Real-Time Detection and Classification of Purkinje-Cell Neural Activity

D.A. Vrijenhoek

Real-Time Detection and Classification of Purkinje-Cell Neural Activity

by

D.A. Vrijenhoek

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday July 5th, 2023 at 01:00 PM.

Student number:	5193133	
Project duration:	September 1, 2021 – July 5, 2023	
Thesis committee:	Prof. dr. ir. Said Hamdioui,	TU Delft, supervisor
	Dr. ir. Dante Muratore,	TU Delft
	Dr. ir. Christos Strydis,	TU Delft
	Dr. ir. Anteneh Gebregiorgis,	TU Delft
	Dr. ir. Muhammad Ali Siddiqi,	TU Delft

TU Delft

Abstract

Purkinje cell is a type of neuron that can be found in the *cerebellum*. What characterises Purkinje cell neural activity is the fact that it exhibits two types of spiking behaviour; the so-called *simple* and *complex* spikes. These two types of spikes are thought to play a role in motor functionality. In order to better understand the relationship between Purkinje cell neural activity and the *motor-cortex*, neuroscientists record such neural activity in mice. However, current experimental setups pose a challenge as they involve a wired connection between the animal's head stage and the recording device, which limits the mouse's natural behaviour by restricting its movement. This work proposes a lightweight neural-spike detection and classification architecture for acquiring Purkinje cell neural activity. The proposed design discards unneeded information, by detecting and classifying spikes in real-time. This type of compression enables data storage on a removable device in the head stage, freeing mice from wires. Its small form factor allows unrestricted movement during experiments, while a power-efficient design ensures long-term operation. The performance of the algorithm has been evaluated using a software implementation, yielding a combined accuracy for detection and classification ranging from 92.74% to 94.54%. The system has been synthesised using the 45 nm Nangate Open Cell library resulting in an ASIC with an area of 0.22 mm^2 and a power consumption of 0.412 mW.

D.A. Vrijenhoek
Delft, July 2023

Contents

List of Figures

List of Tables

1 Introduction

1.1. Motivation & Problem Statement

This thesis is the result of a collaboration between the Computer Engineering lab of the TU Delft and the Whisker Lab. The Whisker Lab is a research laboratory that focuses on studying topics related to the brain, mainly focusing on the brain of mice. For this particular research project, they try to correlate the neural activity of the Purkinje cells, which are located in the *cerebellum*, with neural activity from the *motor cortex* in the brain of a mouse. The neural activity of these Purkinje cells is especially interesting because it manifests in two types of spikes, each with its own distinctive waveform. The exact function of each type of spike within the neural circuits of the cerebellum and motor cortex is not yet known. Splitting these two types of spikes and looking into their relationship with another part of the brain helps to further our understanding of the dependence of different brain functions. This could, potentially, have application in the field of neural implants which aim to restore the (paralysed) patients' ability to move. It is, of course, hard to say what the direct applications could be, due to the early stage of the research. However, it is a promising topic to look into.

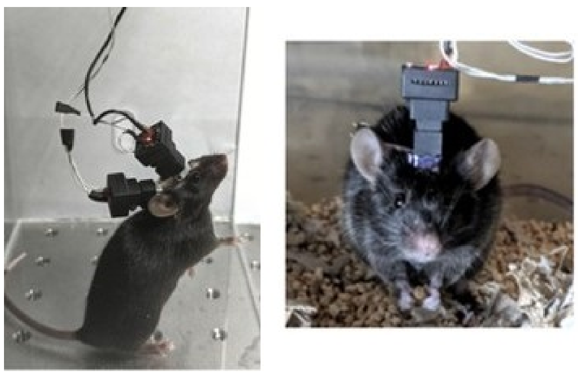
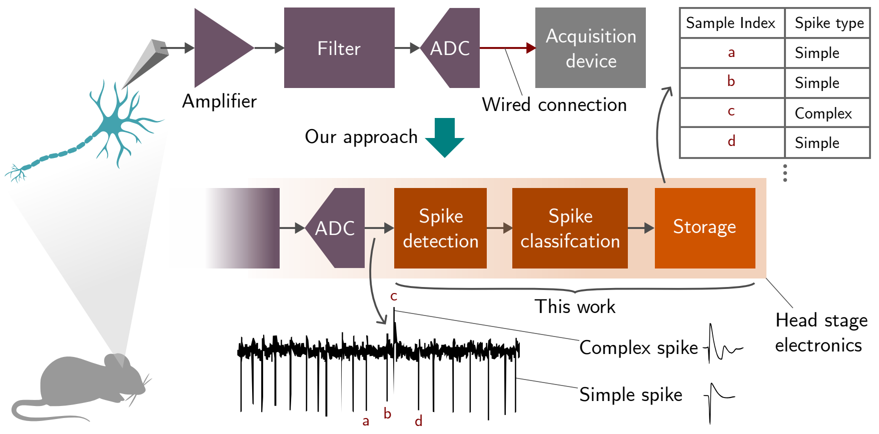


Figure 1.1: Tethered mice are not able to move around freely[9].

What makes the Purkinje cell unique, is the fact that it is one of the largest types of neuron found in the mammalian brain. Out of the two types of spikes, the most common type of spike is called the simple spike and characterised by only having a downward deflection. The second type of spike is called the complex spike and is characterised by having a downward as well as an upward deflection. The current method of data acquisition is recording required data (i.e. neural signals) via a tethered device, which directly streams the neural recordings to an off-site personal computer for data logging (see figure 1.1). Restricting the mouse's movement through this tether hinders mimicking ideal behaviour, so a different method is preferred. In order to tackle this problem, two solutions are possible: either send raw data wirelessly to the off-site computer for logging or save all the data on-chip within the mouse head stage, so it can be downloaded after the experiment has been conducted. The wireless solution introduces a plethora of additional problems, relating to latency, reliability, and power issues, and is not really a viable option without addressing these issues, which is a time-consuming process. Saving the recorded data on-chip for later extraction avoids these problems, but would require a lot of chip real-estate to facilitate (sufficient) data storage. The solution of the Whisker Lab for this problem is simple because they are only interested in the timestamp of the occurrence of the spike, as well as its classification. Currently, there is no solution for this kind of "compression" within the mouse head stage, and this thesis work aims to provide such a solution (see figure 1.2). The solution which is presented in this work also solves another problem: the software implementation of the proposed design can aid in post-processing the recorded experiments and automating the labelling of spikes. Currently, this is performed manually, which is labour intensive work.



Sample Index	Spike type
a	Simple
b	Simple
c	Complex
d	Simple
⋮	

Figure 1.2: Current wired solution for recording Purkinje cell activity and proposed solution to allow free natural behaviour (i.e. omit wired connection).

The spike detection and classification workload is very similar to a more generic workload which is increasingly being utilised in neural *implantable medical devices* (henceforth referred to as neural IMDs), called spike sorting. Spike sorting is the process of identifying the individual contribution of neurons in a recorded signal. Neural signals can be recorded in several ways, each with its own level of precision. The easiest signal type to record is called the EEG and records the neural activity of a large proportion of brain tissue, from outside the skull. To get a more localised view of the neural dynamics, brain activity must be recorded from inside the brain. By means of a recording shank, the neural activity of several thousands of neurons can be recorded. These types of neural recordings are called *Local Field Potentials* (henceforth referred to as LFP). By reducing the size of the recording shanks, the neural activity of individual neurons can be recorded. These types of brain signals are the hardest to acquire but give the most detailed insight of the brain dynamics. The neural activity of individual neurons is called *Action Potentials* (APs) However, the (small) size of the recorded neurons makes it too hard to record the neural activity of just one typical neuron. The recording shanks always record the activity of 3-5 neurons [34]. By applying a spike sorting algorithm to this recorded signal, each recorded spike can be mathematically attributed to one individual neuron and hence the spikes are sorted.

Even though the formal definition of spike sorting and the definition used in this thesis are not identical, they represent the same type of workload and share a great many similarities. Therefore, the work done for this thesis can be utilised in a broader context of neural implants and its scope exceeds that of the application it was intended for.

Even though the concept of spike sorting has been around for 20+ years, there still has not been reached a consensus regarding the preferred method to perform it[34]. Due to the many design considerations that need to be taken into account, no one method is preferred over the others. These design considerations are often about the fact that most neural IMDs have to follow a strict budget in terms of energy consumption and chip area. Moreover, the recorded data often has a poor signal-to-noise ratio with a signal's amplitude of some μV[71]. These factors contribute massively to the fact that a spike sorting system is always a trade-off between available resources and performance and is often tailored for the specific application.

The recording shank cannot be firmly attached to brain tissue and therefore, some electrode drift over time is inevitable. A spike sorting algorithm should therefore be able to deal with these kinds of issues and be robust enough to consistently perform its task. This means that some parameters might need on-the-fly calibration, but generally this means that the algorithm should be designed well. One of the main limitations for

spike sorting algorithms is the fact that the methods that deliver the most performance often introduce a high computational load to the system. These methods often comprise of "traditional" mathematical techniques, such as Principal Component Analysis, which make the spike sorting algorithms well suited for offline spike sorting on a recorded signal.
In order to facilitate real-time spike sorting with the required performance and robustness levels, new techniques need to be devised. The use of *Deep Neural Network* (i.e. DNNs) can be a solution to these limiting factors when applied well. DNNs tend to generalise well, when trained appropriately, and should therefore be able to make a spike sorting algorithm robust enough to operate under varying circumstances (e.g. amplitude change due to electrode drift).

1.2. Research Topics

In this section, the research questions are introduced. In order to address the aforementioned problem statement, the following research question needs to be addressed:

"How to develop a system that is able to detect and classify neural activity from a Purkinje cell in a real-time fashion as an embedded implant?"

In order to answer the main research question, it is often useful to break up the research question into smaller sub-questions which address a specific issue. which are the following:

- *"How can techniques for spike sorting be utilised to detect and classify neural activity from a Purkinje cell in real-time?"*
- *"How can deep learning techniques aid in classifying spiking activity from a Purkinje cell?"*
- *"What techniques can be used to deploy the proposed system as an embedded implant?"*

1.3. Thesis Contribution

The following states the contributions made based on the research opportunities as discussed in 1.1.

The main contribution of this thesis can be divided into three parts: the use of conventional spike-detecting techniques to allow for real-time detection of Purkinje cell neural spikes combined with novel neural network-based classification stage, implementing the resulting algorithm in a lightweight system architecture and utilise this system architecture in a small-form-factor CMOS implementation.

1.3.1. Real-time Detection and Classification of Purkinje Cell Neural Activity

Generally, the detection and classification of neural activity of Purkinje cells is performed manually and offline. This work is not the first attempt to automate this process, as there are other research groups working on detecting and classifying the Purkinje cell's unique neural behaviour (with varying levels of complexity and features). To the author's best knowledge, there exists no other system which aims to detect and classify complex and simple spikes in a *real-time* and *lightweight* fashion apart from this work. Applying techniques from regular spike sorting solutions, the proposed system is able to accomplish its task in an energy-efficient matter. By doing so, it is able to filter out the scarce spiking events and dispose of unwanted information. This greatly reduces requirements for data storage, which in turn leads to conducting longer experiments. The main goal of this work is to design an ASIC for an implantable device. However, the software implementation of the proposed system is already aiding neuroscientists from the Whisker Lab. Manually labelling spiking events from Purkinje cells offline is a cumbersome and time-consuming process. Time and resources which could otherwise have been spent on other activities. The software implementation of the proposed system is able to automate this process and save time and resources.

1.3.2. Lightweight System Architecture

By far the most resource-hungry part of the proposed system is the spike classifier. Given the fact that spiking events are fairly rare, it would be a waste of energy to have the classifier run continuously. Therefore, the proposed system uses a two-tier approach, where the spike detection and classification step are separated. The spike detector runs continuously in a very energy-efficient matter and raises a flag whenever a spike is detected. The system then proceeds to disable the spike detector and save a number of neural samples, which then get fed to the spike classifier. Only at this point, the classifier is enabled and does its job. After the spike is classified, the classifier is disabled again and the spike detector continues to process the incoming neural samples in order to detect the next spike.

1.3.3. CMOS Implementation

In order for the system to run in an embedded context (i.e. as an implantable device), a small-form-factor CMOS implementation is proposed. This CMOS implementation allows for low-power operation for an extended period of time without hindering the mice in their natural behaviour. Several techniques have been applied to the design to make this possible. The aforementioned two-tier approach limits the power consumption of the system by only enabling each module when it is needed. Moreover, through quantisation of the neural network classifier, the computational complexity has been reduced by implementing integer-only arithmetic (contrary to the more commonly used floating-point arithmetic). Through quantisation, the memory usage to store the synaptic weights has also been reduced by a factor of four (8 bits vs. 32 bits). This greatly influences the used chip real-estate and the power consumption of the system as a whole in a positive way.

1.4. Outline

The first chapter after the introduction presents background knowledge of all relevant topics.The topics discussed are a basic background in Purkinje cell studies, background knowledge and state-of-the-art of neural IMDs, and the various steps of spike sorting. Lastly, the basics of artificial neural networks are explained. The next chapter will present the proposed design. It will cover a system overview explaining the whole spike sorter system. Also, a more in-depth analysis of the spike detector and spike classifier is given. The spike sorter's performance was evaluated and the findings of this evaluation can be found in chapter 4. The system was evaluated in two different ways, first off the performance of the algorithm was assessed. Next up, the performance of the ASIC implementation was presented.
This report concludes with a chapter where the proposed system design is summarised. Suggestions for improvements for future iterations are also presented and things that could have been done better are touched upon.

This does not cover all topics treated in this thesis, hence background knowledge of neural IMDs as a whole is also given. The data acquisition part of these systems is treated, as well as how these systems stimulate the surrounding brain tissue. Lastly, different workloads between the recording and stimulation steps are discussed. This is a nice transition to a section explaining the basics of spike sorting, which is such a workload. Of course, for both topics, the current state-of-the-art in their respective fields are also presented. Lastly, the basics of artificial neural networks are explained. This gives the reader the required background knowledge to understand the material with regard to the spike classifier, which is implemented as such an artificial neural network.

2 Background & State-of-the-art

This chapter presents background knowledge for all the topics touched upon by this work. In case it applies, the state-of-the-art of these topics is also discussed. The first section is about Purkinje cells. Their physiology is discussed as well as different systems that are designed to classify Purkinje neural activity. The next section presents a brief overview of neural implantable medical devices or neural IMDs. The recording of neural activity is discussed as well as the stimulation of neural tissue. Lastly, different types of workloads for neural IMDs are presented. These workloads can be used to close the loop by enabling recording-driven stimulation. One of those workloads is called spike sorting, which will be discussed in detail. Lastly, a brief introduction to artificial neural networks will be given.

2.1. Purkinje Cells

Purkinje cells are neurons which are found in the *cerebellum*, a brain region generally found in the lower part of the brain. This kind of neuron is named after its discoverer, the Czech anatomist Jan Purkyně [2]. The first topic discussed in this chapter is Purkinje cells, especially what makes them unique from other types of neurons. All relevant information with regard to the context of this thesis work is provided. Moreover, the state-of-the-art with respect to classifying neural activity of Purkinje cells is also given. Several state-of-the-art systems for classifying Purkinje cell neural activity will also be discussed.

2.1.1. Physiology

As mentioned before, Purkinje cells can be found in the cerebellum (also called the cerebellar cortex). This part of the brain is responsible for governing motor functions. The Purkinje cells can be found on the outer edge of the cerebellar cortex, where they form neural circuits with neighbouring Purkinje cells as well as neurons located deeper in the cerebellar cortex. Being one of the biggest types of neurons, Purkinje cells have a large number of dendrites (extensions of neurons that enable them to connect to one another). The resulting neural circuits play an important role in learning and controlling movements, as well as coordination [23].
Another differentiating factor that set Purkinje cells apart from other types of neurons is the fact that they exhibit two types of spikes, called the simple and complex spike [77] (see figure 2.1). As can be seen in figure 2.1, a simple spike is characterised by a downward reflection of the neural signal, after which the spike is over. As the name suggests, the complex spike exhibits more complex behaviour as it initially reflects downwards, followed by an upwards overshoot. The spike ends with a high-frequency (dampened) oscillation. In general, neurons exhibit spiking behaviour, also known as firing. This behaviour is achieved by accumulating a certain concentration of Sodium or Potassium Ions. This causes a membrane potential to occur (i.e. *action potential*), which in turn is perceived as neural activity [34]. Purkinje cells also use concentrations of Calcium Ions to generate action potentials [81]. By doing so, Purkinje cells are able to generate complex spikes as well. Most of the spiking activity of Purkinje cells consists of simple spikes, while the occurrence of complex spikes is relatively scarce. The simple spiking frequency of a Purkinje cell ranges between 17 and 150 Hz, while the complex spiking frequency is about 1-3 Hz [57].

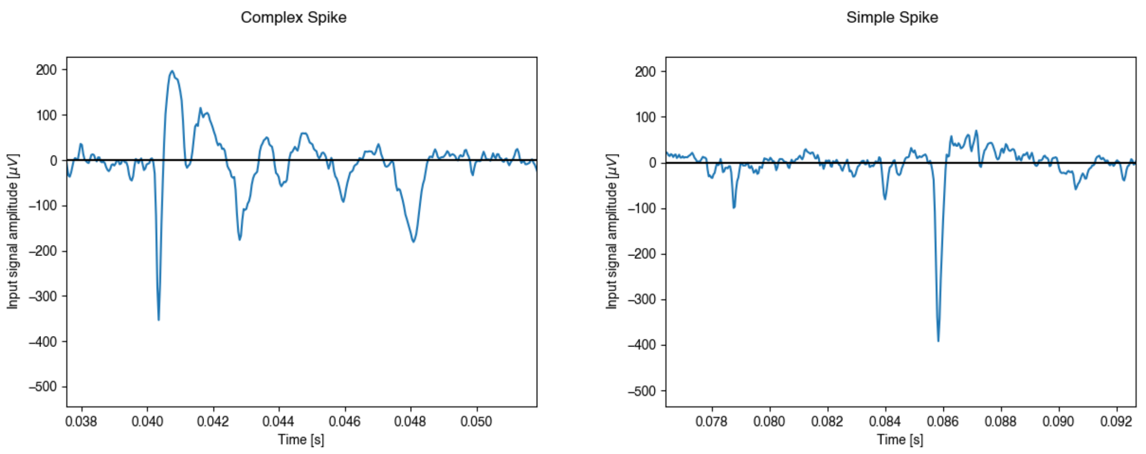


Figure 2.1: Two types of spikes characterising the Purkinje cell. The complex spike can be seen on the left side and the simple spike on the right side.

2.1.2. Classifiers

The relatively rare occurrence of complex spikes, together with their complex waveform result in the fact that the exact functionality of a complex spike remains a topic of research. In order to look into this topic, the neural dynamics of Purkinje cells need to be mapped. This can be done by recording neural activity from Purkinje cells. These recordings can be processed by a system that is able to detect and classify both types of spikes. The requirements for such a system depend greatly on the research project. For example; in some cases, it might suffice to process prerecorded neural signal offline, while other cases require online detecting and classification.
Markanday *et al.*[44] propose a system that uses *Deep Neural Network* techniques for detecting and classifying complex spikes from prerecorded neural recordings from monkeys. Their proposed solution (see figure 2.2) consists of a *Convolutional Neural Network* which takes both *action potentials* (i.e. APs) and *local field potentials* (i.e. LFPs) as input. The neural network is able detect and classify complex spikes from these recordings. Its functionality extends beyond the regular classification of spikes as the system is also able to provide the start and end times of the detected complex spikes. Moreover, the system is able to process neural recordings which contain spiking activity of several Purkinje cells and identify which neuron is causing the detected spike. In order to optimise the performance of the system, three post-processing steps are introduced. The first post-processing step corrects a mismatch between reported start and end times and actual start and end times. The system does this by realigning the detected complex spike and by doing so it can correct the timing window up to ± 2 ms [44]. The second step involves mapping the recorded APs and LFPs on a two-dimensional plane, 2 ms after the occurrence of a CS spiking event. After a dimensionality reduction, the acquired features are fed to a clustering algorithm to pick out false positives (e.g. simple spiking event). This is the third post-processing step. Optionally, Markanday *et al.* suggest other post-verification steps, such as tracking electrode drift through the tissue. The authors do not report an explicit performance metric and state that their system is on par with human experts. Considering that their proposed design is relatively complex (31,482 floating-point 32-bit values) and requires two types of neural recordings, it is not a feasible solution for online embedded deployment.

Sedaghat-Nejad *et al.* proposed a method to sort the spikes of Purkinje cells as well and they have called their algorithm PSort[70]. The PSort algorithm (which is implemented using Python) takes APs and LFPs as input. Both neural signals are sampled with a frequency of 24,414 Hz and are band-pass filtered between 100 Hz and 10 kHz (*AP*) and 3 and 300 Hz (*LFP*). Initially, the algorithm will filter both input signals again using a 4$^{\text{th}}$ order Butterworth filter with a band-pass of 50-5000 Hz and 10-200 Hz respectively. To suit the specific conditions of the data, these parameters can be tweaked using the provided GUI. The next step is to detect the occurrence of spiking events. This happens under the assumption that *simple spikes* produce negative peaks in the *AP* channel and that *complex spikes* produce positive peaks in the *LFP* channel. Simple thresholding is then employed to separate spikes from the background noise. In order to calculate the threshold, PSort computes the histogram of peaks for each channel. On these histograms, PSort will attempt to fit a Gaussian

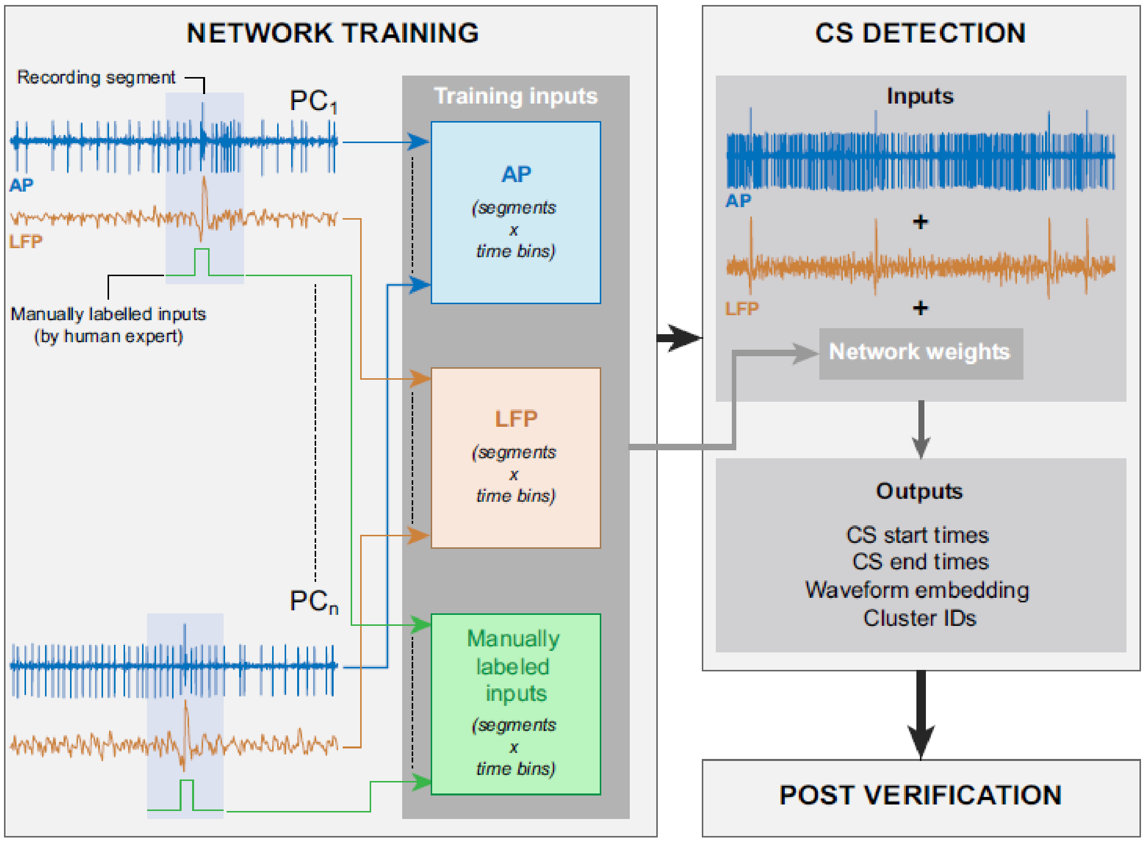


Figure 2.2: Graphical representation of the workflow of the proposed algorithm by Markanday *et al.* [44].

Mixture Model, which will result in two Gaussian distributions per channel. The Gaussians describe the distribution of the noise floor as well as the signal of interest (i.e. the neural spikes). The intersection between both Gaussians is used as a threshold. Since this method is applied, the result of this step is two thresholds; one threshold which is used to detect *simple spikes* from the *AP* channel and another threshold which is used to detect *complex spikes* from the *LFP* channel. The next step is to relate peaks in one input channel to peaks in the other input channel. PSort offers three methods to do so: *simple spike* index, *simple spike* template, and *complex spike* template. In the first instance, there is no data to compute a template for either spike type. Thus, the spike index is used to calculate a template. Both types of spikes are characterised by an initial negative deflection. So, initially, these sharp negative deflections (*SS* index) are used to align the *complex spike* or *simple spike* waveforms. A distinction between the two types of spikes is made using the threshold which detected them. In case of the *complex spikes*, PSort will move the *complex spike* template (with a width of 3 ms) over a 5 ms time-window on the *AP* signal and it will calculate the correlation between the template and the AP in the template window. The timestamp that results in the maximum correlation value is then saved. The time window, which again is 5 ms, is centred around the detected *LFP* peak. It is not centred symmetrically as the time window starts 4 ms before the *LFP* peak and therefore ends 1 ms after the *LFP* peak. Similar to the other functionalities of PSort, the construction of both templates can be configured by the user via the GUI.

Apart from detecting and classifying Purkinje cell spikes, PSort also offers additional features. PSort is equipped with a clustering module which clusters detected spikes. Being one of the key features of PSort, it allows the user to (automatically) form groups of spike waveforms and study their statistical interaction with other spikes. PSort is equipped with two feature reduction algorithms, namely UMAP which was proposed in [45], and Principle Component Analysis. The user is then able to specify which clustering algorithm can use the reduced feature set for clustering. The first option is to apply the Gaussian Mixture Model to the feature set. For this clustering algorithm, the user needs to specify the number of clusters as well as their respective centroids. This will not be the case for the other two clustering algorithms, which are able to do their job in an automated fashion. The second choice for clustering the spike waveforms is the use of the ISO-Split algorithm and, lastly, the third option is the use of the HDBSCAN (hierarchical density-based spatial clustering of applications with noise) algorithm. PSort is also able to detect outliers (i.e. spikes that do not really fit in any of the clusters) using the Local Outlier Factor Density algorithm. References to all these algorithms can be found in [70]. The last feature of PSort is its dissect module, which allows the user to better inspect individual spikes.

The performance of the PSort system has been evaluated using a dataset containing neural recordings from the cerebellum of both mice, marmosets and macaques. In total, the dataset contains recordings of 300 different Purkinje cells. The recordings from these three types of animals were found to be relatively similar, although differences were reported on average spiking rate, duration of simple spike suppression after the occurrence of a complex spike and the interspike interval distribution (i.e. the distribution of spikes over time). In order to set the baseline, an expert manually labelled the simple and complex spikes in the aforementioned dataset and results from applying PSort to this dataset were then compared to this baseline. It was found that PSort and the expert agreed on roughly 99% of the simple spikes and 92% of the complex spikes. The median differences between PSort and the expert annotation's spike timing (i.e. the start time of the spike) were 0.06 ± 0.01 ms (median ± median absolute deviation, MAD) for simple spikes and -0.12 ± 0.05 ms for complex spikes. It is hard to say whether or not PSort is more optimistic or conservative than the expert, because PSort identified 74 simple spikes that were not identified by the expert. However, Psort also missed 382 simple spikes that were annotated by the expert. Similarly, PSort identified 111 complex spikes that were not identified by the expert (effectively raising the average complex spiking rate from 1 Hz to 1.19 Hz, a 19% increase). PSort missed 8 complex spikes that the expert did identify. PSort was also compared to another PC sorting algorithm, namely the one proposed by Markanday *et al.*[44], discussed previously. Only the detected *complex spikes* of PSort were used for comparison since the CNN-based system of [44] is only able to detect *complex spikes*. Since no ground truth was established, it was difficult to draw a fair comparison. The authors of [70] attempted to solve this by calculating a similarity score. This score is calculated by measuring the number of matched spikes (using a 1 ms time window) and dividing this by the total number of detected spikes by both systems, minus the number of matched spikes. It was found that the similarity score between both systems was 99.0% and 93.0% for two clusters of *complex spikes*.

Even though the goal of [70] is similar to the goal set out for this work, the context of the application differs significantly. Whereas this works aims to automate the detection and classification of PC neural activity in an *embedded* and *real-time* fashion, PSort aims to provide offline spike detection and classification. Because it operates offline, its requirements with respect to the computational load are more relaxed compared to those set for this work. An example of these relaxed requirements is the use of templating since this is a relatively expensive method in terms of memory usage and computation. Moreover, PSort offers some features which are not of interest in the scope of this work. One of these features is, for example, the clustering module. Another such feature is the great degree of configurability. As mentioned before, a great many aspects of PSort can be configured by the user, such as filter parameters and template settings. This allows the user to tailor the algorithm to their specific needs, however, the algorithm will be less robust out-of-the-box. The aim of this work is to deliver a robust system that is able to operate adequately under varying circumstances (within limits). Lastly, and very importantly, PSort requires two different types of neural inputs signal (similar to [44]), namely *LFPs* and *APs*.

2.2. Neural IMDs

A neural *implantable medical device* is a device that is (partly) implanted in the brain to monitor and stimulate brain tissue. These neural IMDs come in a variety of configurations and cover a diverse application range. Neural IMDs are either capable of recording neural activity or stimulating brain tissue and often combine both features in a closed-loop system. Neural IMDs can either be fully implanted or partly, where often only the recording and/or stimulation devices are implanted. Computational heavy tasks can then be performed outside of the host body by streaming the recorded data (either wirelessly or tethered) or process it directly on the implant.

2.2.1. Data Acquisition

Neural IMDs typically work one of three types of neural signals, namely EEG/ECoG, local field potentials, and action potentials. These three types of neural signals differ in the amount of information they contain (i.e. the number of neurons contributing to the recorded signal) as well as the complexity of data acquisition. An example of all three signals can be found in figure 2.3.

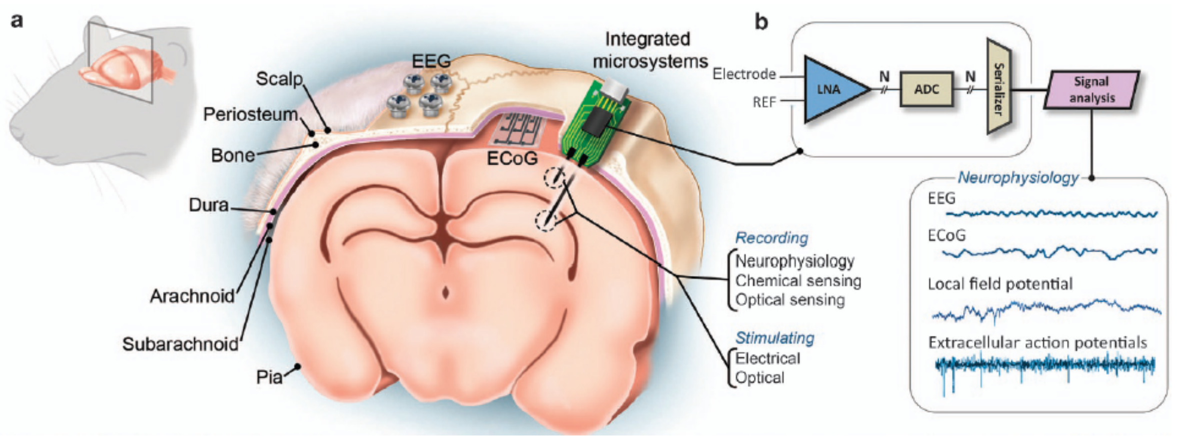


Figure 2.3: Illustration of a rat's brain together with various recording and stimulation techniques. On the right side of the figure is an overview of the four types of neural signals. [71].

In this regard, EEG/ECoG contains neural activity from millions of neurons. It is recorded using electrodes on the scalp (EEG) or on the direct surface of the brain (ECoG). EEG/ECoG signals are typically used to identify neural activity in certain frequency bands. These frequency bands are commonly referred to as "brain waves". However, generally speaking, EEG/ECoG are not often used for neural IMDs, due to the fact that the EEG/ECoG signals are obtained through recording electrodes on the scalp which makes it impractical for long-term use.

A more commonly used neural signal is recording the LFP of a number of neurons (several thousands of neurons). The LFP is the combined potential of all neurons in the vicinity of the recording electrode array which is located inside or on the brain tissue. It is therefore better suited for long-term deployment and it provides detailed insights in the neural dynamics of the targeted brain region. Recording LFPs is possible through an electrode grid [43], which is a high-density electrode grid, placed inside the brain tissue. An example of such a grid can be found in figure 2.4.

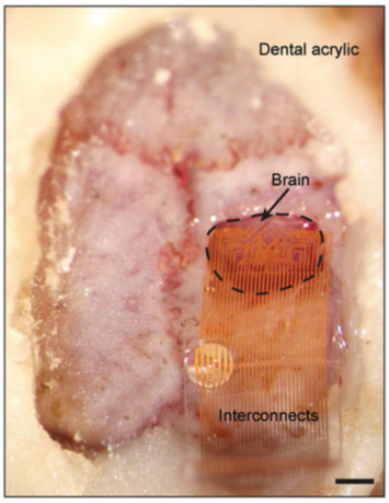


Figure 2.4: Example of a recording electrode grid [78].

Lastly, some neural IMDs work with the recorded action potentials of the targeted neurons. This is the most detailed form of neural recordings which is also the hardest to record. Generally a very small recording electrode (or several on some occasions) is placed on a shank-like object (see figure 2.5) which is embedded in the brain tissue. These shanks often contain several recording electrodes packed on small surface, with areas from 25-150 μm^2 [12]. These recording electrodes are able to record the action potentials of three to five neurons that are near the recording shank. Action potentials give a very detailed overview of the neural dynamics within a part of the brain since it records the individual spiking behaviour of neurons. However, its scope is very limited since it is restricted in terms of the number of neurons that can be recorded.

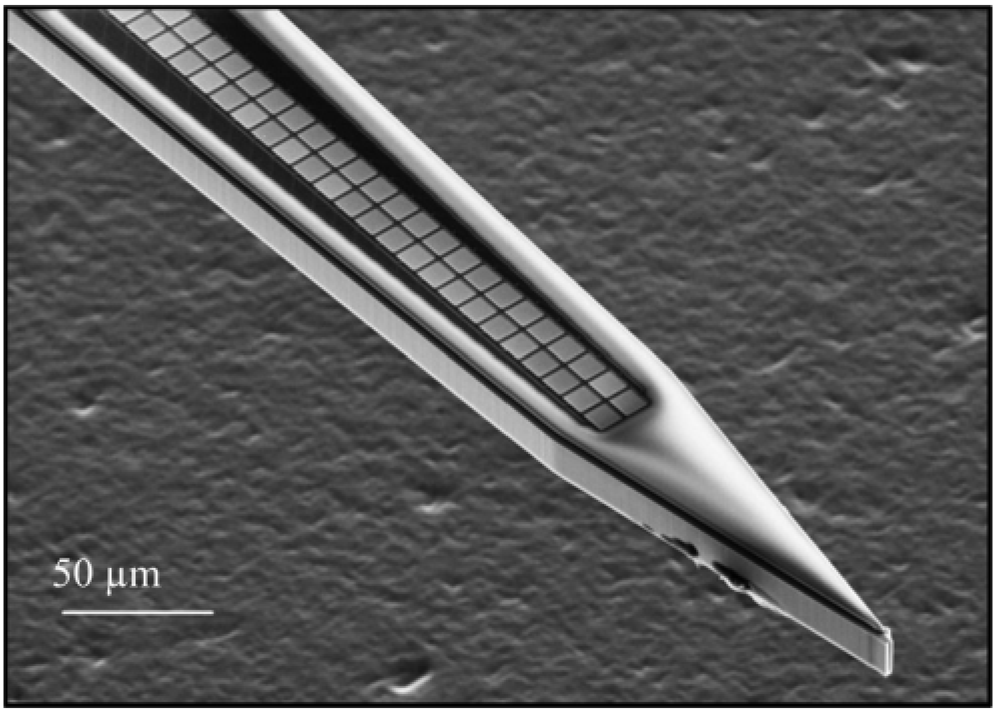


Figure 2.5: Example of a recording shank [68].

Some recording systems are also able to deal with several types of neural signals. For example, [14] presents a neural recording system that is able to process both LFPs and APs. Through timesharing of the available hardware, the system can process up to 16 input recording channels simultaneously. The analog front-end (i.e. AFE) of the system comes with a low noise amplifier, after which the amplified signal is digitised using a 10-bit ADC. Since LFPs and APs require different filter parameters (i.e. LFPs are band-filtered 1-300 Hz and APs are band-filtered between 0.3-10 kHz), the system comes with adaptive 2^{nd} order digital IIR filter. By implementing an LMS algorithm (i.e. least mean square), the system is able to cancel slow recording electrode offset, which would otherwise degrade the signal's integrity. [40] describes an implantable platform for neural recordings and is designed to handle recorded action potentials. The system is not outfitted with a recording probe, instead, it is compatible with several commercially available recording probes. The signal conditioning stage of the system is comprised of an amplifier. This amplified signal passes through several cascaded filters which form a pass-band filter. First off, the signal is passed through 3^{rd} order Butterworth low-pass filter which is tuneable. What follows is a 1^{st} order high-pass filter in the analog domain. After digitisation (through a 16-bit ADC), the signal is again high-pass filtered using a 1^{st} order IIR filter. The cut-off frequencies are 300 Hz and 5 kHz respectively.

The development of neural IMDs is not merely a matter of academic research as commercial parties are also investigating its possibilities. The San Francisco-based company Neuralink developed a neural IMD platform that is capable of recording and processing several thousands of channels[48]. The system works with recording threads containing 32 recording electrode contacts which are spaced by 50-75 μm (depending on the configuration), the dimensions of the electrode pads are 14x24 μm. Up to 96 of these recording threads can surgically be implanted in the brain and are connected to a neural processing ASIC. This ASIC is equipped with programmable amplifiers and band-pass filters to accommodate variations in the signal quality. The ASIC is also equipped with a 10-bit ADC which samples the recordings at 19.3 kHz. Lastly, the ASIC serialises the data coming from all the channels, so it can be read out using a USB-C port. Each ASIC is capable of processing 256 channels in parallel, so 12 ASICs are mounted on a PCA (i.e. *printed circuit assembly*) to achieve the reported number of recording channels (12 x 256 = 3072 channels). Due to the reconfigurable gain and filter properties, the system is able to handle both LFPs and APs, even though it is designed to handle APs in the first place.

2.2.2. Tissue Stimulation

Some neural IMDs are equipped with the means to stimulate neural tissue. This is commonly referred to as *deep brain stimulation* (i.e. DBS). Neurons communicate which each other through electrical pulses (i.e. neural spikes) and these pulses are generated by building up a potential through the separation of ions in the cell's membrane[8]. This difference in ion concentration is achieved through manipulating so-called ion channels in the cell. The purpose of neural stimulation in the context of neural IMDs to is open these ion channels. This will cause the neuron to lose some of its action potentials and cause it to fire off a spike less. In this sense, neural stimulation has an inhibitory effect on the brain region of interest which in turn suppresses the effects of Parkinson's disease, essential tremor, and epileptic seizures[15, 82].

Several methods of neural stimulation are possible, each with its own merits and challenges. Currently, the most used type of neural stimulation is electrical stimulation since this method is relatively easy to apply [26]. One of the main drawbacks of using electrical stimulation is the fact that it tends to damage the tissue surrounding the stimulation electrode [82]. Over time, this will lead to the growth of scar tissue around said electrode, which in turn reduces the effectiveness of DBS. Electric stimulation of brain tissues comes in a variety of configurations. For example, DBS can be performed using a constant current[61] or constant voltage source[60]. A significant difference between the effectiveness of constant current and constant voltage were not found, although constant current stimulation seems to be the safer option since it leads to less scarring[58]. The stimulation pulses can also be either mono or biphasic in nature. Similarly to constant current stimulation pulses, biphasic stimulation pulses tend to lead to less scarring and are considered the safer option. However, no significant difference have been found between monophasic and biphasic stimulation pulses [91].

Another way to stimulate neural tissue is through optical means. This method eliminates many of the downsides of electrical stimulation. However, as of now, it is not the preferred method of DBS since preparing the tissues for optical stimulation. The brain tissue itself does not respond to light pulses and needs to be genetically altered in order to respond to the light pulses[4]. Apart from the ethical concerns this method raises, it is relatively hard to genetically alter the brain region of interest to be sensitive to light. However, it is not impossible and it would lead to little to no scarring to the surrounding brain tissue. Apart from this, it is also possible to very precisely stimulate certain neurons since surrounding tissue is not affected. Neurons can be made sensitive to light across the entire spectrum, which adds to this fact. Lastly, μLEDs can be used to provide the required light source[13], which are less stringent on the energy budget of a neural IMD. An example of such a μLED can be found in figure 2.6. This would greatly aid the battery life of the system, which lengthens the duration of deployment, especially for embedded devices. It is a very promising technique but due to various reasons, its utility remains merely academic.

2.2.3. Workloads

Ideally speaking, a neural IMD is capable of both recording neural activity as well as stimulating brain tissue, since stimulating brain tissue without feedback is not desirable and the merits of merely recording neural activity are not as significant as using the recorded data directly as feedback for stimulation. When a neural IMD is equipped with both functionalities and these are integrated in a closed-loop feedback system, the neural IMD often employs an operation of some kind of recorded neural activity to govern the stimulation settings.
One of the most prominent workloads in closed-loop neural IMDs is the stimulation control algorithm. Such a feedback algorithm is able to control the stimulation parameters (e.g. the amplitude, duration, and duty cycle of the stimulation waveform) based on signature features in the recorded neural signal [33]. Adjusting the stimulation parameters can be based on several features which can be extracted from the recorded neural signal. Often if such a feature exceeds a calibrated threshold, stimulation of neural tissue is required. This is called event or activity-driven stimulation [75]. For example, neural biomarkers (i.e. a measurable neural event) can be detected if the spectral power of the signal exceeds a certain threshold [96]. If such a biomarker is detected, it can be suppressed through stimulation. Another way to detect these biomarkers is by looking at the number of spiking events in a (sliding) time window[16, 32, 67]. When the number of spikes exceeds a certain threshold, then stimulation is enabled to stir an inhibitory effect. Another example of activity-driven stimulation is by looking at the phase-lock value of the incoming neural signal. This value tells something about the tendency of a neuron to fire a spike at a particular phase. Kassiri *et al.* propose to use a CORDIC-based processor to calculate this value and neural stimulation is enabled if the PLV value exceeds a certain threshold[27].

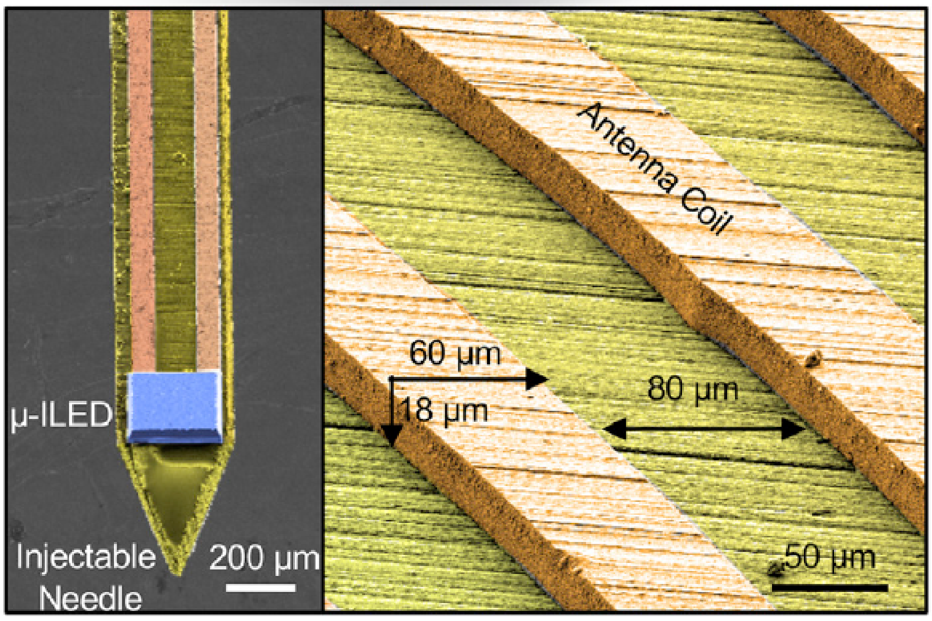


Figure 2.6: Example of a μLED on a needle, the right side depicts part of the (coil) antenna[74].

However, not all feedback algorithms are *event-driven*, and some aim to continuously adjust the stimulation parameters. For example, Stanslaski *et al.* propose a closed-loop neural stimulator that uses a *support vector machine* to control the stimulation parameters[76]. Most systems that use a continuous feedback loop borrow techniques from control engineering, such as PID controllers, both in the digital[86] and analog[38] domain.

Others aim to utilise deep learning ideas to generate patient-specific stimulation waveform patterns[51]. Lastly, event-driven stimulation does not necessarily need to be in the context of a feedback loop. Stimulation can be timing-based and depend on events detected from LFP neural signals[59]. Using canonical correlation analysis, the system proposed in [59] can detect the intent to push a lever in a rat's motor cortex. Based on this intent it can stimulate the spinal cord of the rat in order to trigger a muscle response. The application is the reanimation of a (lost) limb. In a way, this system can bridge a (broken) connection in the brain and restore motor functions in case of paralysis.

Another category of workloads for neural IMDs frequently seen is compressed sensing. Generally speaking, the Nyquist-Shannon sampling theorem states that to accurately reconstruct a periodic signal, one must sample it at twice the highest frequency of interest[73]. This could potentially lead to sampling frequencies up to 40 kHz for sampling *action potentials*[87]. This could lead to resource overhead when trying to process the sampled data. Compressed sensing is a signal processing technique that allows for accurately reconstructing a sampled signal without adhering to the Nyquist-Shannon sampling theorem. The resulting (reconstructed) signal will pose less overhead to further processing steps. Another advantage of utilising compressed sensing is the reduction of required bandwidth for wireless streaming of telemetry[6], which can be as much as two to three orders of magnitude [40]. Others report a more than four-fold increase in energy efficiency compared with state-of-the-art FPGA-based event-driven neurostimulators [67] (with wireless data transmission capabilities). Compressed sensing can also be used to reduce the computational load for a neural IMD. Because of compressed sensing, feedback loops for closed-loop neurostimulators can be simplified without compromising for performance[37]. This can for example be done by using a 2^{nd} order *differential pulse code modulator* (i.e. DPCM) in conjunction with an encoding engine [42]. This encoding engine is based on an adaptive Golomb coding algorithm which compresses the recorded neural data based on optimal parameters obtained from the signals in real-time[42]. Compressed sensing can also be used partially. For example, only detected spikes can be fed to a compressor module[3]. These compressed spikes are then processed further and because of their simplified representation will lead to a reduced computational load for the system. Another popular method of compressed sensing is through the *discrete wavelet transform* (DWT) [7, 24]. By doing so, only a subset of the wavelets (and therefore corresponding coefficients) is used in further steps. For example, DWT is a commonly used technique in the detection of spikes (see 2.3.1) and by only considering a part of the DWT, adequate spike detection can still be achieved [24].

The last common category of neural IMD workloads is the cancellation of recording artefacts that negatively impact a given system's performance. Generally, there are two types of artefacts that might end up in the recorded signal. The used recording electrode might introduce recording artefacts when it moves (slightly) when deployed. [39] presents a PD-controller that allows for the removal of these recording artefacts in the signal of interest. The amplitude of (recorded) neural signals is inversely proportional to the neuron's distance to the recording electrode. The amplitude of the neural signal will therefore vary as the recording electrode can not be firmly attached without damaging surrounding tissue. This problem will also cause recording artefacts. [56] aims to cancel out these types of recording artefacts by implementing an auto-ranging loop, by means of a DC servo loop (i.e. *DCSL*). Such a loop tries to keep the amplitude of the neural signal constant. Doing so mitigates the negative effects of varying amplitudes (relative over time). Lastly, recording artefacts can also arise by the DC offset which recording electrodes introduce[14]. These offsets slowly vary over time and should, ideally, be cancelled out in a dynamic fashion. Fathy *et al.* aim to solve this problem by utilising an LMS (i.e. *least mean squares*) interference-canceling filter. The least mean square values are used to configure an adaptive high-pass filter, which filters out the offset.

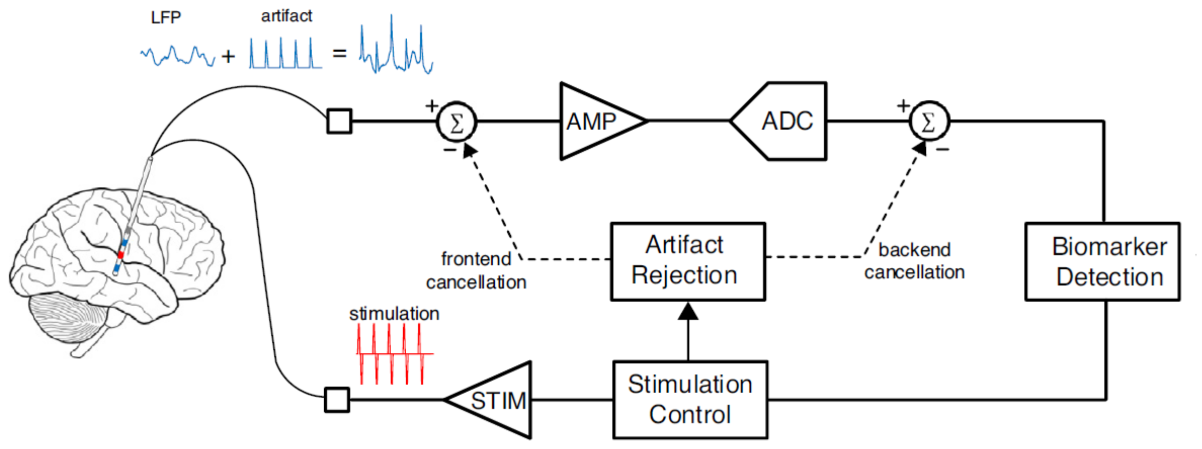


Figure 2.7: Example of a closed-loop neurostimulator IMD with stimulation artefact removal [95].

Stimulation of brain tissue also causes artefacts to show up in the recorded neural signal, especially when it concerns electrical stimulation[84]. The stimulation artefacts have a negative impact on the effectiveness of the feedback system. One way of dealing with these stimulation artefacts is to cancel them through interpolation[96]. Since the stimulation waveform is known, it is possible to reconstruct the neural signal without these stimulation artefacts by estimating it[96]. When brain tissue is stimulated, the recorded signal will be a combination of the neural activity as well as the simulation waveform. Since the stimulation waveform is known upfront, the stimulation artefacts can easily be removed in the *analog front-end* (i.e. AFE) of the neural IMD, by simply subtracting the stimulation waveform from the recorded signal using an amplifier[95]. An example of a closed-loop neurostimulator IMD with (stimulation) artefact cancellation can be found in figure 2.7.

Yet another common workload for neural IMDs is spike sorting. In the context of this work, spike sorting is an important workload and is therefore treated indepth in the following section.

2.3. Spike Sorting

Spike sorting is the practice of identifying the spiking activity of individual neurons from a recorded neural signal through mathematical means. Recorded signal implies that spike sorting algorithms are only able to operate on pre-recorded sets of neural samples, however, plenty of spike sorting solutions are also able to operate in a real-time fashion. Some recording electrodes in neural IMDs are able to record APs, however, it is inevitable that the APs of several neighbouring neurons are picked up by the recording electrode. So in order to get the most insights into the neural dynamics happening in the brain, it is required to know the individual contribution of each neuron. This is where spike sorting comes into play. In the literature, it is generally agreed upon that spike sorting can be broken down into four distinct steps. The spike sorting algorithm takes neural samples as input in a sequential way. The first step is to detect the occurrence of a spiking event from these samples. This is called the detection step. In order to properly sort the spikes, the detected spikes are aligned in a consistent way, during the alignment step. These (aligned) spike waveforms are classified to identify from which neuron they originate. Often, the number of features (i.e. every sample point of the spike waveform) is too big to efficiently perform the classification step. So after aligning the spike, a feature reduction step is needed to achieve a smaller feature set. This (reduced) feature set is then used in the last step where the spike is classified. An overview of the spike sorting process can be found in figure 2.8.

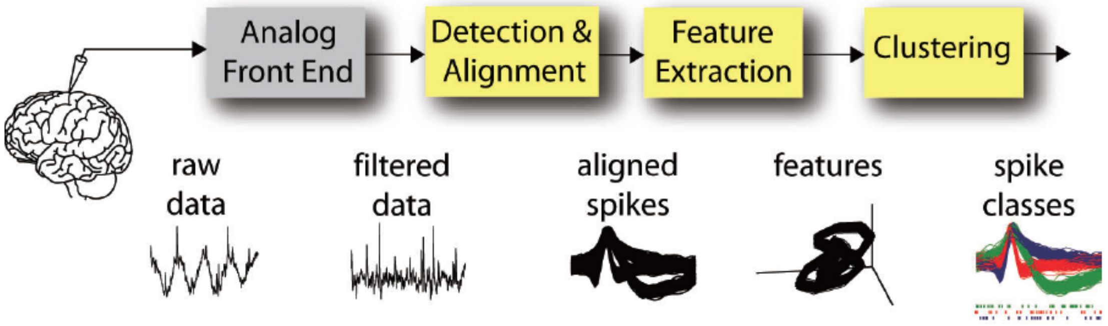


Figure 2.8: The spike sorting process[25]

The next subsections will provide an in-depth analysis of each step, together with the most commonly used methods in the field of spike sorting.

2.3.1. Detection

Detecting spikes from a recorded neural signal can be a cumbersome task due to the nature of the input signal (i.e. the neural signal). The neuron's spike often is not clearly distinguishable from the background noise due to the high noise floor of the recorded signal. Generally speaking, simple thresholding (i.e. amplitude thresholding) of the neural signal does not yield satisfactory results with respect to the detection accuracy, even though it is still employed in some cases [53][94][30][49]. In order to increase the potential of amplitude thresholding, one could also look at the absolute sample values in order to use both the negative and positive threshold simultaneously [50]. Another factor that contributes to the need for more sophisticated detection methods is the fact that recording neural activity with this resolution (i.e. recording action potentials from individual neurons) inherently introduces variation in the recorded signal. Because of the fact that such a recording electrode cannot be fixated due to the soft nature of the brain tissue, amplitudes of recorded spikes will differ over time. So detection of spikes by comparing the amplitude of a candidate spike with a given threshold, always boils down to a trade-off between detection performance and lack of robustness, while also considering restrictions on resource consumption.

Several methods have been proposed to mitigate the lack of robustness by looking at other features of the recorded signal. Most of the spike detection methods happen in the digital domain after the recorded neural recordings have been converted to a binary representation. However, one method remains in the analog domain for detecting spikes by making use of memristors [18]. Gupta *et al.* make use of the fact that the resistive properties of a memristor depend on both the voltage over its terminals as well as the duration of applying this voltage. The resistance of this memristor changes in a characteristic way since spike waveforms tend to remain constant over time. Via a readout circuit, this change in resistance is measured and used to detect a spiking event. However, their proposal tends to suffer from saturation in the event of many spikes in a short period of time, which facilitates the need for a reset signal. This also tends to happen when there are a lot of recording artefacts. The authors recognise that their work is a promising candidate for future spike-sorting implants, but it is still in an early development phase.

Digital methods range from traditional mathematical methods to newer methods, such as employing artificial intelligence techniques. For example, [54] proposes a spike sorting *spiking neural network* (i.e. SNN) which encodes incoming neural recordings in a spike train. The amount of spikes in the input spike train of the SNN (i.e. the *encoded* neural recordings, so not the actual spikes in the neural recordings) is determined by the amplitude of the input signal. The occurrence of a spiking event can then be determined by looking at the number of spikes in the input spike train (which, to avoid confusion, is the encoded representation of the input neural signal). Even though SNNs are a promising deep learning technique, the performance of this system could not match some of its more traditional alternatives, both in terms of accuracy and resource consumption. Another proposed system that leans on deep learning techniques is the system proposed by Rácz *et al.*[63]. Their proposed system uses a *Recurrent Neural Network* (i.e. RNN) in conjunction with a *Convolutional Neural Network* (i.e. CNN) in order to detect spikes from an incoming stream of neural recordings. This detection method is not suitable for real-time embedded spike detection as it features a relatively large number of neurons (and hence parameters that need to be saved in memory). Moreover, the performance of the system is not adequate as on no occasion (tested on 10 datasets) did it exceed a detection accuracy level of 88%.
An example of a spike detection system which is based on a more traditional mathematical technique employs the use of wavelet transforms. The method described in [17] uses *discrete wavelet transforms* (DWT) for spike detection. This results in a set of wavelet coefficients which differ when a spike occurs. The spike is detected by applying a threshold to these coefficients. The DWT method yields good results in signals with a lot of background noise. However, generally speaking, it does not perform really well relative to the resources it consumes.

The most widely used method for spike detection employs ideas from the radio engineering. Mukhopadhyay and Ray were the first to make use of the Non-linear Energy Operator (also called the Teager Energy Operator) as a method for detecting spikes [47]. Originally, it was used to quantify sinusoidal signals which were modulated using either AM or FM. Applying the non-linear energy operator (henceforth referred to as NEO) to a signal accentuates the high-frequency components of the signal. The high-frequency components are at the base of sudden increases (or decreases) in the signal's amplitude, which is very typical in neural spikes (hence the name; spikes). Moreover, the relatively cheap computational cost of the operator makes it an ideal candidate for spike detection. It only requires two multiplications and one addition to calculate a NEO value. It is therefore the preferred method for spike detection in spike sorting systems, both offline [28, 80] and online systems [11, 25, 69, 89, 90, 93].

One of the disadvantages is that NEO-based spike detection still relies on thresholding in order to detect spikes. Applying NEO to a neural recording signal accentuates the spikes and decreases the influence of background noise, however, choosing the threshold dictates how well the spike detector will perform. When NEO first became the more commonly used method for spike sorting, this threshold was often determined by calculating the average NEO value over the recorded experiment and multiplying this average by an empirically found constant[47]. Later papers proposed methods to dynamically calculate this threshold [31][88] to make the spike detector robust against varying recording circumstances (which is often the case for neural recordings).

2.3.2. Alignment

A spike cannot be detected if it has not occurred yet, so often the detection point lies somewhere halfway through the spike's waveform. In order to properly sort the spikes, the detected spikes need to be aligned in order to capture its entire waveform. Properly aligning spikes also increases the consistency of the remaining two steps. However, given the fact that this step is often overlooked in terms of optimisation and is barely mentioned in the literature, its total influence on a spike sorter system is significantly less compared to the other three steps. Jongkil *et al.*[53] mention that the detected spike is aligned by including the 20 neural samples before the point of detection. Another way to align detected spikes is by looking at the peak of the spike and using this as a reference point for spike alignment [80, 92]. Valencia and Alimohammad propose a system that also uses the peak of the detected spike as aligning point. Their proposed design saves 24 samples before the spike's peak as well as 48 samples following the spike's peak (capturing the spike's waveform in 72 samples total) [25]. A more methodic procedure to determine the alignment point is proposed in [46]. The alignment point of a detected spike is found by applying a centroid filter to the recorded neural signal. The main idea is to view a spike's waveform as a polygon with the time on its x-axis and its amplitude on y-axis. By finding the centroid of the surface of the spike, the optimum alignment point is found. The centroid filter can efficiently be implemented, relatively speaking, in hardware as an 8^{th} order *FIR* filter.

2.3.3. Feature Extraction

When a detected spike gets aligned and fed to the classification stage, the set of neural samples in the captured spike waveform that the classifier uses can be regarded as the feature set. The size of this feature set is rather large and spike sorting systems could benefit from reducing the feature set in order to alleviate the computational load on the spike classifier [53]. Generally speaking, it is beneficial to a spike sorter's efficiency to implement an additional step where certain features of the detected spikes are extracted (hence the name of the third step: *feature extraction*). The spike classification is then run on this reduced feature set (i.e. dimensionality reduction of the spike). Extracting features from spike often boils down to a trade-off between effectiveness (how distinctive are the features) and cost-efficiency (the complexity of the feature extractor). A simple feature extractor which results in a feature set with little variation, offloads the computational load to the classification step, as the classifier needs to work with input that is harder to classify. A complex feature extractor will ease the computational load on the classifier but will require more resources. Therefore, a variety of features can be used, such as looking at the spike's integral or (double) derivative [17].

For offline spike sorting, the most used method for feature extraction is by applying a *Principal Component Analysis* (i.e. PCA) to the detected spikes. Generally speaking, the first and second principal component scores are used as feature set [17, 34, 53]. The main downside of this method is that it is computationally heavy and not suitable for embedded applications. Due to its resource requirements, it is not used for real-time (embedded) spike sorting. However, some systems aim to overcome these obstacles by reducing the complexity of a PCA. For example, neural networks have been trained to approximate the correct principal components given certain input spikes [35]. This results in an online spike sorting system that is more resource-efficient relative to its traditional counterpart (i.e. an actual PCA) without compromising performance. It should be noted that this PCANN is part of a larger neural network that also does clustering (i.e. classification), so it is not clear if the PCANN could be used as a standalone module.

An alternative way to reduce the feature set is applying a DWT to the detected spikes (see 2.3.1 and using the resulting coefficients as a feature set [65]. This VLSI implementation is able to perform online spike sorting and is suitable for embedded systems. The reduced feature set (i.e. the DWT coefficients) is used to assign a *neural fingerprint ID* to each neuron. These fingerprints are stored in a look-up table, which is then used to point to the responsible neuron of the detected spike. DWT is a suitable method for neural signals with high-noise content [65], but cannot generalise well. Hence the authors of [65] report an *overall* spike sorting accuracy of 84.5%.

Another feature that can be used for spike classification is the double derivative of the detected and aligned spike as proposed by [25]. The double derivative of each neural sample is not calculated continuously. Rather, the double derivative of the neural is calculated periodically and thus reducing the feature set. This is a cost-effective solution for feature extraction since it is relatively simple to calculate a number of double derivative values. Because of this, the VLSI implementation of their design is able to handle 64 channels of neural data, while only consuming 130 μW of energy. The authors elaborate on the double derivative idea by first decomposing the detected spike over seven frequency bands by filtering the spike[92]. The double derivative of these seven bands is then calculated and the extrema of these double derivative values are used as a feature set.

One could extend the double derivative idea by also taking the first derivative into account. In [52], a feature extraction method is proposed that looks at the first and second derivative of a detected spike and uses the extrema as spike features. The shape of spikes from individual spikes remains relatively constant over time (both in amplitude and width), under the assumption that the recording electrode stays fixed [52]. Taking this, together with the fact that each neuron has a distinctive spike, into account, the first and second derivatives of a spike waveform are excellent candidates for a (reduced) feature set. What adds to this is the fact that it is relatively cheap to compute these values, since only requires some registers and an adder. [19] shows that it can even be done in the analog domain, although this would require some additional steps if the rest of the spike sorting system operates in the digital domain.

Feature extraction does not necessarily mean that one needs to extract the feature indirectly (e.g. applying a DWT to the spike). For some features, it suffices to process the spike and extract the feature directly. For example, [89] describes a full spike sorting system. In which feature extraction happens by filtering the detected spikes through a band-pass filter. The peaks of the spikes are the reduced feature set. The parameters of the filter (i.e. the passband) have been determined through spectral analysis of real neural recordings. The pass-band is chosen such that the influence of the noise is minimised and the peaks of the spikes form a distinctive enough feature set. This is a simple method to reduce the feature set, but it might be subjected to the conditions of the data and specific applications might need recalibrating the filter's parameters.

Another popular feature set is by looking at the zero-crossing time interval[22]. The authors of this paper propose using two of these time intervals (named zero-crossing features 1 and 2, i.e. ZCF1 and ZCF2) as the reduced feature set. These two-time intervals are defined as the time interval between the first and second zero-crossing after the detection moment and the time interval between the second and third zero-crossing after the detection moment. This method is simple to implement because it only requires detecting a change in the sign bit of the neural samples and a counter to determine the length of the time interval. The authors recognise that determining ZCF1 and ZCF2 is computationally cheap but also offloads the computational load to the classification phase. However, they also mention that it outperforms more conventional methods such as PCA.

Lastly, in [72] a framework for on-implant spike sorting is presented. The framework is split up into an internal and external part. The internal module performs unsupervised spike sorting and the external module calculates the parameters used by the internal module for online spike sorting, this means that the spike sorting method is split up into two parts: an offline training phase on the external module and the main spike sorting phase on the internal module. What sets this system apart in terms of feature extraction is the fact that different features are considered and the most salient (i.e. the feature which stands out the most) is used. The proposed method is compared with a variety of feature extraction methods normally used and outperforms them significantly.

2.3.4. Classification

The last step in the spike sorting step is to classify which of the (known) neurons is the source of the detected spike. A graphical representation of this last step can be found in figure 2.9. Most spike sorting systems are designed to make a distinction between three and five individual neurons [66], making it a multiclass classification problem. One way to solve such problems using traditional mathematical means (i.e. not using machine learning techniques) is by forming clusters inside the reduced feature set, where each cluster represents a neuron. A popular clustering algorithm used in spike sorting is the k-means clustering algorithm (or variations thereof, such as moving centroid k-means clustering) [66]. The main advantage of this algorithm is that it is simple and fast, however, its main drawback is that it requires supervision. This is another way of saying that the user of the spike sorting system needs to specify *k*. This is a design trade-off when it comes to spike sorting systems. Supervised classifiers are superior in terms of performance, however, it requires extensive effort to determine the number of neurons that need to be sorted. On the other hand, unsupervised classifiers are able to learn how many neurons are contributing to the recorded neural signal, with their main drawback being that they generally perform poorly relative to their supervised counterparts [17]. Generally speaking, good-performing spike classifiers tend to use a lot of hardware resources and are therefore not suited for embedded deployment [66]. One solution to this problem is to offload the classification stage to an offline post-processing step [25]. The embedded device will only detect and align the spikes and save the extracted features. These features are downloaded after the experiment has been conducted for classification. However, this solution is not feasible if the entire spike sorting system needs to run in real-time in an embedded environment. Traditional spike classifier methods leave little headroom in terms of resource optimisation [66]. More and more spike sorting systems are, therefore, discovering the merits of applying deep neural network techniques to the classification step of spike sorting, because these neural network classifiers have the potential to be scaled down without compromising for performance [20, 55].

Classifying spikes does not require a complex neural network necessarily. Kim and Kim [28] proposed a spike sorting system that uses a *multilayer perceptron* (i.e. *MLP* classifier (see 2.4.1) in conjunction with a NEO-based spike detector. Their classifier uses only one hidden layer containing 10 neurons, which keeps

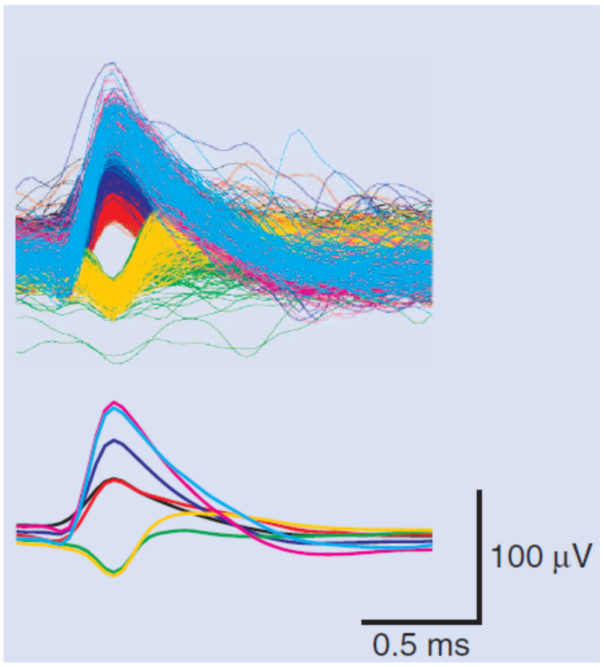


Figure 2.9: Visual representation of clustered spikes. There are five clusters, each representing a spike. The picture contains multiple numerous spiking events from each neuron (top). The average spike waveform is depicted in the bottom part[17].

to network's size in check sufficiently well for it to be suitable for embedded deployment. The spike sorting design they proposed is suitable to run on an FPGA and its classification accuracy is over 90% for several datasets containing neural recordings with varying SnRs. Similarly, in [85] a spike classifier is proposed which is also based on an MLP neural network. This network is trained in such a way that the overlap in the spikes' feature map is minimised, which in turn results in better classification results. The network's size, however, significantly increases as the number of hidden layers is set to two, containing 600 and 60 neurons per hidden layer respectively.

Zhu *et al.*[97] propose another neural network-based spike classifier that also uses an *MLP* architecture. They altered the network's activation function to an exponential activation function. Because of this, the network is easily trained and tends to generalise well for classification problems such as sorting spikes. The network was implemented on both Matlab and an FPGA-based implementation. It was found that the FPGA-based implementation was able to correctly identify spikes 44.37 times faster than the Matlab script, without compromising for performance. This makes the design suitable for real-time operation. An overall classification result of 93.83% was found, using 32-bit floating-point weights and activations.

The feature extracting neural network from Li *et al.*[35] has already been treated in 2.3.3. Their spike sorting system is comprised of two neural networks (in a hybrid configuration). The second neural network (i.e. the classifier) takes the output of the *feature extracting* PCA Neural Network and performs clustering. This neural is a so-called *self-organising map* (i.e. SOM) neural network which allows it to do clustering without supervision. Therefore, no training stage with a predetermined number of clusters is necessary. Both neural networks (i.e. the PCANM and SOM NN) are merged into one neural network, which is the reason why the authors refer to it as a hybrid neural network. Their design is able to correctly classify spikes 97.91% of the time [35].

Valencia and Alimohammad [80] propose a combination of NEO-based spike detection and neural network-based spike classification. In order for their design to run in an embedded environment as an ASIC (i.e. *application specific integrated circuit*), they propose to apply an optimisation technique called binarization to their neural network classifier. By doing so, the weights and activations of the neurons are binarized by turning the full precision weights/activations into one-bit values, effectively reducing the memory and computational load significantly. This resulted in an ASIC design that only uses 0.33 mm^2 of chip real-estate in a 0.18 μm technology while consuming 2.02 μW energy from a 1.8 V power supply and using a 24 kHz clock.

All the neural network-based spike classifiers which have been discussed up to this point use the MLP architecture (also called *fully-connected* neural networks), which is a relatively simple type of neural network.

More complex neural network architectures are usually beneficial for performance, albeit with an increase in computational load [79]. Rácz *et al.*[63] propose a spike sorting system that uses a combination of *recurrent neural network* (i.e. RNNs) and *convolutional neural networks* (i.e. CNNs) to detect and classify neural spikes. The spike detection neural network has already been discussed in subsection 2.3.1 and this neural network consists of a combination of an RNN and CNN to detect spikes. An additional CNN is utilised to perform the spike classification. Given the size of the system, it is not suitable for embedded operations. However, the authors recognise that several issues need to be addressed in order for their design to be suitable for a functional real-time BCI. The aim of their proposal was to show the benefits of using deep learning techniques and stepping away from the current spike sorting paradigm.

The issue with using neural networks for spike sorting (i.e. from detection to classification) is the fact that the neural network will be active all of the time, while the occurrence of spikes is relatively rare. This means that the spike sorter system is using energy while it could remain idle. Bernert and Yvert try to solve this problem in their proposed SNN-based spike sorting system. The network is continuously supplied with neural recording samples, which are fed through several layers to identify the activity of neurons. In order to cope with energy loss during the interspike period (i.e. the time interval between two consecutive spikes), special "attention neurons" are used in the network. These neurons can have an inhibitory effect on the remaining neurons, which inhibits their activity during the interspike time interval. These neurons do not detect spikes themselves, that is part of the functionality of the rest of the system. They are merely able to detect possible spiking activity in the recorded neural signal and they "tell" the rest of the network to pay attention by not inhibiting the rest of the neurons anymore. The network is trained in an unsupervised manner and is, therefore, a suitable candidate for embedded deployment, since no prior knowledge of the target neurons is required. The network was assessed to see how well it performed and an overall classification accuracy of 85% was reported. This shows that the use of SNNs for spike sorting is a promising technique.

However, one of the main issues with the use of SNN classifiers is the fact that they are generally hard to train. [54] aims to solve this problem by implementing an SNN spike classifier that is equipped with a K-means supervised training engine. This combines traditional clustering techniques with 3^{rd} generation neural networks. The SNN has three output neurons, which is based on the maximal number of separate neurons (i.e. number of classes). The K-means training module can periodically be activated in order to improve the classifier's performance and it can be shared among several spike classifiers if the system is scaled up. The spike sorting system has been implemented in 90 nm technology and the SNN spike classifiers use ~40 nW for one classification operation, which is approximately 4 times less than the k-means classifier used to train the SNN.

2.4. Neural Networks

Neural networks are a field within artificial intelligence, where biological neural circuits are mathematically modelled. This section aims to provide the reader with a basic understanding of neural networks. The basic topology of a neural network is discussed, as well as a more detailed description of the neurons in the network. Lastly, the algorithm used to train the network is discussed and, additionally, some important parameters for training the neural network are presented.

2.4.1. Network Topology

Neural network topologies vary greatly in structure and complexity, each with its own merits regarding the context of their application. Many of these topologies are beyond the scope of this work and therefore not treated. This subsection will focus on perceptron neural networks only, since they fit the application context of this work. Perceptron neural networks are considered the simplest of all feed-forward neural network topologies and come in two variants: the single-layer and multilayer perceptrons [62]. In case of single-layer perceptrons, the input neurons are directly connected to the output neurons and no additional steps are performed. As the name suggests, multilayer perceptrons contain additional neurons in between the input and output neurons. These additional neurons are said to be contained in hidden layers as their output is only considered internally. The perceptron neural network is a mathematical model, modelled after neurons in the brain. Instead of electrical pulses, they work with numerical values. Like mentioned before, neurons are arranged in layers, where the outputs of all neurons from the previous layer form the input for each neuron. Hence, the name fully connected neural networks is a more common name for single and multilayer perceptrons. A graphical representation of a fully connected neural network can be found in figure 2.10.

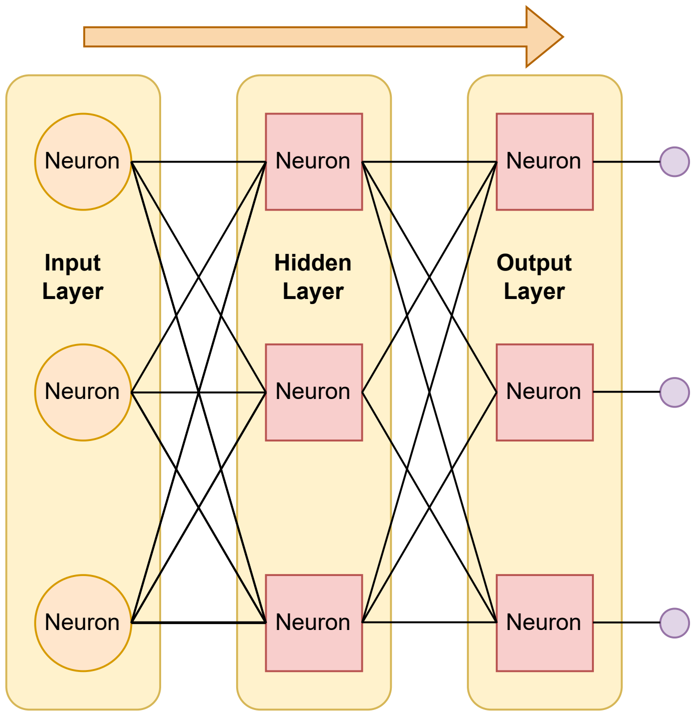


Figure 2.10: Example of an MLP with one hidden layer and three neurons in each layer.

2.4.2. Artificial Neurons

In the previous subsection, the basic network topology for fully connected neural networks (i.e. FCN) was discussed. This subsection is dedicated to what actually happens in each artificial neuron during operation. As mentioned before, all neurons in an FCN are connected to each other in order of layers. Each neuron outputs a numerical value (called the post-activation output) which gets sent to all the neurons in the next layer (in case it does not concern the output layer). So each neuron will collect all the outputs from neurons in the previous layer and multiply each output with a certain (numerical) weight and accumulate the result in one value. Each neuron has its own bias which it will add to the *MAC result* (i.e. multipy and accumulate). The resulting value is called the pre-activation output. These weights are updated in the training phase of the neural network in order for the neural network to behave as expected given certain input. The neuron then proceeds to generate its eventual output by applying an activation function to the result of the MAC operation. A common activation function is the ReLU (i.e. *rectified linear unit*)[5], which causes the neuron to output 0 if the MAC result is less than 0 and the MAC value itself otherwise. Another popular activation function is the *sigmoid* activation function[5]. This activation function is a popular choice for binary classifiers as it outputs the likelihood of the correct classification. Graphical representations of both activation functions can be found in figure 2.11.

2.4.3. Backpropagation

Updating the weights and biases of a neural network is done via a process called backpropagation, which is a supervised learning algorithm [5]. It is used to optimise the weights and biases of the neurons in the network in order to minimise the error between the predicted output (i.e. the output of the neural network) and the ground truth (i.e. the labels). A loss function is used to quantify the error (or loss). Backpropagation works by computing the gradient of this loss function with respect to the weights and biases in the neural network. This gradient is then used to update the weights and biases in the opposite direction to the gradient, effectively reducing the error. Backpropagation is based on the chain rule of differentiation and is used to compute gradients for each layer of the network in a backward direction. It starts from the output layer and then computes the gradient for each subsequent layer in a recursive manner until the input layer is reached. The gradients are then used to update the weights and biases using an optimisation algorithm, such as gradient descent.

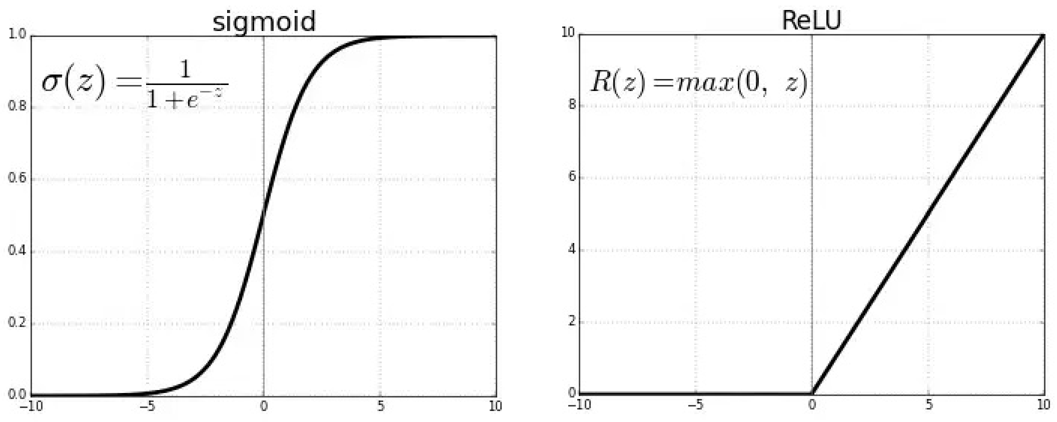


Figure 2.11: Visual representation of the ReLU and sigmoid activation function[64].

2.4.4. Training Neural Networks

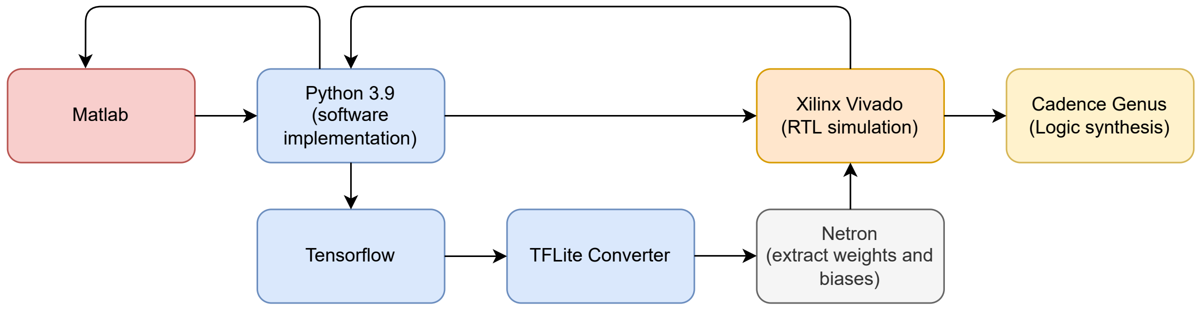
A neural network is trained using a dataset containing samples of input data together with the desired outcome (called labels). The dataset is split up into two datasets for validation purposes. This will result in a training dataset and a validation dataset. This train/validation split generally favours the training dataset, often used ratios (called the training ratio in this work) vary between 70:30 and 80:20. The dataset needs to be sufficiently large in order for the network to generalise well, meaning that the network will perform well for any given input (within reason of the context) and not just converge to perform well for the training samples. The verification is therefore not used for training, in order to see how well the network responds to unknown input.

The network uses the training samples several times to update the network's weights and how often the network iterates over the training dataset is referred to as the number of epochs (or simply epochs). The number of epochs influences the performance of the network up to some point, as the network will eventually converge to a stable state and no significant changes in performance will be achieved. The ideal number of epochs greatly depends on the complexity of the network and is therefore considered a design choice. Lastly, each training sample will result in some error which can be used to update the network's weights. This can be a cumbersome process when the network's training dataset is sufficiently large as a great number of training samples needs to be considered. Often it suffices to accumulate the error over several training samples and update the weights using the average error value. This greatly speeds up the training process without compromising performance. A common way to do this is to update the weights after a batch of input samples has been processed. The amount of input samples in this batch is referred to as the batch size.

3 Proposed Spike Sorter Design

3.1. Design Approach

Given the fact that applying spike sorting principles to the (detection and) classification of the neural activity of Purkinje cells is a novel idea, implementing and testing the system required a certain level of flexibility to allow for easy changes and inevitable bug fixes. Secondly, the data which was used to verify the functionalities of the system came in such a format, that it would need much more processing in order to be useful. These are the main two reasons why the system was first, in its entirety, implemented in software. After verifying that it worked properly it was then implemented as a hardware design using VHDL. Verification of the hardware design was done by checking if its output (obtained through RTL simulations) matched the output of the software implementation.
Another example of the usefulness of this approach is the fact that it was relatively easy to construct and train/test various neural network topologies to see which one would be most suitable. After deciding upon a neural network classifier, training the neural network in software was also relatively easy. The obtained weights can be used in a hardware implementation. The purpose of the hardware implementation is to do inference only, so it does not feature the required subsystems to allow for training and updating the weights. Moreover, the used machine learning libraries are extensively optimised and well-documented, so implementing all required parts from scratch (e.g. algorithm for training the neural network) will (very) likely result in a non-optimal solution.
Based on the two-tier spike sorting approach, certain flexibility with this design approach was possible. Specifically, this meant that work on the hardware description of the spike detection module could already start while the software implementation of the spike classifier module was not yet finalised.



Matlab
Python 3.9 (software implementation)
Xilinx Vivado (RTL simulation)
Cadence Genus (Logic synthesis)
Tensorflow
TFLite Converter
Netron (extract weights and biases)

Figure 3.1: Flow diagram of the used toolchain.

The software implementation of the spike sorter was implemented using python, because of its ease of use and well-supported libraries (especially in regards to the neural network-based classifier). The data used to test and verify the system was formatted in Matlab and saved in a Matlab-based file format (which could be handled by a Python library). The hardware implementation was written in VHDL and simulated using the Xilinx Vivado software. An overview of the used toolchain can be found in figure 3.1. As mentioned before, Matlab is used to pre-process the data. Python is used to implement the detector and classification module in software. The detector module does not require additional tooling. When successfully implemented as a Python program, it can be translated to a hardware description in VHDL. This implementation can be simulated using Xilinx Vivado. The classifier needs additional tooling to train the neural network, this is done using TensorFlow. The neural network is then converted to a TFLite model. By doing so, the network's weights and activations get quantized to integers. A program called Netron [41] is used to visualise the neural network and extract the weights and biases. The weights and biases are then used to create a hardware description of the classifier in VHDL. Similarly to the detector, the VHDL functionalities of the classifier are verified using Vivado. Both modules are combined resulting in the complete system. When the complete system is verified using Vivado, Cadence Genus is used to perform a logic synthesis of the design.

3.2. Proposed System Overview

In terms of hierarchy, the resulting work from this thesis will not operate entirely in an independent fashion. This means that it was designed under certain assumptions. The spike sorter will be used in conjunction with another neural activity monitoring system. This system will analyse the LFPs of a different brain section and a (yet to-be-defined) top module will process the data from both systems and handle the timing synchronisation. So the assumptions under which this work was concluded is that an external module needs to process the output of the spike sorter and deal with the clock synchronisation.
An overview of the spike sorter system can be seen in figure 3.2. The two main modules of the system are highlighted by two inner dotted squares. The system is fed with 10-bit input samples and is filtered using a 1st-order exponential IIR filter. The *Non-Linear Energy* operator (i.e. NEO) is applied to the filtered signal and the resulting energy indicator is filtered yet again by another 1st order exponential IIR filter, albeit with different filter coefficients: filter-specific information can be found in section 3.3. The output value of the second filter is then compared with a previously calculated threshold. The threshold calculator[88] is able to dynamically (re)calculate the threshold and minimise the performance loss in case of changes to the input signal (e.g. changes in amplitude). More information on how this threshold is calculated will be provided in section 3.3.2.
In case of the detection of a spike, the next 40 neural samples are saved in a memory block. These 40 samples are then be fed to the neural network classifier which runs the inference on the input spike. The classifier proceeds to output its activation value for further (offline) post-processing steps.

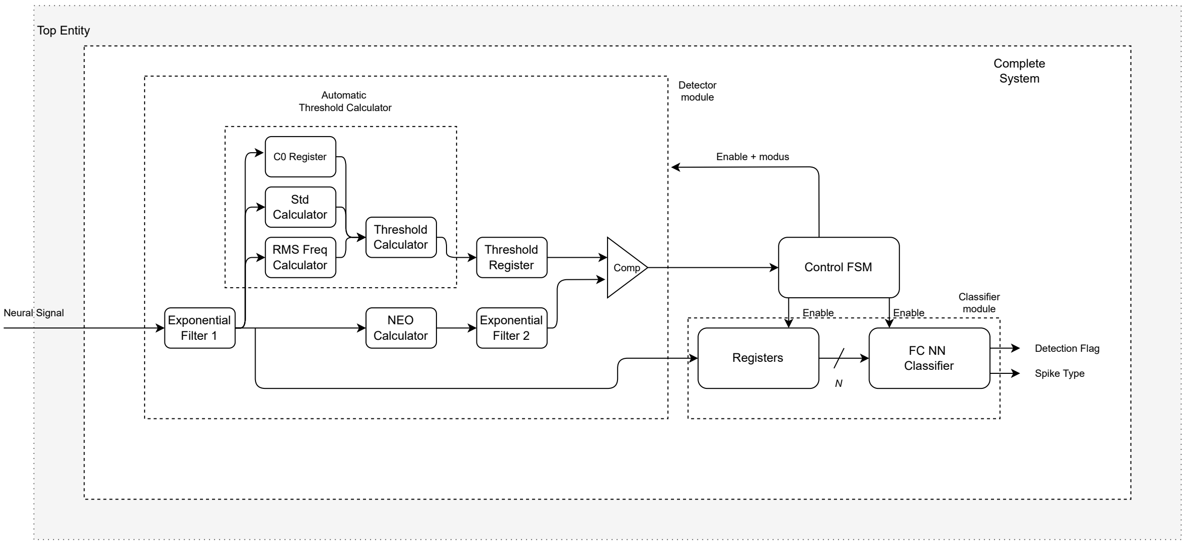


Figure 3.2: General schematic of the spike sorter. The detector and classifier modules can be seen, together with control logic.

The need for constantly running all of the modules in the spike sorter is not a necessity, given the fact that the occurrence of spikes is relatively rare. In fact, once the threshold has been calculated, the only block in the system that runs continuously is the NEO calculator and the comparator. Once a spike has been detected, there is no need to try to detect another one, whilst running inference on the initial spike. This means that there is a "flow" of information through the system which needs some kind of control logic to govern it. For the spike sorter system, a *finite state machine* (FSM) is implemented for this exact purpose and a visual representation of the FSM can be seen in figure 3.3.

The initial state is the *RESET* state, in which all signals are reset to their initial value. This state is not explicitly defined as a state in the hardware implementation and after one clock cycle, the state machine will transition to the *INIT* state. The spike sorter will only be in this state when the system is initially started or reset and the purpose of this state is to calculate a threshold that is used in the spike detection. This means that all other subsystems will be disabled. When the threshold calculator converges to a threshold, a flag will be set indicating that the threshold is ready. At this point, the control block will transition into the *RUNNING* state in which the NEO calculator is enabled and the system will perform spike detection. There is no need to enable the threshold calculator or classifier. This is the state in which the system will be most of the time. When a calculated NEO value exceeds the threshold, a flag will be set, indicating the detection of a spike. At this point, the system will be in the *DETECTION* state and in this state, the next 40 filtered input samples will be saved. It should be noted that the precision of the samples is reduced to 8 bits, in order to facilitate proper classification. This is done by dividing the 10-bit input sample by 4, which practically is performed by only saving the 8 MSBs of the input signal (i.e. a bit-shift to the right by 2). Again, the exact motivation and additional information will be provided in later sections (3.4). The system will always be in this state for 40 consecutive clock cycles, after which it will go to the *CLASSIFICATION* state, in which, the saved spike will be classified. The system will transition back into the *RUNNING* state in order to repeat the whole process.

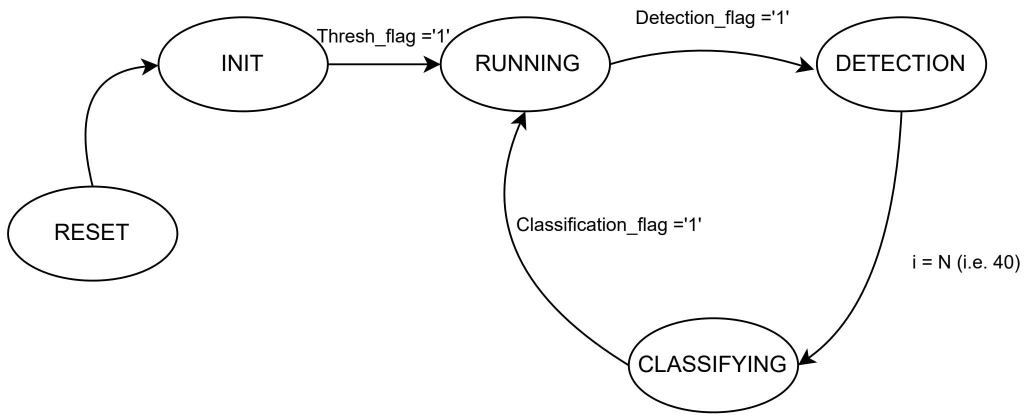


Figure 3.3: FSM graph of the main control FSM.

The main feature of the on-board threshold calculator is to have an adaptive threshold that minimises performance loss when the input signal conditions change. Relatively speaking, it takes some time for the threshold to converge to the right value. So when the threshold is recalculated, it is desirable that the system can continue to detect and classify spikes using the previous threshold. Adding this feature to the control logic will result in a significant increase in the state space and complexity of the control FSM. Given the fact that all this happens inside of the detector module, is opted to implement a separate control FSM for the detection module specifically. The FSM graph can be found in figure 3.4. There are three states in which the spike detector operates. The spike detector can either calculate a threshold, calculate the NEO values of the input signal or do both simultaneously. The initial state of the spike detector is the state in which it only calculates the threshold (depicted as *Threshold_calc* in figure 3.4). After the first initial threshold is calculated, the control FSM will take the system to the regular operating state (called *NEO* in figure 3.4) in which the actual spike detection is done. It is also possible (and recommended) to recalculate the threshold without seizing normal operation. This can be done by setting an external pin. The reason behind this is the fact that the threshold needs to be calculated periodically. As mentioned at the beginning of the chapter, it is assumed that an external top module will handle such timing issues. Hence, the spike sorter can recalculate the threshold per demand.

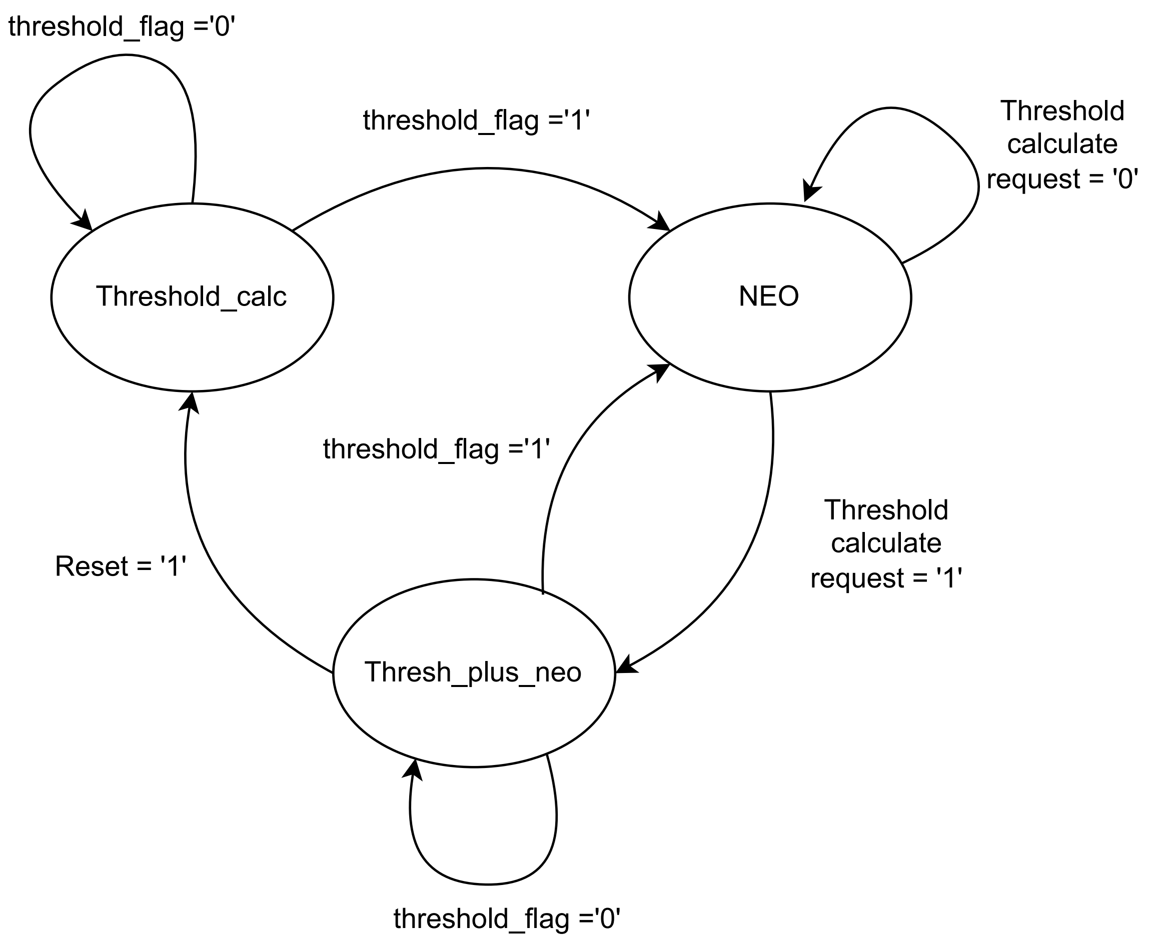


Figure 3.4: FSM graph of the detector control FSM.

3.3. Spike Detection

This section will describe the inner workings of the spike detector module, which consists of two main modules and a comparator. These two modules, i.e. the NEO calculator and threshold calculator, will be explained in more detail in the following subsections. Moreover, the spike detector module contains two first-order infinite impulse response (IIR) exponential filters. The transfer function of these filters is the following:

$$y_{IIR}(n) = \alpha x(n) + (\alpha - 1) y(n-2) \tag{3.1}$$

The first filter in the spike sorter's schematic (figure 3.2) has a value of $\frac{1}{4}$ for α (equation 3.1) and is used to smooth the input signal to mitigate high-frequency noise components which have an adverse effect on the performance of the spike detector. This particular value allows for a low computational burden because it can be implemented by simple bit-shifts and multiplications. The second filter in the schematic (figure 3.2) is used to estimate the expected NEO values ($E[\Psi(n)]$) [88]. Normally, a moving average filter is used for this very purpose, but this results in significant overhead in terms of chip area as several registers are required. In order to minimise this overhead, [88] proposes to use an exponential IIR filter to approximate a moving average filter. Such a filter would only require one register. They suggest a value of $\frac{3}{32}$ for α (equation 3.1) for the best approximation of a moving average filter with length N=15 while keeping the computational burden low (i.e. the use of bit-shifts and multiplications only).

3.3.1. NEO Detector

Spike detection using the non-linear energy operator (NEO) is a relatively simple method of detecting spikes and is also well-suited to differentiate spikes from the background noise. The reason behind this is the fact

that the NEO operator accentuates the instantaneous rise in the signal's energy, which becomes more apparent during a spiking event. It, therefore, seems to filter out the effect of the noise on the recorded signal. Originally it was used to quantify AM-FM modulation in sinusoidal signals, and Mukhopadhyay first recognised its spike detection capabilities in 1998 [47].
By applying the NEO operator (equation 3.2 to the recorded brain signal, one gets the NEO values of this signal. In order to actually detect spikes, a threshold is used (section 3.3.2). A spiking event will occur when the calculated NEO value exceeds this threshold.

$$\Psi(x(n)) = x(n)^2 - x(n-1)x(n+1) \tag{3.2}$$

Equation 3.2 shows the NEO operator $\Psi()$ of neural signal *x(n)*. Because of the second term in the equation, applying the NEO operator requires two registers to save *x(n)* and *x(n+1)*. The basic operation can be achieved by using two multipliers and an adder and is therefore relatively cheap in terms of computational load.

3.3.2. Automatic Threshold Calculator

In order to utilise NEO for spike detection, a threshold is still required for the actual detection. For offline analysis of a neural recording consisting of N samples, this threshold is often calculated using 3.3. This method was first described in [90].

$$Thr_{\Psi(n)} = \frac{8}{N}\sum_{j=0}^{N}\Psi_j(n) \tag{3.3}$$

However, this solution is not feasible for real-time applications, since it requires a large number of samples for an accurate approximation. It is also susceptible to changes in the spiking frequency and can lead to poor performance. Yang and Mason propose a hardware-efficient automatic threshold calculator which adapts to changes in the recorded neural signal [88].

Setting the threshold such that the spike detector is not too conservative or optimistic requires studying the statistical properties of the neural signal. A thorough analysis of these properties can be found in [88], but a brief summary will be provided in this work for completeness.
A recorded neural signal *x(n)* can be considered as a spike train signal (*s(n)* together with background noise *v(n)* as can be seen in 3.4.

$$x(n) = s(n) + v(n) \tag{3.4}$$

Assuming the spike train signal and background noise are uncorrelated, the expected NEO value can be expressed as:

$$E[\Psi(x(n)) = E[\Psi(s(n))] + E[\Psi(v(n))] \tag{3.5}$$

The term which dictates the performance of the NEO-based spike detector in 3.5 is the last one (i.e. $E[\Psi(v(n))]$), so setting the threshold to an optimal value requires studying the probability density function of $E[\Psi(v(n))]$. The noise component of the recorded neural signal can be represented as a weighted sum of K frequency components and can be expressed as:

$$v(n) = \sum_{k=1}^{K} B_k cos(\Omega_k n + \theta_k); \Omega_k = 2\pi\frac{f_k}{f_s} \tag{3.6}$$

Where B_k and θ_k are the amplitude and phase offset of the k^{th} frequency component f_k. f_s is the sampling frequency of the neural signal. When the noise is expressed as in 3.6, $E[\Psi(v(n))]$ can be approximated as [10]:

$$E[\Psi(v(n))] \approx \sum_{k=1}^{K} B_k^2\Omega_k^2 = \sum_{k=1}^{K} B_k^2 * \frac{\sum_{k=1}^{K} B_k^2\Omega_k^2}{\sum_{k=1}^{K} B_k^2} \tag{3.7}$$

The first term in equation 3.7 (i.e. $\sum_{k=1}^{K} B_k^2$) describes the signal's energy in the frequency domain and can be expressed in the time domain using Parseval's Theorem (equation 3.8):

$$\sum_{n=0}^{N-1}[v(n)]^2 = \frac{N}{2}\sum_{k=1}^{K}B_k^2 \tag{3.8}$$

The second term in equation 3.7 (i.e. $\frac{\sum_{k=1}^{K}B_k^2\Omega_k^2}{\sum_{k=1}^{K}B_k^2}$) is referred to as the RMS frequency of the recorded signal and is denoted as Ω_{RMS}^2. This is the weighted sum of the squared radial frequencies of the signal. Combining 3.7 and 3.8 yields:

$$E[\Psi(v(n))] = \frac{2}{N}\sum_{n=0}^{N-1}[v(n)]^2\Omega_{RMS}^2 \tag{3.9}$$

It is generally assumed that the background noise of neural signals has a Gaussian distribution with a mean around zero [34]. Therefore, $\frac{\sum_{n=1}^{N-1}[v(n)]^2}{\sigma^2}$ can be described by a chi-square distribution with N degrees of freedom:

$$\frac{\sum_{n=1}^{N-1}[v(n)]^2}{\sigma^2}\ {}^2(N) \tag{3.10}$$

Using the fact that *v(n)* can be expressed as a chi-square distribution, so $E[\Psi(v(n))]$ can be expressed as a chi-square distribution by substituting 3.9 into 3.10. This yields the following expression:

$$Z^{noise} \equiv \frac{N}{2\sigma^2\Omega_{RMS}^2}E[\Psi(v(n))] \sim \chi^2(N) \tag{3.11}$$

NEO represents the instantaneous rise in a signal's energy and is defined as the sum of the square of the signal's amplitude and the signal's frequency squared [21]. Therefore the threshold Thr_Ψ can be expressed as:

$$Thr_\Psi = C_0\sigma_n^2\Omega_{RMS}^2 \tag{3.12}$$

From equation 3.12 it becomes apparent that the threshold indicates the energy of the signal's noise floor. So in order to detect spikes within a reasonable limit of false detections, $E[\Psi(v(n))]$ needs to be compared to this threshold. This yields:

$$Prob[E[\Psi(v(n))] > Thr_\Psi] \tag{3.13}$$

$$= Prob[E[\Psi(v(n))] > C_0\sigma_n^2\Omega_{RMS}^2] \tag{3.14}$$

$$= Prob[\frac{N}{C_0\sigma_n^2\Omega_{RMS}^2}E[\Psi(v(n))] > \frac{NC_0}{2}] \tag{3.15}$$

$$= 1 - F_{\chi^2}(\frac{NC_0}{2}, N) \tag{3.16}$$

Equation 3.16 shows that this probability can be calculated using the cumulative distribution function of a chi-square distribution (i.e. F_{χ^2}), which is a function of C_0 and N only. C_0 is some scalar indicating the sensitivity of the spike detector to background noise and N is the length of the moving average filter approximating $E[\Psi(v(n))]$. Both values can be determined offline using heuristics functions. [88] found the optimum values for C_0 and N to be 9.5 and 15 respectively.

To be able to calculate a new threshold for spike detection, two things need to be known. The amplitude of the noise (i.e. its standard deviation) needs to be approximated, as well as its RMS frequency. The Fourier transform of the signal is required to calculate its RMS frequency. This operation is computationally heavy, relatively speaking. It has been shown that it can be approximated by calculating the average zero-crossing frequency of the signal [88], which is better solution in terms of hardware efficiency. [88] proposes a simple circuit to calculate the value of Ω_{RMS} by comparing the sign bits of two consecutive samples using an *XOR* gate. Every time the signal encounters a zero-crossing, the XOR will output a *1*. The number of *1's* are accumulated in a register for 2^{N_z} clock cycles. The final output of the RMS frequency calculator is given by dividing the number of zero-crossings (n_z by twice the total number of clock cycles and multiplying this by 2π to get the angular frequency. This yields the following expression:

$$\Omega_{RMS} = \pi \frac{n_z}{2^{N_z}} \tag{3.17}$$

To avoid floating point arithmetic (due to the π term in equation 3.17), a value of *100/32* (i.e. 3.125) is used instead. By doing so, equation 3.17 can in its entirety be calculated by bit-shifts and multiplications only.

Calculating the standard deviation (σ_n) of the noise is slightly more complex. [88] proposes a method to do this in digital hardware without adding significant chip real-estate overhead. They utilise a statistical theory that the probability of Gaussian noise exceeding its σ_n is exactly 0.159. Their proposed system converges to a value of σ_n such that it is exceeded 15.9% of the time by the background noise. This is done by comparing the neural signal to the estimated σ_n (i.e. σ_n^{est}). If the amplitude of the neural signal exceeds σ_n^{est}, then the comparator will produce a *1* as output, otherwise, a *0* will be outputted. A counter will keep track of the outputs of the comparator and count the number of *1s* over M clock cycles. The result from this counter is then compared to 0.159*M, which is the desired value. The estimate σ_n^{est} is updated using the difference between the actual and desired output. The estimated value is updated every M clock cycles until a convergence detector has detected the convergence of the estimated value of σ_n.

3.4. Spike Classifier

After the detection of a spike, the spike waveform is saved for classification. The type of classifier used to do this is a neural network. The following section motivates the design of the neural network classifier in terms of its topology and characteristic features (e.g. activation functions). Two different network topologies are presented and discussed, each with its own advantages and disadvantages. Moreover, techniques will be presented to make the neural network run efficiently in dedicated a hardware. Lastly, two post-processing steps are discussed which aid in increasing the system's performance.

3.4.1. Binary Classifier

Because the spike sorter will be embedded in a neural implant, it is vital to keep the amount of spent resources in check. This often implies reducing the complexity of a system and finding the optimal trade-off between performance and area/energy consumption. The classification problem itself can be regarded as a binary classification problem (i.e. simple or complex spike), so the network needs only one output neuron. Because this spike sorter does not utilise dimensionality reduction, the full feature set is used as input for the neural network (i.e. all the samples of the spike). The type of neural network used was a *multilayer perceptron* [5], i.e. a feed-forward fully connected network (see figure 3.5). This network has an input vector length of forty, features a hidden layer consisting of four neurons, and, as mentioned before, has one output neuron. The resulting 164 weights (signed bytes) are saved using DFlipflops in the synthesised design.

Being a binary classifier, the network is trained on a dataset consisting of two types of spikes, complex and simple spikes. These spikes are captured in waveforms of 40 neural samples. Two examples of training samples can be found in figure 3.6. Normally, neural networks that work on these binary classification problems use the sigmoid activation function, because its output directly corresponds to the likelihood of a correct classification. However, implementing such an activation function is not trivial, as the sigmoid outputs a value between 0 and 1 (sometimes -1 and 1). This implies that floating-point arithmetic should be used. Moreover, the sigmoid activation function is not a linear function (equation 3.18), which adds to the computational complexity. Quite often, this type of activation function is implemented as a lookup table, which might be a considerable burden on the resource budget.

$$f(x) = \frac{1}{1+e^x} \tag{3.18}$$

As can be observed in figure 3.6, the two types of spikes differ significantly from each other. If the network is trained properly, it might be robust enough to still perform well enough while using a simpler activation function. A less complex activation function is called the Rectified Linear Unit:

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise} \end{cases}$$

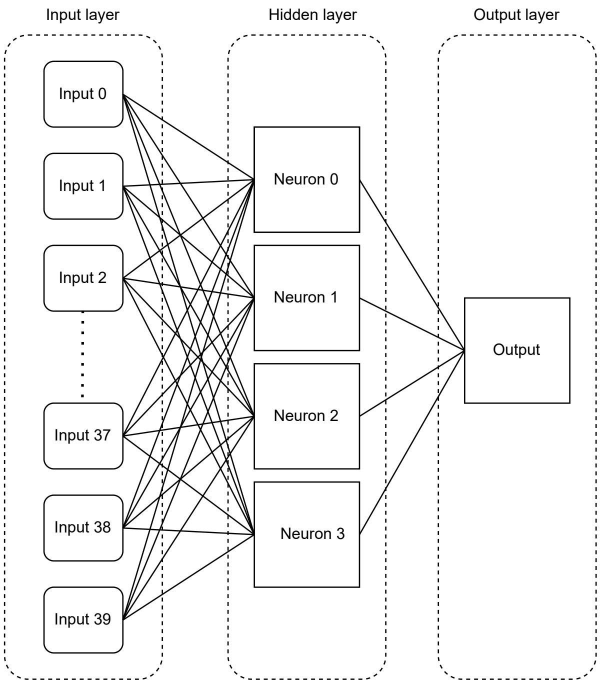


Figure 3.5: Visual representation of neural network classifier with one output neuron.

The use of this activation function significantly reduces the computational load of the neural network and is therefore used in all 4 of the neurons.

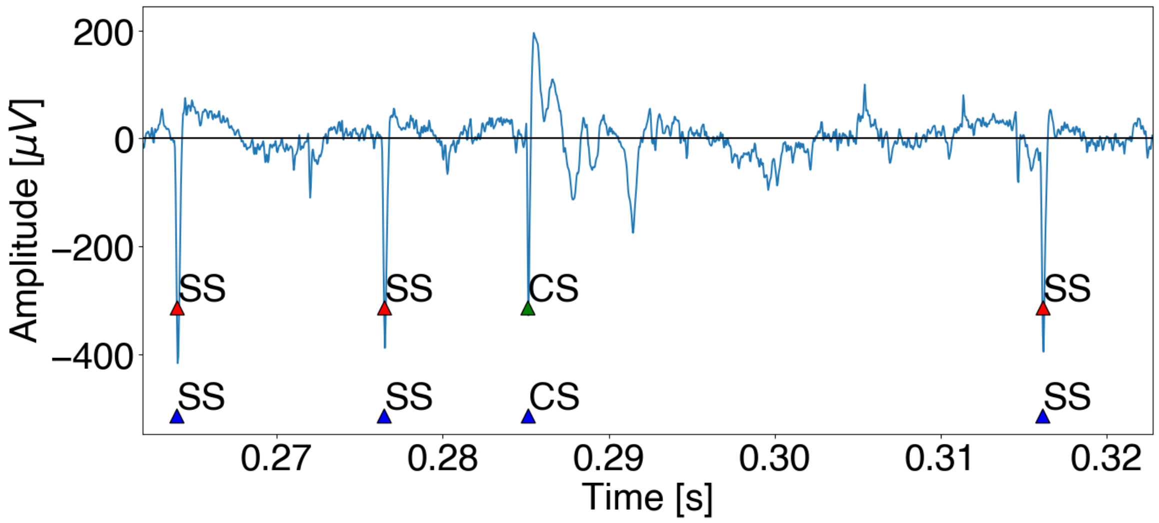


Figure 3.6: Recorded waveforms of annotation (blue) and detected simple (red) and complex spikes (green).

Conventionally, software implementation of neural network uses floating-point number representation which adds to the computational burden of the classifier. Through a technique called quantisation [20], the network is transformed to integer format to increase the efficiency of the arithmetic. The weights and activation values of the neurons are mapped to signed 8-bit (signed) values according to this quantisation scheme[20], where r is a real number, q is an integer and S and Z are quantisation parameters:

$$r = S(q - Z) \tag{3.19}$$

Similarly, biases are mapped to 32-bit signed values. Calculating the output of one layer of a fully-connected neural network is equivalent to performing a matrix multiplication of the weight matrix and an input matrix.

Since both the weights and activations can be expressed in this quantized form (3.19), the output calculation of one layer can be expressed in the following form (3.20) and can be rewritten to 3.21, where M = $\frac{S1S2}{S3}$. Since *S1, S2* and *S3* are all static quantisation factors, *M* can be calculated offline.

$$S_3(q_3^{(i,k)} - Z_3) = \sum_{j=1}^{N} S_1 = (q_1^{(i,j)} - Z_1)S_2(q_2^{(j,k)} - Z_2) \tag{3.20}$$

$$q_3^{(i,k)} = Z_3 + M\sum_{j=1}^{N}(q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2) \tag{3.21}$$

It was empirically determined that *M* is in the interval (0,1) [20] which makes it unusable for integer-only arithmetic. *M* can therefore be expressed in the following form (3.22) which preserves the value of *M* while facilitating integer-only arithmetic:

$$M = 2^{-n}M_0 \tag{3.22}$$

Where M_0 is in the range of [0.5,1) and n is a positive integer. Multiplication with M_0 can be implemented by fixed-point multiplication, while multiplication with 2^{-n} can be implemented by a simple bit-shift to the right [20].

When we take *q1* to be the weight matrix and *q2* to be the input matrix, then *q3* will be the activation matrix (equation 3.21). Since both *q1* and *q2* are in *int8* format, the accumulation of the product of both requires 32 bits. So in order to get 8-bit output values for the output matrix *q3*, the final result in the 32-bit accumulator needs to be scaled down. This is done by performing said fixed-point multiplication and bit-shift (3.22). After scaling down, a saturating cast to int8 is performed, because the result of the scaled-down value might still exceed the upper or lower bound of a signed byte. The ReLU activation function is fused in this saturating cast.

The neural network was trained on a dataset containing a total number of *63504* spikes with an equal distribution of complex and simple spikes. This dataset was constructed using the spike detection module and the neural recordings from the Whisker Lab. The composition of the data structure containing the neural recordings (and labels of spike occurrence) can be found in table 3.1.

	Data
name	Name of recording
wave	Array of sample points
cs	Timestamps complex spikes
ss	Timestamps simple spikes
start	Start time
stop	Stop time
time_vector	Array of timesteps

Table 3.1: Composition data structure.

Because the spike classifier performs inference on the spikes detected by the spike detector, the spikes in the training dataset need to be very similar to the actually detected spikes. The spikes in the neural recording are manually labelled, thus, the exact timestamp can differ from the timestamp provided by the automatic spike detector. Because of this, not all spikes in the datasets were used for training. Instead, the spike detector was used to construct the training dataset for the neural network. First of all, the spike detector returned an array of all detected spike occurrences, and the subsequent 40 samples of all true positive spikes were saved. Lastly, a label in the form of either a *1* (simple spike) or a *0* (complex spike) was added. Because these neural recordings are still in 10-bit format, the whole waveform (not the label) was divided by 4 in order for it to be suitable for the neural network (which requires 8-bit inputs). Even though the occurrence of complex spikes is less frequent than that of simple spikes, the training dataset was balanced by including equal numbers of both simple and complex spikes (called undersampling). This prevents the neural network from being biased too much towards one class in a classification problem such as this one.

Training Parameters Binary Classifier	
Size Dataset	63504
Training Ratio	0.75
Epochs	100
Batch size	10
Loss function	Binary cross-entropy
Optimiser	Adam

Table 3.2: Training parameters binary classifier.

A summary of all relevant training parameters can be found in table 3.2. As mentioned before, 75% of the spikes in the dataset are used for training, while the remaining 25% is used for testing the trained network. The weights of the network are updated over 100 epochs, in batches of 10. The loss function used for training is the binary cross-entropy loss function. As the name suggests, this loss function is eminently well-suited for a binary classification problem. This loss function works well if the target is 0 or 1, which is the case for this classifier. Updating the weights is optimised by using the Adam optimiser algorithm. Adam[29] is an improved version of *stochastic gradient descent* and varies the learning rate in order to converge to a solution faster.

3.4.2. Multi-class Classifier

One of the drawbacks of using a binary classifier in this system is that the classifier assumes that the input spike originates from a Purkinje cell. This, and the fact that the detector detects *false positives*, has a detrimental effect on the system's ability to detect complex spikes. The spike detector will report a relatively low number of false positives under normal operation, relative to the total number of spikes. The total number of spikes mainly consists of simple spikes. Therefore, classifying a *detected* false positive spikes as simple spikes only mildly affects the sorter performance when it comes to simple spikes. Quite the opposite is true when it comes to complex spikes because of their lower frequency of occurrence. Note the number of false negatives generally tends to be lower (i.e. all complex spikes are detected) than the number of false positives.
One solution for this problem is to mitigate the (inevitable) shortcomings of the spike detector by adding an extra class to the classifier. Extending the classifier to also be able to classify false positives, gives the system the opportunity to catch faulty spike detections in a later stage. In order to accomplish this, the network topology needs to be revisited. In case of a binary classifier, one output neuron suffices for both classes. The number of output neurons need to be extended to three, one for each class. The output neuron with the highest activation corresponds to the most likely class.
Training such a network requires a slightly different approach to the binary classifier. Each training sample is labelled using a one-hot vector (i.e. vector where all but one indices have the value zero). This causes the network to converge to a state where the activation of the output neurons is in line with the labels (i.e. 1 for the implied class and 0 for the other two classes). Adding an extra output class also introduces an extra degree of uncertainty to the classification problem, since the likelihood of correctly classifying a given spike decreases. As a result, the overall sorter accuracy might decrease and the overall variation might increase. The drawback of this extra degree of uncertainty might be offset by the fact that the overall *complex spike* sorter accuracy will increase. This makes it a worthwhile trade off.

A new design space exploration was performed (see section 4.1.2), because of the revised neural network topology. Building upon the result from the initial design space exploration (see 4.1.1), a final network was chosen which can be seen in figure 3.7. The choice of the activation function was kept the same, as well as the number of input samples (i.e. 40). An additional layer was added and the number of neurons in the hidden layer was increased to eight. An overview of the training details can be found in table 3.3. Most parameters were kept the same, apart from the number of training samples. In order to preserve the symmetry in the dataset, the number of simple spikes, complex spikes and false positives was kept the same. The number of false positives is the limiting factor in this sense as they are not as abundant as the simple spikes or even complex spikes. Another key difference with table 3.2 is, of course, the choice of loss function. Essentially speaking both loss functions differ only slightly. As the name suggests, *binary* cross-entropy is tailored for binary classification problems, whereas *categorical* cross-entropy is geared towards multi-class classification problems.

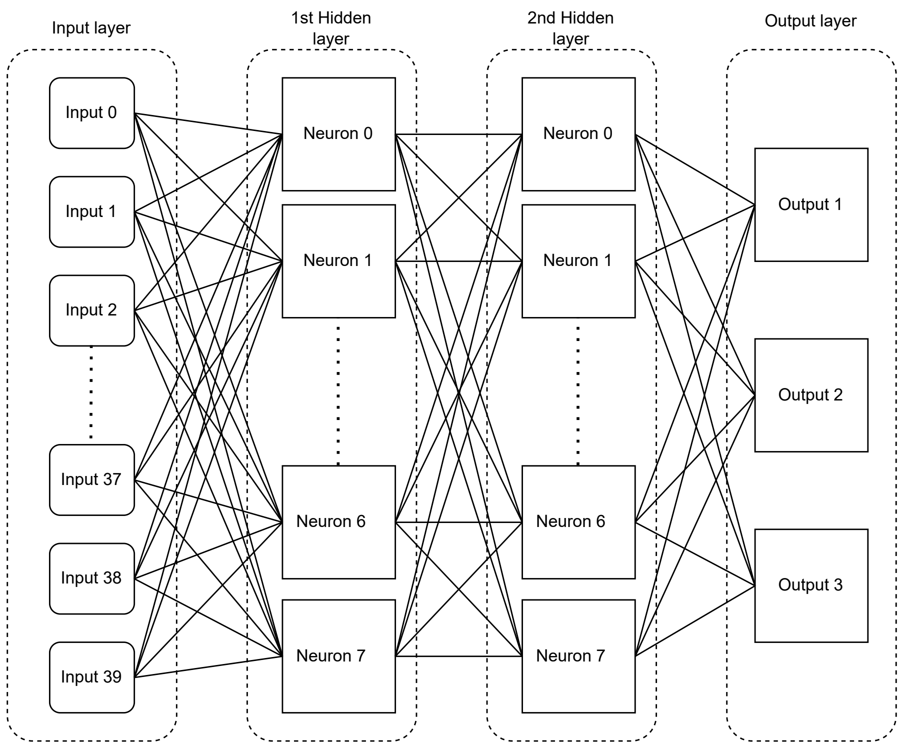


Figure 3.7: Visual representation of neural network classifier with three output neurons.

Training Parameters Multi-class Classifier	
Size Dataset	17854
Training Ratio	0.75
Epochs	100
Batch size	10
Loss function	Categorical Cross-Entropy
Optimiser	Adam

Table 3.3: Training parameters multi-class classifier.

3.5. Post-Processing

Additional steps can be undertaken to improve the performance of the system by post-processing the sorter results. Two post-processing steps are introduced, both of which can be applied offline. The main goal of this thesis is to aid in the data acquisition of Purkinje cell neural activity for research purposes. The post-processing steps described in the subsequent subsections use several parameters. The value of some of these parameters can be calculated, but not in all cases. For those parameters, input from the Whisker lab was needed. Their requirements with respect to these parameters were used to give them a value.

3.5.1. Detection

The first post-processing step aims to mitigate shortcomings in the detection algorithm by introducing a *dead zone* after a detection event. What tends to happen is that one actual spike results in several spike detections, since the calculated NEO value will exceed the threshold several times in a short amount of time. The time interval between these detection events is often in the sub-millisecond domain, which is not possible due to the anatomy of Purkinje cells. Neurons need time to build up the required charge (i.e. ion concentration) to produce a spike[57], which typically exceeds 1-2 ms. Moreover, the likelihood of a neuron firing two consecutive spikes right after one another (assuming it is able to produce a spike of course) is slim. The duration of the detection dead zone can therefore be more than the required time for the neuron to recharge. For this work, a dead zone interval of 2 ms is chosen but this dead zone can also be extended to 4 ms. A small dead zone interval can lead to a larger number of false positives but will result in a smaller number of false nega-

tives. The opposite is true for a larger dead zone interval.
One of the fundamental differences between complex and simple spikes is the way the Purkinje cell produces them. The Purkinje cell uses a concentration of calcium ions to produce a complex spike and a concentration of sodium ions to produce a simple spike[81]. Because the Purkinje cell uses two kinds of ions, no additional time is needed to restore the needed concentration. It is therefore possible to have a complex spike right after a simple spike (or vice versa). Generally speaking, there is still at least 1 ms between the occurrence of one type of spike followed by the other type of spike. This slightly complicates this post-processing step. The solution to this is to break up this post-processing step into two steps further. First off, we discard any additional spike detections which are less than 1 ms after an initial spike detection. The second step is applied after the spikes have been classified. If two spike events of the identical type happen within a 2 ms time frame, the most recent spike event will be ignored.

3.5.2. Classification

The following post-processing step only applies to the binary classifier. As previously mentioned, one of the shortcomings of this approach is the fact that the classifier tries to classify each detected spike as either type. Often when it concerns a false positive (i.e. neither of both spike types), the classifier output will be close to the classification threshold. Ideally, the classifier output of a complex spike is 0 and 1 for a simple spike, corresponding to the training labels. Therefore, the classification threshold is in the middle of this interval and equal to 0.5. To catch false positives, all spike events which are relatively close to this threshold are ignored. Similar to the detection post-processing step, a dead zone is introduced. In this case the dead zone lies within the classification dimension. All spike events within a certain range situated around the classification threshold are ignored. The bounds of this interval are determined empirically by averaging the classification output of all false positive complex (i.e. lower bound) and simple (i.e. upper bound) spiking events. It was found that a considerable amount of simple spikes were falsely disregarded, resulting in a significant rise in false negative simple spikes. The purpose of the post-processing step is to address the negative impact of false positives on the sorter's performance, especially with regard to complex spikes. Given that this does not appear to be problematic for simple spikes, which occur in large numbers relative to the number of complex spikes and false positive spike detections, it is logical to exclude this post-processing step for them. As a result, the upper bound for the dead zone interval is set at the classification threshold of 0.5.

4 Results

In this chapter, the test results of the spike sorter are presented. The system is evaluated in two different ways. First off, the performance of the algorithm is evaluated in terms of its prediction accuracy. Because the spike sorter is comprised of two modules, the detector and the classifier, each of these modules is evaluated separately as well. This is to see which of the two modules influences the performance of the overall system the most. Also, the performance of the original neural network is compared to that of its quantized counterpart. Since the spike sorter will perform its functionalities embedded in a neural implant, the power consumption and on-chip area of the synthesised ASIC design are also evaluated.

4.1. Design Space Exploration

This section describes the results from both design space explorations. The motivation behind the choice of parameters for both design space explorations is also provided. The first design space exploration describes the path of finding the relative optimum network topology for the binary classifier. Later on it was found that the performance of this type of classifier suffered from false positive spike detections. The possibility of extending the classifier to three classes was therefore explored and a second design space exploration was required as the complexity of this neural network varied from the binary classifier neural network. The second design space exploration builds upon the results found in the first design space exploration.

4.1.1. Binary Classifier

In order to explore the effectiveness of using a simple fully-connected neural network, a test network with 32 input samples, eight neurons in the hidden layer, and one output neuron was constructed. The preliminary results were very promising (approximately 95% test accuracy). So in order to find an optimal solution, several other networks were implemented and tested. Because of the restricted resource budget of the system, a trade-off had to be made between the performance of the spike sorter and its resource consumption. Exploring this design space can be time-consuming, because of the many possible combinations which can be explored. To limit the time spent on this, only a few parameters were changed. These specific parameters were chosen, because they have a significant effect on both the performance of the classifier as well as on the resources. The design space exploration was performed by varying said parameters and training the resulting network on 75% of the total dataset. The remaining 25% of the samples were used to validate the network. This test accuracy was then used as an indicator to see how well a certain combination performed.
One of the parameters which was changed is the topology of the neural network. In total, 6 different networks were compared, ranging from a simple neural network without a hidden layer to a more complex network with two hidden layers and dozens of neurons. Generally, the deeper the network and the more neurons there are in the network, the better it performs in terms of (test) accuracy. However, this will also result in a significant increase in weights and biases that need to be stored. The number of stored weights scales exponentially when additional neurons are added to a given layer. This also implies that the consumed area and power consumption will scale with it. Adding layers to the network will cause the number of stored weights to scale linearly, but have a different effect on the performance of the system. Broader neural networks (i.e. with more neurons per layer and fewer layers per network) tend to converge faster. Deeper neural networks (increased number of hidden layers with relatively low numbers of neurons per layer) generally require ex-

tensive and proper training [36, 83]. The number of neurons chosen in the different network topologies are all powers of two, in case some addressing was needed. By choosing a power of two, the full address space can be used. No addressing was eventually needed, so the number of neurons is quite arbitrary in this sense. Different topologies were chosen to represent a design space with networks of varying complexity.// Another parameter that could influence the performance of the classifier is the width of the input waveform. Each sample can be considered as a feature of the spike, so by increasing the feature set, one generally increases the performance as well. However, this increase in performance is bound by the fact that spikes have a certain duration, which is relatively constant. 32 input samples were used for initial testing and the number of input samples was either incremented or decremented by four to establish a design space. Smaller or bigger steps could have been chosen, but steps of four allowed for a variety of design choices without making the exploration unnecessarily long.
After identifying the two parameters of interest, neural networks with different configurations were trained and tested. A summary of this combination with corresponding test results can be found in figure 4.1. All networks were trained over 50 epochs with a batch size of 16. The number of epochs was lowered compared to the training setting of the eventual network, to facilitate faster training. Due to the stochastic nature of the neural network, results might vary slightly when the experiment is repeated. It can be seen in figure 4.1, that increasing the input vector size has a positive effect on the test accuracy up to 40 samples. Increasing the input size beyond this point yields little improvement.

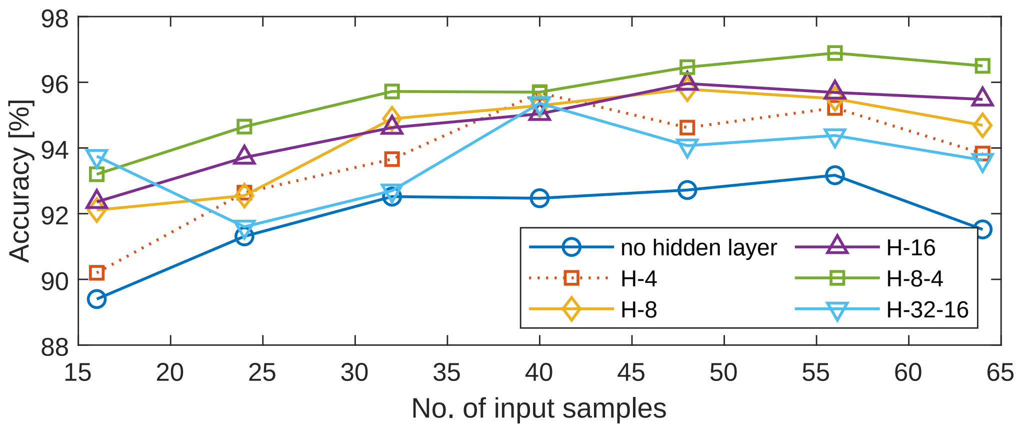


Figure 4.1: Test accuracy of different neural networks as a function of their input vector size.

It can also be observed that the most simple network of the six, the network without a hidden layer (*no hidden layer* in legend), is not able to keep up with the more complex networks. The other five networks all seem to converge to a test accuracy of approximately 95.50% for 40 input samples. In some cases, the test accuracy actually decreases if the input vector size is increased even more. This might be due to the fact that increasing the sample length of the spike might introduce extra sample points that are either not part of the spike anymore, or add little to the distinctiveness of the spike. However, increasing the input sample width still improves the test accuracy somewhat in most cases. This gain in performance is not really worthwhile considering the additional cost in terms of chip real-estate and energy consumption. Based on this, the optimal number of input samples is set at 40. Out of the five networks (not considering the network without a hidden layer since it performs considerably worse), the network with four neurons in the hidden layer (H4 in the legend) and 40 input samples performed the best considering its complexity.
Based on the exploration of the design space, it was therefore established that a fully-connected neural network with 40 input neurons and 4 neurons in the hidden layer, is the optimal solution for classifying neural spikes on-implant. It should be noted, of course, that this is the optimum solution for the design space explored in this work. Another set of parameters might yield different results. Also, if the design space was broadened (i.e. more parameters), the results might have been different. Considering the duration of such an exploration, it might just be too time-consuming for this work. In fact, most of the combinations tried out during this exploration may suffice for this application. In brief, the decision to go for this particular network (FC4 with 40 input samples) is grounded, but there is likely a better configuration in terms of costs/performance.

4.1.2. Multi-class Classifier

The addition of an extra output class in the case of the multi-class neural network classifier gave rise to a second design space exploration. Using a neural network with one hidden layer with four neurons and three output neurons is not sufficient to produce satisfactory classification results. Therefore a new network topology needs to be determined with the aim of minimising the resource overhead and maximising the classification performance. The second design space exploration builds upon the results of the first design space exploration in terms of the width of the input samples. One of the conclusions of the first design space exploration was the fact that enough information about the spikes is captured in 40 neural samples. Each network tested in the second design space exploration, therefore, uses 40 input samples as well. The parameters which were varied in this second design space exploration are the breadth (i.e. the number of neurons per layer) and depth (i.e. the number of hidden layers) of the network. The influence of both parameters on the performance and resource overhead varies as discussed in section 4.1.1.

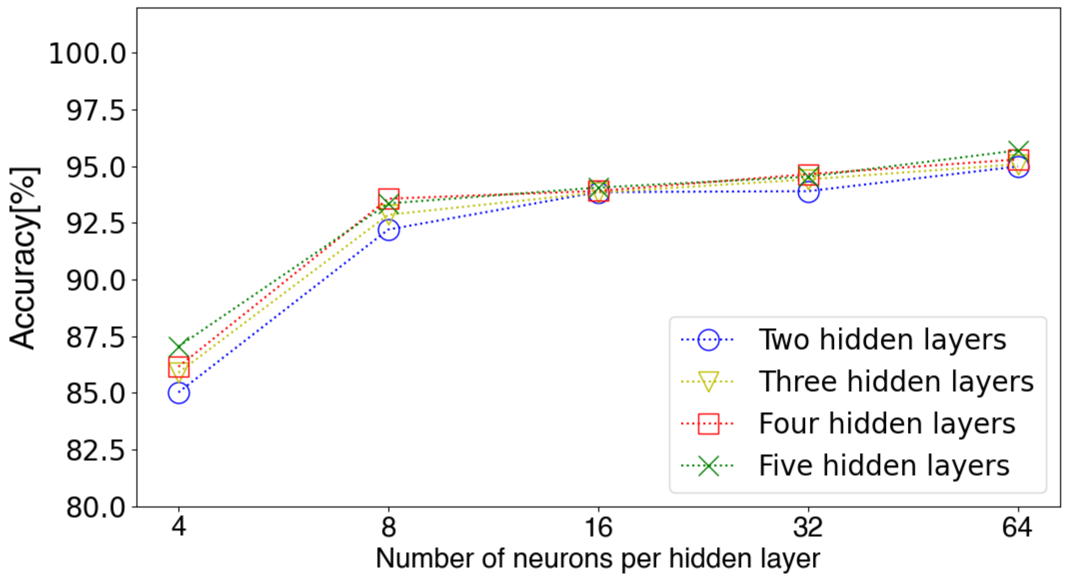


Figure 4.2: Test accuracy of different neural networks as a function of the number of neurons per hidden layer.

The results of the second design space exploration can be seen in figure 4.2. A clear trend can be seen as the number of neurons in the hidden layers increases. A significant increase in performance can be observed when the number of neurons in each hidden layer is increased from four to eight. Increasing the number of neurons per hidden layer beyond this point does not seem to aid the performance of the classifier, as the classification accuracy only increases slightly. It follows the expectation that increasing the number of hidden layers is beneficial to classification accuracy. The differences tend to be greater for lower neuron counts, albeit they are still relatively small.

Figure 4.2 shows that the benefit of adding additional hidden layers does match the additional overhead it poses in terms of resource consumption. Each added hidden layer essentially doubles the number of weights that need to be stored. The choice, therefore, falls on the neural network with the least number of hidden layers explored, which amounts to a network with two hidden layers. It can also be concluded from figure 4.2 that adding neurons to each hidden layer beyond eight neurons is not beneficial to the overall system performance. The total number of weights increases quadratically with the increase of neurons per hidden layer and therefore does not warrant a slight increase in classification accuracy. Similar to the first DSE, the list of parameters and their range that could be explored is extensive. Given, the time constraints of this thesis, only a subset could be explored. From the explored options the neural network with two hidden layers, each with eight neurons, seems to be the optimal solution in terms of performance and resource overhead. This network will suffice in the context of this work, but it is highly likely that a more optimal network exists in terms of cost/performance.

Dataset	**Number of Experiments**	**Complex Spike**	**Simple Spikes**	**Total**
Initial	63	18 130	1 071 778	1 089 233
New	134	37 485	2 317 966	2 355 451

Table 4.1: Table containing a summary of the datasets.

4.2. Evaluation Setup and Performance Metrics

This section will elaborate on the data used to verify the performance of the system. A brief explanation of the evaluation methodology is also provided. This explanation includes a summary of the performance metrics as well as a summary of the test setup.

4.2.1. Data

The spike sorter was tested and evaluated on two datasets, each containing data from several experiments (see table 4.1). The data structure of one experiment can be found in table 3.1. The first dataset contains data of better quality than the data in the second dataset. More specifically, the signal-to-noise ratio is better in case of the first dataset. The noise floor is not known in either of the datasets, however. So this claim cannot be supported by a definitive number. The person who recorded the experiments visually inspected the data and grouped it into two datasets. In order to evaluate the system, it was tested using each of these experiments, which resulted in a number of performance figures. This data has been plotted to show how well the system behaves over different sample sets (i.e. the individual experiments). The first dataset, the one containing the good data, is called the *initial dataset*, because on this dataset the system was initially tested. This dataset contains 63 different experiments from 35 were suitable to be used. The second dataset, containing data of lesser quality, is called the *new dataset* and contains 134 different experiments of which 66 were actually used. The reason why not all experiments were suitable for evaluating the spike sorter, is the fact that the data of some experiments did not accurately depict the situation in case the spike sorter is actually deployed. An example would be an experiment where there is excessive recording electrode drift (some recording electrode drift is unavoidable). Saturated sample points (significantly higher than other sample points) were also frequently encountered, as well as offsets in the data. In this last case, all the sample points received some offset in such a way that the neural signal was not concentrated around the x-axis (the threshold calculator works under the assumption that the noise is zero-mean).

4.2.2. Evaluation Setup

How well the spike sorter performs, is determined by its sorting accuracy. This in turn is influenced by the performance of the spike detector, as well as the performance of the spike classifier. Each requires their own specific method in order to determine its accuracy. If the spike detector can detect all the spikes at the right time, then you will have a flawless system. Such a detection is called a true positive (T_p). It is also possible that the spike detector detects a spike when there is none, this is called a false positive (F_p). A detector which is too sensitive will exhibit a lot of false positives. When a detector is not so perceptive, it will tend to miss a lot of spikes. These are called false negatives (F_n). These three statistics need to be collected in order to calculate the accuracy according to equation 4.1. The classical formula for calculating an accuracy figure also takes into account true negatives (i.e. T_n). During this evaluation, this parameter is omitted. The reason for this is the fact that the number of samples that correctly *not* detected greatly exceeds any of the other possibilities, due to the fact that the sampling frequency is significantly higher than the spiking frequency of neurons.
What is important for this is to correctly determine which of the three categories a detected spike belongs to. This can be determined by using the provided labels in the datasets. These labels indicate the timestamps of the occurrence of both types of spikes. However, the data is manually labelled, so not all timestamps will correspond to the exact timestamp of the detected spike. So in order for a detected spike to be regarded as a true positive, it needs to be sufficiently close to the labelled spike. For evaluating the spike detector, this time interval was chosen to be the duration of such a spike, which is 1 ms on average. So if a detected spike is within 1ms of the labelled spike, it is counted as a true positive. In case a detected spike is not in range of any of the labelled spikes, it is counted as a false positive. Lastly, the remaining number of labelled spikes are counted as false negatives, as they are not detected.

$$Accuracy = 100\% * \frac{T_p}{T_p + F_p + F_n} \tag{4.1}$$

In case of determining the accuracy of the spike classifier, things are less complicated. Test datasets of each recorded experiment are constructed similarly to the way the training dataset is constructed (see section **??**), with the only difference being that the training dataset contains spikes from all experiments of both datasets and each test set contains spikes from only one experiment. These entire test datasets are fed to the spike classifier and the prediction outcome is compared to the label. The accuracy is calculated by dividing the number of right predictions by the total number of spikes in the test dataset. The performance difference between the normal and quantized neural networks is evaluated the same way.
Lastly, these two methods are combined to determine the overall accuracy of the spike sorter. Essentially, the evaluation setup is the same as the evaluation setup of the spike detector. The only difference is that the two types of spikes are evaluated separately, instead of looking at all the labelled spikes combined. A figure of accuracy is still calculated using equation 4.1, where the total number of true positives, false positives and false negatives is the sum of these values for both types of spikes.
As the software implementation of the system allows for more flexibility, the performance of the system is evaluated using the software implementation. Since the hardware design is essentially the same as the software implementation, it is assumed that the evaluation also holds for the hardware design. This is under the assumption that the quantization of the neural network does not degrade the performance significantly.

Synthesising the logic design based on the RTL description will yield some results in terms of area and power consumption, and timing. This will be discussed in section 4.4.

4.3. Accuracy

The performance of the spike sorting algorithm is evaluated by looking at its detection, classification and overall sorting accuracy. First, the two separate parts of the algorithm are evaluated, followed by a performance evaluation of the entire spike sorting algorithm. Because the initial neural network is not going to be included in the physical implementation of the algorithm, the performance difference between the initial and quantized neural network is also treated.

4.3.1. Spike Detection

The detection accuracy of all experiments in the initial dataset and the new datasets are depicted in barcharts as can be seen in figure 4.3 and 4.4 respectively. The first thing that can be observed in both plots is quite some variation between experiments. There are some notable dips in detection accuracy. For example, experiments 6 and 37 of the initial datasets perform significantly worse compared to other experiments. The reason behind this lack of performance is the fact that the detection algorithm detects a lot of false positives. A closer (visual) inspection of these datasets reveal that the detected false positives are in fact spikes from neurons in the background of the recording site. These spikes have not been labelled as activity from a Purkinje cell as they are not. However, the spike detector does not discriminate between neurons and detects spikes if they are there. Proper placement of the recording electrode can mitigate this problem.

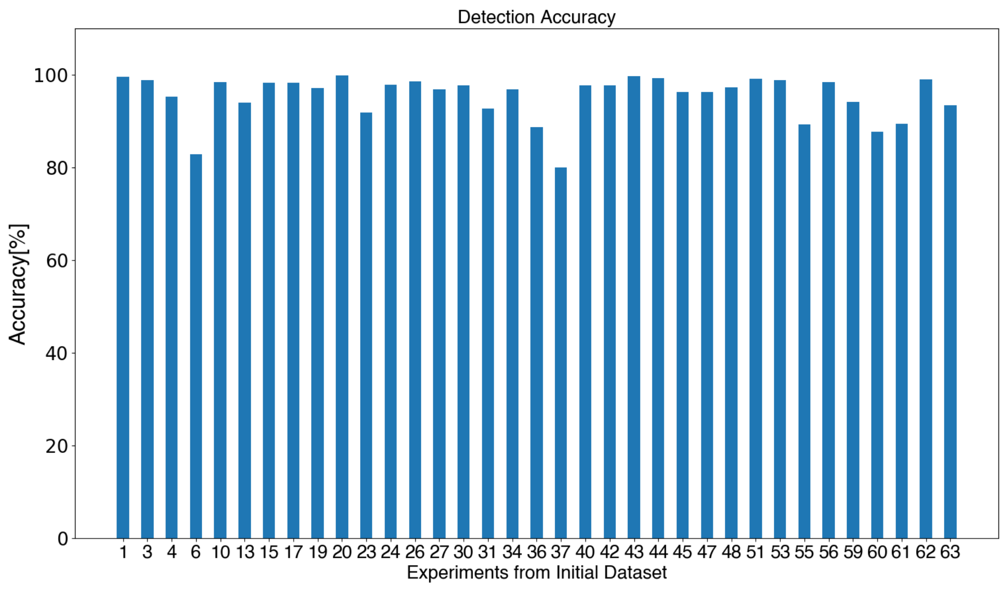


Figure 4.3: Detection accuracy of all experiments in the initial dataset.

In order to get better insights into the performance of the spike detector, it would also make sense to look at the distribution of the detection accuracies of both datasets. Figure 4.5 contains two boxplots corresponding to both datasets. Several interesting observations can be made from these boxplots. The first one being that most of the dips in figure 4.3 are considered outliers in the corresponding boxplot. To a lesser extent, this is also the case for figure 4.4. Another observation that can be made is the fact that there is more variation in detection accuracy when it comes to the new dataset. Intuitively this makes sense as the initial dataset contains neural recordings of "better quality" which facilitates the detection algorithm.

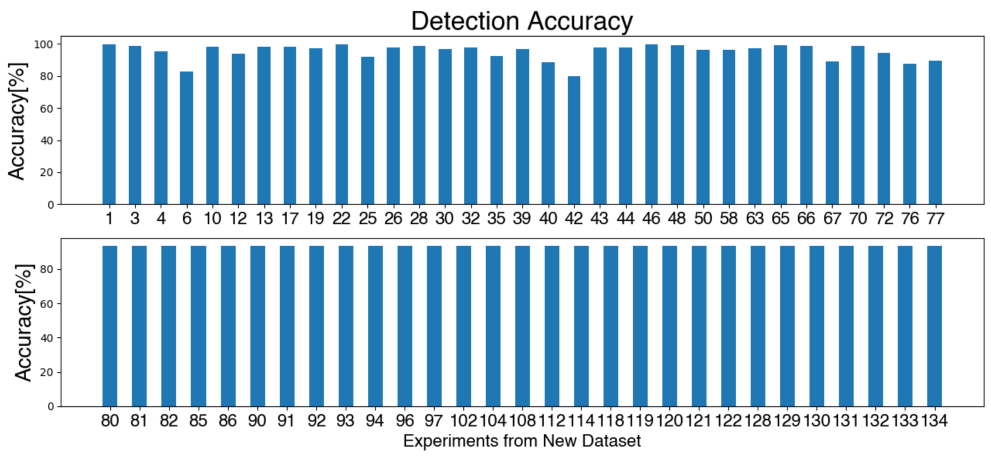


Figure 4.4: Detection accuracy of all experiments in the new dataset.

Another key observation is the fact that there seems to be more variation in the lower 50% of the data in both datasets. This means that the average detection accuracy does not give a clear indication of the performance of the spike detector. This is because the performance data is skewed due to the presence of (unrealistic) noisy datasets. Instead, it would make more sense in this situation to use the median instead. A summary of all relevant metrics can be found in table 4.2 or 4.3.

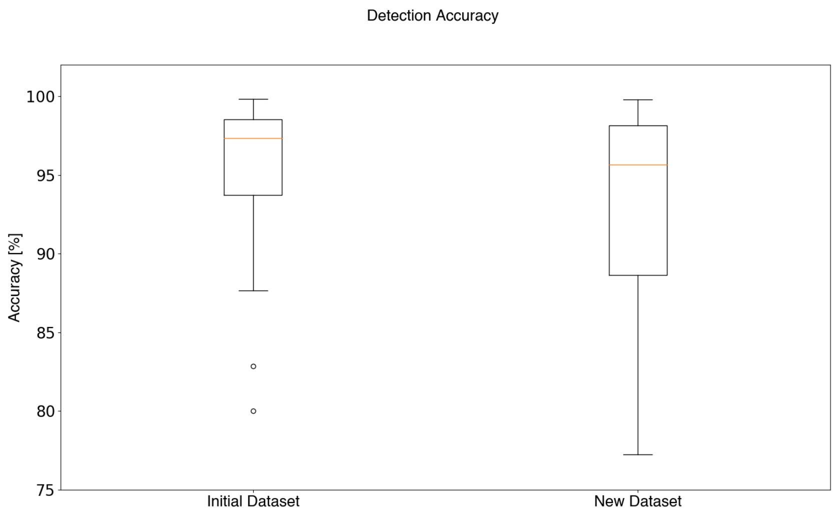


Figure 4.5: Boxplots of detection accuracies of both datasets.

4.3.2. Spike Classification

Classification Accuracy

Similarly to the evaluation of the detection accuracy, the spike classifier was evaluated by seeing how well the classifier performed on every single experiment (i.e. neural recording session). Both the binary and multi-class classifiers have been evaluated. The results from these evaluations are visualised in barcharts which can be seen in figures 4.6, 4.7, 4.8 and 4.9. The first 20 experiments are shown to maintain readability. The complete results can be found in appendix A

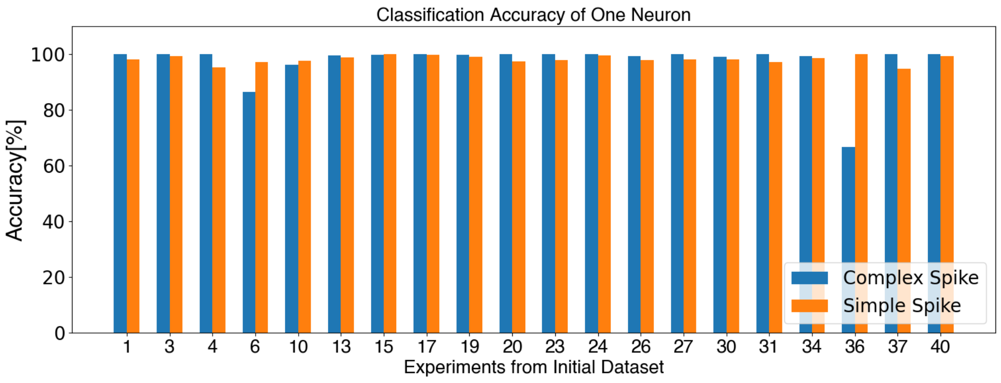


Figure 4.6: Classification accuracy of the first 20 experiments in the initial dataset using a neural network with one output neuron.

It can be seen that the binary classifier is able to identify both types of spikes with at least 95% accuracy in most cases. This figure is a bit lower in case of the multi-class classifier, where most experiments report a classification accuracy of 90% or more. It is interesting to see that some experiments (for example experiments 6 and 36 of the initial dataset) show similar dips in performance when compared to the detection accuracy. However, this does not seem to be the case for the multi-class classifier. There is also a significant difference between complex and simple spike accuracy for some experiments which use the binary classifier, notably experiments 6 and 36 of the initial dataset (figure 4.6) and experiments 6 and 28 of the new dataset (figure 4.8). This does not seem to be the case for the multi-class classifier, which shows great consistency for most experiments (figure 4.7 and 4.9).

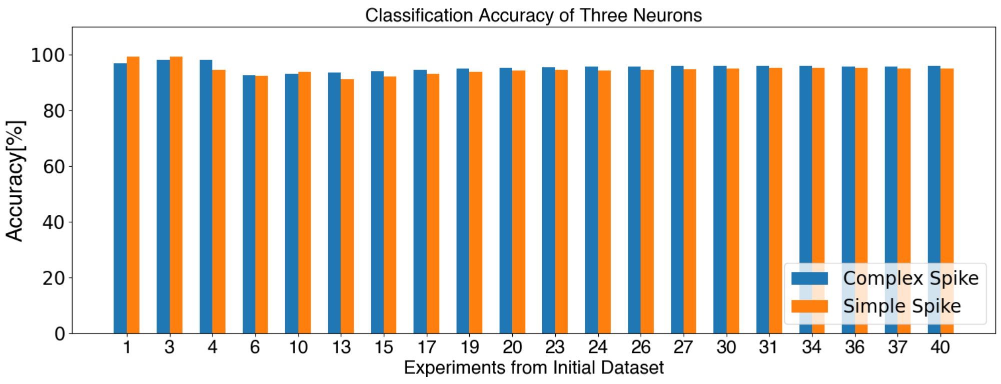


Figure 4.7: Classification accuracy of the first 20 experiments in the initial dataset using a neural network with three output neurons.

Yet again it would make sense to look at the distribution of the classification accuracies to discover other trends in the classification data. Figure 4.10 contains four boxplots showing the performance distribution of simple and complex spikes for both datasets (using the binary classifier). The main takeaway from these boxplots is the fact that the classification performance over all experiments is very skewed. There seems to be a lot of variation beneath the 50^{th} percentile, which seems to be the result of the apparent dips which can be observed in the bar charts (figure 4.6 and 4.8).

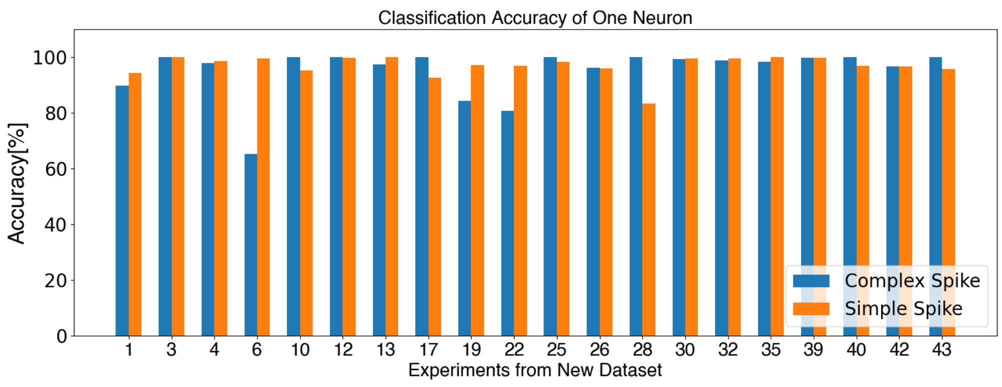


Figure 4.8: Classification accuracy of the first 20 experiments in the new dataset using a neural network with one output neuron.

The boxplots showing the performance distribution of the multi-class classifier show a different story (figure 4.11). Overall, the multi-class classifier can not match the performance of the binary classifier. This makes sense since the classifier needs to deal with an additional class which adds another degree of uncertainty. Generally speaking, the data does not tend to be as skewed compared to the binary classifier. This consistency is also reflected in the bar charts (figure 4.7 and 4.9), as the differences between complex and simple spike accuracies tend to be minor.

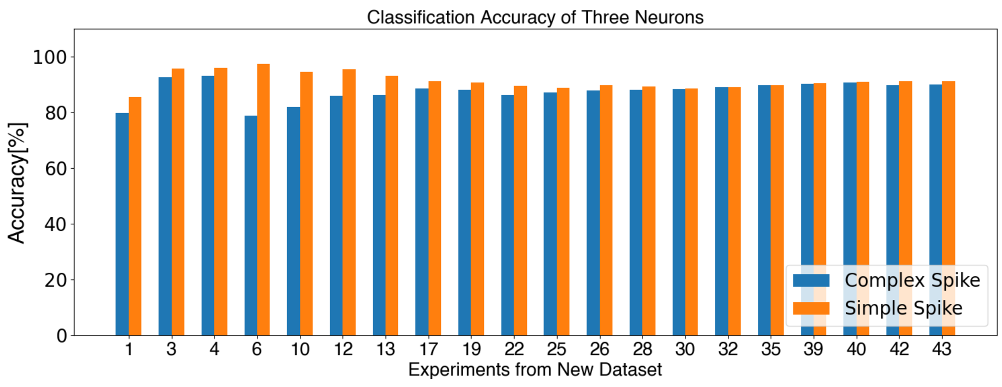


Figure 4.9: Classification accuracy of the first 20 experiments in the new dataset using a neural network with three output neurons.

Due to the classification performance being skewed for the binary classifier, using the mean value gives a pessimistic view of the classifier's performance and the median would be a more suitable metric to give an indication of how well the classier performs. In case of the multi-class classifier, the mean and median are very similar because the data does not tend to show some degree of skewness. In this sense, it is fair to also use the median as a performance metric for the multi-class classifier. By doing so a fair comparison between the binary and multi-class classifiers can be made. In general, the binary classifier outperforms the multi-class classifier by about 5%. However, both classifiers were evaluated using a set of *true positives* complex and simple spikes. The reason to use a multi-class classifier is to mitigate the negative influence of falsely detected spikes in the *whole* system. The setback in raw classification accuracy might therefore be offset by an increase in *overall* performance. Therefore making the trade-off worthwhile. The next subsection will elaborate on the system's overall performance.

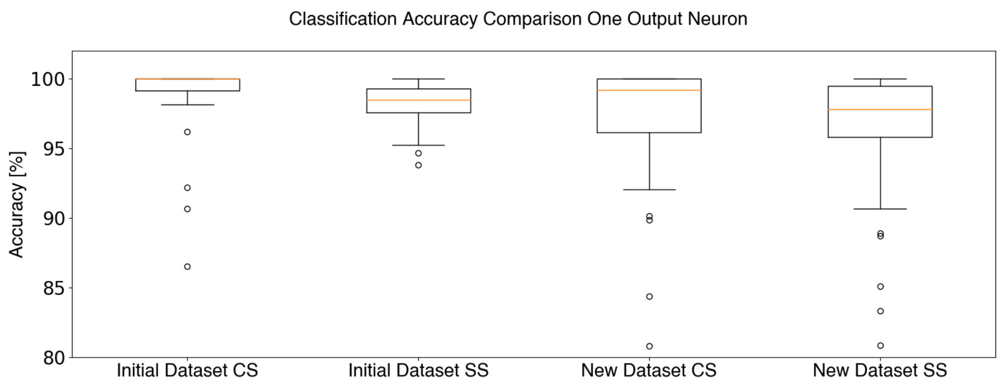


Figure 4.10: Boxplots of classification accuracies of both datasets for the classifier with one output neuron.

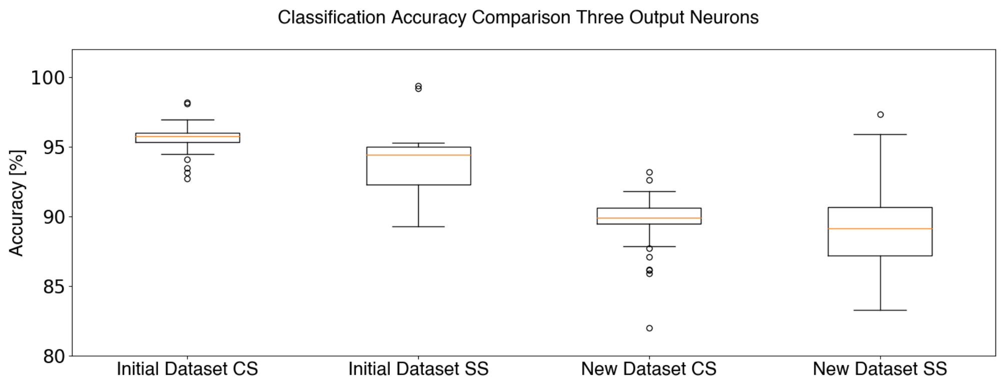


Figure 4.11: Boxplots of classification accuracies of both datasets for the classifier with three output neurons.

Normal vs. Quantised Network

Generally speaking, quantizing a regular neural network has an adverse effect on the performance of said network [20]. It is therefore important to evaluate the impact of quantizing the neural network classifier on the performance of the classifier, as the quantized version will be implemented in hardware. In order to compare the performance of both neural networks, the evaluation of the regular neural network needs to be redone with the quantized neural network as a classifier.

Out of the 107 total experiments, only three of them showed different results. In all other 104 cases, the classification accuracy was identical up to two digits after the decimal point. In case of experiment 46 of the initial dataset, the difference between the regular and quantized network is 4.09% (absolute difference). For experiment 47, the difference is 0.99% and for experiment 48 the difference is -1.43%.

4.3.3. Spike Sorter

After evaluating the performance of each individual module (i.e. detection and classification module), both modules in conjunction were evaluated. Similarly to section 4.3.1 and 4.3.2, bar charts and boxplots are used to evaluate the system's overall performance. Figure 4.12 and 4.13 show the performance distribution of the binary classifier-based system and the multi-class classifier-based system on both the initial and new datasets. What stands out from these different boxplots is the fact that the overall sorter performance for the multi-class system is less compared to the binary classifier system. The multi-class spike sorter is also showing more variation. The boxplots illustrating the sorter's accuracy in detecting simple spikes appear to resemble this scenario. Moreover, the multi-class system seems to be doing better for complex spikes, in the context of the entire system. This makes sense since the multi-class option was explored to increase the complex spike sorting performance. The purpose of the multi-class classifier is to detect false positive spikes. The origin of these spikes might come from neurons residing in the background.

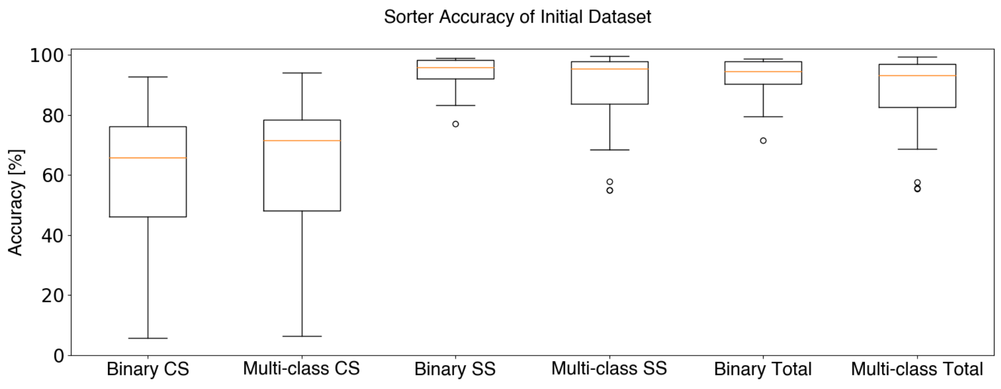


Figure 4.12: Distribution comparison of the accuracies of the detection and classification modules and the entire system. This concerns the experiments in the initial dataset. Results from both types of classifiers are shown.

The spikes from these background neurons slightly resemble simple spikes, hence a number of actual simple spikes might be considered false positives. The overall accuracy is the weighted average of both the complex and simple spike accuracy and since simple spikes are in abundance relative to complex spikes, their influence on the overall accuracy is significant. This is reflected in the overall performance of the system, resulting in less performance with more variation for the multi-class classifier.

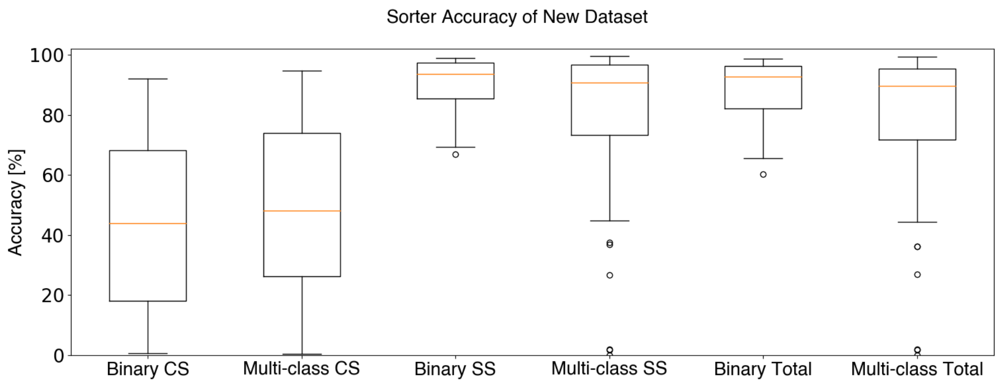


Figure 4.13: Distribution comparison of the accuracies of the detection and classification modules and the entire system. This concerns the experiments in the new dataset. Results from both types of classifiers are shown.

Since the sorter is a combination of both the detection and classification module, it is interesting to see the influence of both on the overall performance of the system. Figures 4.14, 4.15, 4.16 and 4.17 show the detection, classification, and overall accuracy for binary and multi-class based systems, for both datasets. Only the first ten experiments of each dataset have been shown to maintain readability. Complete results can be found in the appendix. 5.3.

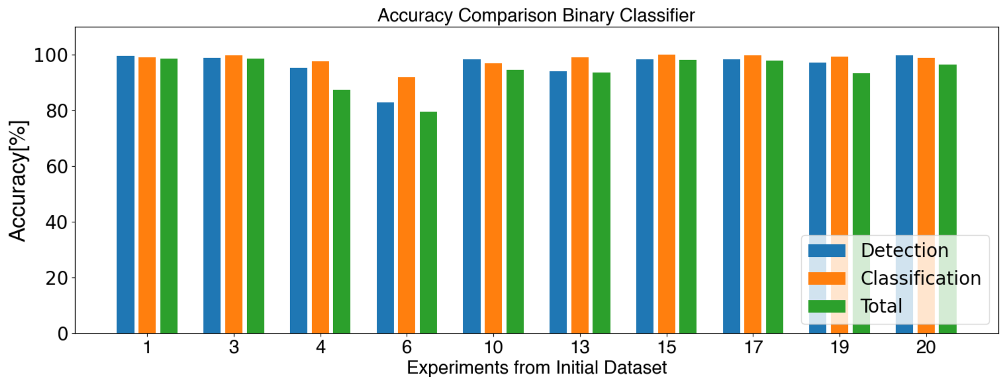


Figure 4.14: Accuracy comparison of the first ten experiments in the initial dataset. The performance of the individual modules can be compared with the performance of the entire system.

Some interesting observations can be made from both figures, which can be used to give an indication of which module can be considered the bottleneck of the system in terms of performance. Also, the functionality of both types of classifiers can be verified by looking at the performance contribution of each module. The multi-class classifier is used to compensate for false positive spike detections but is less well-equipped for classifying complex and simple spikes. The binary classifier is well-equipped for classifying simple and complex spikes but suffers in case the spike detector is performing relatively poorly. This can also be observed in figure 4.14 and 4.16, experiment 6. Experiment 6 reports a relatively poor detection accuracy of 82.87%. The multi-class and binary classifiers perform very similarly when it comes to classifying complex and simple spikes (median of 91.84% vs. 92.40%, respectively). However, only the multi-class-based system is able to properly deal with the shortcomings of the detector module. In the case of the multi-class classifier, the total accuracy is 94.24% vs. 79.41 (figure 4.14 and 4.16, experiment 6) for the binary classifier-based system.

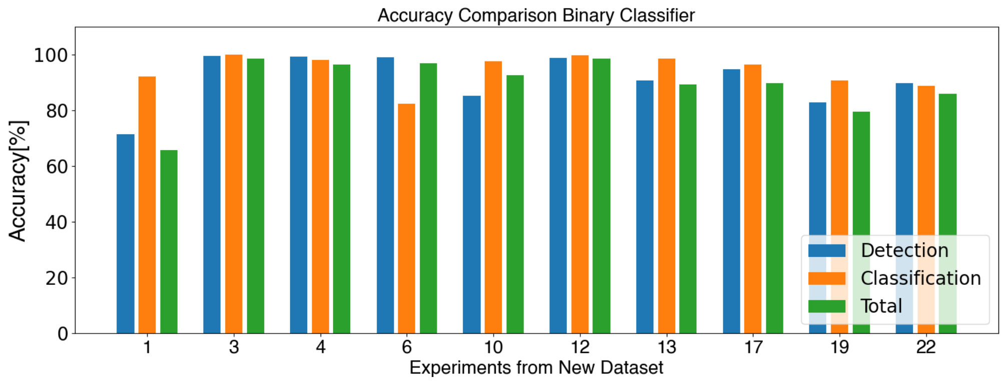


Figure 4.15: Accuracy comparison of the first ten experiments in the new dataset. The performance of the individual modules can be compared with the performance of the entire system using a binary classifier.

For experiments where the detector module does function adequately, for example, experiment four of the initial dataset, the binary classifier-based system is expected to perform better than the multi-class-based system. In these situations, nearly all spike waveforms which are fed to the classifier are either complex or simple spikes (i.e. *true positive* spike detections). The binary classifier is better equipped in this situation to correctly classify the input spike. Experiment four of the initial dataset reports a detection accuracy of 95.34%. Both classifiers perform relatively similarly (97.86% vs. 96.36%, for the binary and multi-class classifier respectively). The binary classifier-based system is better able to translate this to a sufficiently high total accuracy of 87.34% (see figure 4.14), while the multi-class-based system only reaches 82.58% (figure **??**).

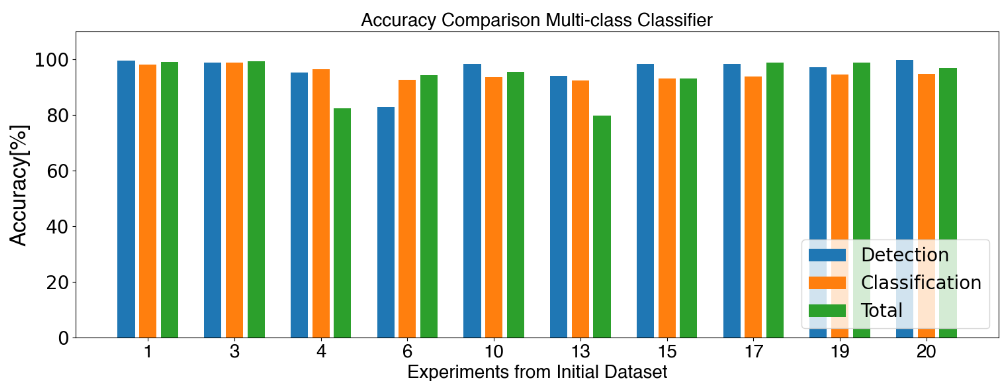


Figure 4.16: Accuracy comparison of the first ten experiments in the initial dataset. The performance of the individual modules can be compared with the performance of the entire system using a multiclass classifier.

The effects of the binary and multi-class classifiers on the total performance of the system only become apparent in situations where either of the sub-modules do not perform adequately well. In most cases, the performance is similar for both classifiers. The distribution of the system's performance (figure 4.12 and 4.13) does reveal that adding an extra output class to the classifier does introduce additional variation (standard deviation of 12.85% and 24.94% for the initial and new dataset respectively) to the system, which is not necessarily desirable. This can also be seen in table 4.2 and 4.3. In some cases, using an additional output class is effective in increasing the performance of the system. However, due to the added resource overhead and variation, it is not worth the trade-off. Adequately applying post-processing steps as described in section 3.5 can already mitigate some of the problems false positive spike detections pose, without adding to the variation of the system.

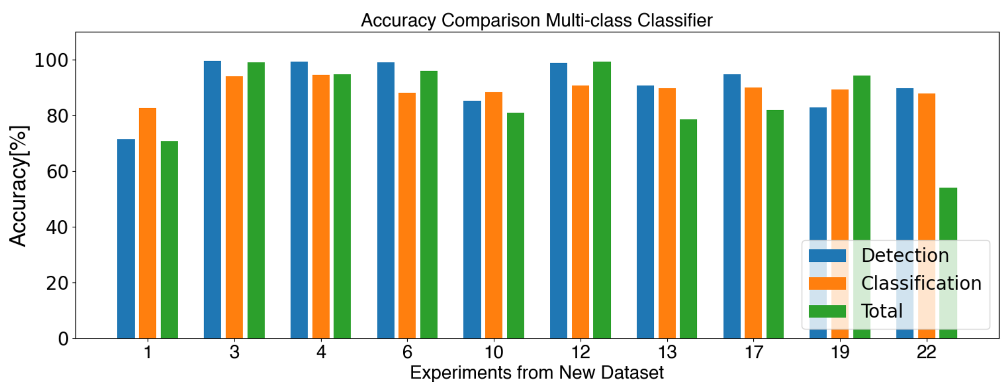


Figure 4.17: Accuracy comparison of the first ten experiments in the new dataset. The performance of the individual modules can be compared with the performance of the entire system using a multiclass classifier.

	Detection		**Classification CS**		**Classification SS**	
Dataset	*initial*	*new*	*initial*	*new*	*initial*	*new*
Mean	95.35%	97.34%	97.79%	96.61%	98.21%	96.61%
Median	92.76%	95.62%	100%	99.20%	98.50%	97.81%
Std. Dev.	4.85%	6.87%	6.17%	7.54%	1.54%	4.04%

	Complete System Totaal		**Complete System CS**		**Complete System SS**	
Dataset	*initial*	*new*	*initial*	*new*	*initial*	*new*
Mean	92.85%	88.82%	59.32%	43.70%	94.22%	90.60
Median	94.54%	92.74%	65.84%	43.97%	95.92%	93.65%
Std. Dev.	6.41%	9.73%	23.59%	27.83%	5.31%	8.41%

Table 4.2: Table containing results of the performance evaluation using the binary classifier.

Figure 4.12 and 4.13 give an insight into the distribution of all three types of performance metrics, to see how the detection and classification accuracy influence the overall accuracy. In line with expectations, the overall accuracy seems to follow the trend which the detection and classification accuracies exhibit. It can be observed that the variation in the overall levels of accuracy (i.e. the accuracy of the complete system) increases. This in turn leads to fewer outliers.

	Detection		**Classification CS**		**Classification SS**	
Dataset	*initial*	*new*	*initial*	*new*	*initial*	*new*
Mean	95.35%	97.34%	95.56%	89.39%	93.86%	89.19%
Median	92.76%	95.62%	95.77%	89.89%	94.45%	89.17%
Std. Dev.	4.85%	6.87%	1.11%	2.54%	2.15%	2.85%

	Complete System Totaal		**Complete System CS**		**Complete System SS**	
Dataset	*initial*	*new*	*initial*	*new*	*initial*	*new*
Mean	88.06%	78.06%	62.78%	48.83%	88.93%	79.35%
Median	93.15%	89.56%	71.58%	48.06%	95.37%	90.67%
Std. Dev.	12.85%	24.94%	22.15%	25.21%	13.34%	25.21%

Table 4.3: Table containing results of the performance evaluation using the multi-class classifier.

4.4. ASIC Performance

A logic synthesis of the RTL description of the proposed system has been performed using the Nangate 45 nm Open-Cell library [1]. This open-source library is developed in order to facilitate research, testing and exploring EDA design flows. Because of this, design synthesised using this library cannot be fabricated. This in turn is the reason why only a logic level synthesise was performed and not a complete place-and-route, which would require additional library files describing the physical characteristics of the used technology node. The results are therefore not final, but give a good estimation of the eventual performance of the physical design. A summary of the synthesis report can be found in table 4.4. Normally speaking, the synthesis reports the estimated chip area, the maximum power consumption and the timing slack relative to the configured clock period. Because the used clock frequency follows the sampling frequency and is therefore relatively low, the slack is not reported as it is not of interest. In order to get a better idea of the resource consumption of the individual modules, they were also synthesised separately. Hence, three entries can be found in table 4.4. As can be seen in this table, the classifier takes up the bulk of the chip real-estate. However, the occurrence of a spike is relatively rare so the classifier is used sparsely. The classifier needs two clock cycles to perform a classification. Therefore the energy needed per classification is 24.8 nW, assuming a 24414 kHz clock frequency.

	Detector	**Classifier**	**Complete System**
Area	6240 μm^2	15896 μm^2	22136 μm^2
Power Consumption	0.109 mW	0.303 mW	0.412 mW

Table 4.4: Table containing a summary of the logic synthesis results.

5 Conclusion

5.1. Summary

Presented in this work is an ASIC design that is able to take 10-bit neural recordings of action potentials of a single Purkinje cell of a mouse and detect the occurrence of a spiking event. Purkinje cells manifest more elaborate neural behaviour, as this type neuron is able to generate two types of spikes (complex and simple spikes), each with its own distinctive waveform. The proposed design is also able to capture the waveform of a detected spike and classify it as either a complex or simple spike. After the classification has been done, the ASIC will set a detection flag and output the classification result which is used in the offline post-processing step. Both the detection and classification can be done in real-time, making the ASIC suitable for applications in neural implants. The spike detection is performed by applying the non-linear energy operator to the recorded neural signal. By doing so, the instantaneous rise in the signal's energy is amplified. In case of a spiking event, the energy of the signal already increments relatively fast as it is a spike. Detecting neural spikes using this method still requires a threshold. The ASIC also features a dynamic threshold calculator, which is able to periodically recalculate the threshold to maximise the detection performance. This is needed due to the fact that traditional thresholding methods are sensitive to the ever-changing circumstances of recording neural activity. Due to the nature of brain tissue (i.e. it is soft), it is not possible to firmly anchor a recording electrode without damaging surrounding tissues. This causes electrode drift which influences the amplitude (and duration to some extent) of the recorded spikes. Combining this with a variable firing rate of the Purkinje cells (and neurons in general) leads to the need for adaptive thresholding. The use of a neural network as a spike classifier is grounded using the same motivation. Neural networks are well-equipped to show robust performance under varying circumstances. Captured spike waveforms are sent to a fully-connected neural network to be classified. The power of such neural networks is the fact that they generalise well given proper training. This allows the neural network to still perform its task adequately despite the varying amplitude of the recorded spikes. Two types of neural network classifiers were explored. The binary classifier assumes that the spike waveform at its input is either a simple or complex spike. It will try to classify false positive spike detections as either spike, resulting in poor performance for complex spikes. This is because complex spikes are relatively rare and the influence of false positive detections is therefore relatively large. Part of this problem can be mitigated in post-processing steps. Another solution to this problem was to extend the classifier to three classes by adding false positives as an output as well. It was found that both the binary and multi-class classifier showed similar results for the overall performance of the system. Because the multi-class classifier has a more intricate network topology compared to the binary classifier, which leads to higher resource overhead, the binary classifier is considered the preferred solution.

Tested on two large datasets containing real-life recordings of Purkinje cells of mice, the complete system was able to correctly detect and classify Purkinje cell neural activity 92.85% and 88.82% of the time on average. To be more precise: 94.22% and 90.60% for simple spikes, likewise 59.32% and 43.70% for complex spikes. However, the data resulting from the performance evaluation is not evenly distributed around the mean. This skewness causes the mean to be relatively low and hence the median gives a better indication of how well the system performs. The median overall accuracy level on both datasets is 94.54% and 92.74%. For both the mean and median, a discrepancy in performance is caused by the difference in signal-to-noise ratio between both datasets. The system performs better on the dataset containing better recordings. The system has been synthesised using the 45 nm Nangate Open Cell library resulting in an ASIC with an area of 22136 μm^2 and a power consumption of 0.412 mW.

The main problem statement presented in the introduction of this work was twofold; the current solution to detecting and classifying neural activity from Purkinje cells requires streaming the neural recordings through a wired connection to a personal computer. These recordings would then be manually processed in order to detect and classify the occurrences of spiking events. The first aspect of this problem statement relates to the wired connection. This hinders mimicking the ideal behaviour of the mice from which the neural activity is recorded and needs to be replaced by another solution. This (general) solution entails compressing the recorded data by only saving the spike class and the timestamp if its occurrence. The second aspect of the problem statement relates to the manual processing of the recorded data. Ideally, this process would be automated. The essence of this problem is captured in one research question which in turn broken down into three sub-questions. These questions are repeated for completeness:

"How to develop a system that is able to detect and classify neural activity from a Purkinje cell in a real-time fashion as an embedded implant?"

- *"How can techniques for spike sorting be utilised to detect and classify neural activity from a Purkinje cell in real-time?"*
- *"How can deep learning techniques aid in classifying spiking activity from a Purkinje cell?"*
- *"What techniques can be used to deploy the proposed system as an embedded implant?"*

The answer to the first sub-question is that not all steps of spike sorting are needed. By applying the Non-Linear Energy operator to the recorded neural signal, spikes can be detected from the background noise using a (dynamic) threshold. This first step is important for the functionality of the proposed system since the neural activity needs to be detected in the first place. The entire waveform of the detected spike is used to classify it, hence the third step (i.e. feature extraction) is omitted. Because of this, the spike does not need to be aligned, which is the reason why the second step (i.e. spike alignment) is also omitted. Classifying the detected spike (the fourth step in the spike sorting paradigm) is required and done through the use of neural networks. This reduces the computational complexity required by traditional methods for spike classification. This also answers the second sub-question. The differences between simple and complex spikes are quite noticeable and hence a relatively simple neural network suffices for the classification of both types of spikes. The detection stage of the proposed system is very resource efficient and very well-suited for an embedded application. By converting the neural network to 8-bit integer-based weights and activations, the classifier can also be deployed in an embedded context such as a neural implant. Despite this, the classifier is still responsible for most of the consumed resources. By splitting up the detection and classification stage, the resource-hungry classifier is only used relatively sparsely. This allows the system to be deployed as an embedded implant. Combining these three answers also answers the main research question.

5.2. Discussion

The system was trained and evaluated using datasets containing neural recordings with manually labelled spiking events. It is highly unlikely that a person would be accurate 100% of the time when it comes to detecting and classifying spikes from neural recordings. It is therefore inevitable that the "ground truth" contains errors on its own. So the performance evaluation is relative to this ground truth and might (and probably does) differ from a real-world situation. The proposed system can either outperform the person(s) who performed the manual labelling or be worse at it, it is impossible to know. The difference will likely be slight, however, it needs to be kept in mind nonetheless.
The proposed system performs its task under the assumption that no spiking event will occur for four milliseconds after the occurrence of another spike. This "dead zone" of 4 ms is quite conservative and the system might behave differently in terms of its performance when another value is chosen (2 ms for example, which is quite optimistic). The influence of the dead zone on the performance of the proposed system should be looked into further in order to establish the optimum value to maximise the performance of the proposed system.
Furthermore, the evaluation method used in this work can be considered quite optimistic. If a detected spike

falls within a time frame of 1 ms of the labelled spike, it is assumed to be a true positive. This time interval was chosen because of the average length of a simple spike (of the two types of spikes, the one with the shortest duration) is 1 ms. Evaluating the performance of the proposed design using this method makes sense when the manual labelling is performed inconsistently. In this case, the labelled timestamp can deviate quite a bit from the occurrence of the actual spike. Being lenient by accepting detected spikes as true positives using a relatively large time frame (the aforementioned 1 ms), can mitigate the influence of poor manual labelling. However, it is an optimistic way of evaluating the design if most of the manually labelled timestamps are quite accurate. This is very likely the case, so optimising the way the system is evaluated could give better insights in how well the system performs.

5.3. Opportunities for future research

Even though the initial results are promising, the next iterations of the proposed design may benefit from several additions and improvements. For example, the neural network classifier can be optimised even further. The neural network used in this work is a fully-connected and even though it is relatively simple it still consumes most of the silicon resources to store the synaptic weights. Not of these weights contribute significantly to the performance of the classifier. Via a technique called pruning, the number of weights could be reduced without compromising the performance of the classifier. This could result in a significant increase in the efficiency of the classifier both in terms of power and area.
The main problem of the proposed design is the fact that it is built under the assumption that the recording electrode is placed in such a way that it only records the neural activity from one Purkinje cell. If the position of the electrode shifts over time, it might also pick up activities from surrounding neurons. This activity will also be detected and classified, resulting in a number of false positives. A suggestion for future iterations might be to extend the classifier from a binary classifier to a ternary classifier. With the added third class, the classifier will also be able to identify false positives and disregard these detected spikes. Naturally, a study has to be conducted to determine the feasibility of such a solution in order to maintain a certain level of performance. Also, extending the classifier might result in a silicon resource overhead which is not worth the added benefits. This is a trade-off which needs to be looked into.
The third suggestion for future research is to extend the design to process multiple recording channels. The current design is single-channel and has a relatively low clock frequency which is the same as the sampling frequency (i.e. 24 kHz). The design can be altered to run with a higher clock frequency so the system is able to process multiple channels. Extra care should be given to allow the system to properly process simultaneous spikes on different channels. The adaptive threshold calculator is able to calculate a new threshold for 64 individual channels continuously in a sequential matter [88].

All the aforementioned suggestions for further development of the proposed design assume the use of CMOS technology, which has its own benefits and drawbacks. The type of classifier (artificial neural network) lends itself well to be implemented using CMOS technology. Constant developments in the field of neural networks lead to the proposition of new types of neural networks. One such type of neural network is called a spiking neural network (SNN) which differs quite significantly from traditional neural networks. Instead of using numerical values as input, synaptic weights, and outputs, information is encoded as a series of spikes called spike trains. This type of neural network aims to mimic biological neural dynamics even further compared to more traditional neural networks. The use of such a spiking neural network as a spike classifier comes with a number of advantages and disadvantages. One such advantage is the fact that these spiking neural networks can be implemented in hardware using memristive devices. These can lead to an area reduction of up to four times (depending on the type of memristor) over regular CMOS technology. This would greatly reduce the silicon real-estate occupied by the design. Which in turn would reduce the power consumption of the system or allow for the expansion of the number of channels. Often, spiking neural networks are less complex compared to their ANN counterparts. This would reduce the needed area even further. However, spiking neural networks operate in the analog domain so a conversion step is needed to take the digitally detected spikes back to the analog domain for classification. This would mean that the neural signal is converted from analog to digital by the recording front-end, so a digital spike detector could do its job and detect spike occurrences from the recorded neural activity. These detected spikes would then be converted back to the analog domain so that they can be encoded as a spike train in order to be classified by the SNN. In order to interpret the output of the classifier, the output spike train needs to be converted back again to the digital domain for further processing. This process would be rather inefficient because, generally speaking, analog-to-digital (ADC) and digital-to-analog converters tend to be a burden on the energy budget for systems such as this one. Several solutions might solve this problem. For one, the detection of spikes might also be done in the analog domain, which would lead to only a conversion step after the spike has been detected. It is hard to say what such an implementation would do to the complete system in terms of its performance. Another solution would be to abandon the two-tier approach and implement an SNN which would take care of both the detection and classification stages. The power of this two-tier approach is the fact that it is energy efficient because the classification stage takes up the bulk of the energy budget and spiking events are relatively rare. So implementing an SNN, which entails the entire process, might have a detrimental effect on the energy consumption of the system. However, the merits of using an SNN might also prove worthwhile for an application such as this one.

Another big issue with SNNs that needs to be addressed is the fact that training such networks is a hard task and not as straightforward as training an ANN. Unless trained properly, the performance of the SNN (detector and) classifier might not suffice for the intended application.

Even though quite some hurdles need to be taken, the use of SNNs for detecting and classifying neural activity might prove fruitful and should at least be looked into.

A
Comprehensive Classifier Results

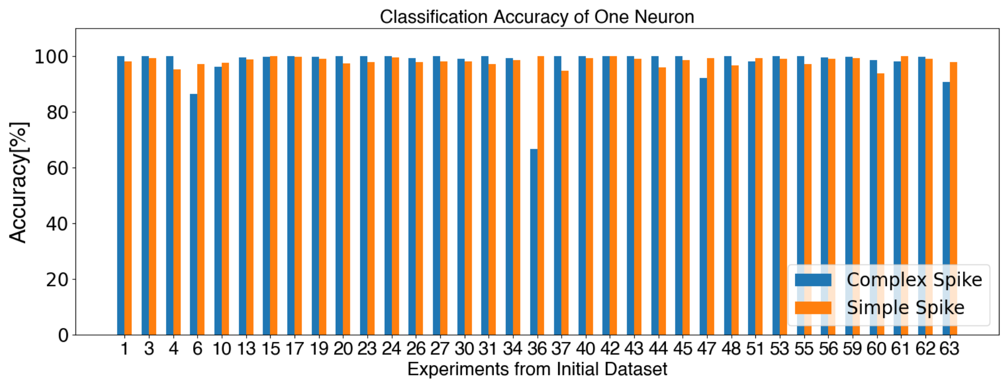


Figure A.1: Classification accuracy of all experiments in the initial dataset using a neural network with one output neuron.

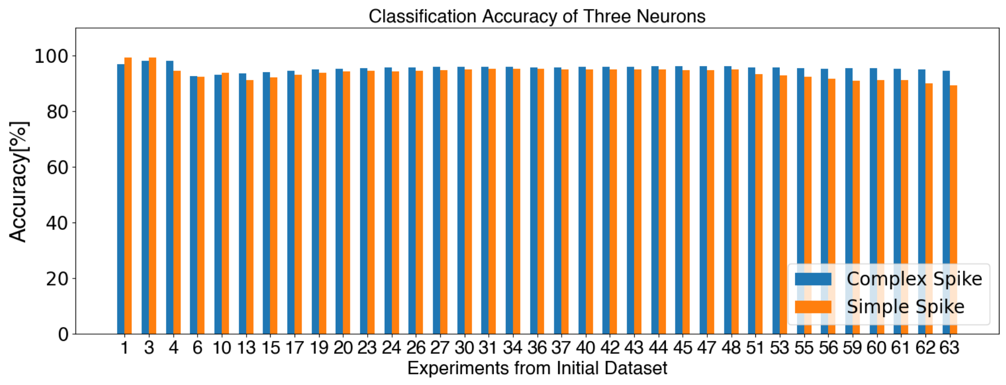


Figure A.2: Classification accuracy of all experiments in the initial dataset using a neural network with three output neurons.

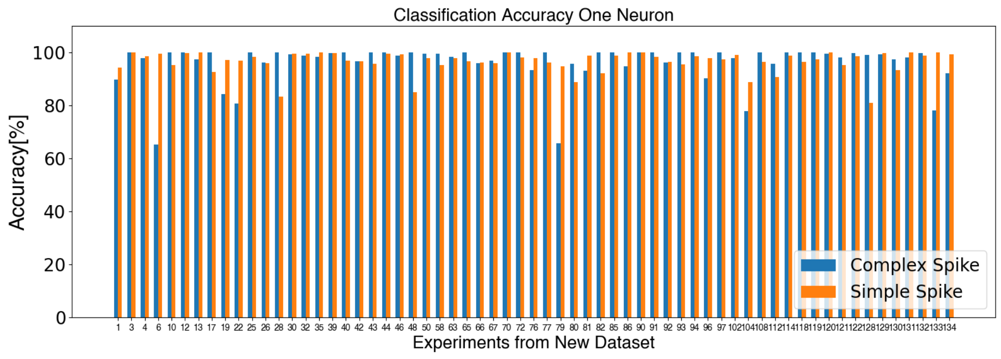


Figure A.3: Classification accuracy of all experiments in the new dataset using a neural network with one output neuron.

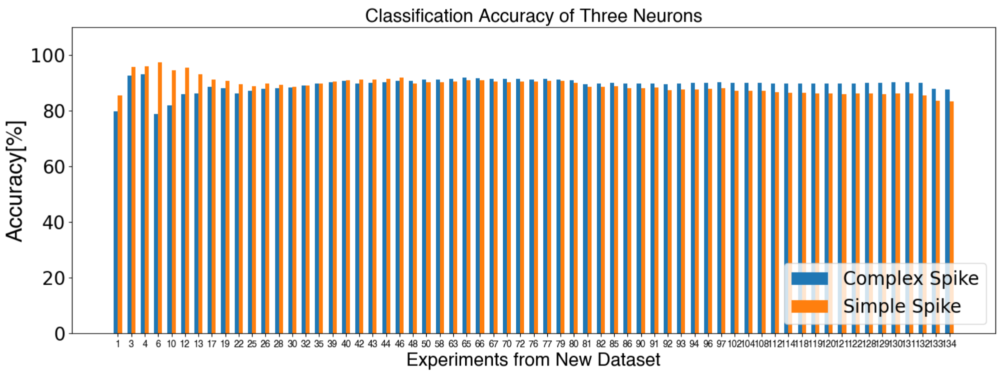


Figure A.4: Classification accuracy of all experiments in the new dataset using a neural network with three output neurons.

B
Comprehensive Sorter Results

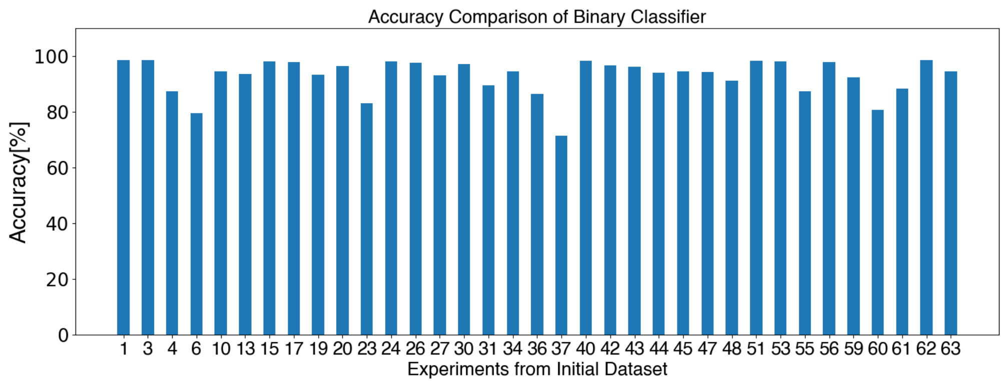


Figure B.1: Sorter accuracy of all experiments in the initial dataset using a neural network with one output neuron.

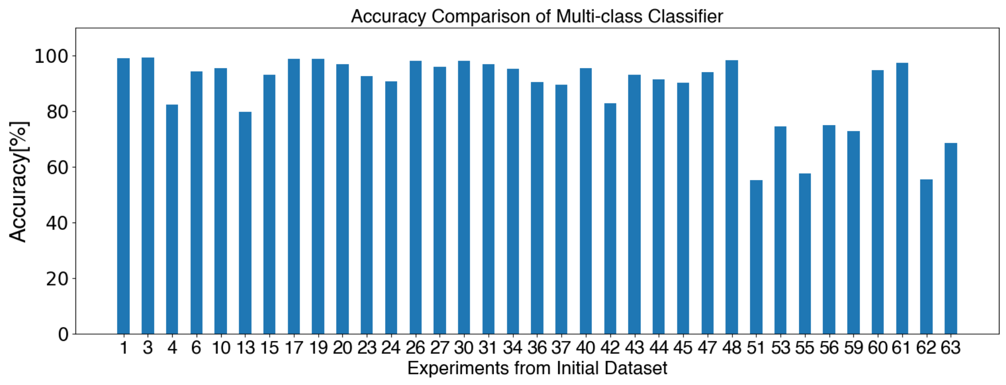


Figure B.2: Sorter accuracy of all experiments in the initial dataset using a neural network with three output neurons.

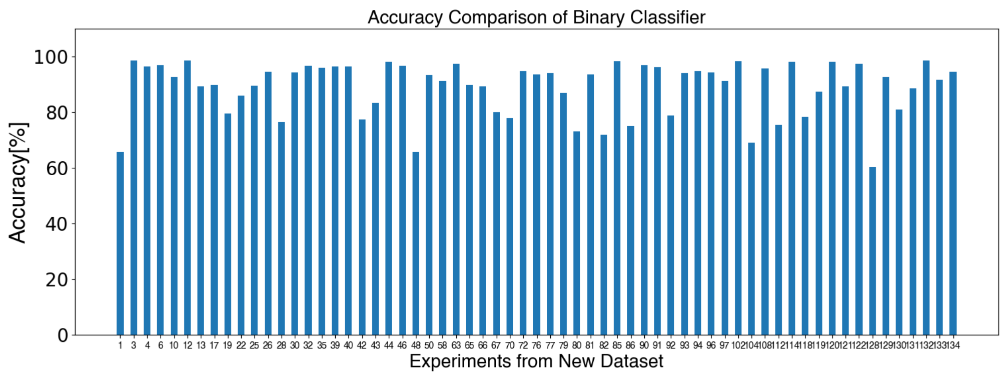


Figure B.3: Sorter accuracy of all experiments in the new dataset using a neural network with one output neuron.

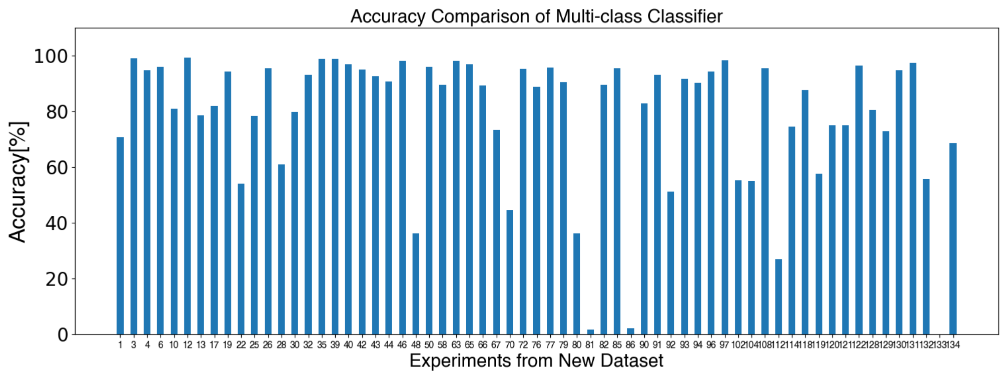


Figure B.4: Sorter accuracy of all experiments in the new dataset using a neural network with three output neurons.

Bibliography

[1] Open-cell library, Aug 2019. URL `https://si2.org/open-cell-library/`.

[2] Purkinje cell, Jan 2023. URL `https://en.wikipedia.org/wiki/Purkinje_cell`.

[3] Nazanin Ahmadi-Dastgerdi, Hossein Hosseini-Nejad, Hadi Amiri, Afshin Shoeibi, and Juan Manuel Gorriz. A vector Quantization-Based spike compression approach dedicated to multichannel neural recording microsystems. *Int J Neural Syst*, 32(3):2250001, December 2021.

[4] Jokubas Ausra, Stephanie Munger, Amirhossein Azami, Alex Burton, Roberto Peralta, Julie Miller, and Philipp Gutruf. Wireless battery free fully implantable multimodal recording and neuromodulation tools for songbirds. *Nature Communications*, 12, 03 2021. doi: 10.1038/s41467-021-22138-8.

[5] C.M. Bishop. *Neural networks for pattern recognition.* Oxford University Press, USA, 1995.

[6] Zainul Charbiwala, Vaibhav Karkare, Sarah Gibson, Dejan Markovic, and Mani B. Srivastava. Compressive sensing of neural action potentials using a learned union of supports. In *2011 International Conference on Body Sensor Networks*, pages 53–58, 2011. doi: 10.1109/BSN.2011.28.

[7] Baibhab Chatterjee, K Gaurav Kumar, Shulan Xiao, Gourab Barik, Krishna Jayant, and Shreyas Sen. A 1.8μW 5.5 mm3 adc-less neural implant soc utilizing 13.2pj/sample time-domain bi-phasic quasi-static brain communication with direct analog to time conversion. In *ESSCIRC 2022- IEEE 48th European Solid State Circuits Conference (ESSCIRC)*, pages 209–212, 2022. doi: 10.1109/ESSCIRC55480.2022.9911420.

[8] Stuart F Cogan. Neural stimulation and recording electrodes. *Annu Rev Biomed Eng*, 10:275–309, 2008.

[9] Andres de Groot, Bastijn JG van den Boom, Romano M van Genderen, Joris Coppens, John van Veldhuijzen, Joop Bos, Hugo Hoedemaker, Mario Negrello, Ingo Willuhn, Chris I De Zeeuw, and Tycho M Hoogland. Ninscope, a versatile miniscope for multi-region circuit investigations. *eLife*, 9:e49987, jan 2020. ISSN 2050-084X. doi: 10.7554/eLife.49987. URL `https://doi.org/10.7554/eLife.49987`.

[10] Dimitrios Dimitriadis, Alexandros Potamianos, and Petros Maragos. A comparison of the squared energy and teager-kaiser operators for short-term energy estimation in additive noise. *Signal Processing, IEEE Transactions on*, 57:2569 – 2581, 08 2009. doi: 10.1109/TSP.2009.2019299.

[11] Anh Tuan Do and Kiat S. Yeo. A hybrid neo-based spike detection algorithm for implantable brain-ic interface applications. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2393–2396, 2014. doi: 10.1109/ISCAS.2014.6865654.

[12] B. Dutta, A. Andrei, T. D. Harris, C. M. Lopez, J. O'Callahan, J. Putzeys, B. C. Raducanu, S. Severi, S. D. Stavisky, E. M. Trautmann, M. Welkenhuysen, and K. V. Shenoy. The neuropixels probe: A cmos based integrated microsystems platform for neuroscience and brain-computer interfaces. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 10.1.1–10.1.4, 2019. doi: 10.1109/IEDM19573.2019.8993611.

[13] Alexander Tarnavsky Eitan, Shirly Someck, Mario Guillermo Zajac, Eran Socher, and Eran Stark. Outan: An on-head system for driving μled arrays implanted in freely moving mice. *IEEE Transactions on Biomedical Circuits and Systems*, 15(2):303–313, 2021. doi: 10.1109/TBCAS.2021.3068556.

[14] Nader Sherif Kassem Fathy, Jiannan Huang, and Patrick P. Mercier. A digitally assisted multiplexed neural recording system with dynamic electrode offset cancellation via an lms interference-canceling filter. *IEEE Journal of Solid-State Circuits*, 57(3):953–964, 2022. doi: 10.1109/JSSC.2021.3116021.

[15] Gabriel Gagnon-Turcotte, Guillaume Bilodeau, Olivier Tsiakaka, and Benoit Gosselin. Smart autonomous electro-optic platforms enabling innovative brain therapies. *IEEE Circuits and Systems Magazine*, 20(4):28–46, 2020. doi: 10.1109/MCAS.2020.3027220.

[16] Gabriel Gagnon-Turcotte, Guillaume Bilodeau, Olivier Tsiakaka, and Benoit Gosselin. Smart autonomous electro-optic platforms enabling innovative brain therapies. *IEEE Circuits and Systems Magazine*, 20(4):28–46, 2020. doi: 10.1109/MCAS.2020.3027220.

[17] Sarah Gibson, Jack W. Judy, and Dejan Marković. Spike sorting: The first step in decoding the brain: The first step in decoding the brain. *IEEE Signal Processing Magazine*, 29(1):124–143, 2012. doi: 10.1109/MSP.2011.941880.

[18] Isha Gupta, Alexander Serb, Ali Khiat, Ralf Zeitler, Stefano Vassanelli, and Themis Prodromakis. Real-time encoding and compression of neuronal spikes by metal-oxide memristors. *Nature Communications*, 7:12805, 09 2016. doi: 10.1038/ncomms12805.

[19] Han Hao, Jiahe Chen, Andrew G. Richardson, Jan Van der Spiegel, and Firooz Aflatouni. A 10.8 μw neural signal recorder and processor with unsupervised analog classifier for spike sorting. *IEEE Transactions on Biomedical Circuits and Systems*, 15(2):351–364, 2021. doi: 10.1109/TBCAS.2021.3076147.

[20] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *CoRR*, abs/1712.05877, 2017. URL `http://arxiv.org/abs/1712.05877`.

[21] J.F. Kaiser. On a simple algorithm to calculate the 'energy' of a signal. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 381–384 vol.1, 1990. doi: 10.1109/ICASSP.1990.115702.

[22] Awais M. Kamboh and Andrew J. Mason. Computationally efficient neural feature extraction for spike sorting in implantable high-density recording systems. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 21(1):1–9, 2013. doi: 10.1109/TNSRE.2012.2211036.

[23] Masanobu Kano and Masahiko Watanabe. Chapter 4 - cerebellar circuits. In John Rubenstein, Pasko Rakic, Bin Chen, and Kenneth Y. Kwan, editors, *Neural Circuit and Cognitive Development (Second Edition)*, pages 79–102. Academic Press, second edition edition, 2020. ISBN 978-0-12-814411-4. doi: https://doi.org/10.1016/B978-0-12-814411-4.00004-4. URL `https://www.sciencedirect.com/science/article/pii/B9780128144114000044`.

[24] Ioannis Karageorgos, Karthik Sriram, Ján Veselý, Michael Wu, Marc Powell, David Borton, Rajit Manohar, and Abhishek Bhattacharjee. Hardware-software co-design for brain-computer interfaces. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 391–404, 2020. doi: 10.1109/ISCA45697.2020.00041.

[25] Vaibhav Karkare, Sarah Gibson, and Dejan Markovic. A 130- w, 64-channel neural spike-sorting dsp chip. *Solid-State Circuits, IEEE Journal of*, 46:1214 – 1222, 06 2011. doi: 10.1109/JSSC.2011.2116410.

[26] Hossein Kassiri, Sana Tonekaboni, M. Tariqus Salam, Nima Soltani, Karim Abdelhalim, Jose Luis Perez Velazquez, and Roman Genov. Closed-loop neurostimulators: A survey and a seizure-predicting design example for intractable epilepsy treatment. *IEEE Transactions on Biomedical Circuits and Systems*, 11 (5):1026–1040, 2017. doi: 10.1109/TBCAS.2017.2694638.

[27] Hossein Kassiri, Sana Tonekaboni, Muhammad Salam, Nima Soltani, Karim Abdelhalim, Jose Velazquez, and Roman Genov. Closed-loop neurostimulators: A survey and a seizure-predicting design example for intractable epilepsy treatment. *IEEE Transactions on Biomedical Circuits and Systems*, PP:1–15, 07 2017. doi: 10.1109/TBCAS.2017.2694638.

[28] Kyung Hwan Kim and Sung June Kim. Neural spike sorting under nearly 0-db signal-to-noise ratio using nonlinear energy operator and artificial neural-network classifier. *IEEE Transactions on Biomedical Engineering*, 47(10):1406–1411, 2000. doi: 10.1109/10.871415.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL `https://arxiv.org/abs/1412.6980`.

[30] Ondrej Klempir, Radim Krupicka, Jan Krusek, Ivan Dittert, V Petráková, Václav Petrák, and Andy Taylor. Application of spike sorting algorithm to neuronal signals originated from boron doped diamond micro-electrode arrays. *Physiological research*, 69, 05 2020. doi: 10.33549/physiolres.934366.

[31] Ermis Koutsos, Sivylla E. Paraskevopoulou, and Timothy G. Constandinou. A 1.5 w neo-based spike detector with adaptive-threshold for calibration-free multichannel neural interfaces. In *2013 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1922–1925, 2013. doi: 10.1109/ISCAS.2013.6572243.

[32] Yiran Lang, Rongyu Tang, Yafei Liu, Pengcheng Xi, Honghao Liu, Zhenzhen Quan, Da Song, Xiaodong Lv, Qiang Huang, and Jiping He. Multisite simultaneous neural recording of motor pathway in Free-Moving rats. *Biosensors (Basel)*, 11(12), December 2021.

[33] Jongwoo Lee, Hyo-Gyuem Rhew, Daryl R. Kipke, and Michael P. Flynn. A 64 channel programmable closed-loop neurostimulator with 8 channel neural amplifier and logarithmic adc. *IEEE Journal of Solid-State Circuits*, 45(9):1935–1945, 2010. doi: 10.1109/JSSC.2010.2052403.

[34] Michael S Lewicki. A review of methods for spike sorting: the detection and classification of neural action potentials. *Network: Computation in Neural Systems*, 9(4):R53–R78, 1998. doi: 10.1088/0954-898X_9_4_001. URL `https://doi.org/10.1088/0954-898X_9_4_001`. PMID: 10221571.

[35] Hongge Li, Pan Yu, and Tongsheng Xia. Efficient hybrid neural network for spike sorting. *Journal of Systems Engineering and Electronics*, 24(1):157–164, 2013. doi: 10.1109/JSEE.2013.00020.

[36] Nian Liu and Nayyar A. Zaidi. Artificial neural network: Deep or broad? an empirical study. In Byeong Ho Kang and Quan Bai, editors, *AI 2016: Advances in Artificial Intelligence*, pages 535–541, Cham, 2016. Springer International Publishing. ISBN 978-3-319-50127-7.

[37] Xilin Liu, Milin Zhang, Tao Xiong, Andrew G. Richardson, Timothy H. Lucas, Peter S. Chin, Ralph Etienne-Cummings, Trac D. Tran, and Jan Van der Spiegel. A fully integrated wireless compressed sensing neural signal acquisition system for chronic recording and brain machine interface. *IEEE Transactions on Biomedical Circuits and Systems*, 10(4):874–883, 2016. doi: 10.1109/TBCAS.2016.2574362.

[38] Xilin Liu, Milin Zhang, Andrew G. Richardson, Timothy H. Lucas, and Jan Van der Spiegel. Design of a closed-loop, bidirectional brain machine interface system with energy efficient neural feature extraction and pid control. *IEEE Transactions on Biomedical Circuits and Systems*, 11(4):729–742, 2017. doi: 10.1109/TBCAS.2016.2622738.

[39] Xu Liu, Juzhe Li, Tao Chen, Wensi Wang, and Minkyu Je. A neural recording and stimulation chip with artifact suppression for biomedical devices. *Journal of Healthcare Engineering*, 2021:1–11, 08 2021. doi: 10.1155/2021/4153155.

[40] Song Luan, Ian Williams, Michal Maslik, Yan Liu, Felipe De Carvalho, Andrew Jackson, Rodrigo Quian Quiroga, and Timothy G Constandinou. Compact standalone platform for neural recording with real-time spike sorting and data logging. *Journal of Neural Engineering*, 15(4):046014, may 2018. doi: 10.1088/1741-2552/aabc23. URL `https://dx.doi.org/10.1088/1741-2552/aabc23`.

[41] Lutzroeder. Lutzroeder/netron: Visualizer for neural network, deep learning, and machine learning models. URL `https://github.com/lutzroeder/netron`.

[42] Qier Ma, Liyuan Guo, Seyed Mohammad Ali Zeinolabedin, and Christian Mayr. Ultra-low power and area-efficient hardware accelerator for adaptive neural signal compression. In *2021 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4, 2021. doi: 10.1109/BioCAS49922.2021.9644999.

[43] Omid Malekzadeh-Arasteh, Haoran Pu, Ahmad Reza Danesh, Jeffrey Lim, Po T. Wang, Charles Y. Liu, An H. Do, Zoran Nenadic, and Payam Heydari. A fully-integrated 1µw/channel dual-mode neural data acquisition system for implantable brain-machine interfaces. In *2021 43rd Annual International Conference of the IEEE Engineering in Medicine Biology Society (EMBC)*, pages 5780–5783, 2021. doi: 10.1109/EMBC46164.2021.9630058.

[44] Akshay Markanday, Joachim Bellet, Marie E. Bellet, Junya Inoue, Ziad M. Hafed, and Peter Thier. Using deep neural networks to detect complex spikes of cerebellar purkinje cells. *Journal of Neurophysiology*, 123(6):2217–2234, 2020. doi: 10.1152/jn.00754.2019. URL `https://doi.org/10.1152/jn.00754.2019`. PMID: 32374226.

[45] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2018. URL `http://arxiv.org/abs/1802.03426`. cite arxiv:1802.03426Comment: Reference implementation available at http://github.com/lmcinnes/umap.

[46] Benjamin Metcalfe, C.T. Clarke, Nick Donaldson, and John Taylor. A new method for neural spike alignment: The centroid filter. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, PP:1–1, 06 2017. doi: 10.1109/TNSRE.2017.2716822.

[47] S. Mukhopadhyay and G.C. Ray. A new interpretation of nonlinear energy operator and its efficacy in spike detection. *IEEE Transactions on Biomedical Engineering*, 45(2):180–187, 1998. doi: 10.1109/10.661266.

[48] Elon and Musk. An integrated brain-machine interface platform with thousands of channels. 2019. doi: 10.1101/703801.

[49] Joaquin Navajas, Deren Barsakcioglu, Amir Eftekhar, Andrew Jackson, Timothy Constandinou, and Rodrigo Quian. Minimum requirements for accurate and efficient real-time on-chip spike sorting. *Journal of neuroscience methods*, 230:51–64, 04 2014. doi: 10.1016/j.jneumeth.2014.04.018.

[50] I. Obeid and P.D. Wolf. Evaluation of spike-detection algorithms fora brain-machine interface application. *IEEE Transactions on Biomedical Engineering*, 51(6):905–911, 2004. doi: 10.1109/TBME.2004.826683.

[51] Gerard O'Leary, David M. Groppe, Taufik A. Valiante, Naveen Verma, and Roman Genov. Nurip: Neural interface processor for brain-state classification and programmable-waveform neurostimulation. *IEEE Journal of Solid-State Circuits*, 53(11):3150–3162, 2018. doi: 10.1109/JSSC.2018.2869579.

[52] Sivylla Paraskevopoulou, Deren Barsakcioglu, Mohammed Saberi, Amir Eftekhar, and Timothy Constandinou. Feature extraction using first and second derivative extrema (fsde) for real-time and hardware-efficient spike sorting. *Journal of neuroscience methods*, 215, 02 2013. doi: 10.1016/j.jneumeth.2013.01.012.

[53] Jongkil Park, Gookhwa Kim, and Sang-Don Jung. A 128-channel fpga-based real-time spike-sorting bidirectional closed-loop neural interface system. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(12):2227–2238, 2017. doi: 10.1109/TNSRE.2017.2697415.

[54] Rakshit Pathak, Saurav Dash, Anand Kumar Mukhopadhyay, Arindam Basu, and Mrigank Sharad. Low power implantable spike sorting scheme based on neuromorphic classifier with supervised training engine. In *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 266–271, 2017. doi: 10.1109/ISVLSI.2017.54.

[55] M. Prezioso, Y. Zhong, D. Gavrilov, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. Likharev, and D. Strukov. Spiking neuromorphic networks with metal-oxide memristors. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 177–180, 2016. doi: 10.1109/ISCAS.2016.7527199.

[56] Norberto Pérez-Prieto, Ángel Rodríguez-Vázquez, Manuel Álvarez Dolado, and Manuel Delgado-Restituto. A 32-channel time-multiplexed artifact-aware neural recording system. *IEEE Transactions on Biomedical Circuits and Systems*, 15(5):960–977, 2021. doi: 10.1109/TBCAS.2021.3108725.

[57] Indira M. Raman and Bruce P. Bean. Ionic currents underlying spontaneous action potentials in isolated cerebellar purkinje neurons. *Journal of Neuroscience*, 19(5):1663–1674, 1999. ISSN 0270-6474. doi: 10.1523/JNEUROSCI.19-05-01663.1999. URL `https://www.jneurosci.org/content/19/5/1663`.

[58] Fernando Ramirez de Noriega, Renana Eitan, Odeya Marmor, Adi Lavi, Eduard Linetzky, Hagai Bergman, and Zvi Israel. Constant current versus constant voltage subthalamic nucleus deep brain stimulation in parkinson's disease. *Stereotact Funct Neurosurg*, 93(2):114–121, February 2015.

[59] Vaishnavi Ranganathan, Jared Nakahara, Soshi Samejima, Nicholas Tolley, Abed Khorasani, Chet T. Moritz, and Joshua R. Smith. Neuralclip: A modular fpga-based neural interface for closed-loop operation. In *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 791–794, 2019. doi: 10.1109/NER.2019.8717135.

[60] Stefan Reich, Markus Sporer, Michael Haas, Joachim Becker, Martin Schüttler, and Maurits Ortmanns. A high-voltage compliance, 32-channel digitally interfaced neuromodulation system on chip. *IEEE Journal of Solid-State Circuits*, 56(8):2476–2487, 2021. doi: 10.1109/JSSC.2021.3076510.

[61] Hyo-Gyuem Rhew, Jaehun Jeong, Jeffrey A. Fredenburg, Sunjay Dodani, Parag G. Patil, and Michael P. Flynn. A fully self-contained logarithmic closed-loop deep brain stimulation soc with wireless telemetry and wireless power management. *IEEE Journal of Solid-State Circuits*, 49(10):2213–2227, 2014. doi: 10.1109/JSSC.2014.2346779.

[62] F. Rosenblatt. The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January 1957.

[63] Melinda Rácz, Csaba Liber, Erik Németh, Richárd Fiáth, János Rokai, István Harmati, István Ulbert, and Gergely Márton. Spike detection and sorting with deep learning. *Journal of Neural Engineering*, 17(1): 016038, jan 2020. doi: 10.1088/1741-2552/ab4896. URL `https://dx.doi.org/10.1088/1741-2552/ab4896`.

[64] Himanshu S. Activation functionsnbsp;: Sigmoid, relu, leaky relu and softmax basics for neural networks and deep..., Aug 2021. URL `https://himanshuxd.medium.com/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c`

[65] Zaghloul Saad Zaghloul and Magdy Bayoumi. Adaptive neural matching online spike sorting vlsi chip design for wireless bci implants. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 977–981, 2015. doi: 10.1109/ICASSP.2015.7178115.

[66] Maryam Saeed, Amir Ali Khan, and Awais Mehmood Kamboh. Comparison of classifier architectures for online neural spike sorting. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(4): 334–344, 2017. doi: 10.1109/TNSRE.2016.2641499.

[67] Pasquale Davide Schiavone, Davide Rossi, Yan Liu, Simone Benatti, Song Luan, Ian Williams, Luca Benini, and Timothy Constandinou. Neuro-pulp: A paradigm shift towards fully programmable platforms for neural interfaces. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 50–54, 2020. doi: 10.1109/AICAS48895.2020.9073920.

[68] Jörg Scholvin, Justin P. Kinney, Jacob G. Bernstein, Caroline Moore-Kochlacs, Nancy Kopell, Clifton G. Fonstad, and Edward S. Boyden. Close-packed silicon microelectrodes for scalable spatially oversampled neural recording. *IEEE Transactions on Biomedical Engineering*, 63(1):120–130, 2016. doi: 10.1109/TBME.2015.2406113.

[69] Laszlo Schäffer, Zoltan Nagy, Zoltan Kincses, Richard Fiáth, and Istvan Ulbert. Spatial information based osort for real-time spike sorting using fpga. *IEEE Transactions on Biomedical Engineering*, 68(1):99–108, 2021. doi: 10.1109/TBME.2020.2996281.

[70] Ehsan Sedaghat-Nejad, Mohammad Amin Fakharian, Jay Pi, Paul Hage, Yoshiko Kojima, Robi Soetedjo, Shogo Ohmae, Javier F. Medina, and Reza Shadmehr. P-sort: an open-source software for cerebellar neurophysiology. *Journal of Neurophysiology*, 126(4):1055–1075, 2021. doi: 10.1152/jn.00172.2021. URL `https://doi.org/10.1152/jn.00172.2021`. PMID: 34432996.

[71] John Seymour, Fan Wu, Kensall Wise, and Euisik Yoon. State-of-the-art mems and microsystem tools for brain research. *Microsystems Nanoengineering*, 3:16066, 01 2017. doi: 10.1038/micronano.2016.66.

[72] Mohammad Shaeri and Amir Sodagar. A framework for on-implant spike sorting based on salient feature selection. *Nature Communications*, 11:3278, 06 2020. doi: 10.1038/s41467-020-17031-9.

[73] C.E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, jan 1949. doi: 10.1109/jrproc.1949.232969. URL `https://doi.org/10.1109/jrproc.1949.232969`.

[74] Gunchul Shin, Adrian Gomez, Ream Al-Hasani, Yu Ra Jeong, Jeonghyun Kim, Zhaoqian Xie, Anthony Banks, Seung Min Lee, Sang Youn Han, Chul Yoo, Jong-Lam Lee, Seung Lee, Jonas Kurniawan, Jacob Tureb, Zhongzhu Guo, Jangyeol Yoon, Sung-Il Park, Sang Bang, Yoonho Nam, and John Rogers. Flexible near-field wireless optoelectronics as subdermal implants for broad applications in optogenetics. *Neuron*, 93, 01 2017. doi: 10.1016/j.neuron.2016.12.031.

[75] Larry E. Shupe, Frank P. Miles, Geoff Jones, Richy Yun, Jonathan H. Mishler, Irene Rembado, R. Logan Murphy, Steve I. Perlmutter, and Eberhard E. Fetz. Neurochip3: An autonomous multichannel bidirectional brain-computer interface for closed-loop activity-dependent stimulation. *Frontiers in Neuroscience*, 15, 2021.

[76] S. Stanslaski, Pedram Afshar, Peng Cong, Jon Giftakis, Paul Stypulkowski, David Carlson, David Linde, Dave Ullestad, Al-Thaddeus Avestruz, and Timothy Denison. Design and validation of a fully implantable, chronic, closed-loop neuromodulation device with concurrent sensing and stimulation. *IEEE transactions on neural systems and rehabilitation engineering : a publication of the IEEE Engineering in Medicine and Biology Society*, 20:410–21, 01 2012. doi: 10.1109/TNSRE.2012.2183617.

[77] W T Thach. Somatosensory receptive fields of single units in cat cerebellar cortex. *Journal of Neurophysiology*, 30(4):675–696, 1967. doi: 10.1152/jn.1967.30.4.675. URL `https://doi.org/10.1152/jn.1967.30.4.675`. PMID: 6035687.

[78] Klas Tybrandt, Dion Khodagholy, Bernd Dielacher, Flurin Stauffer, Aline Renz, Gyorgy Buzsáki, and János Vörös. High-density stretchable electrode grids for chronic neural recording. *Advanced Materials*, 30: 1706520, 02 2018. doi: 10.1002/adma.201706520.

[79] Raghavendra U, U Rajendra Acharya, and Hojjat Adeli. Artificial intelligence techniques for automated diagnosis of neurological disorders. *European Neurology*, 82, 11 2019. doi: 10.1159/000504292.

[80] Daniel Valencia and Amir Alimohammad. Neural spike sorting using binarized neural networks. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, PP:1–1, 12 2020. doi: 10.1109/TNSRE.2020.3043403.

[81] Jan Voogd and Mitchell Glickstein. The anatomy of the cerebellum. *Trends in Neurosciences*, 21(9): 370–375, 1998. ISSN 0166-2236. doi: https://doi.org/10.1016/S0166-2236(98)01318-6. URL `https://www.sciencedirect.com/science/article/pii/S0166223698013186`.

[82] Abraham Vázquez-Guardado, Yiyuan Yang, Amay Bandodkar, and John Rogers. Recent advances in neurotechnologies with broad potential for neuroscience research. *Nature Neuroscience*, 23:1522–1536, 12 2020. doi: 10.1038/s41593-020-00739-8.

[83] Jun Wang and Shengchen Li. Comparing the influence of depth and width of deep neural network based on fixed number of parameters for audio event detection. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2681–2685, 2018. doi: 10.1109/ICASSP.2018.8461713.

[84] Yuwei Wang, Hongrui Luo, Yang Chen, Zihao Jiao, Quan Sun, Lei Dong, Xinlei Chen, Xiaofei Wang, and Hong Zhang. A Closed-Loop neuromodulation chipset with 2-level classification achieving 1.5-vpp CM interference tolerance, 35-db stimulation artifact rejection in 0.5ms and 97.8%-sensitivity seizure detection. *IEEE Trans Biomed Circuits Syst*, 15(4):802–819, September 2021.

[85] Jasper Wouters, Fabian Kloosterman, and Alexander Bertrand. A neural network-based spike sorting feature map that resolves spike overlap in the feature space. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1175–1179, 2020. doi: 10.1109/ICASSP40776.2020.9053530.

[86] Lijuan Xia, Ahmed Soltan, Junwen Luo, Eric Chester, and Patrick Degenaar. A flash-fpga based rodent control system for closed-loop optogenetic control of epilepsy. pages 1–5, 05 2018. doi: 10.1109/ISCAS.2018.8351355.

[87] Jia-Min Xu, Ce-Qun Wang, and Long-Nian Lin. [multi-channel in vivo recording techniques: signal processing of action potentials and local field potentials]. *Sheng Li Xue Bao*, 66(3):349–357, June 2014.

[88] Yuning Yang and Andrew J. Mason. Hardware efficient automatic thresholding for neo-based neural spike detection. *IEEE Transactions on Biomedical Engineering*, 64(4):826–833, 2017. doi: 10.1109/TBME.2016.2580319.

[89] Yuning Yang, Sam Boling, and Andrew J. Mason. A hardware-efficient scalable spike sorting neural signal processor module for implantable high-channel-count brain machine interfaces. *IEEE Transactions on Biomedical Circuits and Systems*, 11(4):743–754, 2017. doi: 10.1109/TBCAS.2017.2679032.

[90] Yuning Yang, Sam Boling, and Andrew J. Mason. A hardware-efficient scalable spike sorting neural signal processor module for implantable high-channel-count brain machine interfaces. *IEEE Transactions on Biomedical Circuits and Systems*, 11(4):743–754, 2017. doi: 10.1109/TBCAS.2017.2679032.

[91] Yue Yuan, Lvpiao Zheng, Zhouyan Feng, and Gangsheng Yang. Different effects of monophasic pulses and biphasic pulses applied by a bipolar stimulation electrode in the rat hippocampal ca1 region. *BioMedical Engineering OnLine*, 20(1):25, Mar 2021. doi: 10.1186/s12938-021-00862-y. URL `https://doi.org/10.1186/s12938-021-00862-y`.

[92] Majid Zamani, Dai Jiang, and Andreas Demosthenous. An adaptive neural spike processor with embedded active learning for improved unsupervised sorting accuracy. *IEEE Transactions on Biomedical Circuits and Systems*, 12(3):665–676, 2018. doi: 10.1109/TBCAS.2018.2825421.

[93] Majid Zamani, Dai Jiang, and Andreas Demosthenous. An adaptive neural spike processor with embedded active learning for improved unsupervised sorting accuracy. *IEEE Transactions on Biomedical Circuits and Systems*, 12(3):665–676, 2018. doi: 10.1109/TBCAS.2018.2825421.

[94] Pu-Ming Zhang, Jin-Yong Wu, Yi Zhou, Pei-Ji Liang, and Jing-Qi Yuan. Spike sorting in multi-channel extracellular recordings of retinas. In *International Conference on Neural Networks and Signal Processing, 2003. Proceedings of the 2003*, volume 1, pages 712–715 Vol.1, 2003. doi: 10.1109/ICNNSP.2003.1279374.

[95] Andy Zhou, Benjamin C Johnson, and Rikky Muller. Toward true closed-loop neuromodulation: artifact-free recording during stimulation. *Current Opinion in Neurobiology*, 50:119–127, 2018. ISSN 0959-4388. doi: https://doi.org/10.1016/j.conb.2018.01.012. URL `https://www.sciencedirect.com/science/article/pii/S095943881730243X`. Neurotechnologies.

[96] Andy Zhou, Samantha Santacruz, Benjamin Johnson, George Alexandrov, Ali Moin, Fred Burghardt, J.M. Rabaey, Jose Carmena, and Rikky Muller. A wireless and artefact-free 128-channel neuromodulation device for closed-loop stimulation and recording in non-human primates. *Nature Biomedical Engineering*, 3, 01 2019. doi: 10.1038/s41551-018-0323-x.

[97] Xiaoping Zhu, Longtao Yuan, Dong Wang, and Yaowu Chen. Fpga implementation of a probabilistic neural network for spike sorting. In *2010 2nd International Conference on Information Engineering and Computer Science*, pages 1–4, 2010. doi: 10.1109/ICIECS.2010.5677694.