recorded stereo-electroencephalography data and the other for the ground truth.

5.3.3 Data Visualization

The developed visualization utility functions were reutilized in this chapter. As explained in the Proposed Solution section, the procedure we followed to create an optimal HFO Detector relies on understanding the nature of the data. That is why a visualization module follows every implementation step that modifies the input. To start with, the synthetic raw sEEG data is represented in figure 5.4. Image a) contains the signal during the whole recording for 12 channels, highlighting the heterogeneity of SNRs. Images b) and c) incrementally focus on a smaller time window. Notice, however, that the ripple annotated by the yellow box is hard to detect as it is. The oscillatory activity is not visible because the signal is not filtered in the appropriate band, 80-250Hz, as demonstrated in the filtering segment 5.4.1. Figure B.1 provides a comparable example from the clinical source.

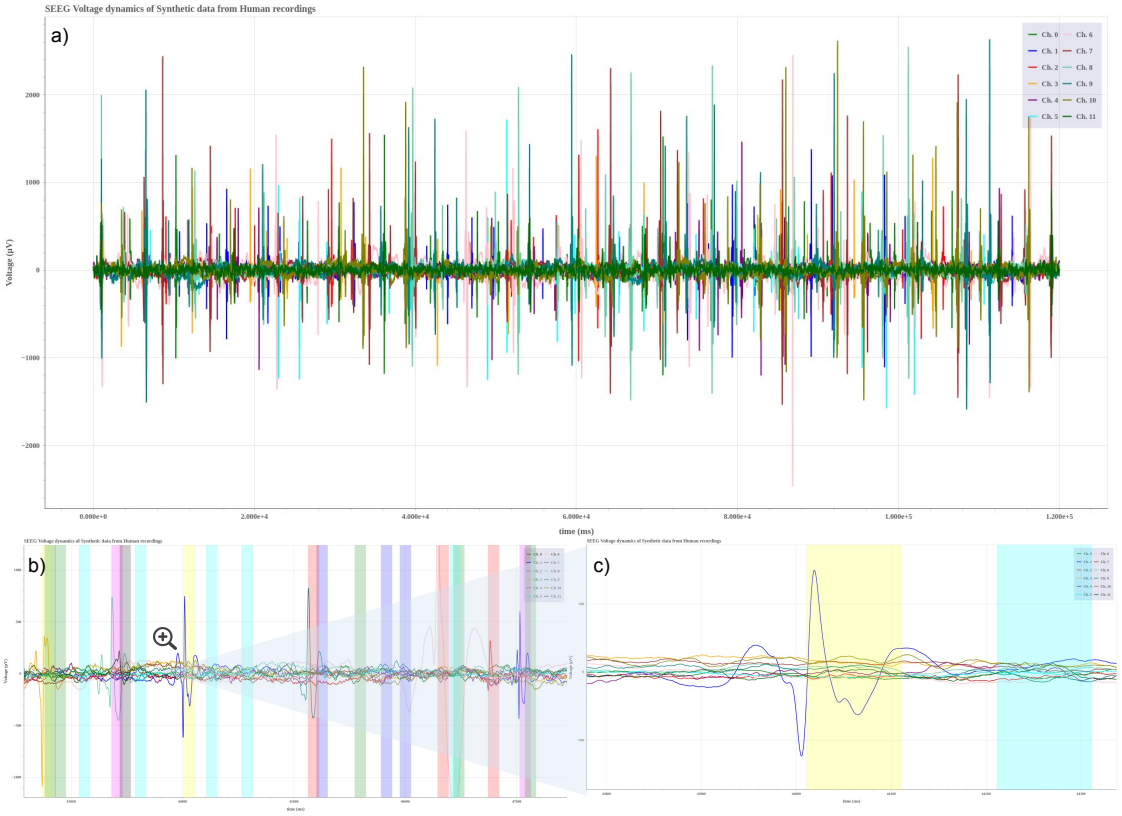


FIGURE 5.4 – Line plot of the processed synthetic sEEG data at increasing levels of resolution. For visualization purposes, 12 representative channels were selected. Images b) and c) contain colored boxes representing different annotated events. Image c) shows a Spike+Ripple event in greater detail.

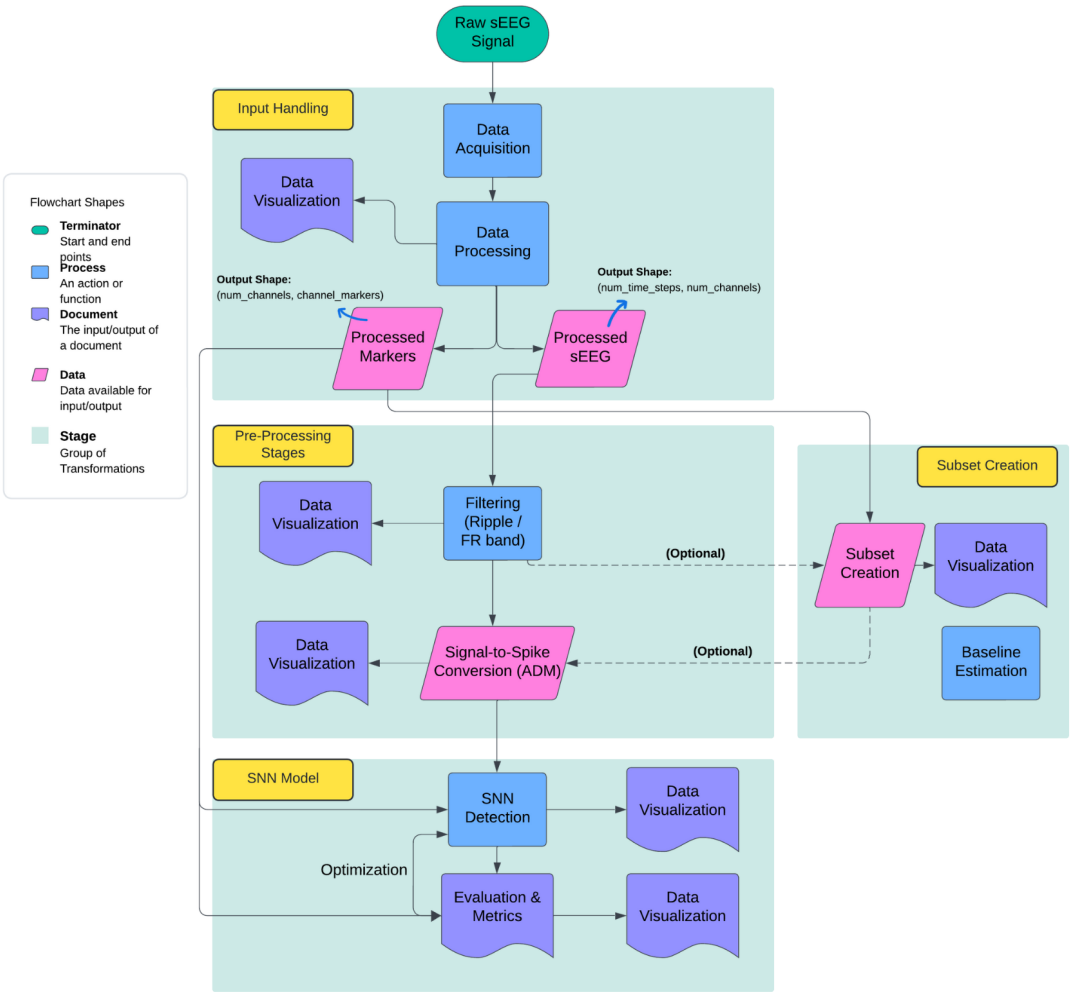


FIGURE 5.5 – Data Pipeline of the HFO Detector, from Data Collection to the Model's Evaluation

5.4 Proposed Solution

The HFO Detector is organized into multiple stages that can be divided into four main modules: Input Handling, Pre-Processing, Subset Creation, and SNN Model. Because Loihi and even SNNs are up-and-coming technologies, current applications are sparse and spread among different industries. That being the case, finding the optimal HFO detection process was exploratory work, which is why we tested several configurations and pre-processing techniques. Anyhow, figure 5.5 presents an overview of the series of processes that data goes through, also known as the data pipeline, from the raw sEEG signal to the classification evaluation and metrics. The next sub-sections will cover each implementation step of the pipeline.

5.4.1 Filtering

The first Pre-Processing Stage is Filtering. Based on [53], since we are working with iEEG data that contains higher resolution, the signal may be filtered in both the ripple and FR frequency ranges. However, filtering can also introduce artifacts due to the presence of sharp and brief non-HFO events, so we sought to study the impact of the filtering strategy on the detection results. Thus, in addition to filtering the sEEG signals in the ripple (80-250Hz) and FR band (250-500Hz), we assessed the performance of an HFO detector that simultaneously detected ripples and FR by filtering in the 80-500Hz range (HFO Band). The filtered signals are plotted in figure 5.6.

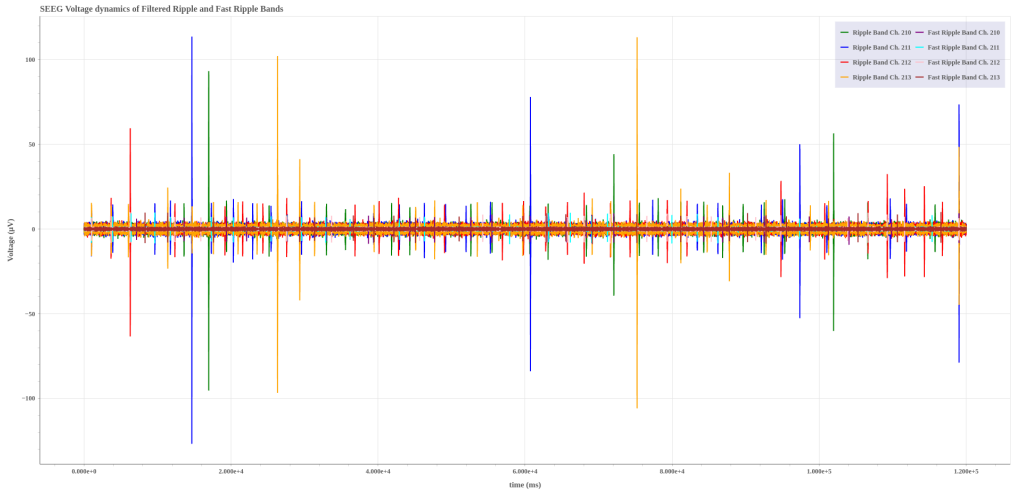
Although the development of the SNN runs in software through the Loihi simulation, it is important to remember we aim to deploy the solution to neuromorphic hardware. Thus, every pre-processing step requires a hardware implementation that avoids adding significant delay to the system. In this case, a combination of analog filters can be utilized to form three operation amplifiers configured to form a *Tow-Thomas* resonating architecture [53]. Again, for the scope of this work, we did not have access to a physical Loihi device, so the hardware implementation is mentioned as a preparation for future work. Regarding the software implementation, we used digital Butterworth filters since they are a good approximation of the proposed hardware filtering strategy.

5.4.2 Signal-to-Spike Conversion

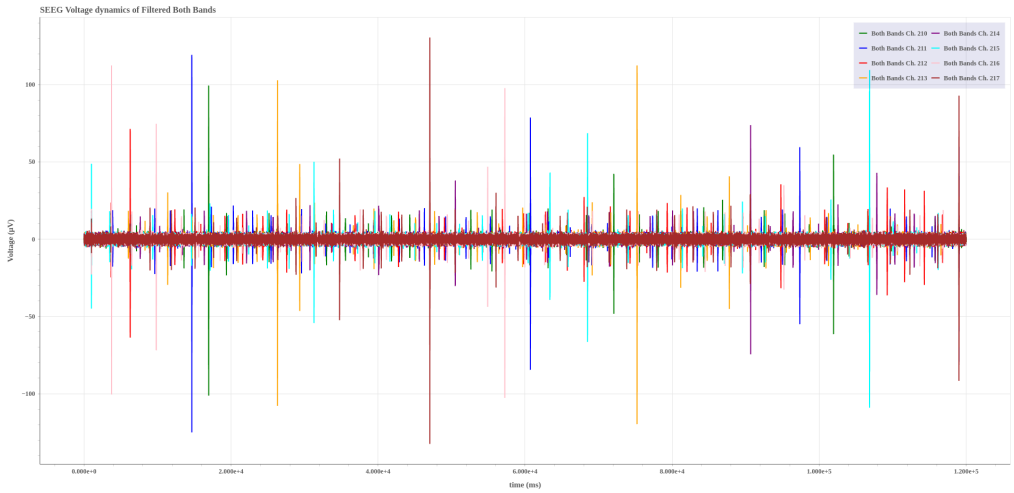
This is when the neuromorphic solution starts to diverge from the deep-learning counterpart. The ANN HFO Detector would require dividing the signal into overlapping windows, which, in turn, would be the input pieces of the model. However, this is unnecessary with an SNN because the network is designed to receive activations in real-time so that a continuous input can be fed directly to the network. Nevertheless, the spiking network cannot comprehend the filtered sEEG data, so it is necessary to find a proper conversion algorithm to transform the analog signal into digital pulses.

After performing trials with several heuristics, we converged toward a solution initially proposed by Sharifshazileh et al. [69] that transforms the input signal into UP and DOWN (DN) spikes. These two spike trains capture sudden amplitude surges or drops in the input signal to ultimately represent the oscillatory activity of high-frequency oscillations. This way, when the SNN receives intermittent UP and DN stimulation in a short period of time, as represented in example a) of figure 5.7, the network should spike, announcing the presence of a high-frequency oscillation. Hence, the UP spikes indicate when the signal increases in amplitude more than a user-defined threshold (v_{thresh_up}), and the DN spikes mark when the signal decreases in amplitude more than a user-defined threshold (v_{thresh_dn}). With these engineered features, the spiking rate depends on the derivative of the signals and the user-defined thresholds, providing the SNN with an indirect metric of the signal's shape.

The performance of the HFO Detector is heavily dependent on this pre-processing stage. After all, the activation activity arriving at the SNN corresponds to the output of the signal-to-spike



(A) Filtered sEEG in the Ripple and Fast Ripple bands.



(B) Filtered sEEG in the combined band (80-500Hz).

FIGURE 5.6 – Line plots of the filtered synthetic sEEG data during the full recording for different filtering bands. A subset of the 960 channels is presented for visualization purposes.

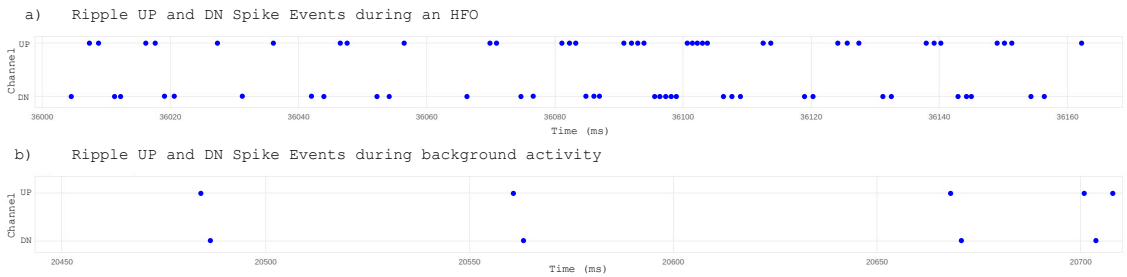


FIGURE 5.7 – UP and DOWN spike train segments. Image a) shows a segment containing frequent oscillatory activity, while b) corresponds to background activity.

conversion. To ensure the UP and DN spikes bring relevant information to the model, the configurable thresholds (v_{thresh_up} and v_{thresh_dn}) must be selected appropriately. In the results section, the classification metrics of the detector are compared using different threshold selections. We performed a Baseline Estimation 5.4.2.2 to find the amplitude ceiling of the background activity and, consequently, calculate the optimal thresholds using distinct heuristics.

The signal-to-spike conversion can be implemented in hardware by applying an asynchronous delta modulator (ADM) block right after the filters, which converts the analog signal into the UP and DN digital pulses according to the changes in amplitude [53]. We emulated this behavior in Python through a dedicated module that iteratively searches the filtered sEEG and builds the corresponding UP and DN spike trains as the thresholds are exceeded. The final spike trains hold a list of timestamps representing the time steps the network shall be stimulated and the algorithm's implementation is shown in Figure C.3.

5.4.2.1 Subset Creation

We previously mentioned the relevance of finding suitable UP and DN thresholds for the signal-to-spike conversion. The conversion is part of the pre-processing pipeline, so independently of the provided input signal, the thresholds will be fixed parameters. We must then calculate these edge values using a dataset that represents the variability of sEEG recordings, which, as we know, have varying SNRs and are susceptible to noise. With that in mind, we found the need to create a custom subset containing synthetic sEEG data from different channels of the same SNR category. Hence, the resultant Subset signals have the same duration as the original synthetic sEEGs but combine snippets from different channels to promote heterogeneity. An extra reason supporting the creation of this Subset is to increase the frequency of annotated events. Because we want to capture the noise floor that separates baseline activity from the HFOs, the implemented procedure benefits from having a higher density of relevant events. Note that each filtered signal (ripple, FR, and HFO bands) has different noise floors, so the subset creation, baseline estimation, and signal-to-spike conversion were applied separately.

The subset creation starts by defining the pool of channels that the merging algorithm will harness. Here, we opted for the highest SNR channels of any brain region since the SNR of those channels was the closest to the SNR of the clinical dataset. Next, to ensure that the rupture points of the Subset, i.e., the points where signals from distinct channels meet, are as camouflaged as possible, we attributed appropriate values for the length of each segment ($segment\_length$) and the padding at the end of each segment ($segment\_padding$) based on the documented duration range of ripples and FRs (table 2.3) and the duration of these events in our dataset. Ergo, selecting sEEG windows of 500 ms where the annotated event begins at least 200 ms away from the end of the segment guarantees that the signal will be in a background activity state by the end of that same segment.

The third and final step corresponds to the segment-by-segment building of the custom signal. For each half-second segment, we search for a channel with an annotation obeying the

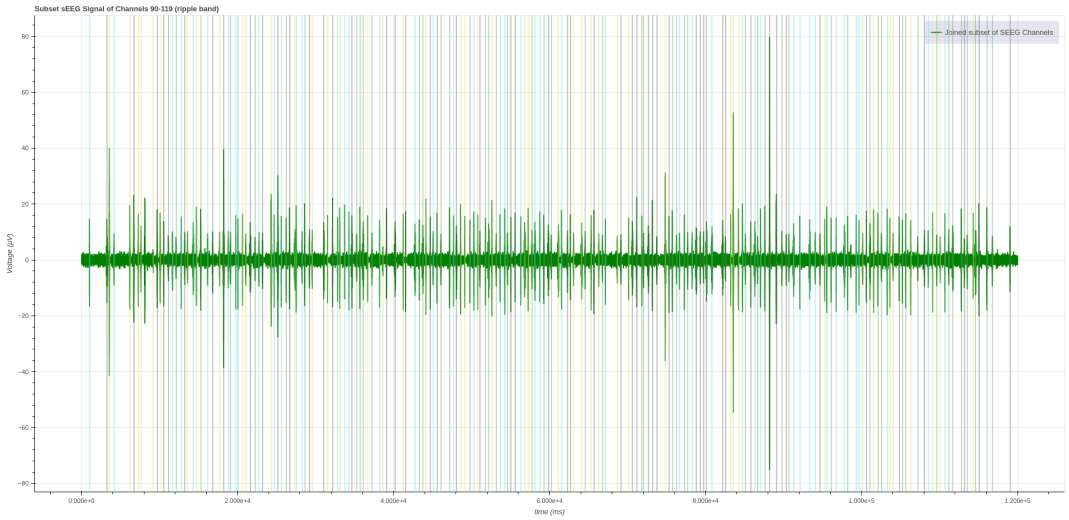


FIGURE 5.8 – sEEG signal of the created Subset in the ripple band.

segment_length and *segment_padding* restrictions while promoting the maximum variety of channels. If such a channel exists, it is appended to the output signal. Otherwise, a random channel is chosen to represent that time frame. The output of this subroutine is a single-channel signal and the ground truth markers that belong to it. Figures 5.8 and B.2 endorse the increase of annotated events in the signal, going from a frequency of 18 HFOs/minute to about 100 HFOs/minute.

5.4.2.2 Baseline Estimation

After building the custom signal, we can finally proceed to estimate the background activity threshold values. These thresholds, v_{thresh_up} and v_{thresh_down}, will be then used to perform the signal-to-spike conversion. To separate the baseline from the relevant events, the signal was windowed into non-overlapping 150 ms segments, followed by an extraction of the maximum signal amplitude of each window. Figure 5.9 separates the time segments with the dotted red lines, highlighting the local maxima with red circles. The choice of the window length is vital for the remaining algorithm. We chose 150 ms after computing the HFO frequency of the custom Subset (1.67 events/second), which means that there is an event, on average, every 0.6 seconds. Therefore, with a time window of 150 ms, about 25% of the segments will include non-baseline activity. Consequently, about a quarter of the local maxima correspond to the maximum amplitude of high-frequency oscillatory events. The distribution of local maxima can be observed in figures 5.10 and 5.11.

Given a distribution of local maximum amplitudes where roughly 25% of the windows correspond to events of interest, the next step involves finding suitable metrics to estimate the baseline amplitude. Hence, we adopted different strategies to extract the thresholds:

1. Lowest quartile (Q1) as the baseline amplitude.
2. Mean as the baseline amplitude.
3. Median (Q2) as the baseline amplitude.

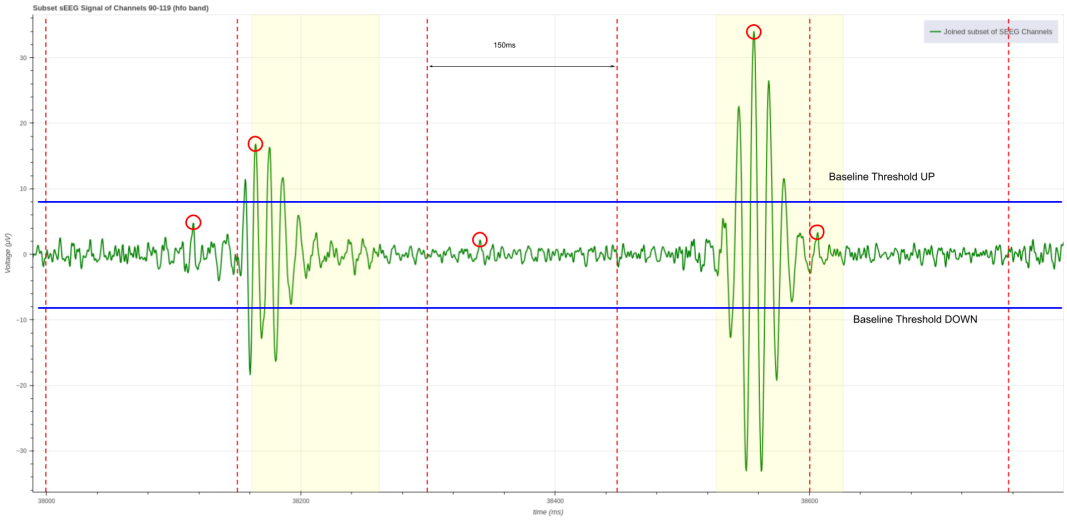


FIGURE 5.9 – Illustration of the Baseline Estimation Algorithm. The red circles show the local maxima, which are utilized to derive the UP and DN thresholds (blue lines).

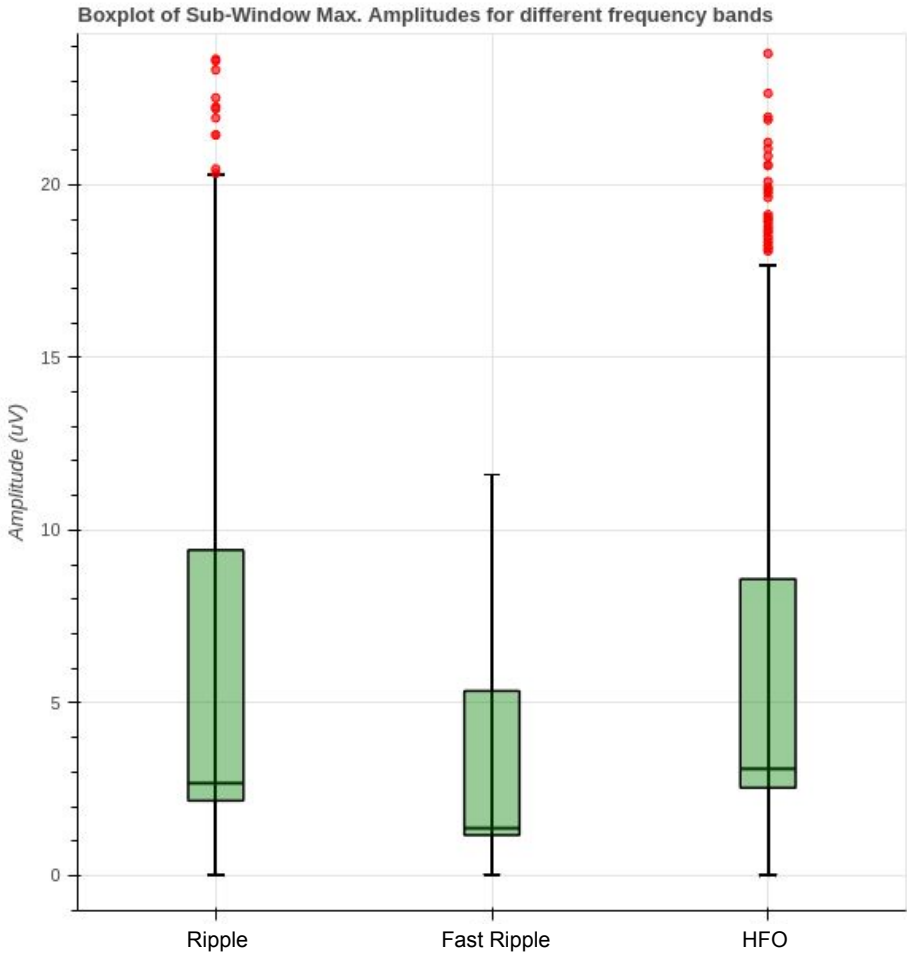


FIGURE 5.10 – Boxplot with the distribution of local maxima amplitudes for each frequency band.

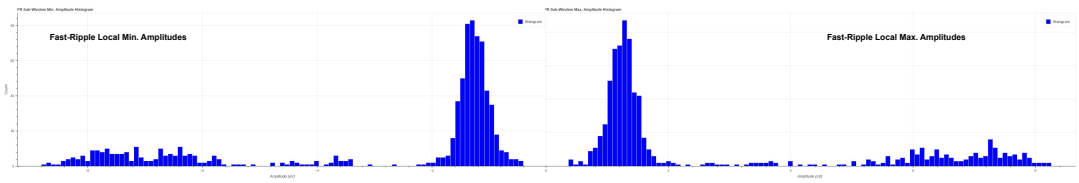


FIGURE 5.11 – Histogram of local maxima and minima for the FR frequency band. The amplitude is represented in the x-axis, and the height of each bar corresponds to the frequency of the bin.

4. 75-percentile (Q3), followed by a scaling factor as the baseline amplitude.

5. 70-percentile as the baseline amplitude.

In alternative 4, the scaling factor is added to scale down the thresholds in an attempt to capture relevant information from the baseline signal that impacts the HFO detection. The scaling factor differs for the ripple, FR, and HFO bands, taking the values 0.6, 0.3, and 0.45, respectively. Lastly, v_{thresh_down} is calculated by multiplying v_{thresh_up} by -1 since the signal is roughly symmetric around the x-axis, as verified by the FR histogram of local minima and maxima 5.11 and table 5.3. The Results section presents the best-performing techniques and their respective parameters.

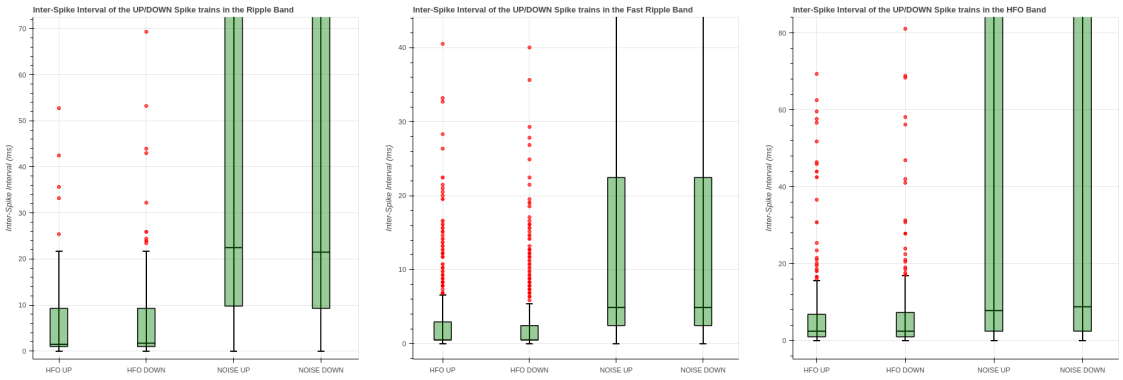
5.4.2.3 Analyzing the Spike Trains

At this point in the data pipeline, the input was converted from a continuous signal to spikes (UP and DN spike trains). All that is left is to design an SNN architecture that is able to discern the spiking activity during high-frequency oscillations from the baseline activity. However, we have no guidelines on how we should configure the network's neurons, namely their synaptic and membrane potential time constants, which is why this intermediate step is crucial. An analysis of the inter-spike interval (ISI) in the UP and DN spike trains is performed, where we aim to identify the time scale between activations during HFO activity versus noise.

Figure 5.12 exhibits the distribution of the ISI of UP and DN spikes during relevant oscillatory events compared to noise. In the ideal scenario, the inter-spike interval distribution during HFOs would be completely disjoint from the distribution during baseline activity. While that is

	Mean	Std. Deviation	Median
Ripple Local Max. Amplitudes	6.128	6.644	2.677
Ripple Local Min. Amplitudes	-6.082	6.622	-2.720
FR Local Max. Amplitudes	2.792	2.460	1.375
FR Local Min. Amplitudes	-2.874	2.581	-1.397
HFO Local Max. Amplitudes	6.317	6.762	3.093
HFO Local Min. Amplitudes	-6.287	6.753	-3.114

TABLE 5.3 – Statistics of the distribution of local maximum and minimum amplitudes according to the baseline estimation algorithm for different frequency bands.



(A) ISIs in the Ripple band (B) ISIs in the FR band (C) ISIs in the HFO band

FIGURE 5.12 – Boxplots of ISIs for each frequency band. Each sub-figure has 4 boxes representing the distribution of the ISI for the UP train during HFO, DN train during HFO, UP train during Noise, and DN train during Noise, respectively.

not the case, there is a clear separation between the activation interval during HFO versus background activity, most noticeable in the ripple band (sub-figure A). A common property of the three frequency bands is the ISI similarity between the UP and DN spike trains, so we will focus our analysis on the UP trains as a representation of both. Looking at the Ripple band first, the median ISI on regions with annotated events is 1.465 ms (IQR: [0.977, 9.278] ms), while otherwise, the median is 22.461 ms (IQR: [9.767, 439.941] ms). From the total generated spikes in this band, 83.18% of them occur inside an HFO. Observing the FR band, 46.57% of spikes are located inside the region of a relevant event instead. A decrease was expected because the signal filtered in the 250-500Hz range has a significantly lower amplitude, thus more susceptible to noise being mistaken with relevant data. The median interval between spikes during HFOs is 0.488 ms (IQR: [0.488, 2.930] ms) versus 4.883 ms (IQR: [2.441, 22.461] ms) during background activity. Finally, in the entire HFO band, the median ISI during relevant events is 2.441 ms (IQR: [0.977, 6.836] ms), with 70.33 % of the activations belonging to this scenario, whereas the baseline's median ISI is 7.813 ms (IQR: [2.441, 167.236] ms).

The performed ISI analysis showcases an example using the created Subset as input. Because the insights from this step will affect the time constants of our SNN model, as explained in section 5.4.3, we use the custom Subset to find a reference value for the time constants to make our SNN robust to slight variations in the signal due to noise or different recording conditions. The ISI distribution for channel 90 of the synthetic dataset is also included in B.3.

5.4.3 SNN Model

Inspired by Indiveri's team HFO detector[69], the developed SNN identifies pertinent oscillatory activity through a layer of Feature Neurons, where each neuron is tuned to react to rippling behavior in a short frequency window. As an example, some neurons might spike in response to 80-100Hz HFOs, while others respond to 140-160Hz HFOs.

5.4.3.1 SNN Architecture

Figure 5.13 depicts the adopted architecture, comprised of three layers:

1. **Custom Input Layer**: Lava Process that sends, for every time step, the input spikes to the remaining network. The input comprises two spike trains, UP and DN spikes, generated via the ADM algorithm explained previously in 5.4.2.

2. **Dense layer**: The Dense Layer receives the UP and DN spikes and propagates them to the Feature Neurons Layer, so the number of output nodes of this layer is equal to the number of feature neurons. In turn, each output node of the Dense Layer receives an excitatory (red lines) and an inhibitory current (blue lines) from the UP and DN nodes, respectively. On top of that, for each excitatory/inhibitory connection pair, the corresponding weight is drawn from a distribution function to introduce variability in the intensity of a single activation. Contrarily to the excitatory synapses, the weight of the inhibitory connections is negative to propagate a negative action potential to the feature neurons.

3. **Feature Neurons**: Final layer composed of a determined number of refractory leaky integrate-and-fire neurons supporting two input currents (inhibitory and excitatory) with different synaptic time constants (τ_{exc} and τ_{inh}). This neuron model was developed for the scope of this solution (*ConfigTimeConstantsRefractoryLIF*) and is further explained in segment 5.4.3.2. Each feature neuron is tuned to detect HFOs in a small window of the frequency spectrum, acting like a band-pass filter. The neurons' dynamics follow a distribution function to capture HFOs in different frequencies and each processing node is classified as either a silent, noisy, or detector neuron in the Feature Neurons Classification stage 5.4.4.

5.4.3.2 Developed Neuron Model

In the *in vitro* practical study, we implemented the refractory version of the CUBA LIF model, the default neuron model in Lava and Loihi. Even so, that model was too simplistic to represent the dynamics required for the HFO detector. In this case, we need to define different time constants for the excitatory and inhibitory synaptic connections since the interplay between the two decays will provide the band-pass behavior to the neurons. Moreover, the CUBA LIF included in the library only supports a single decay value for the entire neural layer for both the membrane potential decay (d_v) and current decay (d_u). This is not desirable since we cannot add variability inside a layer if the underlying time constants are the same, which forces all neurons to behave equally in response to the same input. Hence, we created our custom refractory CUBA LIF model with distinct excitatory and inhibitory synaptic time constants and support for different time constants on a neuron-to-neuron basis, named *ConfigTimeConstantsRefractoryLIF*. The equations governing the model are the following:

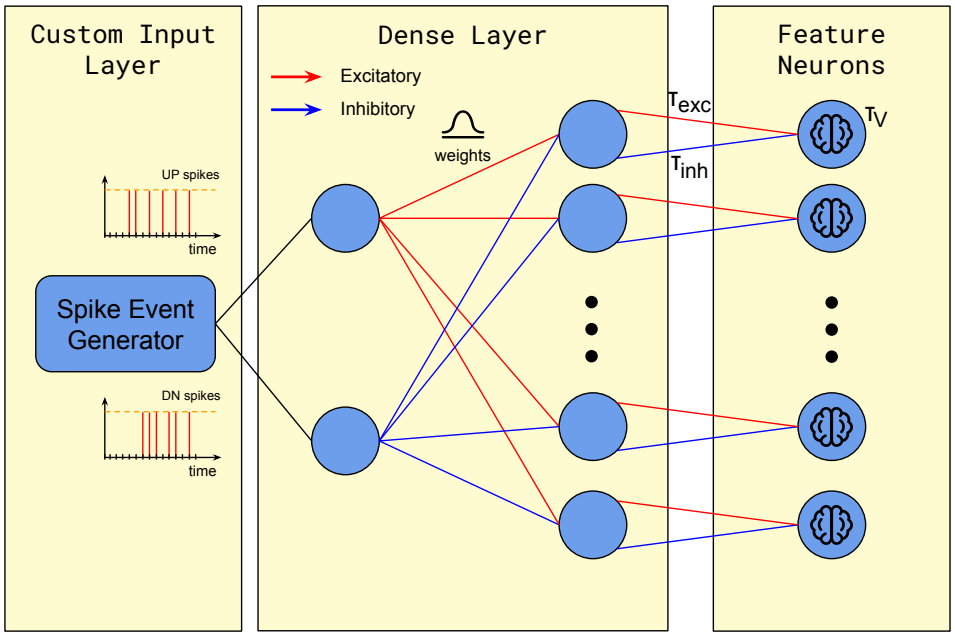


FIGURE 5.13 – Architecture of the SNN HFO Detector.
The weights of the Dense layer follow a distribution function around a pre-determined value, with each excitatory/inhibitory pair having the same absolute value but opposite sign.

1. The excitatory synaptic current, $U_{exc}(t)$, evolves according to the equation:

$$U_{exc}(t) = U_{exc}(t-1) \cdot (1 - du_{exc}) + I_{exc} \tag{5.1}$$

2. The inhibitory synaptic current, $U_{inh}(t)$, follows the same principle:

$$U_{inh}(t) = U_{inh}(t-1) \cdot (1 - du_{inh}) + I_{inh} \tag{5.2}$$

3. The synaptic current, $U(t)$, corresponds to the sum of the E/I currents:

$$U(t) = U_{exc}(t) + U_{inh}(t) \tag{5.3}$$

4. The membrane potential, $V(t)$, is updated as in the traditional CUBA LIF model:

$$V(t) = V(t-1) \cdot (1 - dv) + U(t) + bias \tag{5.4}$$

5. When $V(t) > V_{threshold}$, the neuron fires a spike and resets the voltage, staying in the refractory state for *refractory_period* steps:

$$V(t) \leftarrow 0 \ (Discontinuous\ dynamics) \tag{5.5}$$

In equations 5.1 and 5.2, I_{exc} and I_{inh} represent the excitatory and inhibitory synaptic current the neuron receives at timestep t. Whilst the *bias* represents a constant increment of the membrane potential for specific scenarios. Due to each neuron having a unique du_{exc}, du_{inh}, and dv, there are 5 degrees of freedom to configure each neuron's response to an input, the aforementioned decay values and the I_{exc} and I_{inh} values. However, we mentioned in the SNN architecture's definition that the intensity of the excitatory and inhibitory activations is the same, condensing to 4 degrees of freedom. The fourth variable is controlled by the weight of the Dense Layer associated with each neuron. In comparison, the original CUBA LIF model has a single degree of freedom, the activation intensity I_t.

The extra capabilities of the developed neuron model are apparent. Because we believe it can be helpful for other use cases outside this work, we proposed adding it to Lava's repertoire of models, and the corresponding pull request is currently under review. The source code is included in figure C.4.

5.4.3.3 Network Parameters

That said, we have everything necessary to run the SNN. However, numerous parameters must be tested to find the combination returning the best results. Getting the fixed parameters out of the way first, only the refractory period and the number of steps were pre-determined. The former is equal to the causality window of the frequency band we are predicting. So, the refractory period is set to 180 ms for the ripple and HFO detector (both bands) and 80 ms for the fast ripple detector. Regarding the number of steps, it is equal to the sample size of the given input file. The input can come from three different sources: a channel of the synthetic dataset, the custom subset, or a channel of a patient's sEEG. On the other hand, the tunable variables are explained below:

1. **Chosen baseline algorithm**: The baseline algorithm determines the strategy employed to convert the continuous signal into spikes. Each strategy is associated with different UP/DN spike trains that will be fed to the network. Additionally, the baseline algorithm also affects the excitatory synaptic time constants. The conducted ISI analysis 5.4.2.3 for each UP/DN spike train returned distribution statistics (mean, standard deviation, and IQR) that will be used to create a heterogeneous neural population with different du values.

2. **Distribution strategy of the excitatory synaptic time constants**: This parameter specifies how the distribution of du_{exc} values is created. One strategy is to use the IQR of the inter-spike intervals to generate a normal distribution of excitatory synaptic time constants with the same IQR. The second strategy relies on the mean and standard deviation of the ISI to model the normal distribution of time constants. Because the time constants must be positive, the last strategy uses the mean and standard deviation to generate a log-normal distribution of the time constants, which is strictly positive by nature.

3. **Inhibitory synaptic time constant subtract interval**: So that neurons can spike in response to a specific frequency range but not to inputs going over or below that range, we

introduce an inhibitory time constant that is slightly smaller than the excitatory one. Hence, the amount subtracted from the excitatory time constant is derived from a uniform distribution around a set range, and a value is picked for each neuron. The range [0.1 ms, 1 ms] returned suitable results.

4. **dv Distribution**: As for the membrane potential time constant, since it is another degree of freedom, we decided to let it vary across neurons following a normal distribution. $dv \sim N(\mu = 15ms, \sigma = 10ms)$ allowed capturing HFOs in our datasets.

5. **Weights distribution**: Following the same logic, the weights of the dense layer also follow a normal distribution. In particular, $w \sim N(\mu = 0.2, \sigma = 0.1)$ yielded the best results according to our tests.

How well the configurable parameters are chosen will determine the predictive capabilities of the model. As explained in sub-section 2.2.4, not all SNNs require a local learning rule. In this case, we opted to develop a solution that is similar to Reservoir Computing, explained in segment 2.2.4.4. Our SNN has a middle layer with engineered entropy, resulting in different neurons capturing different patterns from the input. The Output Layer, where the relevant computations are filtered from the reservoir, is explained in sub-section 5.4.4.

While it would be interesting to analyze the SNN's ability to learn using supervised learning or unsupervised local learning rules, we considered it a future step and something that should be explored after understanding the SNN dynamics and how they can be configured based on a deep understanding of the data being fed. Another set of problems arises when considering supervised learning rules, such as the training dataset's dimension, the ground truth robustness, and how to model the error function of the temporal-coded SNN and its propagation along layers. In the Future Work section, we delve deeper into this route and how we would add learning capabilities to the existing model.

5.4.4 Feature Neurons Classification

Finally, this is the last step of the data pipeline, responsible for classifying each feature neuron into one of three classes: Silent Neuron, Noisy Neuron or Detector Neuron. Due to the entropy of the LIF layer, we know that each neuron has different dynamics and consequently activates in response to distinct-natured HFOs. The randomness of the process causes some nodes to act as we intend and others to detect features we are not looking for, which is why we need to do the classification of the feature neurons as an extra step during training. A production version already acknowledges which neurons are capturing relevant features, so it can skip this phase.

Anyhow, the neuron's classification is conducted using an annotated signal and analyzing the ground truth against the neuron's firing timings. If a feature neuron does not spike during the whole experiment, it is classified as a Silent Neuron. Otherwise, if the neuron does spike, we look at the relevancy ratio of the stimulations, which corresponds to the percentage of spikes that succeed an HFO marker inside its causality window. Ergo, another tunable parameter,

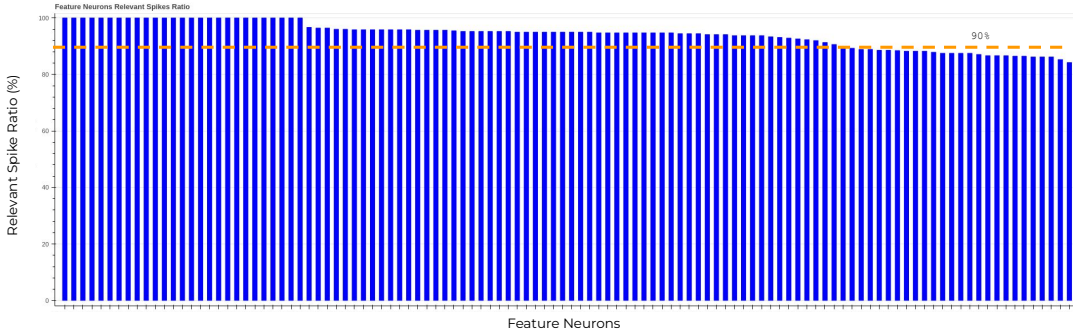


FIGURE 5.14 – Barplot of relevant spike ratio of a Feature Neuron Layer detecting ripples on the custom Subset. The bars are ordered by the descending value of the y-coordinate, and only a fraction of the neurons are represented for better visualization.

relevant_ratio_threshold, defines what is the minimum relevancy ratio to consider a computational unit a Detector Neuron. The neurons that exceed this limit are classified as Detectors, while the others are considered Noisy Neurons. A plot of the relevant spike ratio for a given *in silico* neuronal population is depicted in figure 5.14.

As we will see in the Results section, the SNN detects high-frequency oscillatory events (ripples, FRs, or both) when enough feature neurons spike in fast succession, at most taking *causality_window* ms. The number of neurons that need to spike is defined by a variable named *num_consensus_neurons*. For most acquired results, only one feature neuron's spike was required to detect an HFO ($num\_consensus\_neurons = 1$).

5.5 Results

This section compiles the results of the *in vivo* practical experiment. It is split into three parts: the dynamics of the modeled neurons, the classification metrics on the synthetic dataset, and testing the HFO detection on the clinical patients' dataset.

5.5.1 Dynamics of the Modeled Neurons

Similar to the Activity Burst detection study, this segment aims to give an intuition of what happens to the SNN neurons during execution. The analysis is performed on an HFO detector tuned to the ripple band and fed with the created Subset's data. For that, the voltage and current dynamics of five feature neurons are shown representing the whole layer. Each selected neuron is illustrated by a different line color in figure 5.15. The figure presents three rows with an increasing time resolution. Looking separately at each row, the membrane potential is located in the first column and the synaptic current in the second.

In the upper image, we observe the neuron's behavior at the population scale. Although it is difficult to draw conclusions regarding a single channel's behavior, it is clear that some neurons are more responsive to input spikes than others. The orange neuron stands out particularly since

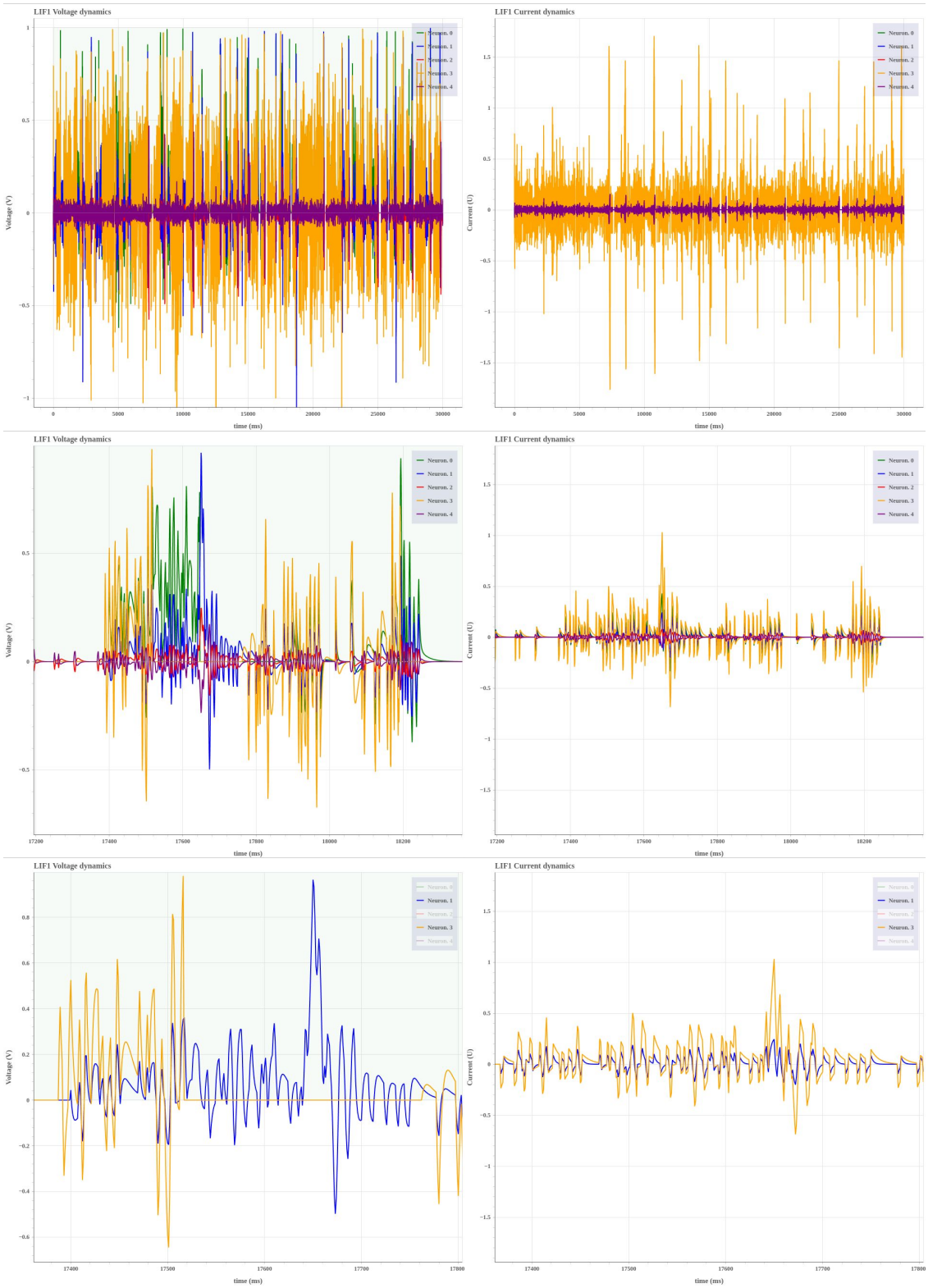


FIGURE 5.15 – Voltage and Current Dynamics of the Ripple Detection on the created Subset at different time scales.

Baseline Algorithm	Accuracy	Precision	Recall	F1-Score	Relevancy Ratio
Q1	99.99%	95.12%	98.98%	97.02%	96%
Mean	99.98%	93.00%	94.42%	93.70%	95%
Median	99.99%	95.05%	97.46%	96.24%	95%
Q3 + Scaling Factor	99.97%	95.74%	95.74%	95.74%	95%
70-Percentile	99.97%	88.42%	94.42%	91.85%	85%

TABLE 5.4 – Classification Metrics of the Ripple Detector (80-250Hz) using different Baseline Estimation Algorithms.

its corresponding current chart obscures the dynamics of the remaining neurons due to its amplitude interval. The input spike intensity is modulated by the weights of the Dense Layer, which varies on a neuron-to-neuron basis. Analyzing the middle image, the oscillatory behavior of the computing unit's current and voltage becomes apparent. The undulatory pattern is due to the intertwined arrival of excitatory and inhibitory spikes that bring a positive and negative net current, respectively. It is important to mention that the represented synaptic current corresponds to the sum of the excitatory and inhibitory currents at every time step. At the greatest time resolution, we hid the output of some neurons to make the spike of neuron three easier to visualize. The voltage cumulatively increases in a wave motion until reaching $v_{threshold}$, which equals 1. For the next $refractory\_period$ ms, the membrane potential of the spiking neuron remains at the baseline level.

5.5.2 HFO Detection on the Synthetic Dataset

First, we evaluated the performance of the HFO detector on the synthetic dataset. As covered in segment 5.4.3.3, multiple network parameters were configurable, so we comprehensively tested their combinations and hereby present the main findings and results. Similarly to the classification metrics of the *in vitro* study 4.5.1.2, the TN count is a misleading statistic, so the models should be benchmarked according to the precision, recall, and f1-score.

To isolate the influence of the chosen baseline algorithm on the classification metrics, tables 5.4, 5.5, and 5.6 exhibit, in order, the prediction results of the ripple detector, fast ripple detector and HFO detector, using different baseline detection algorithms. The SNN received UP/DN spikes derived from the created synthetic subset for this analysis. Interestingly, there is no universal baseline algorithm that performs the best for all frequency ranges. In the ripple detector, estimating the baseline using the first quartile strategy (Q1) yields the best results, with a f1-score of 97.02%. Meanwhile, the third quartile (Q3), followed by a scaling factor, achieved the best predictions in the FR and HFO detectors, with a f1-score of 97.96% and 90.02%, respectively. The fast ripple detector was the overall best performer, with the results from the ripple detector being slightly worse. In comparison, when we filter the sEEG signal in the whole HFO band (80-500Hz), the prediction outcomes deteriorate, which was expected. Moreover, notice that the finest relevancy ratio guiding the classification of neurons as detector neurons or noisy neurons, varies by strategy and frequency band.

Baseline Algorithm	Accuracy	Precision	Recall	F1-Score	Relevancy Ratio
Q1	99.98%	90.45%	100%	94.99%	95%
Mean	99.97%	88.73%	94.97%	91.75%	90%
Median	99.99%	95.57%	97.49%	96.52%	95%
Q3 + Scaling Factor	99.99%	96.00%	100%	97.96%	95%
70-Percentile	99.98%	92.42%	97.99%	95.12%	92%

TABLE 5.5 – Classification Metrics of the FR Detector (250-500Hz) using different Baseline Estimation Algorithms.

Regarding the distribution strategy of the excitatory synaptic time constants, we tested the FR detector using the best-performant baseline algorithm and remaining parameters. By varying only the distribution strategy (table 5.7), the log-normal distribution approach produced the best results.

As a result of the parameter comparison, the optimal configuration of the HFO detector of each band was found. Hence, we proceeded to evaluate the model on single channels of the synthetic dataset. Tables 5.8, 5.9, and 5.10 contain the ripple, FR, and HFO detector results on five individual channels of the synthetic dataset, as well as the performance averaged across these five channels. The results are aligned with what was observed previously. The FR detector remains the best performer, followed by the ripple detector and, finally, the HFO detector. There is a slight decrease in performance when detecting single channels of the synthetic dataset, but the precision, recall (sensitivity), and f1-score metrics remain very significant, performing better than the traditional HFO detectors and comparably to the deep-learning detector [3], as we will clarify in the discussion part. There is a visible decrease in performance detecting HFOs on synthetic channel 0, with a remarkably low sensitivity on the ripple and HFO band. The reason for this is unknown, but it could be due to the model neglecting relevant patterns or mislabeled events.

To facilitate the process of choosing the right parameters, the classification metrics are exported into *JSON* files following the example of figure B.4.

5.5.3 HFO Detection on the Clinical Dataset

The SNN detectors of high-frequency oscillatory events were then tested on the data from epilepsy patients. We had access to sEEG recordings from four patients, so we applied the same classification configuration utilized for the synthetic data. Accordingly, tables 5.11, 5.12, and 5.13 showcase

Baseline Algorithm	Accuracy	Precision	Recall	F1-Score	Relevancy Ratio
Q1	99.96%	86.92%	92.79%	89.76%	85%
Mean	99.94%	81.74%	88.74%	85.10%	80%
Median	99.96%	88.26%	91.44%	89.82%	90%
Q3 + Scaling Factor	99.96%	88.65%	91.44%	90.02%	90%
70-Percentile	99.94%	86.32%	82.43%	84.33%	85%

TABLE 5.6 – Classification Metrics of the HFO Detector (80-500Hz) using different Baseline Estimation Algorithms.

Distribution Strategy	Accuracy	Precision	Recall	F1-Score
IQR	99.99%	96.08%	98.49%	97.27%
Mean and Std. Dev.	99.96%	95.14%	100%	97.51%
Log Normal	99.99%	96.00%	100%	97.96%

TABLE 5.7 – Classification Metrics of the FR Detector (250-500Hz) using different du_{exc} Distribution Strategies.

Synthetic Dataset	Accuracy	Precision	Recall	F1-Score
Ch. 0	99.99%	100%	72.5%	84.06%
Ch. 1	99.99%	95.65%	91.67%	93.62%
Ch. 2	100%	100%	100%	100%
Ch. 3	100%	96.0%	100%	97.96%
Ch. 4	99.99%	92.31%	100%	96.0%
Average	99.99%	96.79%	92.83%	94.33%

TABLE 5.8 – Classification Metrics of the Ripple Detector (80-250Hz) with optimal parameters on the Synthetic Dataset.

Synthetic Dataset	Accuracy	Precision	Recall	F1-Score
Ch. 0	100%	92.31%	100%	96.0%
Ch. 1	100%	95.83%	95.83%	95.83%
Ch. 2	100%	96.0%	100%	97.96%
Ch. 3	100%	96.0%	100%	97.96%
Ch. 4	100%	100%	87.50%	93.33%
Average	100%	96.03%	96.67%	96.22%

TABLE 5.9 – Classification Metrics of the FR Detector (250-500Hz) with optimal parameters on the Synthetic Dataset.

Synthetic Dataset	Accuracy	Precision	Recall	F1-Score
Ch. 0	99.98%	90.48%	52.78%	66.67%
Ch. 1	99.98%	89.19%	91.67%	90.41%
Ch. 2	99.99%	88.89%	88.89%	88.89%
Ch. 3	100%	91.67%	91.67%	91.67%
Ch. 4	99.99%	86.11%	86.11%	86.11%
Average	99.99%	89.27%	82.22%	84.75%

TABLE 5.10 – Classification Metrics of the HFO Detector (80-500Hz) with optimal parameters on the Synthetic Dataset.

Clinical Dataset	Accuracy	Precision	Recall	F1-Score
Patient 1	99.94%	86.73%	67.1%	75.44%
Patient 2	99.93%	85.04%	78.19%	81.45%
Patient 3	99.95%	81.82%	79.22%	80.34%
Patient 4	99.7%	78.58%	68.82%	73.35%
Average	99.9%	83.04%	73.33%	77.65%

TABLE 5.11 – Classification Metrics of the Ripple Detector (80-250Hz) with optimal parameters on the Clinical Dataset.

the results of the Ripple, FR, and HFO detectors on the distinct patients. The metrics of each patient correspond to the averaged performance across five selected channels. The final row includes the average performance of the detector in the clinical dataset.

The predictions on the clinical dataset suffer a notable drop in performance. The ripple detector is the one that adapted better to the new data, returning a precision of 83.04%, 73.33% sensitivity, and 77.65% f1-score. Contrastingly, the FR and HFO detectors had issues detecting many annotated events, resulting in a lower true positive rate (recall). The clinical sEEG recordings are vulnerable to noise and artifacts, causing some electrode contact points to capture unusual electrical activity. Contrastingly, the synthetic dataset used for training is much more uniform and predictable. In fact, we noticed the SNN is able to detect HFOs in certain channels much better than others, so the noisier channels have a negative impact on the average performance. Moreover, the process of ground truth generation is prone to errors, and we verified instances where more than one annotation is identified at a distance smaller than the causality window of the respective relevant event. When that happens, for example, two ripples spaced less than 180 ms, the event should be merged into a single annotation, which is not the case here. Lastly, Patient 4's recorded signal has a lower quality, making the SNN and specialists' task of identifying ground truth events more challenging. That is why the predictions for Patient 4 are comparatively worse.

Despite the performance drop, the predictive capabilities of the presented model surpass the performance of the CNN and LSTM proposed by Moreira [3], which had an average f1-score of 52.8% and 46.9% for ripples and FR, respectively. A reason for this might be potential overfitting to the synthetic data, which degraded the deep-learning methods results. In turn, the SNN, with its pool of feature neurons, can capture diverse dynamics of activity and still deliver valuable

Clinical Dataset	Accuracy	Precision	Recall	F1-Score
Patient 1	99.96%	90.03%	37.62%	48.54%
Patient 2	99.94%	85.52%	82.40%	83.81%
Patient 3	99.87%	92.53%	8.87%	15.69%
Patient 4	99.96%	81.63%	58.0%	67.81%
Average	99.93%	87.43%	46.72%	53.96%

TABLE 5.12 – Classification Metrics of the FR Detector (250-500Hz) with optimal parameters on the Clinical Dataset.

Clinical Dataset	Accuracy	Precision	Recall	F1-Score
Patient 1	99.92%	86.25%	45.48%	59.26%
Patient 2	99.95%	58.87%	49.73%	48.86%
Patient 3	99.93%	90.60%	40.91%	48.32%
Patient 4	99.83%	27.69%	57.78%	36.99%
Average	99.91%	65.85%	48.48%	48.36%

TABLE 5.13 – Classification Metrics of the HFO Detector (80-500Hz) with optimal parameters on the Clinical Dataset.

predictions when exposed to another dataset.

5.6 Discussion

The outcomes of this practical study, where we focused on building an SNN capable of detecting complex electrophysiology patterns, advocate the idea that it is possible to make a neuromorphic version of existing detectors with rivaling classification metrics. We cover all the steps data goes through, from the sEEG acquisition to the online detection, including our strategy to convert the input to a format the SNN can comprehend.

As for the classification metrics, our model delivers competitive results when testing it with the synthetic dataset. The ripple detector averages a precision of 96.79%, a sensitivity of 92.83%, and a f-measure of 94.33%. The FR detector achieves an even higher f1-score (96.22%) due to a higher recall (96.67%), even though the precision (96.03%) is lower than the ripple detector. The HFO detector achieves 89.27% precision, 82.22% sensitivity, and an 84.75% f-measure. Table 5.14 compares the prediction metrics with known HFO detectors. The SNN is able to detect more relevant events with significantly greater precision than traditional detectors. Even when compared with the deep learning alternatives, the metrics indicate that the neuromorphic system is slightly more accurate. The sensitivity of the CNN and LSTM is higher but at the cost of decreased precision, so the f1-score that balances these two properties gives a slight edge to the event-driven algorithm in both ripple and fast ripple bands. The HFO band is not compared because the other detectors do not test filtering on the 80-500Hz frequency band.

For the reasons mentioned in the clinical dataset results, the detection performance on the patients' sEEGs declined, with about 18% lower f1-score for the ripple band and a greater decrease on the other frequency ranges. This phenomenon also occurred in the deep learning models but with an even more notable performance drop, as shown in table 5.15. The implemented detector with a neuromorphic design adapts better to the clinical data than the traditional neural networks, retaining a notably higher precision in comparison. In particular, the neuromorphic ripple detector stands out, registering an f1-score increase of roughly 20%. Anyhow, there is clear space for improvement when it comes to the application in a clinical setting. For instance, having access to a larger clinical dataset would enable performing another training step to fine-tune the model to the

Synthetic Dataset		Precision	Recall	F1-Score
Traditional Detectors	STE	96.2% / 51.9%	55.6% / 30.0%	70.5% / 38.0%
	STl	51.2% / 63.4%	80.7% / 100%	62.7% / 77.6%
	HIL	81.3% / 42.9%	67.5% / 35.6%	73.8% / 38.9%
	MNI	56.3% / 40.1%	80.6% / 57.5%	66.3% / 47.3%
	Delphos	53.3% / 48.8%	87.0% / 91.5%	66.1% / 63.3%
DL-Models	CNN	89.8% / 92.5%	96.5% / 99.6%	93.0% / 95.9%
	LSTM	90.2% / 92.4%	97.0% / 99.2%	93.5% / 95.7%
SNN		96.8% / 96.0%	92.8% / 96.7%	94.3% / 96.2%

TABLE 5.14 – Comparison of the Classification Metrics of existing Detectors with our SNN in the ripple and FR bands (Synthetic Dataset). Adapted from [3].
The metrics for the ripple and FR band are separated like so: ripple_metric / FR_metric

new data source, thus adapting to the clinical recording properties, such as different spike-to-noise ratios.

All in all, the event-driven detectors provide valuable classification metrics, rivaling existing deep learning implementations. Therefore, deploying such algorithms on neuromorphic designs, which bring significant energy efficiency gains and enable real-time operation, seems to be a promising solution. This practical study demonstrates the feasibility of building such a system on Loihi, focusing on the software implementation side and preparing for the transition to a hardware solution.

Clinical Dataset		Precision	Recall	F1-Score
DL-Models	CNN	55.78% / 48.78%	51.63% / 53.90%	50.30% / 45.50%
	LSTM	51.28% / 45.05%	68.55% / 58.30%	55.30% / 48.30%
SNN		83.04% / 87.45%	73.33% / 46.72%	77.65% / 53.96%

TABLE 5.15 – Comparison of the Classification Metrics of deep-learning Detectors with our SNN in the ripple and FR bands (Clinical Dataset). Adapted from [3].
The metrics for the ripple and FR band are separated like so: ripple_metric / FR_metric

Chapter 6

Conclusion

6.1 Conclusion

This dissertation presents three use cases of neuromorphic technology applied to the neuroengineering and bioelectronic fields. As the first experimental work from the NCN Lab with Intel's neuromorphic research chip, the primary goals were to validate the system's ability to analyze neuronal activity and highlight the advantages of the design. We accomplished both objectives during this project. The *in vitro* practical study focused on showing the development process to build and deploy neuromorphic algorithms in Loihi and includes a thorough analysis of the system's performance and energy efficiency. While the *in vivo* experiment demonstrates how it is possible to build SNNs able to detect more complex patterns of activity, involving, for instance, creating a data pipeline and defining custom neuronal models in Lava.

As opposed to most existing works in the field, we had access to proprietary, high-quality data to test our detectors, namely the *in-vitro* neuronal cultures cultivated by our lab and the humans' recordings conducted through a partnership with *Hospital de Santo António*. It was, therefore, interesting to test the implemented models with actual data instead of simulated or synthetic sources. Moreover, since we treated this work as a proof of concept of the Loihi technology stack, special attention was given to documenting and structuring the project. We intend to use the outcomes of this exploratory work to showcase the relevance of the study and request access to a physical Loihi system where we could, for example, implement the hardware version of the system proposed in chapter 5.

One of the greatest advantages of neuromorphic architectures is the power efficiency it can deliver. A detailed analysis of the event-driven algorithms' performance is presented in this study, harnessing the benchmarking structure recommended by Intel Labs. Out of the profiled algorithms, the NB detector consumed the most energy per second, 0.4595 J/s, which is a noteworthy improvement over *von Neumann*-based solutions. Additionally, we showed how the dynamic power consumption only adds up to a small portion of the total power consumption, demonstrating the energy saved by adopting an event-driven design. Another vital property of embedded medical devices is low latency, which must obey strict requirements to support real-time operation. Our

tests indicate that a neuromorphic setup based on Loihi can receive input and reply to it in the order of tens of microseconds, which fulfills most real-time requirements and exceeds the latency of existing non-neuromorphic bioelectronics. All in all, the benchmarking results are a testament to the potential of neuromorphic computing in systems that require low power and real-time operation.

While it is important to develop energy-efficient designs, it would be in vain if the model loses significant predictive capabilities. The classification metrics of the three detected biomarkers are presented in this dissertation and compared to state-of-the-art alternatives. We observed that the neuromorphic detectors were able to classify events in real-time with a similar performance to existing solutions, even surpassing them in some cases. The spatiotemporal-coding nature of SNNs makes them a compelling architecture to process real-time data because the inner state of the *in silico* neuronal population models the time-varying nature of sensory inputs. The membrane potential and synaptic current dynamics were visualized during the practical studies to understand how the classifiers detect relevant patterns, making the SNN more understandable.

In summary, we fulfilled the objectives of this study by developing event-driven algorithms to detect channel bursts, network bursts, and high-frequency oscillations in neuromorphic hardware. We compared our models with traditional alternatives, highlighting the promising future of neuromorphic computing applied to real-time monitoring systems.

6.2 Future Work

Due to the exploratory nature of the work, an immeasurable amount of model configurations and architectures could be tested, so we had to devote our time to the most promising solutions given the scope of the project. To the best of our knowledge, this dissertation examines the use of Intel's neuromorphic chip to analyze electrophysiology biomarkers for the first time, so many research directions can be followed to further investigate the application of neuromorphic technologies for neuronal activity analysis.

First, to transition the HFO detector to a solution running strictly on hardware, the custom-built neuron models have to be extended to include a version that can be deployed to Loihi's hardware. This requires having access to the micro-code documentation from Loihi which is only given upon request to INRC members for privacy reasons. In addition to gaining access to a physical Loihi device, to implement the hardware HFO detector, a suitable online processing system must be configured, starting with the sEEG signal acquisition, going through filtering and signal-to-spike conversion (ADM algorithm), which is then fed to the SNN located in the neuromorphic chip. A suggestion for the pre-processing stages' hardware counterpart is presented in the designated sections. In turn, the neural processing unit will react in real-time to HFOs. Looking further in the future, this could be used for monitoring disease severity or even developing neuromorphic online closed-loop brain-machine interfaces.

Staying on the register of a closed-loop system, which refers to a device that automatically regulates its state during operation, this property could be added to our HFO detector through

a local learning rule. Adding a synaptic plasticity mechanism into the SNN, such as spike-timing-dependent plasticity (STDP), would allow the network to modify its parameters in response to incoming data, thus adapting to different recording conditions or new activity patterns autonomously. For this, an additional layer could be added to our SNN, replacing the Feature Neuron Classification step that manually performs the selection of relevant neurons. This final layer would fire whenever a ripple or fast ripple occurred, and the weights of the artificial synapses connecting to the feature neurons would be updated continuously via the STDP rule.

On a final note regarding the HFO detection, the clinical dataset's dimension was very limited, which removes the possibility of fine-tuning the SNN model according to the clinical environment. Therefore, building a larger clinical dataset where an additional validation phase could be performed is another auspicious direction. It would also be interesting to test the model's performance on non-invasive scalp EEG instead of iEEG.

Naturally, many other neuromorphic computing applications are yet to be explored in the field of neuroengineering and bioelectronic medicine. Following a similar approach to this dissertation, relevant biomarkers can be tracked with highly efficient event-driven algorithms running on neuromorphic hardware, creating useful technologies to monitor our health in real-time or track disease progression.

References

[1] A. A. Fomani, R. Mansour, C. M. Florez-Quenguan, e P. Carlen. Development and characterization of multisite three-dimensional microprobes for deep brain stimulation and recording. *Journal of Microelectromechanical Systems*, 20:1109–1118, 2011.

[2] Alim Louis Benabid. Deep brain stimulation for parkinson's disease. *Curr. Opin. Neurobiol.*, 2003.

[3] Ana Sofia Moreira. A novel approach for identifying epileptogenic zones: Deep learning-based hfo biomarkers in seeg signals, 2023.

[4] Anatol Bragin, Jerome Engel, Charles L. Wilson, Itzhak Fried, e Gary W. Mathern. Hippocampal and entorhinal cortex high-frequency oscillations (100–500 hz) in human epileptic brain and in kainic acid-treated rats with chronic seizures. *Epilepsia*, 40:127–137, 2 1999.

[5] Anca Mihaela Vasilica, Vladimir Litvak, Chunyan Cao, Matthew Walker, e Umesh Vivekananda. Detection of pathological high-frequency oscillations in refractory epilepsy patients undergoing simultaneous stereo-electroencephalography and magnetoencephalography. *Seizure*, 107:81–90, 4 2023.

[6] Andrew B Gardner, Greg A Worrell, Eric Marsh, Dennis Dlugos, e Brian Litt. Human and automated detection of high-frequency oscillations in clinical intracranial eeg recordings, 2007.

[7] Artur Vetkas, Anton Fomenko, Jürgen Germann, Can Sarica, Christian Iorio-Morin, Nardin Samuel, Kazuaki Yamamoto, Vanessa Milano, Cletus Cheyuo, Ajmal Zemmar, Gavin Elias, Alexandre Boutet, Aaron Loh, Brendan Santyr, Dave Gwun, Jordy Tasserie, Suneil K Kalia, e Andres M. Lozano. Deep brain stimulation targets in epilepsy: Systematic review and meta-analysis of anterior and centromedian thalamic nuclei and hippocampus. *Epilepsia*, 63:513–524, 3 2022.

[8] Ben Goertzel. Artificial general intelligence: Concept, state of the art, and future prospects. *Journal of Artificial General Intelligence*, 5:1–48, 12 2014.

[9] Benoıit Benoit, Cré Pon, Vincent Navarro, Dominique Hasboun, Sté Phane Clemenceau, Jacques Martinerie, Michel Baulac, Claude Adam, e Michel Le Van Quyen. Mapping interictal oscillations greater than 200 hz recorded with intracranial macroelectrodes in human epilepsy. *A JOURNAL OF NEUROLOGY*, 2010.

[10] Birgit Frauscher, Fabrice Bartolomei, Katsuhiro Kobayashi, Jan Cimbalnik, Maryse A. van 't Klooster, Stefan Rampp, Hiroshi Otsubo, Yvonne Höller, Joyce Y. Wu, Eishi Asano, Jerome Engel, Philippe Kahane, Julia Jacobs, e Jean Gotman. High-frequency oscillations: The state of clinical research. *Epilepsia*, 58:1316–1329, 8 2017.

[11] Bodo Rueckauer, Iulia Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, e Shih Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 11:294078, 12 2017.

[12] Bohte Sander, Kok Joost, e Poutré Han. Spikeprop: Backpropagation for networks of spiking neurons. *ESANN 2000*, 2000.

[13] Bokeh Development Team. Bokeh: Python library for interactive visualization. https://bokeh.pydata.org/en/latest/, 2018.

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

[15] Catarina Dias, Domingos Castro, Miguel Aroso, João Ventura, e Paulo Aguiar. Memristor-based neuromodulation device for real-time monitoring and adaptive control of neuronal populations. *ACS Applied Electronic Materials*, 4:2380–2387, 5 2022.

[16] Chao Li, Xumeng Zhang, Pei Chen, Keji Zhou, Jie Yu, Guangjian Wu, Du Xiang, Hao Jiang, Ming Wang, e Qi Liu. Short-term synaptic plasticity in emerging devices for neuromorphic computing. *iScience*, 2023.

[17] Y. Chen, H. Zhang, H. Wang, L. Yu, e Y. Chen. The role of coincidence-detector neurons in the reliability and precision of subthreshold signal detection in noise. *PloS one*, 2013.

[18] Da Li, Xinbo Chen, Michela Becchi, e Ziliang Zong. Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. páginas 477–484. Institute of Electrical and Electronics Engineers Inc., 10 2016.

[19] Dan Goodman. Cosyne tutorial 2022 - spiking neural networks. https://neural-reckoning.github.io/cosyne-tutorial-2022/, 2022.

[20] Dan Goodman e Marcus Ghosh. Neuroscience for machine learners. https://doi.org/10.5281/zenodo.10366802, 2023.

[21] Daniel Auge, Julian Hille, Etienne Mueller, e Alois Knoll. A survey of encoding techniques for signal processing in spiking neural networks. *Neural Processing Letters*, 53:4693–4710, 12 2021.

[22] David Sterratt. *Principles of computational modelling in neuroscience*. Cambridge University Press, 2011.

[23] David T. Brocker, Brandon D. Swan, Dennis A. Turner, Robert E. Gross, Stephen B. Tatter, Mandy Miller Koop, Helen Bronte-Stewart, e Warren M. Grill. Improved efficacy of temporally non-regular deep brain stimulation in parkinson's disease. *Experimental Neurology*, 239:60–67, 1 2013.

[24] Domingos Leite de Castro, Miguel Aroso, A. Pedro Aguiar, David B. Grayden, e Paulo Aguiar. Disrupting abnormal neuronal oscillations with adaptive delayed feedback control. *eLife*, 13, 3 2024.

[25] Elena Urrestarazu, Jeffrey D. Jirsch, Pierre LeVan, Jeffery Hall, e J. Gotman. High-frequency intracerebral eeg activity (100–500 hz) following interictal spikes. *Epilepsia*, 47:1465–1476, 9 2006.

[26] Elisa Donati e Giacomo Indiveri. Silicon neuron with programmable ion channel kinematics for bioelectronic applications. Institute of Electrical and Electronics Engineers Inc., 2021.

[27] Elisa Donati e Giacomo Indiveri. Neuromorphic bioelectronic medicine for nervous system interfaces: from neural computational primitives to medical applications. *Progress in Biomedical Engineering*, 5, 1 2023.

[28] Emre O. Neftci, Hesham Mostafa, e Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36:51–63, 11 2019.

[29] Enrico Opri, Stephanie Cernera, Rene Molina, Robert S Eisinger, Jackson N Cagle, Leonardo Almeida, Timothy Denison, Michael S Okun, Kelly D Foote, e Aysegul Gunduz. Chronic embedded cortico-thalamic closed-loop deep brain stimulation for the treatment of essential tremor. *Sci. Transl. Med*, 12:7680, 2020.

[30] Eugene M. Izhikevich. Simple model of spiking neurons, 11 2003.

[31] Eugene M. Izhikevich. Solving the distal reward problem through linkage of stdp and dopamine signaling. *Cerebral cortex (New York, N.Y. : 1991)*, 17:2443–2452, 10 2007.

[32] Eugene M. Izhikevich. *Dynamical systems in neuroscience: the geometry of excitability and bursting*. The MIT Press, 2010.

[33] Andrzej Ponulak Filip;Kasiński. View of introduction to spiking neural networks: Information processing, learning and applications | acta neurobiologiae experimentalis, 2011.

[34] Fourcaud-Trocmé N, Hansel D, van Vreeswijk C, e Brunel N. How spike generation mechanisms determine the neuronal response to fluctuating inputs. *J Neurosci.*, 2003.

[35] Frank R. Freemon. Galen's ideas on neurological function. *Journal of the history of the neurosciences*, 3:263–271, 10 1994.

[36] Freepik. Images by freepik. https://www.freepik.com/.

[37] Garrick Orchard, Ajinkya Jayawant, Gregory Cohen, e Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades benchmarks and challenges for neuromorphic engineering. 2015.

[38] Geon Hwee Kim, Kanghyun Kim, Eunji Lee, Taechang An, Woo Seok Choi, Geunbae Lim, e Jung Hwal Shin. Recent progress on microelectrodes in neural interfaces, 10 2018.

[39] W. Gerstner e R. Brette. Adaptive exponential integrate-and-fire model. *Scholarpedia*, 4(6):8427, 2009. revision #90944.

[40] 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, Johannes Schemmel, Gert Cauwenberghs, John Arthur, Kai Hynna, Fopefolu Folowosele, Sylvain Saighi, Teresa Serrano-Gotarredona, Jayawan Wijekoon, Yingxue Wang, e Kwabena Boahen. Neuromorphic silicon neuron circuits, 2011.

[41] Ginny, Chiranjeev Kumar, e Kshirasagar Naik. Smartphone processor architecture, operations, and functions: current state-of-the-art and future outlook: energy performance tradeoff: Energy–performance trade-off for smartphone processors. *Journal of Supercomputing*, 77:1377–1454, 2 2021.

[42] Hector Garcia Rodriguez, Qinghai Guo, e Timoleon Moraitis. Short-term plasticity neurons learning to learn and forget. 2022.

[43] Intel. Technology brief | taking neuromorphic computing to the next level with loihi 2, 2021.

[44] Ioana Sporea e André Grüning. Supervised learning in multilayer spiking neural networks. *Neural Computation*, 25:473–509, 2 2012.

[45] J. C. Mateus, C. D.F. Lopes, M. Aroso, A. R. Costa, A. Gerós, J. Meneses, P. Faria, E. Neto, M. Lamghari, M. M. Sousa, e P. Aguiar. Bidirectional flow of action potentials in axons drives activity dynamics in neuronal cultures. *Journal of Neural Engineering*, 18, 12 2021.

[46] J. C. de Oliveira, B. M.B. Drabowski, S. M.A.F. Rodrigues, R. M. Maciel, M. F.D. Moraes, e V. R. Cota. Seizure suppression by asynchronous non-periodic electrical stimulation of the amygdala is partially mediated by indirect desynchronization from nucleus accumbens. *Epilepsy Research*, 154:107–115, 8 2019.

[47] J. Jacobs, R. Staba, E. Asano, H. Otsubo, J. Y. Wu, M. Zijlmans, I. Mohamed, P. Kahane, F. Dubeau, V. Navarro, e J. Gotman. High-frequency oscillations (hfos) in clinical epilepsy. *Progress in Neurobiology*, 98:302–315, 9 2012.

[48] Jia Qin Yang, Ruopeng Wang, Yi Ren, Jing Yu Mao, Zhan Peng Wang, Ye Zhou, e Su Ting Han. Neuromorphic engineering: From biological to spike-based hardware nervous systems. *Advanced Materials*, 32, 12 2020.

[49] Jie Yang, Timothée Masquelier, Feng Xu, e Sijia Lu. Linear leaky-integrate-and-fire neuron model based spiking neural networks and its mapping relationship to deep neural networks, 2012.

[50] Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, e Sebastian Millner. *A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling*. IEEE, 2010.

[51] Jolivet R., Schürmann F., Berger T.K., Naud R., Gerstner W., e Roth A. The quantitative single-neuron modeling competition. *Biological Cybernetics*, 2008.

[52] Julia Jacobs e Maeike Zijlmans. Hfo to measure seizure propensity and improve prognostication in patients with epilepsy. *Epilepsy Currents*, 20:338–347, 11 2020.

[53] Karla Burelo, Mohammadali Sharifshazileh, Giacomo Indiveri, e Johannes Sarnthein. Automatic detection of high-frequency oscillations with neuromorphic spiking neural networks, 6 2022.

[54] Kashu Yamazaki, Viet Khoa Vo-Ho, Darshan Bulsara, e Ngan Le. Spiking neural networks and their applications: A review, 7 2022.

[55] Kavyakantha Remakanthakurup Sindhu, Richard Staba, e Beth A. Lopour. Trends in the use of automated algorithms for the detection of high-frequency oscillations associated with human epilepsy. *Epilepsia*, 61:1553–1569, 8 2020.

[56] Kent H. Redford, William Adams, Rob Carlson, Georgina M. Mace, e Bertina Ceccarelli. Synthetic biology and the conservation of biodiversity. *ORYX*, 48:330–336, 2014.

[57] Koji Iida e Hiroshi Otsubo. Stereoelectroencephalography: Indication and efficacy. *Neurologia medico-chirurgica*, 57:375, 2017.

[58] Konstantine P. Panegyres e Peter K. Panegyres. The ancient greek discovery of the nervous system: Alcmaeon, praxagoras and herophilus. *Journal of Clinical Neuroscience*, 29:21–24, 7 2016.

[59] L. Wasser. Hierarchical data formats - what is hdf5? https://www.neonscience.org/resources/learning-hub/tutorials/about-hdf5, 2020.

[60] Lava Development Team. Lava software framework — lava documentation. https://lava-nc.org/.

[61] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29:141–142, 2012.

[62] Marie Engelene J. Obien, Kosmas Deligkaris, Torsten Bullmann, Douglas J. Bakkum, e Urs Frey. Revealing neuronal function through microelectrode array recordings, 2015.

[63] A Martín-Araguz, C Bustamante-Martínez, MT Emam-Mansour, e JM Moreno-Martínez. Neuroscience in ancient egypt and in the school of alexandria. *Rev Neurol.*, 2002.

[64] Maxence Bouvier, Alexandre Valentian, Thomas Mesquida, Francois Rummens, Marina Reyboz, Elisa Vianello, e Edith Beigne. Spiking neural networks hardware implementations and challenges: A survey, 4 2019.

[65] J. Mesquita. Fork from intel lava deep-learning repository with dissertation code. https://github.com/monkin77/monkin-lava-dl/tree/feat/hfo_snn, 2024.

[66] J. Mesquita. Fork from intel lava repository with dissertation code. https://github.com/monkin77/thesis-lava, 2024.

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

[68] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steve Mccoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, e Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning, 2018.

[69] Mohammadali Sharifshazileh, Karla Burelo, Johannes Sarnthein, e Giacomo Indiveri. An electronic neuromorphic system for real-time detection of high frequency oscillations (hfo) in intracranial eeg. *Nature Communications*, 12, 12 2021.

[70] Nader Sehatbakhsh, Monjur Alam, Nazari Alireza, Alenka Zajic, e Milos Prvulovic. *Syndrome: Spectral Analysis for Anomaly Detection on Medical IoT and Embedded Devices*. 2018.

[71] Nicolas Roehri, Francesca Pizzo, Fabrice Bartolomei, Fabrice Wendling, e Christian George Bénar. What are the assets and weaknesses of hfo detectors? a benchmark framework based on realistic simulations. *PLOS ONE*, 12:e0174702, 4 2017.

[72] Nicolas Roehri, Jean-Marc Lina, John C Mosher, Fabrice Bartolomei, e Christian-George Bénar. Time-frequency strategies for increasing high frequency oscillation detectability in intracerebral eeg hhs public access. *IEEE Trans Biomed Eng*, 63:2595–2606, 2016.

[73] Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih Chii Liu, e Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. *Proceedings of the International Joint Conference on Neural Networks*, 2015-September, 9 2015.

[74] Peter U. Diehl e Matthew Cook. Efficient implementation of stdp rules on spinnaker neuromorphic hardware. *Proceedings of the International Joint Conference on Neural Networks*, páginas 4288–4295, 9 2014.

[75] R Zelmann, F Mari, J Jacobs, M Zijlmans, R Chander, e J Gotman. Automatic detector of high frequency oscillations for human recordings with macroelectrodes. 2010.

[76] R. G. Stockwell. Localization of the complex spectrum: the s transform. *IEEE Transactions on Signal Processing*, 44:993, 1996.

[77] R. J. Staba, C. L. Wilson, Anatol Bragin, e Itzhak Fried. Quantitative analysis of high-frequency oscillations (80-500 hz) recorded in human epileptic hippocampus and entorhinal cortex. *Journal of Neurophysiology*, 88:1743–1752, 2002.

[78] R. Kwiatkowski. Gradient descent algorithm — a deep dive. https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21, 2021.

[79] R. S. Zucker. Short-term synaptic plasticity. *Annual review of neuroscience*, 12:13–31, 1989.

[80] Recep Buğra Uludağ, Serhat Çağdaş, Yavuz Selim İşler, Neslihan Serap Şengör, e Ismail Akturk. Bio-realistic neural network implementation on loihi 2 with izhikevich neurons. 7 2023.

[81] Robert A. Nawrocki, Richard M. Voyles, e Sean E. Shaheen. A mini review of neuromorphic architectures and implementations, 10 2016.

[82] Rodrigo Miguel Guerra Da Mota e Almeida Azevedo. Address-event based communication between spiking neural networks (snn) computing cores, 2023.

[83] Romain Brette e Wulfram Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of Neurophysiology*, 94:3637–3642, 11 2005.

[84] Sanjeet S. Grewal, Mohammed Ali Alvi, William J. Perkins, Gregory D. Cascino, Jeffrey W. Britton, David B. Burkholder, Elson So, Cheolsu Shin, Richard W. Marsh, Fredric B. Meyer, Gregory A. Worrell, e Jamie J. Van Gompel. Reassessing the impact of intraoperative electrocorticography on postoperative outcome of patients undergoing standard temporal lobectomy for mri-negative temporal lobe epilepsy. *Journal of Neurosurgery*, 132:605–614, 2020.

[85] Sen Song, Kenneth D. Miller, e L. F. Abbott. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature neuroscience*, 3:919–926, 9 2000.

[86] Shanlin Xiao, Wei Liu, Yuhao Guo, e Zhiyi Yu. Low-cost adaptive exponential integrate-and-fire neuron using stochastic computing. *IEEE Transactions on Biomedical Circuits and Systems*, 14:942–950, 10 2020.

[87] Simon Jonathan Thorpe. Spike arrival times: A highly efficient coding scheme for neural networks, 1990.

[88] Srinjoy Mitra, Stefano Fusi, e Giacomo Indiveri. Real-time classification of complex patterns using spike-based learning in neuromorphic vlsi. *IEEE Transactions on Biomedical Circuits and Systems*, 3:32–42, 2009.

[89] Sumit Bam Shrestha e Garrick Orchard. Slayer: Spike layer error reassignment in time, 2018.

[90] Tao Luo, Xuan Wang, Chuping Qu, Matthew Kay Fei Lee, Wai Teng Tang, Weng Fai Wong, e Rick Siow Mong Goh. An fpga-based hardware emulator for neuromorphic chip with rram. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39:438–450, 2 2020.

[91] Tasbolat Taunyazov, Weicong Sng, Hian Hian See, Brian Lim, Jethro Kuan, Abdul Fatir Ansari, Benjamin CK Tee, e Harold Soh. Robotics: Science and systems ssss corvalis, oregon event-driven visual-tactile sensing and learning for robots. 2020.

[92] Teppei Araki, Lukas M. Bongartz, Taro Kaiju, Ashuya Takemoto, Shuichi Tsuruta, Takafumi Uemura, e Tsuyoshi Sekitani. Flexible neural interfaces for brain implants-the pursuit of thinness and high density, 10 2020.

[93] Thomas N. Theis e H. S. Philip Wong. The end of moore's law: A new beginning for information technology. *Computing in Science and Engineering*, 19:41–50, 3 2017.

[94] Tim P. Vogels e L. F. Abbott. Signal propagation and logic gating in networks of integrate-and-fire neurons. *Journal of Neuroscience*, 25:10786–10795, 11 2005.

[95] Travis DeWolf, Kinjal Patel, Pawel Jaworski, Roxana Leontie, Joe Hays, e Chris Eliasmith. Neuromorphic control of a simulated 7-dof arm using loihi. 2023.

[96] Vasileios Dimakopoulos, Pierre MCrossed D sign©gevand, Ece Boran, Shahan Momjian, Margitta Seeck, Serge VulliCrossed D sign©moz, e Johannes Sarnthein. Blinded study: prospectively defined high-frequency oscillations predict seizure outcome in individual patients. *Brain communications*, 3, 2021.

[97] Weiran Cai, Frank Ellinger, e Ronald Tetzlaff. Neuronal synapse as a memristor: Modeling pair- and triplet-based stdp rule. *IEEE Transactions on Biomedical Circuits and Systems*, 9:87–95, 2 2015.

[98] William A. Catterall, Indira M. Raman, Hugh P.C. Robinson, Terrence J. Sejnowski, e Ole Paulsen. The hodgkin-huxley heritage: From channels to circuits. *Journal of Neuroscience*, 32:14064–14073, 10 2012.

[99] World Health Organization. *Epilepsy: a public health imperative*. World Health Organization, 2019.

[100] Xueyuan She, Saurabh Dash, Daehyun Kim, e Saibal Mukhopadhyay. A heterogeneous spiking neural network for unsupervised learning of spatiotemporal patterns. *Frontiers in Neuroscience*, 14, 1 2021.

[101] Yongqiang Cao, Yang Chen, e Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113:54–66, 5 2015.

[102] Yuying Fan, Liping Dong, Xueyan Liu, Hua Wang, e Yunhui Liu. Recent advances in the noninvasive detection of high-frequency oscillations in the human brain. *Reviews in the Neurosciences*, 32:305–321, 4 2021.

[103] Yuzhuang Xu, Xu Han, Zonghan Yang, Shuo Wang, Qingfu Zhu, Zhiyuan Liu, Weidong Liu, e Wanxiang Che. Onebit: Towards extremely low-bit large language models. 2 2024.

Appendix A

Additional Results of the *In Vitro* Study

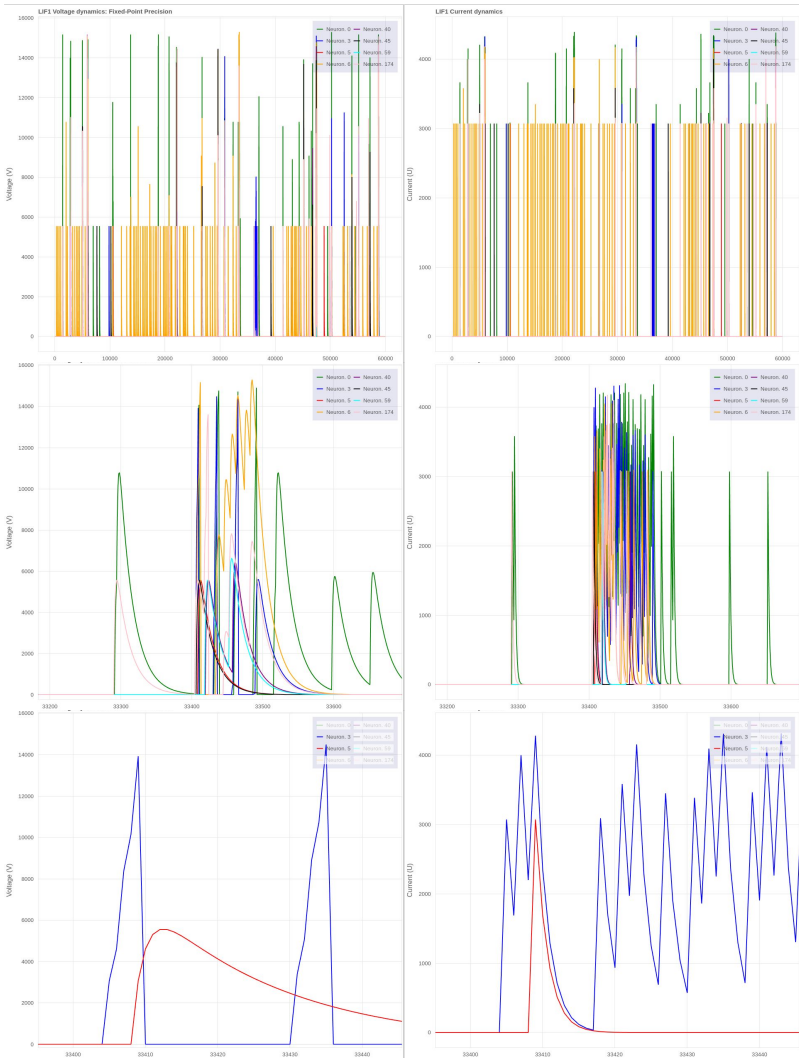


FIGURE A.1 – Voltage and Current Dynamics of the Channel Burst Detection at different time scales (fixed-point precision).

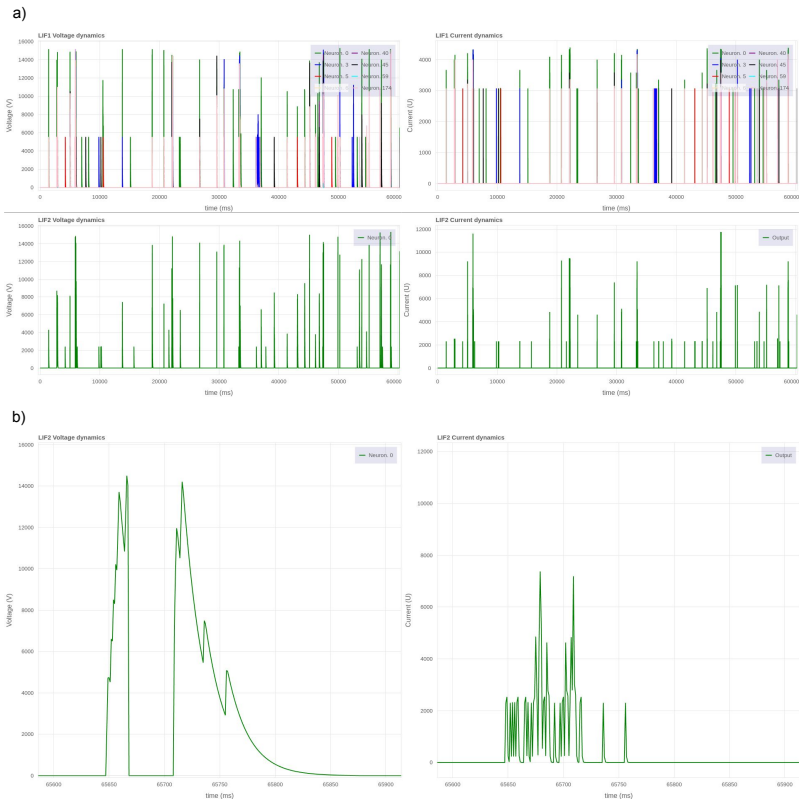


FIGURE A.2 – Voltage and Current Dynamics of the Network Burst Detection at different time scales (fixed-point precision).
a) Dynamics of the Middle and Output LIF layers in a big time window.
b) Detailed look into the membrane potential and current of the Output LIF neuron.

Appendix B

Additional Results of the *In Vivo* Study

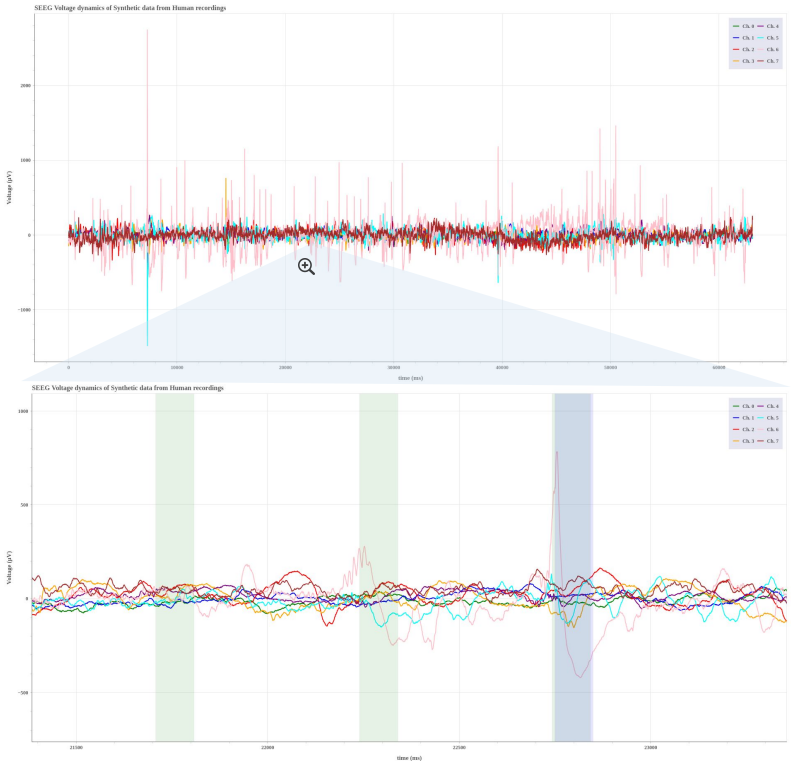
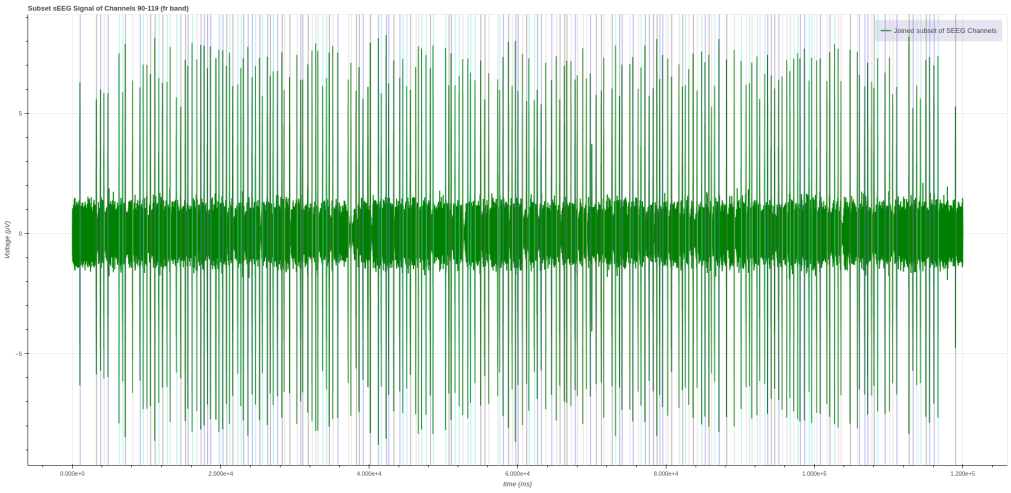
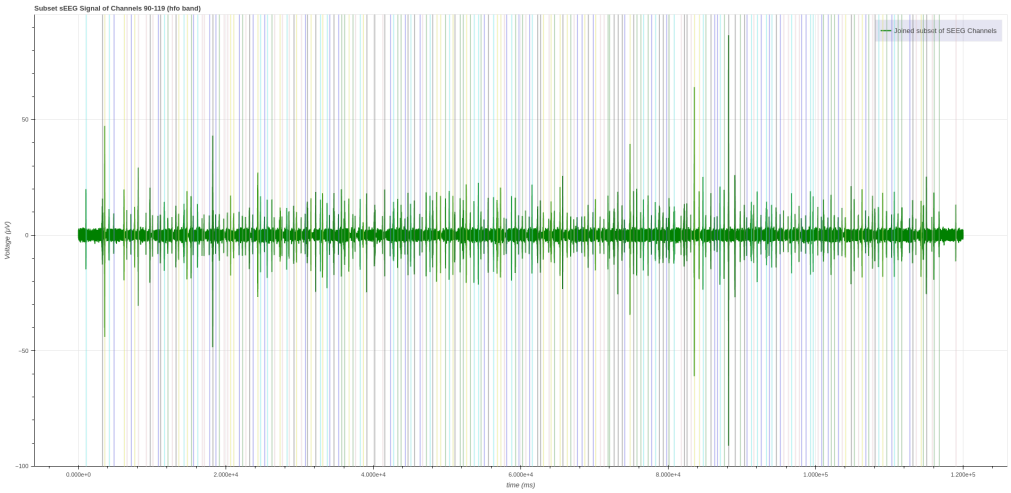


FIGURE B.1 – Line plot of the processed clinical sEEG data of a patient. The image on top shows the entire signal for 8 representative electrodes, while the bottom image zooms into a smaller time window with annotated ripples (green) and FR (blue).

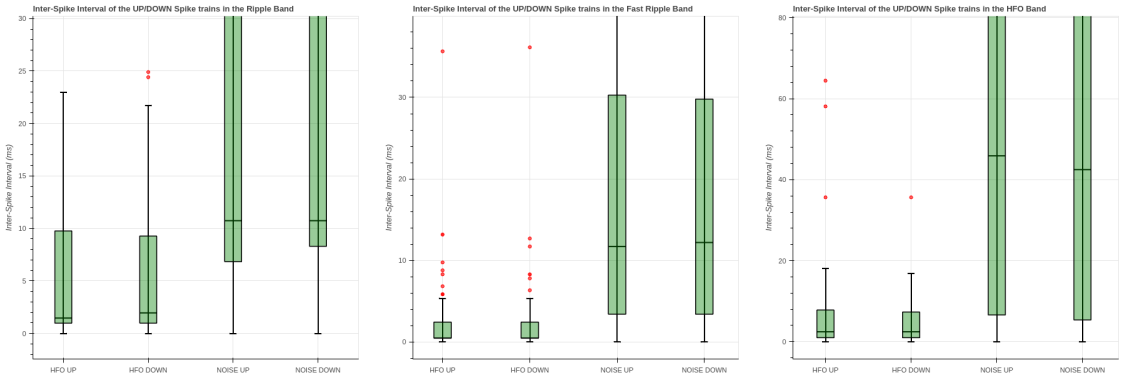


(A) Subset sEEG in the FR band (80-250Hz).



(B) Subset sEEG in the HFO band (80-500Hz).

FIGURE B.2 – sEEG signals of the created Subset in the remaining frequency bands.



(A) ISIs in the Ripple band (B) ISIs in the FR band (C) ISIs in the HFO band

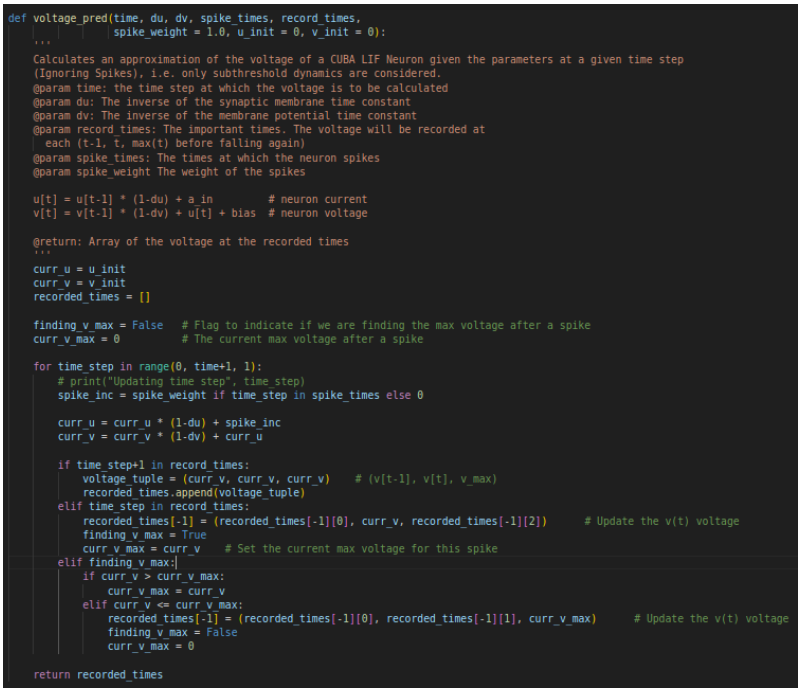
FIGURE B.3 – Boxplots of ISIs for each frequency band of channel 90 from the Synthetic Dataset. Each sub-figure has 4 boxes representing the distribution of the ISI for the UP train during HFO, DN train during HFO, UP train during Noise, and DN train during Noise, respectively.


```
{
    "freq_band": "ripple",
    "confidence_window": 180,
    "feature_neuron_class": {
        "array": [
            2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 1, 1, 0, 2, 2, 2, 1,
            2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 2, 1, 2, 0, 1, 0, 1, 2, 2, 2, 2, 2, 1, 0, 1,
            2, 1, 2, 1, 2, 2, 1, 1, 1, 2, 1, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 1, 0,
            2, 2, 1, 1, 1, 2, 2, 1, 2, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 1, 2, 0,
            0, 1, 1, 1, 1, 2, 2, 1, 0, 0, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 0, 2, 2, 2,
            2, 0, 2, 2, 2, 0, 2, 2, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 1, 2, 1, 1, 0, 2,
            2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1, 1, 0, 2, 1, 2, 2, 1, 2,
            1, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1, 1, 0, 1, 2, 1, 2, 2, 2, 2, 1, 1, 2, 2, 2,
            1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2,
            2, 0, 1, 2, 1, 2, 2, 1, 0, 1, 1, 1, 0, 1, 2, 1, 2, 1, 0, 1, 2, 2, 2, 2, 2,
            2, 2, 2, 1, 1, 2
        ],
        "counts": {
            "silent_neurons": 20,
            "noisy_neurons": 100,
            "detector_neurons": 136
        }
    },
    "num_consensus_neurons": 1,
    "relevant_ratio_threshold": 95.0,
    "metrics": {
        "true_positive": 192,
        "false_positive": 10,
        "true_negative": 119793,
        "false_negative": 5,
        "total_predictions": 120000,
        "accuracy": 99.9875,
        "precision": 95.04950495049505,
        "recall": 97.46192893401016,
        "f1_score": 96.24060150375941,
        "specificity": 99.99165296361527
    }
}
```


FIGURE B.4 – Example of JSON file containing the classification output of the HFO Detector.

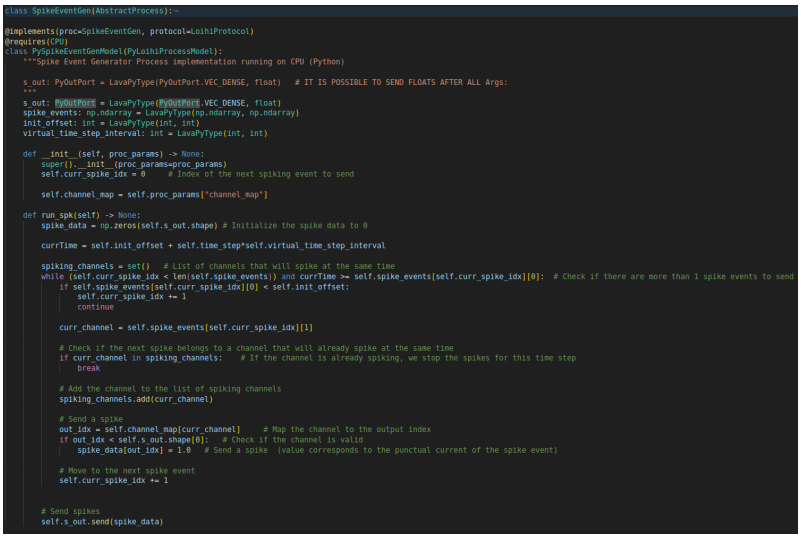
Appendix C

Relevant Code Snippets



```python
def voltage_pred(time, du, dv, spike_times, record_times,
                 spike_weight = 1.0, u_init = 0, v_init = 0):
    '''
    Calculates an approximation of the voltage of a CUBA LIF Neuron given the parameters at a given time step
    (Ignoring Spikes), i.e. only subthreshold dynamics are considered.
    @param time: the time step at which the voltage is to be calculated
    @param du: The inverse of the synaptic membrane time constant
    @param dv: The inverse of the membrane potential time constant
    @param record_times: The important times. The voltage will be recorded at
      each (t-1, t, max(t) before falling again)
    @param spike_times: The times at which the neuron spikes
    @param spike_weight The weight of the spikes

    u[t] = u[t-1] * (1-du) + a_in         # neuron current
    v[t] = v[t-1] * (1-dv) + u[t] + bias  # neuron voltage

    @return: Array of the voltage at the recorded times
    '''
    curr_u = u_init
    curr_v = v_init
    recorded_times = []

    finding_v_max = False   # Flag to indicate if we are finding the max voltage after a spike
    curr_v_max = 0          # The current max voltage after a spike

    for time_step in range(0, time+1, 1):
        # print("Updating time step", time_step)
        spike_inc = spike_weight if time_step in spike_times else 0

        curr_u = curr_u * (1-du) + spike_inc
        curr_v = curr_v * (1-dv) + curr_u

        if time_step+1 in record_times:
            voltage_tuple = (curr_v, curr_v, curr_v)    # (v[t-1], v[t], v_max)
            recorded_times.append(voltage_tuple)
        elif time_step in record_times:
            recorded_times[-1] = (recorded_times[-1][0], curr_v, recorded_times[-1][2])      # Update the v(t) voltage
            finding_v_max = True
            curr_v_max = curr_v    # Set the current max voltage for this spike
        elif finding_v_max:
            if curr_v > curr_v_max:
                curr_v_max = curr_v
            elif curr_v <= curr_v_max:
                recorded_times[-1] = (recorded_times[-1][0], recorded_times[-1][1], curr_v_max)      # Update the v(t) voltage
                finding_v_max = False
                curr_v_max = 0

    return recorded_times
```

FIGURE C.1 – Python utility that models the CUBA LIF neuron subthreshold dynamics.



```python
class SpikeEventGen(AbstractProcess):-

@implements(proc=SpikeEventGen, protocol=LoihiProtocol)
@requires(CPU)
class PySpikeEventGenModel(PyLoihiProcessModel):
    """Spike Event Generator Process implementation running on CPU (Python)

    s_out: PyOutPort = LavaPyType(PyOutPort.VEC_DENSE, float)   # IT IS POSSIBLE TO SEND FLOATS AFTER ALL Args:
    """
    s_out: PyOutPort = LavaPyType(PyOutPort.VEC_DENSE, float)
    spike_events: np.ndarray = LavaPyType(np.ndarray, np.ndarray)
    init_offset: int = LavaPyType(int, int)
    virtual_time_step_interval: int = LavaPyType(int, int)

    def __init__(self, proc_params) -> None:
        super().__init__(proc_params=proc_params)
        self.curr_spike_idx = 0    # Index of the next spiking event to send

        self.channel_map = self.proc_params["channel_map"]

    def run_spk(self) -> None:
        spike_data = np.zeros(self.s_out.shape) # Initialize the spike data to 0

        currTime = self.init_offset + self.time_step*self.virtual_time_step_interval

        spiking_channels = set()  # List of channels that will spike at the same time
        while (self.curr_spike_idx < len(self.spike_events)) and currTime >= self.spike_events[self.curr_spike_idx][0]:  # Check if there are more than 1 spike events to send
            if self.spike_events[self.curr_spike_idx][0] < self.init_offset:
                self.curr_spike_idx += 1
                continue

            curr_channel = self.spike_events[self.curr_spike_idx][1]

            # Check if the next spike belongs to a channel that will already spike at the same time
            if curr_channel in spiking_channels:   # If the channel is already spiking, we stop the spikes for this time step
                break

            # Add the channel to the list of spiking channels
            spiking_channels.add(curr_channel)

            # Send a spike
            out_idx = self.channel_map[curr_channel]     # Map the channel to the output index
            if out_idx < self.s_out.shape[0]:  # Check if the channel is valid
                spike_data[out_idx] = 1.0  # Send a spike  (value corresponds to the punctual current of the spike event)

            # Move to the next spike event
            self.curr_spike_idx += 1


        # Send spikes
        self.s_out.send(spike_data)
```

FIGURE C.2 – Lava Process responsible for feeding the neuronal input to the remaining SNN.

```python
def signal_to_spike(params: SignalToSpikeParameters):
    """
    This functions retuns two spike trains, when the signal crosses the specified threshold in
    a rising direction (UP spikes) and when it crosses the specified threshold in a falling
    direction (DOWN spikes)
    @times (array): Array containing the time points of the signal
    @signal (array): Amplitude of the signal at each time point
    @interpolation_factor (int): upsampling factor, new sampling frequency
    @thr_up (float): threshold crossing in a rising direction
    @thr_dn (float): threshold crossing in a falling direction
    @refractory_period (float): period in which no spike will be generated [same units as time vector]

    Returns:
    - spikes_up (array): spike train for threshold crossing in a rising direction
    - spikes_dn (array): spike train for threshold crossing in a falling direction
    """
    # This variable keeps track of the last recorded membrane potential where a significant oscillatory behavior was detected
    instant_dc = 0   # Initialize the instant DC (Direct Current) variable to 0. (Amplitude at time t)

    # Initialize the spike trains for UP and DOWN spikes
    spikes_up = []
    spikes_dn = []

    # Define variables for the signal and time points
    times = params.times
    amplitudes = params.signal

    idx = 0
    while idx < len(times):
        # If the signal increased in amplitude more than the THRESHOLD_UP value (since the last update to instant_dc), detect an UP spike.
        if amplitudes[idx] - instant_dc > params.threshold_up:
            spikes_up.append(times[idx])    # Add the spike time to the UP spike train

            instant_dc = amplitudes[idx]     # Update the current amplitude to the new value

            idx = skip_refractory_period(times, idx, params.refractory_period)    # Skip the refractory period
        # If the signal decreased in amplitude more than the THRESHOLD_DOWN value (since the last update to instant_dc), detect a DOWN spike.
        elif amplitudes[idx] - instant_dc < params.threshold_down:
            spikes_dn.append(times[idx])    # Add the spike time to the DOWN spike train

            instant_dc = amplitudes[idx]    # Update the current amplitude to the new value

            idx = skip_refractory_period(times, idx, params.refractory_period)    # Skip the refractory period
        else:
            # Move to the next time point without updating the instant_dc (no significant oscillatory behavior detected)
            idx += 1

    return SpikeTrains(up=np.array(spikes_up), down=np.array(spikes_dn))
```

FIGURE C.3 – Signal-to-Spike conversion algorithm.

```python
@implements(proc=ConfigTimeConstantsRefractoryLIF, protocol=LoihiProtocol)
@requires(CPU)
@tag("floating_pt")
class PyConfigTimeConstantsRefractoryLifFloat(AbstractPyConfigTimeConstantsLifModelFloat):
    """Implementation of Configurable time constants Refractory Leaky-Integrate-and-Fire neural process in floating  point precision."""
    s_out: PyOutPort = LavaPyType(PyOutPort.VEC_DENSE, float)
    vth: float = LavaPyType(float, float)
    refractory_period_end: np.ndarray = LavaPyType(np.ndarray, int)

    def __init__(self, proc_params):
        super(PyConfigTimeConstantsRefractoryLifFloat, self).__init__(proc_params)
        self.refractory_period = proc_params["refractory_period"]

    def spiking_activation(self):
        return self.v > self.vth

    def subthr_dynamics(self, activation_in: np.ndarray):
        """Sub-threshold dynamics of current and voltage variables for ConfigTimeConstantsLIF. This is where the 'leaky integration' happens."""
        # Get the excitatory input from a_in -- Positive values increase u_exc
        exc_a_in = np.clip(activation_in, a_min=0, a_max=None)
        # Get the inhibitory input from a_in -- Negative values increase u_inh
        inh_a_in = np.clip(activation_in, a_min=None, a_max=0)

        # Update the excitatory and inhibitory currents
        self.u_exc[:] = self.u_exc * (1 - self.du_exc)
        self.u_exc[:] += exc_a_in

        self.u_inh[:] = self.u_inh * (1 - self.du_inh)
        self.u_inh[:] += inh_a_in

        # Check which neurons are not in refractory period
        non_refractory = self.refractory_period_end < self.time_step

        # Calculate the net current by adding the excitatory and inhibitory currents
        self.u = self.u_exc + self.u_inh   # u_inh is negative

        # Update the voltage of the non-refractory neurons
        self.v[non_refractory] = self.v[non_refractory] * (1 - self.dv[non_refractory]) + (
            self.u[non_refractory] + self.bias_mant[non_refractory])

    def process_spikes(self, spike_vector: np.ndarray):
        """Set the refractory_period_end for the neurons that spiked and Reset the voltage of the neurons that spiked to 0"""
        self.refractory_period_end[spike_vector] = (self.time_step + self.refractory_period)
        super().reset_voltage(spike_vector)

    def run_spk(self):
        """The run function that performs the actual computation during execution orchestrated by a PyLoihiProcessModel using the LoihiProtocol."""
        a_in_data = self.a_in.recv()
        self.subthr_dynamics(activation_in=a_in_data)
        spike_vector = self.spiking_activation()

        self.process_spikes(spike_vector=spike_vector)    # Reset voltage of spiked neurons to 0 and update refractory period
        self.s_out.send(spike_vector)
```

FIGURE C.4 – *ConfigTimeConstantsRefractoryLIF* neuron model in Lava.