ROKAI JÁNOS

Funkcionális Idegtudományok
című program

Programvezető: Dr. Sperlágh Beáta, c. egyetemi tanár

Témavezető: Dr. Márton Gergely, tudományos főmunkatárs

SPIKE SORTING USING DEEP LEARNING

PhD thesis

János Rokai, MD

János Szentágothai Doctoral School of Neurosciences Semmelweis University

Supervisor:	Gergely Márton, Ph.D
Official reviewers:	Péter Barthó, Ph.D
	Csaba Dávid, Ph.D

Head of the Complex Examination Committee:	Alán Alpár, MD, D.Sc
Members of the Complex Examination Committee:	László Acsády, D.Sc
	Zoltán Somogyvári, Ph.D

Budapest
2023

Table of Contents

List of Abbreviations

AE	Autoencoder
ASIC	Application-Specific Integrated Circuits
BCI	Brain-computer interface
Bi-LSTM	Bidirectional long-short term memory
CDBM	Coral Development Board Mini
CNN	Convolutional Neural Network
CNS	Central Nervous System
CPU	Central processing unit
CUA	Coral USB Accelerator
DBS	Distance between clusters
DWT	Discrete Wavelet transform
ECoG	Electrocorticography
EEG	Electroencephalography
FLOPS	Floating point operation per second
FN	False negative
FP	False positive
FPGA	Field-Programmable Gate Array
GPU	Graphical processing unit
ICA	Independent Component Analysis
LFP	Local field potential
LSTM	Long-short term memory
MEA	Microelectrode array
MES	Mean Embedding similarity
MST	Minimum Spanning Tree
MUA	Multiple Unit Activity
NMS	Non-max suppression
NN	Nearest-neighbor
NNCLR	Nearest-neighbor contrastive learning
P2P	Peak to Peak
PCA	Principal Component Analysis

PNR	Positive-negative label ratio
RELU	Rectified Linear Unit
SNR	Signal to Noise Ratio
SPC	Superparamagnetic clustering
SSD	Single Shot Detector
SUA	Single Unit Activity
TEO	Teager energy operator
TES	Template Embedding Similarity
TN	True negative
TOPS	Tera operations per second
TP	True positive
TPU	Tensor Processing Unit
t-SNE	t-distributed stochastic neighbor embedding
VAE	Variational Autoencoder

1. Introduction

1.1. Electrophysiology

The development of electrophysiological techniques has progressed significantly over the centuries. Luigi Galvani's observation in 1791 of electric currents inducing muscle contraction marked a pivotal moment. In 1952, Hodgkin and Huxley's invention of the "voltage clamp" technique and Hubel and Wiesel's use of electrophysiological recordings for understanding higher visual processing further advanced the field. Today, electrophysiology encompasses a wide range of complex solutions.

Extracellular recording, particularly with small-diameter electrodes, has become a dominant technique in neuroscience research. It enables the detection of action potentials from individual neurons, providing insights into neuron-level activity.

Electrophysiology techniques can be categorized based on the recording-site used. Electroencephalography (EEG) measures the summed extracellular activity of cortical pyramidal cells using on-scalp electrodes. It is a non-invasive method but offers low spatiotemporal resolution of brain activity. Electrocorticography (ECoG) involves placing electrodes on the brain surface, providing better spatiotemporal resolution but requiring invasive installation. In-depth electrodes offer the highest spatiotemporal resolution but involve the most invasive installation procedure.

To address the challenge of mixed signals from nearby neurons, spike sorting algorithms were developed. These algorithms separate single-unit activity from multiunit activity, allowing for precise analysis and decoding of neural signals.

This work focuses specifically on the signals and analysis of in-depth electrodes. With their high spatiotemporal resolution, in-depth electrodes distinguish two major components: the low-frequency local field potential (LFP) and the high-frequency extracellular manifestation of action potentials from nearby individual neurons, also known as "spikes."

1.2. Local Field Potential

The extracellular voltage fluctuations generated by the collective activity of neurons in a local region of the brain are referred to as the local field potential. These fluctuations are produced by transmembrane currents, which can originate from various sources, including synaptic activity, fast activity potentials, calcium spikes, intrinsic currents and spike afterhyperpolarizations (1). The contribution of the different currents to the recorded LFP is proportional to their distance from the recording electrode. Due to the multitude of contributing sources, interference occurs both spatially and in the frequency domain. Since distance is the primary factor influencing the nature of LFP, its structure is predominantly determined by the activity of neighboring cells, thereby providing a localized measure of neural activity surrounding the electrode.

These fluctuations can be detected using microelectrodes implanted in the brain tissue. The LFP signal is typically measured as a voltage fluctuation over time, and it has a characteristic frequency range of a few Hz to a few hundred Hz (usually up to 300 Hz).

1.3. Unit Activity

Unit activity, also known as spike, refers to the electrical activity generated by individual neurons, more specifically to the action potentials.

The axonal membrane potential of neurons gives rise to action potentials. These action potentials are constructed by different channels located in the axonal membrane, including voltage-gated sodium (Na^+) channels and potassium (K^+) channels.

The membrane of a neuron at resting state is typically impermeable to most charged ions, which enables ion pumps to distribute different ions in different proportions between the extracellular and intracellular spaces. This process results in higher concentrations of Na^+, Ca^{++}, and Cl^- ions outside of the cell, while K^+ ions are more concentrated inside the neuron. However, passive K^+ channels allow for the flow of K^+ ions between the two sides of the membrane, which maintains the membrane potential close to the potassium reversal potential (resulting in a membrane potential of around -70mV) described by the Nernst equation. In the course of an action potential, voltage-gated Na+ channels open, facilitating the rapid influx of Na+ ions into the neuron, causing the membrane potential to deviate towards the reversal potential of sodium (approximately +55mV). The principles and mechanisms behind the electrical activity of neurons were first described by Hodgkin and Huxley (2–4).

The membrane potential is the result of ionic currents that are able to pass through the membrane in a proportional manner to their permeability, which is regulated by ion channels. At the dendritic level, synaptic activities are convolved and summed, and if the summed value reaches a threshold at the axon initial segment, it can trigger an action potential in an "all-or-nothing" fashion. Fast voltage-gated Na^+ channels are major contributors to the initiation of action potentials. These channels provide the Na^+ currents that are responsible for the sharp depolarizing effect at the start of the action potential.

The effects of the action potential are not limited to intra-cellular processes and can be measured extracellularly with various recording techniques.

The extracellular effect, or the unit activity can be measured with inserted measuring electrodes into the extracellular space, where single unit activity (SUA) and multiple unit activities (MUA) can be recorded.

One important characteristic of SUA is its frequency range. The frequency range of SUA typically falls between 300-3000 Hz. This high frequency range is due to the fast transmembrane processes involved in action potential generation. The amplitude of unit activities recorded with an electrode is strongly influenced by the distance between the source of the activity and the electrode. This can make it challenging to detect spikes from neurons that are further away, as their amplitudes may be lower and more difficult to distinguish from background noise. In addition to frequency, the amplitude and waveform of SUA signals can also provide information about the neuron's behavior.

The activity of the nearby neurons around the measuring electrode can provide information about their role in different processes, contributing to a more complete understanding of various brain functions, such as the thalamus or hippocampus. Electrophysiological recordings have been instrumental in identifying and characterizing different types of neurons and their interactions within neural networks, such as the hippocampal place cells (5,6) or the grid cells (7,8) in the entorhinal cortex. Place cells fire selectively in specific locations or environments, indicating the neural representation of space, while grid cells exhibit a regular firing pattern, representing the animal's location on a two-dimensional plane. These findings have played a substantial role in enhancing our understanding of the neural mechanisms that underlie spatial navigation and memory. The identification and characterization of different types of neurons provide insights into the functional organization of the brain and have implications for understanding various neurological and psychiatric disorders. Extracellular recordings have also been used in neuroprosthetic devices (9–11), where neural activity is recorded and used to control artificial limbs or other devices.

Spike sorting is a critical step in the analysis of extracellular recordings and involves identifying and separating individual SUA signals from other sources of electrical activity. Spike sorting can be a challenging problem due to the variability of SUA waveforms, the presence of noise, and the potential overlap of signals from different neurons. Several algorithms have been developed to address this problem, including template matching, principal component analysis, and clustering algorithms.

1.4. Spike sorting process

1.4.1. Filters and Detectors

The first step in spike detection is usually filtering the wideband data from local field potential signals, low and high-frequency noises. This filtering is often performed in the frequency domain and typically falls between 300 and 3000 Hz (12,13). Once the data has been filtered, various methods can be used to detect the action potentials.

One common approach is the threshold-based method (14–16), where a threshold is set based on the median value of the given channel (eq 1.).

Other methods use different types of energy operators to identify spikes (17–20). Wavelet decomposition is another approach (21), where the signal is decomposed into different frequency bands using wavelets, and spikes are detected in each channel separately.

$$T^c = \frac{\text{median}(D^c)}{0.6745} \quad (1)$$

1.4.1.1. Teager energy operator thresholding TEO

The Teager energy operator (TEO) is a non-linear signal processing technique that provides an estimate of the instantaneous energy of a signal. TEO is used in various applications such as speech processing, image processing, and biomedical signal processing. In spike detection , TEO can be used as a thresholding technique to detect spikes in extracellular recordings (18,22).

TEO is calculated as the difference between the squared value of the signal and the product of the adjacent samples (eq.2). The resulting energy signal emphasizes high-frequency components and suppresses low-frequency components, making it effective for detecting spikes in extracellular recordings.

$$TEO(x,t) = x_t^2 - x_{t-1}x_{t+1} \ (2)$$

The thresholding process involves setting a threshold value based on the estimated noise level of the recording. The signal is then compared to the threshold, and samples above the threshold are considered spike candidates. The TEO thresholding technique is effective at detecting spikes in noisy recordings, as it is less sensitive to low-frequency noise than other thresholding techniques.

One advantage of TEO thresholding is its computational efficiency, as it requires only simple arithmetic operations. Additionally, TEO thresholding can be easily adapted to various types of extracellular recordings, including recordings with different electrode configurations and sampling rates.

However, TEO thresholding has some limitations. It may fail to detect spikes with low energy levels, and it is sensitive to non-stationary noise, which can result in false detections. Therefore, TEO thresholding is often used in combination with other spike detection techniques to improve the overall spike detection performance.

1.4.1.2. Template matching

Template matching is a widely used method in spike detection, where spikes are detected by comparing the recorded waveform with a set of pre-defined templates (23,24). The basic idea is that the shape of the recorded waveform for a given neuron remains constant from one spike to another, and therefore, a template can be constructed for each neuron based on a subset of its spikes. The templates are then used to detect new spikes by computing the correlation between the recorded waveform and each template.

In the template matching approach, the detection process typically involves two steps: template construction and template matching (24). In the template construction step, a set of waveforms that correspond to a single neuron is extracted from the recorded data. These waveforms are then aligned in time, relative to a common reference point, typically the peak or trough of the waveform, to account for small variations in the time of occurrence of the spike. The aligned waveforms are then averaged to produce a template waveform that represents the typical shape of the spikes for that neuron. The template waveform is then normalized to have unit energy and used in the template matching step.

In the template matching step, the recorded waveform is compared with each template waveform using a correlation coefficient or other similarity measure. The template waveform that maximizes the similarity measure is considered to be the best match, and a spike is detected if the similarity measure exceeds a certain threshold. The threshold is typically set to a value that ensures a low false positive rate while still detecting a high percentage of true spikes.

Template matching has several advantages as a spike detection method. It is highly specific to individual neurons and can be used to detect spikes from overlapping neurons with high accuracy. It is also computationally efficient and can be easily implemented on hardware devices. However, it requires a set of pre-defined templates for each neuron, which can be time-consuming to construct and may not capture all the variability in the shape of the spikes. Additionally, the method may not be robust to changes in the waveform shape due to changes in the electrode position or other factors.

1.4.1.3. Discrete Wavelet transform

Discrete Wavelet transform (DWT) is often used to filter out noise and baseline drift, leaving behind the high-frequency spikes (25,26). The DWT decomposes the signal into different scales and time-frequency domains, allowing the extraction of specific signal features at different resolution levels. This decomposition enables a multi-scale analysis of the signal, which can be useful in detecting spikes that occur at different amplitudes and frequency ranges. The DWT coefficients are then thresholded to remove noise and extract only the wavelet coefficients that correspond to the spike waveform. After the thresholding step, the remaining wavelet coefficients can be reconstructed back into a spike waveform. This waveform can then be compared to a template waveform to determine if it matches a known spike shape. If the waveform matches the template, it is classified as a spike.

1.4.1.4. Deep learning methods

More recently, deep learning methods have also been applied to spike detection. For example, a single dense layer has been used to detect spikes (27), and LSTM layers have been used to evaluate recordings per data point (28). These approaches use neural networks to learn patterns in the data that correspond to spikes and can achieve high accuracy in detecting spikes.

In case of (27), fixed-length snippets are extracted from the filtered timeseries, and spike detection is performed individually for each channel. The fixed-length snippets are then inputted into a single dense layer. Following this, a Rectified Linear Unit (RELU) activation function is applied, and the resulting output is aggregated into a single output neuron. The output neuron employs a sigmoid function to produce the probability of the input snippet containing a spike or not.

In a previous study (28), the combination of Convolutional layers and LSTM cells was explored. The aim of this hybrid architecture was to exploit the spatial filtering capabilities inherent in Convolutional layers, while also incorporating the memory and temporal modeling abilities provided by LSTM cells. By integrating these two components, the network was able to simultaneously consider both the local spatial context and the sequential temporal context during the process of making spike detection decisions.

1.4.2. Feature extraction

In spike sorting, the extraction of distinctive features from the spikes is crucial for accurate signal decoding.

After successful data filtering and action potential detection, the spikes need to be realigned for further analysis. This involves binning them into fixed-length windows and aligning based on a temporal reference point, like maximum value or slope (29,30). While this method is crucial for clustering alternatives, it may be limited by high noise corruption (31). Upsampling and super-resolution alignment can mitigate these issues (32).

Among the widely favored techniques for reducing dimensionality is principal component analysis (PCA) (33–38). This method involves creating a matrix of orthogonal basis vectors found within the feature space that encompass the most significant variations. A similar alternative to PCA, the independent component analysis (ICA) has been also used in several studies (39–41).

Other feature extraction methods consist of optimal wavelet transforms (42), wavelet packet decomposition used with support vector machine (43) or Laplacian eigenmaps (16).

1.4.3. Clustering

Clustering plays a crucial role in decoding extracellular action potentials. Clustering algorithms can be categorized as model-based or non-model-based, acknowledging the substantial differences within each group.

Model-based approaches, such as Bayesian methods, expectation maximization, and maximum likelihood estimation, utilize spike probability distributions provided by generative models. These methods exhibit resilience to noise associations and enable cluster visualization (44). On the other hand, non-model-based methods focus on classification tasks. Manual clustering relies on apparent factors such as spike amplitude, duration, and channel location as indicators for patterns. These approaches have progressively given way to minimally supervised or unsupervised methods, including k-means clustering from the partitional subclass.

Learning-based clustering incorporates diverse approaches, ranging from single-layer perceptrons to advanced spiking neural networks, to improve spike sorting performance.

Consensus and ensemble clustering capitalize on the diversity present in various clustering algorithms.

In conclusion, there is no universally applicable clustering solution for spike sorting, and the selection of an algorithm depends on the specific characteristics of the data and feature set under consideration.

1.4.3.1. K-means clustering

K-means clustering is commonly used in spike sorting due to its simplicity, alongside other techniques such as hierarchical, graph-based, fuzzy logic, density-based, grid-based, and learning-based methods. In K-means clustering, spike waveforms are assigned to clusters by minimizing within-cluster variance or maximizing between-cluster separation. The algorithm iteratively updates cluster centroids until convergence is achieved. The choice of the number of clusters (K) is crucial and depends on the complexity of neural activity and desired separation level. Determining the optimal K value can be challenging, requiring domain knowledge and validation techniques like silhouette analysis. A notable limitation of K-means clustering is its underlying parametric assumption of spherical clusters, relying on the Euclidean distances from the centroids. This assumption proves inadequate for representing non-spherical neural data distributions, such as the ellipsoidal clusters encountered during electrode drift. Consequently, this parametric constraint may lead to suboptimal division of clusters and compromises the accuracy of spike sorting outcomes.

1.4.3.2. Superparamagnetic clustering

Hierarchical solutions in clustering analysis utilize Euclidean distance algorithms for optimal filter estimation. Graph-based clustering methods, such as spectral clustering and super-paramagnetic clustering (SPC) in wave_clus, incorporate nearest neighbor interactions. SPC draws inspiration from magnetic particle alignment to guide clustering. It involves randomly assigning spike waveforms to clusters, evaluating clustering energy based on spike similarity, and applying a sampling procedure to minimize energy by iteratively reassigning spikes. The process continues until a low-energy state is achieved, resulting in distinct clusters representing different neurons.

1.4.3.3. HDBScan

Density-based algorithms, like Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBScan), resemble human clustering strategies by focusing on agglomerated regions and low-density belts in the feature space. HDBScan, widely used in data mining and pattern recognition, has potential applications in spike sorting due to its ability to handle density variations and noise. HDBScan incorporates a hierarchical clustering approach to identify dense regions in the data based on point density. It constructs a Minimum Spanning Tree (MST) on the distance graph of the data points to capture the connectivity between these regions. The MST is then condensed into a dendrogram using hierarchical clustering techniques, revealing the hierarchical relationships between clusters of different densities. To determine the final clusters, HDBScan employs a stability-based approach. It assesses the persistence of cluster memberships across different levels of the dendrogram, allowing for the identification of robust and meaningful clusters while disregarding unstable or noise-like structures.

1.4.3.4. ISO-SPLIT

The algorithm in ISO-SPLIT makes two key assumptions about cluster distributions within this feature space. Firstly, it assumes that each cluster originates from a density function that, when projected onto any line, exhibits unimodality with a single region of highest density. Secondly, it assumes that distinct clusters can be separated by a hyperplane, which is characterized by a relatively lower density in its surrounding neighborhood. These assumptions have been observed to hold for the majority of neurons across various brain regions, aligning with similar implicit assumptions found in many other neural clustering methods (45).

1.5. End-to-end / automated solutions

1.5.1. Online spike sorters

On-chip spike sorters represent contemporary tools utilized for decoding neural spikes recorded from neural systems. These sorters predominantly rely on Field Programmable Gate Arrays (FPGAs) (46–49) or Application-specific Integrated Circuits (ASICs) (50–52). Initially designed for conventional hardware, the Osort algorithm was subsequently adapted for FPGA platforms, combined with template matching as a clustering method (46,48). The FPGA implementation of Osort involves optimizing the algorithm to minimize memory access and numerical operations. In the most recent iterations of FPGA-based Osort, spatial correlation between channels is considered, utilizing a spatial window that operates on cluster memory (49). Another noteworthy FPGA-based technique is Hebbian PCA, which entails iteratively training eigenvectors for spike prediction (51,53,54). The growing number of publications focused on hardware-based spike sorting underscores the increasing emphasis on on-chip processing of neural signals (55).

1.5.2. Offline spike sorters

1.5.2.1. Kilosort

Kilosort is a widely recognized and widely used method for end-to-end spike sorting in neuroscience research. Over time, Kilosort has undergone several updates and iterations

(56,57), each incorporating various improvements to its pipeline, enhancing its performance and capabilities. In Kilosort, a series of pre-processing steps are employed to prepare the data for spike sorting. These steps include common average referencing, temporal filtering, and spatial whitening, which collectively aim to eliminate low-frequency fluctuations and correlated noise that may interfere with accurate spike detection and classification. To address common noise sources and fluctuations across channels, Kilosort utilizes zero-phase component analysis, a technique that effectively whitens the data. This process ensures that the data from different channels are adjusted to have comparable statistical properties, reducing the impact of spatial variations in noise and enhancing the accuracy of subsequent spike sorting steps.

The algorithm models mean spike waveforms using singular value decomposition, incorporating customized "private" principal components for each spike. This method effectively reduces residual waveform variance compared to standard PCA approximation per channel.

To infer spike times, cluster assignments, and amplitudes, Kilosort employs an integrated template matching framework. It identifies local maxima of dot products between template waveforms and raw data, considering amplitude proximity to the mean amplitude of the template. Spatiotemporal overlapping spikes are detected using a matching pursuit algorithm. Spike times, templates, and amplitudes are optimized by minimizing a quadratic function. Stochastic batch optimization is utilized to learn the templates.

The overall process of Kilosort involves iterative optimization steps alternating between template matching and template optimization. Post-optimization merging is performed to identify pairs of clusters with continuous spike densities. The algorithm initializes with prototypical spikes and applies scaled K-means clustering based on local spike density.

A key goal of Kilosort is to minimize a cost function that considers the difference between the recorded voltage and the model's predicted voltage. Additionally, the cost function includes a term that restricts the number of spikes with significantly deviant amplitudes. Spike times, amplitudes, templates, and cluster assignments are updated to optimize the cost function. The learning and inference steps are iterated until convergence, with additional steps for spike merging.

1.5.2.2. Spyking Circus

SpyKING CIRCUS (58) is an algorithm specifically designed for robust and accurate spike sorting, offering advanced techniques to handle neuronal spike data recorded from extracellular electrodes. The algorithm consists of two main steps: clustering and template matching, each incorporating various processes to optimize spike sorting performance.

During spike detection, SpyKING CIRCUS identifies spikes by detecting threshold crossings. Extracellular waveforms corresponding to randomly chosen spike times are isolated to capture the characteristics of the detected spikes. These waveforms, known as snippets, are then grouped based on their respective electrode positions. Clustering is performed separately on each group, allowing for the separation of different neurons within the dataset.

To reduce noise and focus on relevant information, the snippets are masked, taking into consideration that a single cell's influence is localized to nearby electrodes. By retaining the signal on electrodes close to the voltage peak, the algorithm effectively reduces the memory requirements for each clustering step. PCA is employed to project the masked snippets into a lower-dimensional feature space. This step aids in feature extraction and dimensionality reduction, a common practice in spike sorting algorithms.

SpyKING CIRCUS utilizes density-based clustering to identify centroids representing putative neurons within each group. The algorithm measures the density of points in the neighborhood of each snippet, taking into account the average distance to the 100 closest points and the distance to the closest point with higher density. By selecting the ten points with the highest ratio of distance-to-density, the algorithm determines the centroids. Remaining snippets are then assigned iteratively to the nearest centroid with the highest density, resulting in the classification of spikes to specific neurons.

To address the issue of overlapping spikes, SpyKING CIRCUS incorporates a template matching step. Templates are extracted from the clusters and consist of averaged extracellular waveforms evoked by putative cells, along with the direction of the largest variance orthogonal to these average waveforms. The algorithm assumes that each waveform triggered by a cell is a linear combination of these components, allowing for variations in amplitude and shape. The template matching process uses a greedy iterative approach inspired by the projection pursuit algorithm, comparing the templates to the raw data. The similarity between the first component of the template and the raw signal is evaluated, and if it falls within predetermined thresholds, the template is matched and subtracted from the raw signal. This process continues until no further matches can be made, resulting in the identification of spikes associated with specific cells.

Upon completion of the algorithm, SpyKING CIRCUS outputs putative cells, characterized by their templates, along with the corresponding spike times at which the templates were matched to the raw data. This comprehensive approach ensures accurate spike sorting and provides valuable information for further analysis and interpretation of neuronal activity.

1.5.2.3. MountainSort4

MountainSort4 (45) is an advanced spike sorting algorithm that utilizes the ISO-SPLIT clustering algorithm as its core component. MountainSort4 is a spike sorting algorithm used to analyze neuronal activity in multi-electrode recordings. It involves several stages of preprocessing, event detection, feature extraction, clustering, and cluster consolidation.

The preprocessing stage includes bandpass filtering each channel of the recorded signal between 600 Hz and 6000 Hz. Spatial whitening is then applied to remove correlations among channels that are not due to the neuronal signals of interest. This whitening step is crucial for separating nearby clusters.

Event detection is performed independently on each electrode using the preprocessed (whitened) data. An event is flagged when it meets two criteria: it exceeds a detection threshold in standard deviations away from the mean, and there is a minimum time difference between two events on the same channel.

Feature extraction is performed by extracting event clips centered around the detected events. PCA is then applied to compute PCA features in a high-dimensional feature space. Clustering is performed using the ISO-SPLIT algorithm in the n-dimensional feature space (n=10). The branch method is used to increase sensitivity in distinguishing between nearby clusters by recursively recomputing PCA features and clustering. Overall, MountainSort4 provides a non-parametric spike sorting approach that handles multi-electrode recordings.

1.6. Challenges

1.6.1. Noise

In electrophysiological recordings, noise can originate from multiple sources, including amplifier and electrode noise, environmental noise, and background neural activity. This noise can introduce distortions and obscuration to spike waveforms, posing challenges in accurately classifying spikes based on their waveform characteristics. Several noise reduction techniques have been developed to address this issue, including filtering, PCA denoising, and wavelet denoising. However, these methods may inadvertently remove or distort crucial signal features, leading to difficulties in spike sorting accuracy.

The foundation of classical signal-detection theory heavily depends on the premise that channels showcase the presence of additive white Gaussian noise, leading to the development of corresponding signal-detection techniques. However, noise in extracellular recordings has been found to deviate from this assumption, being both non-white and non-Gaussian (59). Consequently, many classical techniques are not applicable in this context. Additionally, signal-detection approaches that do not presume Gaussian noise but still rely on understanding the noise distribution encounter obstacles when employed with neural data, primarily due to the absence of precise noise models, especially those that remain valid across diverse experimental configurations.

1.6.2. Lack of ground truth data

One of the main difficulties in spike sorting is the lack of ground truth data. The lack of a gold standard makes it difficult to evaluate the performance of spike sorting algorithms and to compare different methods.

The utilization of synthetic datasets for evaluating spike sorting algorithms offers certain advantages; however, there are notable disadvantages associated with their use. These drawbacks primarily stem from the inherent limitations of synthetic datasets in faithfully representing the complexities and intricacies of real neural data.

One significant disadvantage is the limited representativeness of synthetic datasets. Due to their artificial nature, synthetic datasets often fail to capture the full range of variations and nuances present in real neural recordings. They are typically generated based on simplified assumptions and models, which may not adequately encompass the diverse array of spike shapes, noise patterns, and interneuronal correlations encountered in actual experimental settings. Consequently, the performance assessment of spike sorting algorithms solely based on synthetic datasets may not accurately reflect their real-world capabilities and generalizability.

In contrast, paired datasets, which contain simultaneous recordings of both extracellular and intracellular signals from individual neurons, are an important source of ground truth data. However, paired datasets are relatively scarce and time-consuming to obtain, limiting their use in developing and evaluating spike sorting algorithms.

To address this issue, hybrid datasets have been proposed, which consist of simulated or partially simulated data based on real recordings. These datasets combine the advantages of both simulated and real data, allowing for a greater degree of control over the ground truth while retaining some of the complexity and variability of real data. However, hybrid datasets are still limited in their ability to capture the full range of variability and complexity in real neural recordings, and there may be differences in the statistical properties of simulated and real data that could affect the performance of spike sorting algorithms.

1.6.3. Overlapping spikes

The problem of spatial-temporally overlapping spikes frequently challenges spike sorting algorithms. When different neurons fire within a confined time window, their waveforms can overlap (60). This can cause difficulty in accurately detecting and sorting individual spikes, as they may be merged together or appear as a single large spike, making it difficult to extract suitable subcomponent features for clustering.

To address overlapping spikes, various approaches have been proposed. One straightforward solution involves censoring spikes with double peaks to mitigate the problem, while deconvolution methods can be employed for cases with mild overlap (61). Other strategies include combining pairwise action potential waveform templates at different time shifts to enhance sorting accuracy (62). Alternative methods have also been explored: blind source separation techniques, and careful examination of spike cluster centers can aid in automated template merging and resolving overlapping spikes (34,40,63,64). Wavelet Packets Decomposition and Mutual Information (WM sorting) specifically targets overlapping spikes and has shown superior performance compared to other methods discussed. However, its computational intensity raises concerns about its real-time applicability (65).

Recently, deep learning-based approaches have also been applied to the problem of separating overlapping spikes. For example, some models use recurrent neural networks (66) to model the temporal dependencies of the spike train and estimate the firing times of individual neurons from the overlapping signals.

Nevertheless, the separation of spatial-temporally overlapping spikes remains a challenging problem, and further research is essential to enhance the accuracy and efficiency of spike sorting algorithms in such scenarios.

1.6.4. Drifting

Drift is a prevalent issue encountered in spike sorting and single-unit recordings, arising from the shifting position of recording electrodes over time. The causes and duration of these drifts can vary. Mechanical factors, such as pressure release in the tissue after probe insertion, can result in gradual displacements of cells from their initial positions. Additionally, physiological factors including cardiac and breathing cycles, as well as minor animal movements during recording sessions, can contribute to drift (67–69). Consequently, signals from the same neuron may be detected on different channels at different times, leading to cluster splitting or the formation of erroneously classified groups of action potentials. Furthermore, drifting can affect the shape and amplitude of recorded waveforms, making it challenging to match them with known neuron templates.

Several strategies have been developed to mitigate the impact of drifting in spike sorting. These solutions can be categorized into two main approaches: pre-processing and post-processing (70). Pre-processing drift handling involves non-rigid spatial registration of extracellular signals to correct for drift. This method emulates drift-correction procedures utilized in calcium imaging (71), aiming to maintain the consistency of activity histograms over time (72,73). However, pre-registration introduces distortions to the input data, potentially introducing unwanted artifacts that impact spike sorting performance.

Another approach involves incorporating spike localization and motion correction to facilitate downstream clustering. Precise estimation of spike positions relative to the probe layout can serve as an additional feature for clustering. Advanced techniques such as variational inference (74) or point-source models (75) aimed at estimating spike positions and establishing depth localization. Leveraging estimated positions during clustering enhances the identification and tracking of distinct clusters over time (76,77).

Alternatively, drifts can be addressed at the post-processing stage by dividing the recording into small overlapping temporal intervals and performing sorting independently on each chunk. By analyzing the overlapping regions, connections can be established, and cell identities can be tracked despite minor waveform variations. However, this approach requires running multiple instances of the spike sorting procedure, potentially slowing down the overall processing pipeline.

Overall, the problem of drifting in spike sorting and single-unit recordings is a significant challenge that requires careful consideration in experimental design and data analysis. A combination of multi-channel recordings, drift correction, and denoising techniques can help to mitigate the effects of drifting and improve the accuracy of spike sorting.

1.6.5. Scalability

The advancement of electrode technologies allows for an increase in the number of channels on a single electrode, thereby expanding the ability to gain insights into the activity of larger brain areas. This enhancement enables a superior interpretation of neural population activity (78). However, spike sorting algorithms encounter notable challenges in terms of scalability, necessitating the efficient management of a substantial number of recorded channels and spikes.

Larger electrode arrays and higher channel counts require the processing and analysis of larger data volumes. This entails tasks such as waveform detection, feature extraction, and clustering across multiple channels simultaneously. The complexity of these tasks escalates rapidly with an increasing number of channels, emphasizing the need for efficient algorithms and parallel computing strategies to maintain reasonable processing times. To tackle this challenge, an effective strategy involves embracing a divide-and-conquer processing technique (34). This approach facilitates the identification of duplicated or overlapping spikes, while also furnishing comprehensive spatial information action potentials sources.

The development of scalable algorithms assumes critical importance in extracting meaningful neural information from recorded data while ensuring computational efficiency and accuracy. This necessitates continuous advancements in electrode technologies, as well as the design of effective data processing techniques and algorithms.

2. Objectives

2.1. Applying semi-supervised deep learning methods to spike sorting

Objective 2.1 focuses on utilizing semi-supervised deep learning methods for spike sorting. The primary aim is to develop and implement innovative semi-supervised deep learning architectures specifically designed for spike sorting applications. These architectures will be trained using a combination of labeled spikes, obtained through manual annotation, and synthetic spikes generated through a combination of unsupervised and supervised techniques.

Evaluation of the developed spike sorting solution will involve rigorous testing on both synthetic datasets and real experimental data. Performance metrics such as accuracy, precision, recall, and F1 score will be used to assess the quality of the spike sorting results. Furthermore, the computational efficiency of the proposed method will be thoroughly evaluated to determine its feasibility for real-time or near-real-time applications.

2.2. Developing a deep learning solution for edge devices

Objective 2.2 centers around the development of a deep learning solution that is specifically optimized for Tensor Processing Units (TPUs). This involves adapting existing deep learning architectures and algorithms to efficiently utilize the parallel processing capabilities of TPUs. Special attention is given to designing network architectures that maximize the utilization of TPU's hardware features, while considering memory limitations and communication overhead.

The goal of this objective is to create a deep learning solution tailored for TPUs. By harnessing the power of these specialized hardware accelerators, it becomes possible to achieve local spike sorting efficiently, even with TPUs' compact form-factor. This optimized solution brings the potential for achieving close to state-of-the-art performance in an energy-efficient manner.

3. Methods

3.1. Overview of the working environment and key tools utilized

The working environment in this scenario involves the use of Python 3 along with two important libraries, NumPy and Tensorflow. Python is a popular programming language that is widely used in data science, machine learning, and other scientific applications. NumPy is a library for numerical computing in Python that provides efficient array operations and linear algebra capabilities. Additionally, the recordings are managed through Spikeforest (79), a platform for managing and analyzing extracellular recordings from different sources.

TensorFlow is a popular open-source deep learning framework developed by Google. It provides a comprehensive set of tools for building and deploying deep learning models across a range of platforms and devices. TensorFlow includes a flexible and intuitive programming interface, Keras, that allows developers to define, train, and deploy complex neural networks with ease. It also includes a wide range of pre-built neural network architectures, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers, as well as a variety of optimization algorithms and loss functions.

Spikeforest, on the other hand, is an open-source framework for managing, analyzing, and sharing electrophysiology data from neural recordings. Spikeforest provides a suite of tools for organizing and processing data from multiple recording platforms. The provided database includes synthetic datasets (80,81), paired datasets (58,82) and hybrid datasets as well. It also includes a variety of data analysis tools, such as spike sorting algorithms, and allows users to easily share and collaborate on their data and analysis results.

When used together, Spikeforest and TensorFlow provide a powerful set of tools for analyzing and processing electrophysiological data and building deep learning models for tasks such as spike detection, feature extraction or even end-to-end spike sorting. By converting recordings from Spikeforest into TensorFlow-compatible tfrecord files, users can take advantage of TensorFlow's flexible and efficient data processing capabilities, as well as its extensive library of pre-built deep learning layers.

3.2. Overview of the used datasets

3.2.1. Fiath dataset

The Fiath dataset (83) encompasses nine recordings obtained from a high-density silicon MEA consisting of 128 channels. The MEA features a square-shaped configuration with a 32 × 4 sensor array. The recordings were conducted at a sampling rate of 20 kHz and targeted various neocortical areas, including the somatosensory cortex, parietal association cortex, motor cortex, and cingulate cortex.

To establish spike labels for the dataset, the KiloSort algorithm (56) was employed. Subsequently, the resulting clusters of neural activity underwent manual refinement using Phy and custom-written MATLAB scripts. The manual correction process involved several techniques, such as cluster merging, cluster separation based on Feature-View analysis, and the removal of clusters with fewer than 100 elements, which were considered noise. Clusters demonstrating low inter-unit waveform variability and exhibiting a distinct refractory period were deemed of high quality.

Furthermore, additional custom scripts were applied to refine the results further by removing low-amplitude waveforms identified as units with a peak-to-peak amplitude of less than 60 μV.

3.2.2. Paired datasets

Paired datasets offer valuable advantages as they provide a ground-truth reference for real-world extracellular recordings. However, their utility is constrained by the requirement for intracellular monitoring, resulting in the availability of ground-truth labels for only one neuron. Consequently, the assessment of paired datasets is primarily focused on evaluating the spike detection phase of the spike sorting process.

Since ground-truth labels are available for only one cluster among potentially numerous clusters, the evaluation of clustering becomes limited. Meaningful comparisons can only be made between the ground truth cluster and other clusters. However, evaluating the relationships between different clusters becomes challenging in the absence of ground-truth labels for those clusters.

As a result, in this work, the use of paired datasets is dedicated to the evaluation of spike detection systems.

3.2.2.1. Kampff dataset

The Kampff Lab provided a dataset (82) that utilized both an extracellular silicon MEA and a glass micro-pipette to monitor the potential inside the cell body of individual neurons at varying inter-probe distances, ranging from 800 to 1800 μm deep in the cortex. The measurements were taken at a sampling rate of 30 kHz from the cortex of anesthetized rats. Juxtacellular signals were analyzed using a custom script based on thresholding, resulting in the creation of a binary label for each of the 128-channel recordings in the dataset.

3.2.2.2. Boyden dataset

The Boyden dataset (84) consists of recordings obtained using electrode arrays designed and constructed at the MIT Microsystems Technology Laboratories. The recording sites on the arrays were approximately 9 × 9 μm in size and spaced 2 μm apart in two columns, forming a 64x2 site grid, resulting in a total of 128 channels.

The authors developed an automated system that simultaneously performed patch-clamp recordings and extracellular recordings from the same neuron. By comparing the patch-clamp data with the extracellular recordings, they assessed how well the multielectrode arrays captured the spiking information from the neuron. The dataset includes recordings from the mammalian cortex, specifically investigating how bursting activity affects the performance of spike sorting algorithms on close-packed multielectrode arrays.

3.2.2.3. Yger dataset

The Yger dataset (58) includes recordings of ganglion cells in mice retina using loose patch recordings combined with dense extracellular recordings from 252 channels. The probe layout used is a 16x16 multi-electrode array with 30 μm spacing. However, an 8x8 sub-array was extracted from the original 16x16 array, excluding the four corners, resulting in 64 channels used for the dataset. The recordings were conducted at a sampling rate of 20 kHz.

3.2.3. Hybrid Janelia

Hybrid Janelia dataset comprises a combination of real-world data and synthetic methods to generate a comprehensive recording. The waveform templates used in this study were obtained through high-speed recordings conducted at a sampling rate of 30 kHz, employing electrodes spaced 5 μm apart. These recordings were part of Kampff's Ultra Dense Extracellular Survey, and the corresponding reference can be found in the literature.

Using the real-world waveform templates as a basis, a synthetic dataset was created by introducing various modifications. These modifications involved the addition of noise to simulate real-world conditions, and in some instances, the introduction of drifting. The drifting was implemented using a 2D interpolation technique.

The hybrid nature of the recordings enables the evaluation of spike sorting solutions in a manner that closely mimics real-world extracellular recordings. The Spikeforest platform provides access to individual recordings that are split into two parts, such way, one part can be used for training and validation, while the other part is reserved for testing purposes. Moreover, the recordings within the dataset can be categorized based on the presence or absence of drifting. This categorization provides researchers with flexibility in selecting recordings that align with their specific objectives. Additionally, the dataset encompasses recordings with different channel counts, catering to diverse experimental requirements. For the purposes of this study, 64 and 32 channel recordings were utilized.

3.3. Overview of the evaluation metrics

3.3.1. Standard metrics

Evaluation metrics are used to measure the performance of a model's predictions. Here, we will discuss several evaluation metrics that are commonly used in the context of spike sorting. These metrics are Recall, Precision, F1 (micro, macro, weighted), and Accuracy.

3.3.1.1. Recall

Recall, also known as sensitivity, is a metric that measures the percentage of true positive results that were correctly identified by the model. It is the ratio of the number of true positives to the total number of actual positives (eq. 3). A high recall value indicates that the model is correctly identifying most of the spikes in the data.

$$R = \frac{TP}{TP+FN} \quad (3)$$

3.3.1.2. Precision

Precision, also known as specificity, is a metric that measures the percentage of true positive results out of all the positive results that the model has predicted. It is the ratio of the number of true positives to the total number of positive predictions made by the model (eq. 4). A high precision value indicates that the model has a low rate of false positives.

$$P = \frac{TP}{TP+FP} \quad (4)$$

3.3.1.3. F1 scores

F1 Score is the harmonic mean of precision and recall. It is a measure of the model's accuracy that takes both precision and recall into account. The F1 Score is calculated according to eq. 5. It is a value between 0 and 1, where 1 indicates perfect precision and recall, and 0 indicates the worst possible performance.

$$F_1 = 2 \cdot \frac{R \cdot P}{R+P} \quad (5)$$

$$R_m = \frac{\sum_{j=1}^{C} TP_j}{\sum_{j=1}^{C} (TP_j + FN_j)}, \qquad P_m = \frac{\sum_{j=1}^{C} TP_j}{\sum_{j=1}^{C} (TP_j + FP_j)}, \qquad F_1^m = 2 \cdot \frac{R_m \cdot P_m}{R_m + P_m}. \quad (6)$$

$$F_1^M = \frac{1}{C} \sum_{j=1}^{C} F_{1_j} \quad (7)$$

$$F_1^W = \frac{\sum_{j=1}^{C} n_j \cdot F_{1_j}}{\sum_{j=1}^{C} n_j} \quad (8)$$

In multi-class classification problems, we can calculate the F1 Score for each class. The Micro F1 Score (eq. 6) takes TP, FP and FN globally across categories (C). The Macro F1 Score (eq. 7) is the unweighted average of the F1 Scores of all classes. The Weighted F1 Score is the weighted average of the F1 Scores of all classes, weighted by the number of samples in each class.

3.3.1.4. Accuracy

Accuracy is a metric that measures the percentage of correct predictions made by the model out of all the predictions made. It is the ratio of the number of correct predictions to the total number of predictions made by the model (eq. 9). Accuracy is not always a good metric for evaluating the performance of a model in spike sorting, as it can be misleading in the presence of imbalanced classes or when the dataset contains many noise events.

$$A = \frac{TP}{TP+FN+FP} \quad (9)$$

Recall, Precision, and F1 Score are commonly used metrics for evaluating the performance of a model. Micro F1, Macro F1, and Weighted F1 are useful when dealing with imbalanced datasets.

3.3.2. Custom evaluation methods

3.3.2.1. xSpeed

A custom metrics, xSpeed is introduced to assess the relative acceleration of the sorting speed compared to the actual recording duration. The xSpeed metric serves as a quantitative indicator of the efficiency exhibited by the sorting algorithm, highlighting its ability to process data at an accelerated pace.

To compute xSpeed, two primary parameters are considered: the record duration, representing the total duration of the recorded data, and the sorting time, indicating the time required by the algorithm to complete the sorting process. The xSpeed value is obtained by dividing the record duration by the sorting time, quantifying the factor by which the sorting speed surpasses the actual recording duration (eq. 10).

$$xSpeed = \frac{Record\ duration}{Sorting\ time} \quad (10)$$

A higher xSpeed value denotes a more efficient sorting process, signifying that the algorithm can process the data at a significantly faster rate compared to the duration of the recorded data. This observation implies that the algorithm exhibits timely data handling capabilities, potentially leading to reduced overall processing time for sorting operations.

3.3.2.2. MES

The evaluation of the self-supervised model involved the examination of multiple similarity matrices. To construct the Mean Embedding Similarity (MES) matrix, waveform samples were processed by the model's encoder to generate feature vectors in a latent space. These vectors were then grouped according to their ground truth cluster identity. Using the Euclidean distance calculation, the distances between the feature vectors within each group were determined (eq.11). The resulting distance values were used to populate a similarity matrix (s), which facilitated the assessment of cluster separability within the latent space. To normalize the values in *s*, the minimum and maximum values of *s* were determined. By subtracting the minimum value from each element ($s_{i,j}$) and dividing the result by the difference between the maximum and minimum values, the values in s were normalized. Subsequently, 1 was subtracted from each value to obtain the normalized value for MES (eq. 12).

$$s_{i,j} = \left\| \frac{1}{N} \sum_n Encoder(x_{i,n}^{w}) - \frac{1}{M} \sum_m Encoder(x_{j,m}^{w}) \right\|_2 \quad (11)$$

$$MES_{i,j} = 1 - \frac{s_{i,j} - \text{minimum}\,(s)}{maximum(s)} \quad (12)$$

3.3.2.3. DBS

To create the Distance Between Clusters matrix (DBS), the distances between the different clusters on the channel axis (x^{ch}) were considered. The process of constructing the matrix involved several steps. First, a standard distance matrix (*d*) was created using eq. (13), which calculates the distance between two clusters based on their positions on the channel axis. Next, this matrix was normalized to ensure that all the values were within a specific range. Finally, the values in the matrix were inverted using Eq. (14). This resulted in the final distance matrix DBS, which represents the distance between the different clusters on the channel axis. The values in the matrix are normalized and inverted to ensure that they are easy to interpret and compare.

$$d_{i,j} = x_j^{ch} - x_i^{ch} \quad (13)$$

$$DBS_{i,j} = 1 - \frac{d_{i,j} - \text{minimum}\,(d)}{channel\_number} \quad (1)$$

By combining and normalizing the MES matrix and the DBS matrix, a Combined matrix was formed (eq. 15). This matrix integrates both the positional differences between clusters along the channel axis and the similarity of their feature vectors. It provides a comprehensive view of cluster separability, considering both spatial positioning and feature characteristics.

$$Combined = \frac{MES \cdot DBS}{\text{maximum}\,(MES \cdot DBS)} \quad (2)$$

3.3.2.4. TES

Furthermore, the Template Embedding Similarity matrix (TES) was constructed to investigate the distances between the embeddings of cluster-wise averaged waveforms (eq.16).

$$TES_{i,j} = \left\| Encoder\left(\frac{1}{N}\sum_n x_{i,n}^{w}\right) - Encoder\left(\frac{1}{M}\sum_m x_{i,m}^{w}\right) \right\|_2 \quad (3)$$

3.4. Methods related to applying semi-supervised deep learning methods to spike sorting

3.4.1. Autoencoders

Autoencoder models are an unsupervised deep learning paradigm aimed at reproducing input data while performing feature reduction. A pivotal component in autoencoders is the latent layer, which acts as a bottleneck with significantly fewer nodes than the input. The concept of autoencoders was initially introduced in a seminal publication in 1986 (85).

The autoencoder formulation is defined by the following equations:

$$\begin{aligned} \boldsymbol{z} &= Encoder(\boldsymbol{x}), \\ \boldsymbol{x}' &= Decoder(\boldsymbol{z}), \end{aligned} \quad (4)$$

where the encoder *Encoder* and decoder *Decoder* are differentiable functions, often implemented using deep learning architectures. The latent layer z serves as a compressed representation of the input snippet $\boldsymbol{x}$, while $\boldsymbol{x'}$ represents the reconstructed output.

The traditional principle in constructing autoencoder architectures entails mirroring the encoder structure to design the decoder. However, in this study, a more flexible approach will be adopted, which will be discussed in detail later. During training, the autoencoder endeavors to minimize the reconstruction error, which quantifies the discrepancy between the reconstructed output $\boldsymbol{x'}$ and the original input $\boldsymbol{x}$. This reconstruction error serves as the optimization objective for model training.

Deep learning autoencoders are neural networks specifically designed to learn encoding and decoding processes for data compression in a compressed representation. The underlying idea revolves around capturing a low-dimensional representation of high-dimensional data, yielding reconstructed data that closely approximates the original input. Training an autoencoder involves unsupervised learning techniques applied to a given dataset.

Autoencoders comprise two fundamental components: an encoder and a decoder. The encoder learns to compress the input data into a condensed representation, commonly referred to as a latent code. Subsequently, the decoder employs this latent code to reconstruct the original input data. Both the encoder and decoder are typically implemented as neural networks, and their weights are learned through backpropagation and gradient descent algorithms. The training process aims to minimize the reconstruction loss, which quantifies the discrepancy between the original input data and the reconstructed output data (eq.18). The reconstruction loss is typically evaluated using distance metrics such as mean squared error or binary cross-entropy.

$$L(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=0}^{N} \|x_i - x'_i\|_2^2 = \sum_{i=0}^{N} \left\|x_i - Decoder\big(Encoder(x_i)\big)\right\|_2^2, (18)$$

Autoencoders find applications in diverse domains, including data compression, image denoising, anomaly detection, and dimensionality reduction. By learning a compressed representation of high-dimensional data, autoencoders offer a means to reduce storage and memory requirements for datasets. Moreover, autoencoders have been successfully employed in spike detection for spike sorting tasks, where they learn templates of spike waveforms to facilitate the sorting process.

3.4.2. β-VAE

β-VAE, an extension of the variational autoencoder (VAE), builds upon the core concept of autoencoders. As described previously, autoencoders are neural networks that learn a compressed representation of data through an encoder and a decoder. The VAE, a type of autoencoder, introduces a probabilistic framework for modeling the input data and mapping it to a latent space.

The likelihood *q(z|x)* in VAE is approximated by the probabilistic encoder *q*, assuming a Gaussian distribution for *q(z|x)*. This means that the probabilistic encoder outputs the mean and variance of the normal distribution *q(z|x)*, represented as $z_i \sim \boldsymbol{N}(\mu_i(x), \sigma_i(x))$.

In the context of β-VAE, the conditional probability of data *x* given latent variable *z* can be expressed as:

$$p(\boldsymbol{x}|\boldsymbol{z}) = \frac{q(\boldsymbol{Z}|\boldsymbol{X})p(x)}{p(\boldsymbol{z})} \;.(19).$$

Here, $p(\boldsymbol{x}|\boldsymbol{z})$represents the likelihood of observing data x given latent variable *z*, $q(\boldsymbol{z}|\boldsymbol{x})$ is the approximated likelihood derived from the probabilistic encoder, and $p(\boldsymbol{z})$ is the prior distribution of the latent variables.

The prior distribution $p(\boldsymbol{z})$ is assumed to be a normal distribution, denoted as $p(\boldsymbol{z})$ = **N**(0, I), where **N**(0, I) represents a normal distribution with a mean of zero and a covariance matrix of identity. This assumption ensures that the latent variables follow a standard normal distribution, regularizing the latent space.

The loss function of β-VAE, denoted as $L_{\beta-VAE}(\theta, \varphi; \beta)$, combines the reconstruction loss and the Kullback-Leibler (KL) divergence loss in the following way:

$$L_{\beta-VAE}(\theta, \varphi; \beta) = \mathrm{E}_{q_\varphi(\mathbf{z}|\mathbf{x})}[\log p_\theta\,(\boldsymbol{x}|\mathbf{z})] - \beta \mathrm{D}_{\mathrm{KL}}\left(q_\varphi(\mathbf{z}|\mathbf{x}) \;||\; p(\mathbf{z})\right).(20)$$

In equation (20), $E_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[log\, p_\theta\,(\boldsymbol{x}|\boldsymbol{z})]$ represents the expected reconstruction loss, which measures the ability of the decoder to reconstruct the input data. $D_{KL}\left(q_\varphi(\boldsymbol{z}|\boldsymbol{x}) \;||\; p(\boldsymbol{z})\right)$ denotes the KL divergence loss, which ensures that the distribution of the latent space $q(\boldsymbol{z}|\boldsymbol{x})$ remains close to the prior distribution $p(\boldsymbol{z})$. The hyperparameter β controls the trade-off between the reconstruction loss and the KL divergence loss, allowing for the adjustment of the emphasis on disentanglement.

By tuning the value ofβ, the loss function $L_{\beta-VAE}$ promotes the disentanglement of latent variables in the learned representation. A higher β encourages more disentangled representations, where each dimension of the latent space corresponds to a distinct feature of the input data. Conversely, a lower β places greater emphasis on the reconstruction capability of the network.

In summary, β-VAE leverages the autoencoder concept and introduces a modified loss function to encourage disentangled latent representations. Equations (19) and (20) define the probabilistic framework of β-VAE, including the approximation of the likelihood by the probabilistic encoder and the formulation of the loss function. These equations highlight the probabilistic nature of β-VAE and its ability to learn compressed representations while promoting disentanglement.

3.4.3. Model architecture

The basis of the semi-supervised architecture (ELVISort) is a β-VAE, which is customized to fit the target task. The artificial neural network completes the detection, feature extraction and sorting phases. ELVISort was trained by applying the supervised paradigm to the detection/clustering and the unsupervised paradigm to the reconstruction part of the model.

The input of ELVISort is a 2D matrix of electrophysiological signals, where rows correspond to channels and columns correspond to sampling points in time. A subsidiary goal was to train the network to effectively reconstruct the different input patterns from their compressed representations, which are coded by the different states of the latent space of the autoencoder. A proper representation offers the possibility of distinguishing spikes originating from different sources. To achieve this, multiple branches are used while training the autoencoder to ensure the emergence of a well-balanced latent space which is useful for classification and sorting as well.

In spike analysis, time-domain feature extraction is as important as the inspection of space-domain-specific inter-channel relations. To exploit this concept, the main elements of ELVISort are long short-term memory (LSTM) (86), bidirectional LSTM (Bi-LSTM) (87) and 2D convolutional layers (88,89).

During model development, several architecture combinations were tested, including purely 2D or 3D convolutional networks. To create a latent space with satisfactory generalization capability, good reconstruction capability is needed. As a metric to rank the performance of architecture types, the reconstruction loss was used; the architecture described here (**Figure 1.**) proved to be the best among all. In our experience, LSTM layers were superior to convolution layers in terms of reconstruction when they were used alone; however, when combined, they outperformed the single-type architecture models.

To prevent the model to overfit the training data, two regularization methods were used. The method of early stopping was used in conjunction with several dropout layers probability of *0.5*.

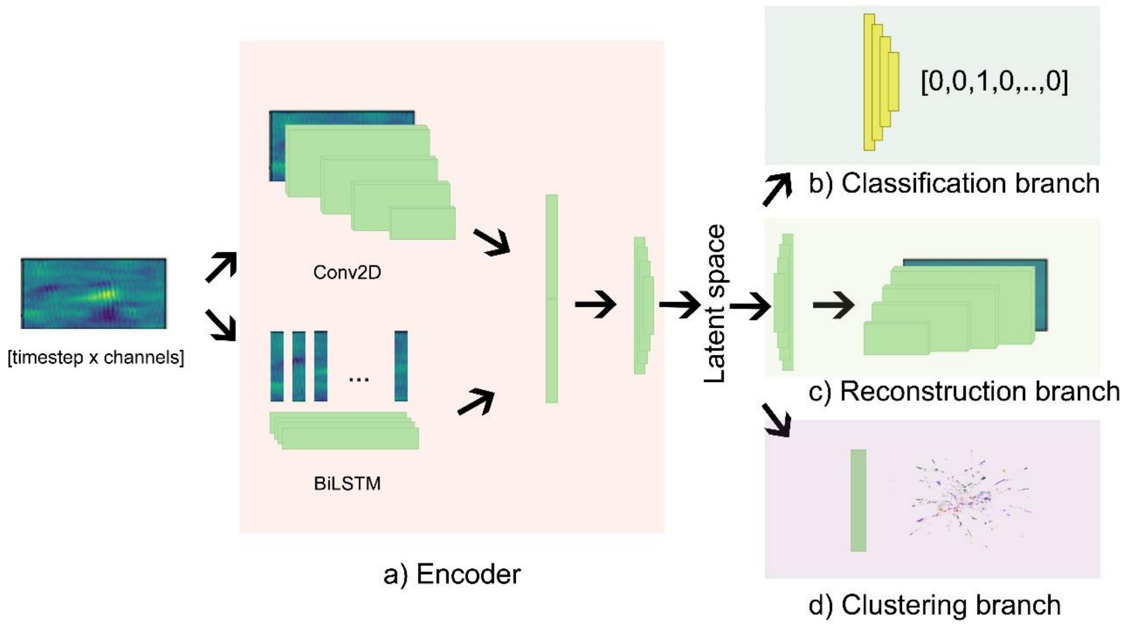


Figure 1. The architecture of ELVISort. The input of the model consists of a 2-dimensional snippet. The encoder (a) contains two branches of different architecture types: BiLSTM and Conv2D. The results of these two branches are concatenated and fed to a series of dense layers, the final dense layer outputs the mean and standard deviation of each latent variable. In contrast to the encoder, the reconstruction branch is using only LSTM and transformer architecture. The output of ELVISort consists of a classification vector (b), a reconstruction of the input (c) and a soft layer assignment vector (d). These are produced by the supervised classifier branch containing dense layers (b)(yellow building blocks), the unsupervised reconstruction (c) and clustering branches (d) (green building blocks) respectively. (90)

The encoder consists of two different branches: the LSTM-based branch processes data in the time domain, having a 2-dimensional matrix as input while the 2D CNN branch extracts spatiotemporal features from a 3-dimensional input. Note that in image processing, the third axis of the input for the first Conv2D layer usually corresponds to the color channels (e.g. R/G/B), in contrast to our model, where samples are lined up along the third axis. For the convolution branch 4 building blocks from GoogLeNet (91) were included beside dropout and convolutional layers. The outputs of the LSTM and

CNN branches are concatenated and combined non-linearly using fully connected layers. The last layer outputs the mean and variance of the latent inference.

In the reconstruction branch, only LSTM elements were used: in our experience, a CNN branch in the reconstruction branch does not ameliorate the overall performance. An improvement in the generalization capability of the model was experienced upon the insertion of attention layers between LSTM layers (92). A custom layer was implemented to handle the inference of the latent variables based on their mean and variance approximated by the encoder. The latent space was constricted to improve clustering, finally a size of 32 was chosen. To further compress information, a hierarchical latent layout was used (93), moreover fully connected layers were applied to the latent variables to further decrease the size of the latent space: the higher latent layer had a total of 8 dimensions, making a total of 40 latent variables. The latent space was visualized using t-distributed Stochastic Neighbor Embedding (t-SNE) (94) and depicted in **Figure 5.**

3.4.4. Preprocessing

Having multiple datasets with different electrode alignments, general data characteristics and data encodings, we made use of the object-oriented features of Python, implementing the preprocessing as a base object and introducing polymorphism to the system only in relation to data and label loading, thus ensuring that every dataset is processed in a similar fashion.

In the preprocessing phase, data is filtered between 300 and 3000 Hz (12)(13) using a Butterworth filter (95) of order 5. After this, a threshold is computed (eq.21) from the median and standard deviation of each channel, according to the formula below (having θ as a multiplier constant for fine-tuning the positive/negative label ratio (PNR) of the generated data).

$$T^c = \frac{\text{median}(D^c)}{0.6745} + \theta \cdot \text{std}(D^c), (21)$$

where D is the filtered data, C is the channel number and T is the threshold. Different, although similar θ values were used for the datasets (Fiath dataset: θ = 3, Hybrid Janelia dataset: θ = 2.5, Kampff dataset: θ = 2) to obtain the best PNR. The following algorithm was performed to generate snippets for the artificial neural network:

Algorithm 1 Sample generation from preprocessed and filtered data

```
Require: $c \in \{0,1,\dots,C\}$ : channel index
Require: $t \in \{0,1,\dots,T\}$ : timestamp
Require: $D_t^c$: preprocessed data-point
Require: $Tr^c$: threshold for the specific channel
Require: $F_t$: filter mask, where $F_t = \begin{cases} 1, & if\ \sum_{i=0}^{C} abs(D_t^i) - Tr^i > 0 \\ 0, & if\ \sum_{i=0}^{C} abs(D_t^i) - Tr^i \leq 0 \end{cases}$
Require: $st$: Snippet timespan in sample
 1: $D_t' \leftarrow \max(abs(D_t)) \cdot F_t$
 2: $t \leftarrow 0$
 3: while $t < T$ do:
 4:     if $D_t' > 0$ then
 5:         $c \leftarrow argmax([D_{t-st/2}', D_{t+st/2}'])$
 6:         if $st - c > st/2$ then
 7:             $t \leftarrow t + c - st/2$
 8:         end if
 9:         $Sample \leftarrow [D_{t-st/2}, D_{t+st/2}]$
10:     end if
11:     $t \leftarrow t + 1$
12: end for
```

The above algorithm generates a filter mask (F_t) for each channel, in which, for each above-threshold data-point the number 1 will be assigned, while for data-points with values not reaching the threshold value, the number 0 is assigned. The whole dataset is filtered with the aforementioned mask, so only the relevant values are kept. For every non-zero data-point, a snippet of length st is generated. If multiple non-zero values are encountered within a snippet length, the one with the largest value will be considered as the center for the particular snippet (Algorithm 1, ln. 5). Important that every snippet is built from the non-filtered data-points, thus maintaining every information for the sorting algorithm.

As snippet timespan, 64 samples for the Fiath and Kampff datasets and 32 samples for the Hybrid Janelia were chosen.

Each recording from the Fiath and Kampff datasets were split into two major parts: training and test data, with the former consisting of the first 70% of the recording and the latter composed of the remaining 30%. The training split was further divided into training and validation splits, using 75% of the data for training and 25% for validation.

In order to obtain a balanced dataset, two counter-measures were taken against the low PNR in the training data: the negative instances were downsampled (only a small random subset of the negative instances were used) and a weighting variable was introduced in the classifier branch loss that penalizes false negatives more than false positives. This weighting variable's value was determined empirically and was found that false negatives needed to be penalized in the Kampff dataset by 3 times more than false positives.

3.5. Methods related to developing a deep learning solution for edge devices

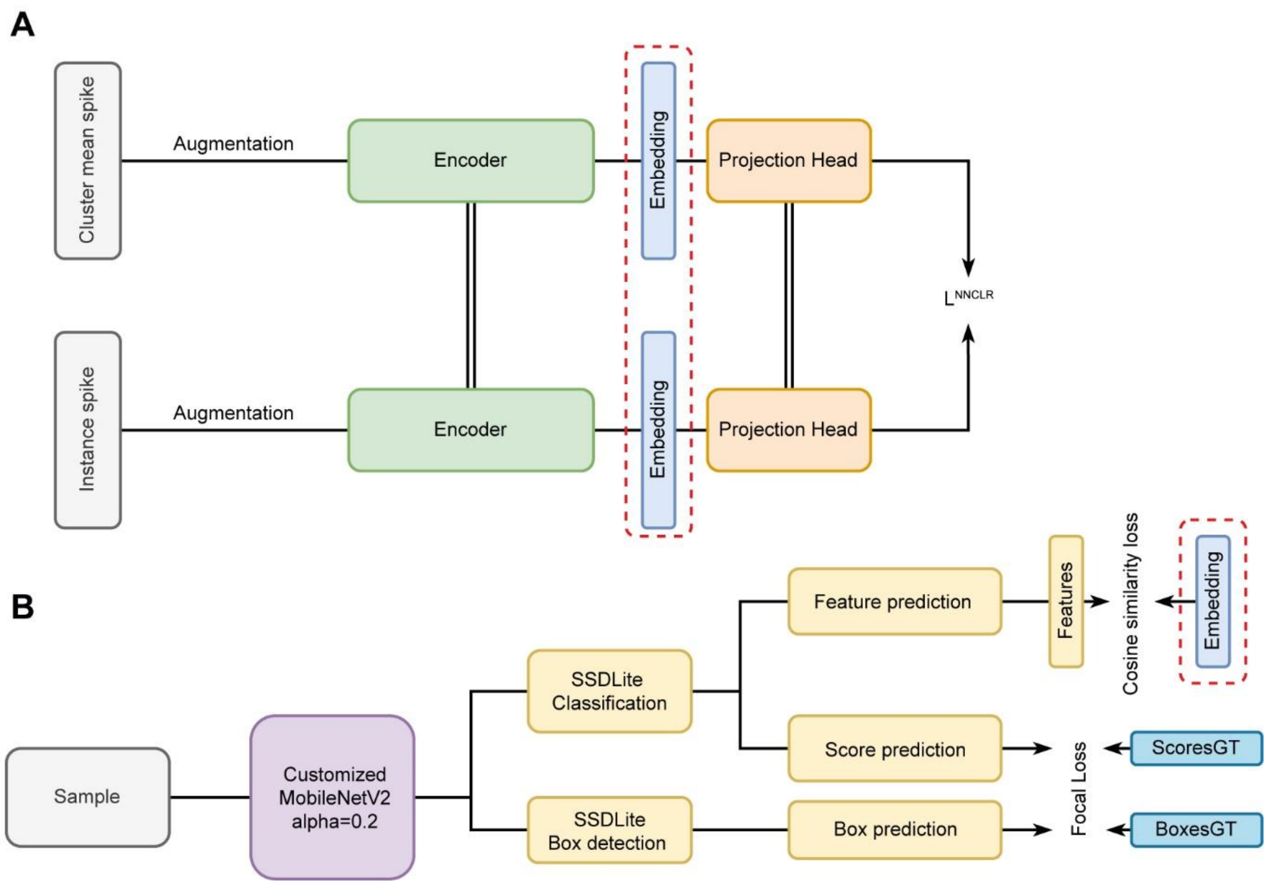


Figure 2. Schematics of the training. In the top subfigure (A), the self-supervised model is depicted during training. Pairs of inputs are provided to the model and are processed by a shared encoder, which produces a feature vector. This feature vector is then passed through a projection head and the NNCLR loss is calculated based on the output of the projection head. The loss is then backpropagated through the model. During inference, only the encoder is used. The resulting feature embeddings are used as labels in the supervised model depicted in the bottom subfigure (B). The supervised model is a single-shot detector type object detection system, which has been modified to include a feature prediction branch. The goal of this branch is to learn the feature embeddings generated by the self-supervised model during training. (96)

3.5.1. Nearest-neighbor contrastive learning

Nearest-neighbor contrastive learning (NNCLR) (97) is a self-supervised learning method based on contrastive learning, in which the model outputs a feature vector. In the contrastive learning paradigm, the model takes in two inputs and generates two different feature vectors. The contrastive loss function then either pulls the vectors together or pushes them apart in the feature space, depending on whether the inputs are considered positive or negative pairs. This results in the model producing similar feature vectors for similar inputs and separable feature vectors for non-similar inputs. The similarity can be (and it is most of the cases) of higher order. To properly utilize this principle and effectively train the model in an unsupervised manner, it is necessary to augment the inputs to produce multiple similar input pairs for the contrastive model.

NNCLR builds upon this principle by using nearest-neighbor to enhance the proximity between different views of the same sample, which are typically produced by data augmentation. In NNCLR, the nearest-neighbor component is used to select a similar point in the feature space (Q) for one of the feature vectors using nearest-neighbor (NN). The dot product is then calculated between the selected similar point and the other feature vector (which is l2 normalized). The other inputs in the mini-batch act as negative samples. The loss function, $\mathcal{L}_i^{NNCLR}$ (eq. 22), is defined as the function of the two feature vectors (z_i *and* z_i^+) which are generated by the model θ from the same input i in the given mini-batch using data augmentation ($z_i = \theta(aug(input_i))$, $z_i^+ = \theta(aug(input_i))$). Instead of calculating the dot product, NNCLR uses NN to select a similar point ($NN(z_i, Q)$) for one of the feature vectors. This selection is used in the loss function to pull the feature vectors for positive pairs together and push the feature vectors for negative pairs apart in the feature space.

$$\mathcal{L}_i^{NNCLR} = -\log \frac{\exp\left(\frac{NN(z_i,Q) \cdot z_i^+}{\tau}\right)}{\sum_{k=1}^{n} \exp\left(\frac{NN(z_i,Q) \cdot z_k^+}{\tau}\right)} \quad (22)$$

3.5.1.1. Preprocessing

To extract relevant features from waveforms using NNCLR, waveforms were extracted from the dataset and the average of the waveforms was calculated for each cluster. Samples with single channel were then extracted from the waveforms which were 105 datapoints long. The waveforms were normalized between [0, 1] to facilitate the learning of the waveforms themselves and avoid overfitting to the signal to noise ratio of the samples. Noise was introduced into the one-dimensional input through augmentations using random scaling (eq. 23) and jittering (eq. 24), which introduced multiplicative and additive noise with a normal distribution, respectively (eq. 25).

$$scaling(x) = x * random.normal(mean = 1, stddev = 0.1) \text{ (5)}$$

$$jittering(x) = \ x + random.normal(mean = 0, stddev = 0.03) \text{ (24)}$$

$$aug(x) = \ jittering(scaling(x)) \text{ (25)}$$

Positive pairs were formed using an instance *i* waveform from a cluster *k* and the average waveform of the same cluster (eq. 26, 27). (**Figure 3.**)

$$input_{i,1}^{k} = aug(x_i^k) \text{ (26)}$$

$$input_{2}^{k} = aug(\frac{1}{N}\sum_{i=1}^{N} x_i^k) \text{ (27)}$$

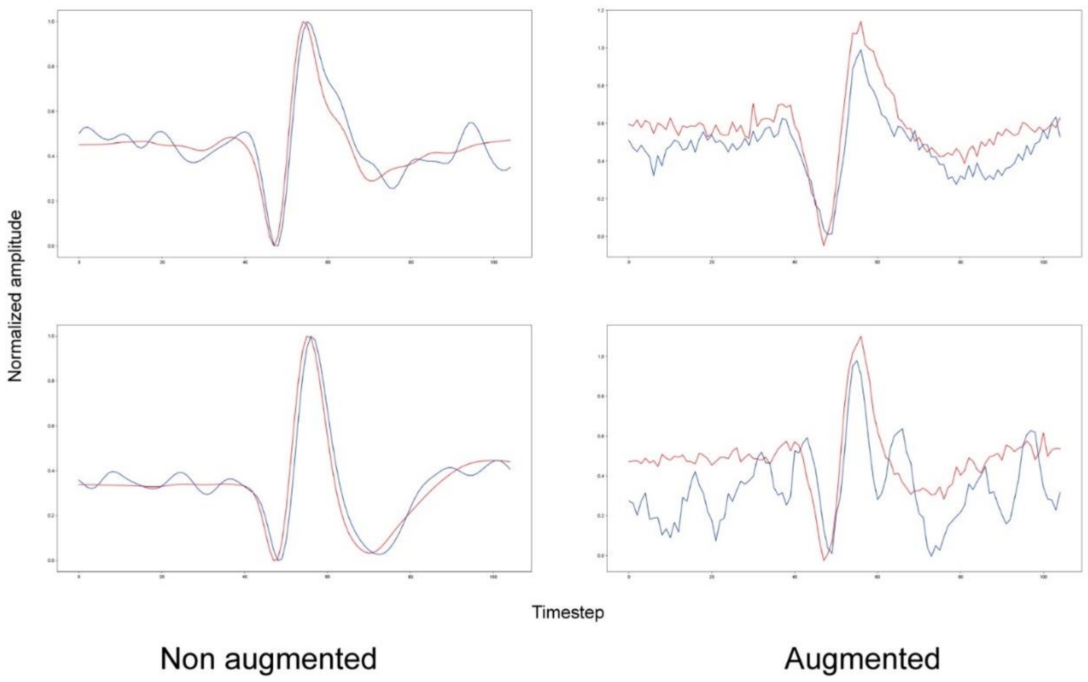


Figure 3. Pairs of waveforms as inputs for self-supervised model. The pairs of inputs for the self-supervised model consists of an instance from a cluster and the mean template waveform of that cluster. The averaged waveform is showed in red, while the instance is depicted in blue. The normalization is done before the augmentation. The left column contains examples of input pairs before augmentation, while the right column contains examples of input pairs after augmentation. (96)

3.5.1.2. NNCLR modell

The base model for NNCLR was constructed using Residual blocks, 1D convolution layers, Dense layers, and Batch normalization layers. (**Figure 4**) The input to the model is a 1D sample with shape (1x105), and the output is a vector with 32 dimensions (1x32). The residual blocks and convolution layers are used to extract features from the input sample, while the batch normalization layers help to stabilize the training process and improve the model's performance. Leaky ReLU activation function after the batch normalization layers introduces a small non-zero gradient for negative input values, allowing the model to learn more robust features. Together, these layers work to transform the input sample into a compact, low-dimensional feature vector that represents the underlying patterns in the data. The model depth is quite shallow, to enhance stability and avoid overfitting.

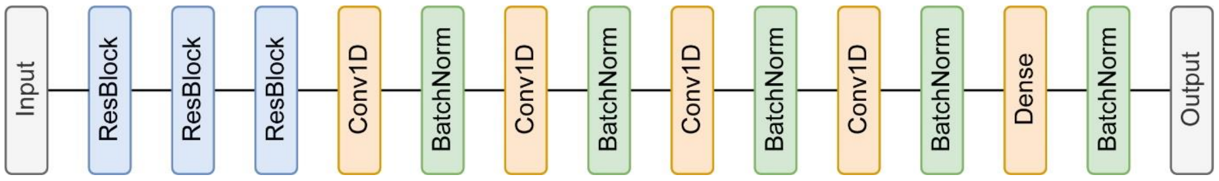


Figure 4. Backbone model for embedding generation for NNCLR. The backbone of the NNCLR is a simple model made of customized 1D Residual Blocks, 1D convolution layers and Batch normalization. After every BatchNormalization layer a LeakyRelu activation layer is present as well, which is not depicted on this figure to increase clarity. The input is a 1-dimensional sample (of shape 1x105), while the output is a vector of 32 dimensions (1x32). (96)

The NNCLR architecture includes a projection head during training, but this block is removed during inference. The remaining backbone model, depicted in **Figure 2.** as the encoder, is used for unsupervised inference phase. In this phase, the feature vector of each mean waveform is extracted and saved as a label for use in the supervised phase.

3.5.2. Single-shot detection

Single-shot detector (SSD) models are anchor-based object detection models that use a set of pre-defined boxes, called anchor boxes or anchors, to identify objects in an input image. These anchor boxes are placed at various locations and scales throughout the image and are designed to overlap heavily, allowing the model to identify objects with high precision. To improve the accuracy of the detection, the model also predicts the transformation parameters that control the positions and sizes of the anchor boxes. This enables the model to adjust the boxes to better match the objects in the image, even if they are shorter or have different aspect ratios than the anchor boxes. During inference, the model outputs a set of confidence scores for each anchor box, indicating the likelihood that the box contains an object. To select the best prediction for each object, non-max suppression (NMS) is applied to the overlapping boxes, taking into account the confidence scores. NMS removes lower confidence boxes that are highly overlapped with higher confidence boxes, leaving only the box with the highest confidence score for each object. This helps to reduce false positive detections and improve the overall accuracy of the model.

The custom anchor system was designed to address the aspect ratio of waveforms when representing them on a 2D plane. The anchor boxes used had a universal width of 5 channels, which was chosen for simplicity to match the width of the ground truth boxes. Using anchor boxes with different widths would not significantly impact performance because the model is able to predict the transformation parameters for the positions and sizes of the anchor boxes, allowing it to adjust the boxes to better fit the objects in the input image. The key factor in improving the precision of the detection is ensuring that the anchor boxes have good overlap with the objects in the image, as this allows the model to accurately predict the transformation parameters.

Ground truth boxes were formed based on ground truth 2D points and had a universal width of 5 channels (covering an electrode space of approximately ~38 μm^2), with the ground truth point placed in the middle. This customization of the labeling generation and anchor system allows the model to accurately predict the transformation parameters of the anchor boxes, enabling it to adjust the boxes to better match the objects in the input image. The ability to predict these transformation parameters is important for improving the precision of the detection. The model predicted 1024 anchor boxes for each sample.

3.5.3. MobileNetV2

The SSD model was chosen based on the limitations of the edgeTPU hardware and the need for a simple yet efficient architecture. MobileNetV2 (98) and EfficientDet (99) were considered as two promising options. EfficientDet had higher accuracies according to previous research, but its greater complexity made it less practical for this use case, as it had lower inference speed. Therefore, MobileNetV2 was selected because it is a lightweight architecture supported by the edgeTPU and is well-suited for systems with limited computational resources.

MobileNetV2 is a neural network architecture designed for efficient mobile and embedded vision applications. It was developed by Google researchers in 2018 as an upgrade to the original MobileNet architecture.

MobileNetV2 has a similar structure to MobileNet, but it introduces several new techniques to improve the efficiency and accuracy of the model. One of the key improvements is the use of residual connections, which allow for better gradient flow and enable deeper networks to be trained.

The MobileNetV2 architecture is composed of a series of building blocks called "inverted residuals". Each inverted residual block consists of three main components: a linear upsampling, an inverted residual, and a linear bottleneck.

The linear upsampling is another 1x1 convolutional layer that increases the number of input channels for the following layer. The next layer in the inverted residual block is a depthwise separable convolutions layer (100), which is computationally efficient and can capture spatial features effectively. The linear bottleneck is a 1x1 convolutional layer that reduces the number of output channels to match the desired output size.

The alpha parameter, also known as the width multiplier, controls the width of the convolutional blocks in MobileNetV2 and determines the trade-off between accuracy and performance. A smaller alpha value results in decreased accuracy but increased performance due to reduced computational requirements, while a larger alpha value leads to increased accuracy but decreased performance. For this study, the MobileNetV2 was customized with an alpha value of 0.2. The MobileNetV2 was also customized by doubling the output dimensions while maintaining the depth of the original model, which greatly improved the model's performance.

The output of the model consists of 3 different branches: box-, score- and feature prediction branches. The score and feature branches have a common, but separate branch from the box branch, branching at the end only. Both main branches use architectural elements from SSDLite introduced with MobileNetV2. The score prediction consists of 2 different classes, where the model predicts the probability that a box is containing a spike or not. The feature prediction branch output has the same dimensions as the feature vectors generated by the previously described self-supervised model. For the box and score prediction, FocalLoss (101) was applied, while for the feature prediction the cosine similarity loss was calculated during training. The provided feature vector for the boxes containing no-spikes was a vector of the same length filled with zero values. At inference, a postprocessing step is added to the model output, where an NMS is performed based on the box and score predictions and based on the results the predicted feature vectors are filtered as well.

3.5.4. EdgeTPU

To run the model in an embedded environment, EdgeTPU chip was chosen, which has as basis a Tensor Processing Unit (TPU) which is a specialized ASIC chip for deep-learning tasks. The TPU is built with a plethora of supported operations, however despite having a large basis of supported operations, it is still considered a limitation in the architecture designing process. To speed up the designing phase, known, supported architectures were chosen from. The EdgeTPU runs the operations in an efficient manner, requiring only 1 Watt per 2 Tera Operations Per Second (TOPS). From the first node that the EdgeTPU encounters as being an unsupported operation, the execution will be performed on the CPU side.

The evaluation of the model speed is done on two different TPU devices: a Coral Development Board Mini (CDBM), which consists of a MediaTek 8167s System on Chip (which integrates a Quad-core Arm Cortex-A35 CPU and an IMG PowerVR GE8300 GPU), 2GB LPDDR3 and a TPU module; a Coral USB Accelerator (CUA) consisting of a TPU module with a USB 3.0 connector. The CUA acts as a peripheral to a PC with a configuration of AMD Ryzen 7 2700X Eight-Core Processor 3.70 GHz CPU, 16GB DDR3 RAM and a Nvidia GeForce RTX 2080 SUPER GPU.

The inference speed is measured on both systems, measuring the net speed of the model inference on the TPU chip, and measuring the additional time needed by the NMS postprocessing. As the NMS is integrated into the model itself, the TFLite library will automatically take care of the data transfer between the CPU and TPU, because NMS runs only on CPU not being supported by the TPU. Thus, the execution can be divided into two major parts: spike detection and feature prediction executed on the TPU, and the postprocessing, like NMS and sorting will be effectuated on the CPU.

To run the proposed model on the TPU, it is necessary to quantize the model, which involves converting the model's parameters from 32-bit float values to 8-bit integers. This process can often result in a performance drop, but to minimize this drop, a quantization-aware training method was applied during the model's training. This technique involves introducing quantization noise during training, which allows the model to learn to be more robust to the effects of quantization. When the model is then quantized for deployment, it should experience a smaller performance drop compared to a model that was not trained with quantization-aware training.

4. Results

4.1. Results related to applying semi-supervised deep learning methods to spike sorting

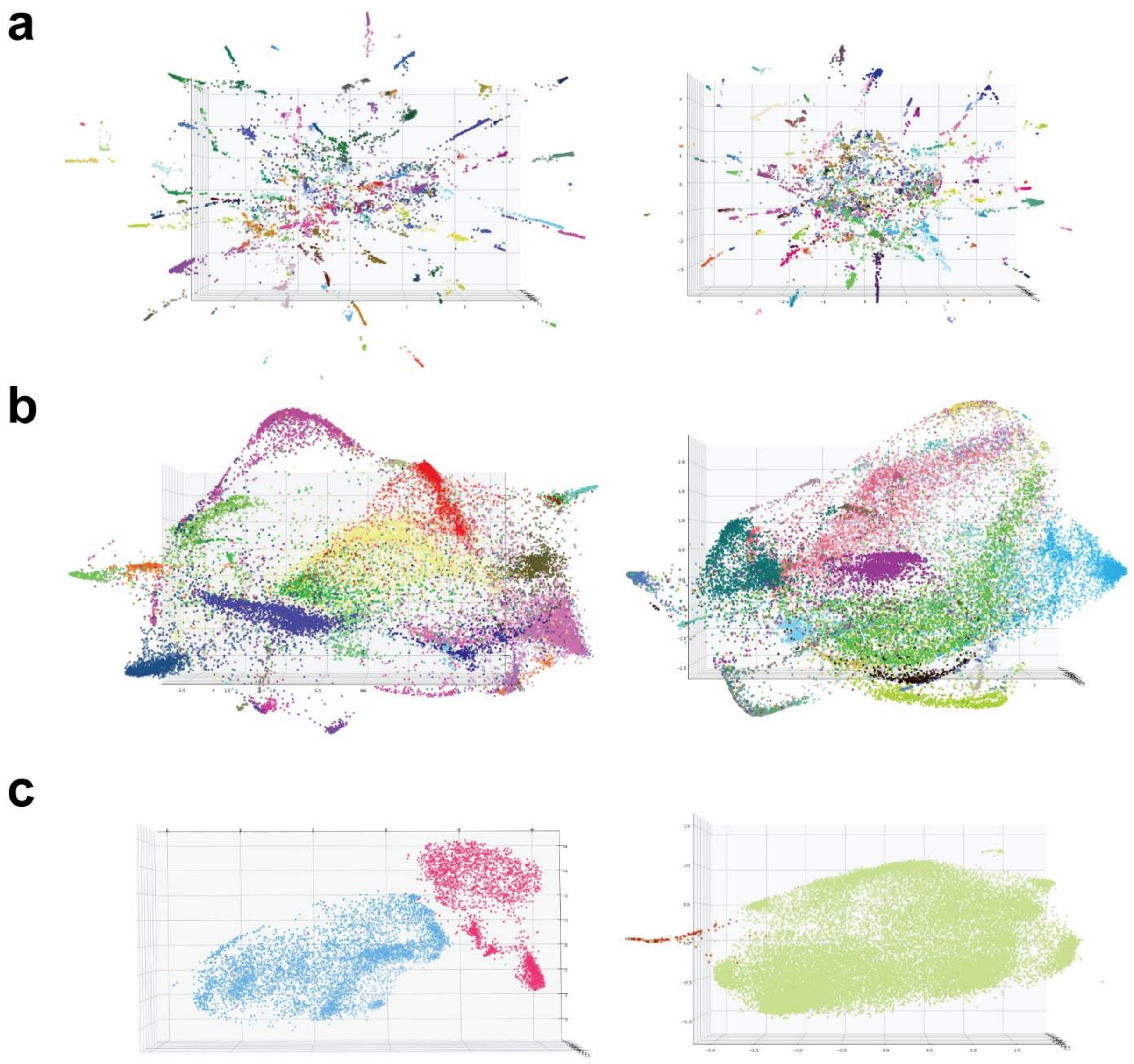


Figure 5. Latent spaces of train and test data for different datasets. The natural clustering capability of ELVISort can be observed on the latent spaces of the training (on the left) and test (on the right) data for the Hybrid Janelia (a), Fiath (b) and Kampff (c) datasets. Points with different colors represent different clusters: in cases (a) and (c) the available ground truth labels (74 and 2 clusters respectively), while in case (b) the manually curated labels provided by KiloSort were used for color coding (42 clusters). To visualize the 40-dimensional latent space, t-SNE was applied. The figures show high spatial separation although they were generated without tuning t-SNE training parameters excessively. ELVISort generates latent variables with high spatial separation between true clusters on training data (left column) and maintains the high separability on never seen data as well (test data - right column). (90)

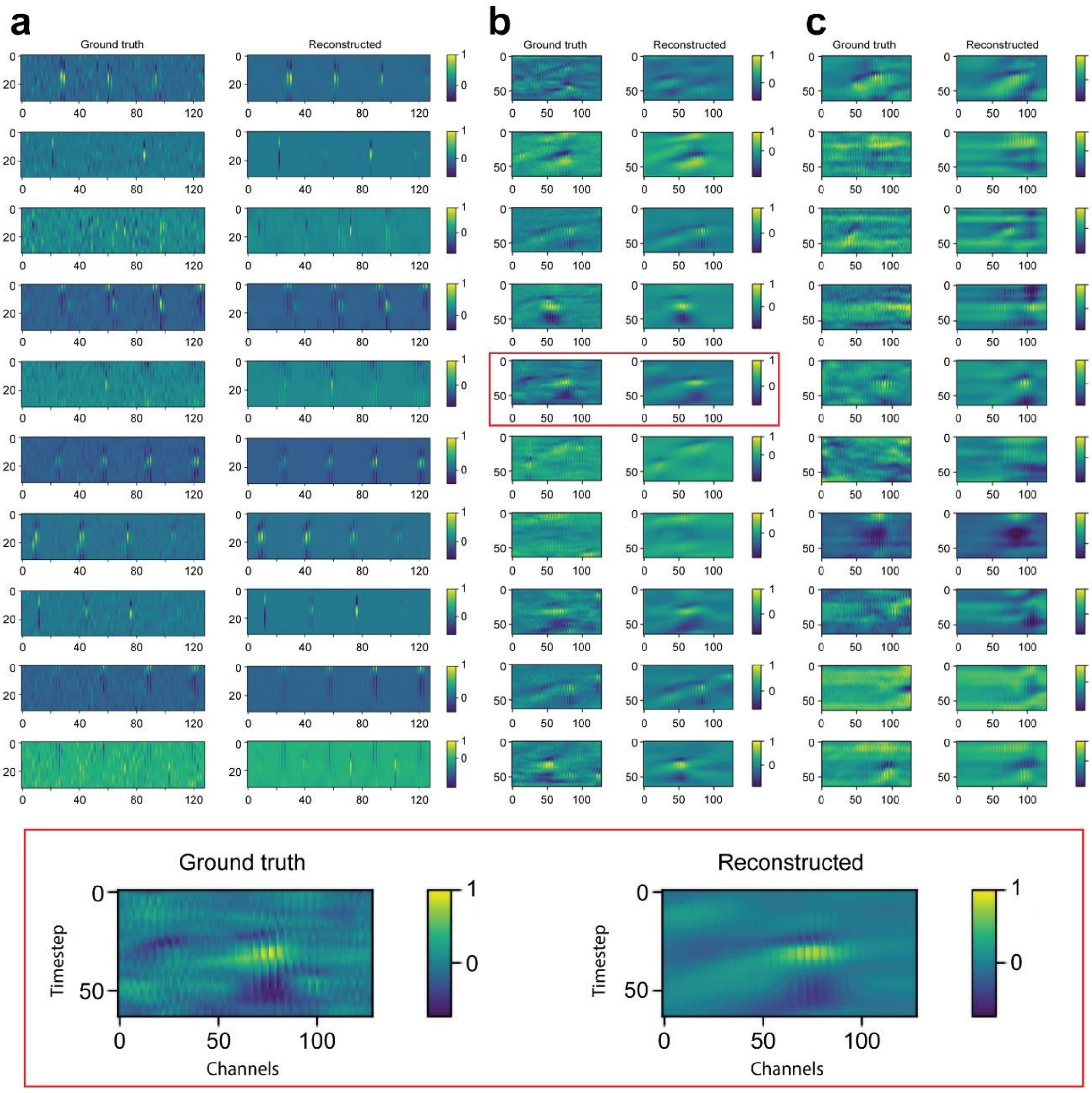


Figure 6. Reconstruction performance of ELVISort. Snippet pairs are from Hybrid Janelia (a), Fiath (b) and Kampff (c) test datasets. Each row corresponds to an individual snippet with the original instance on the left and the reconstructed on the right. The horizontal and vertical axes represent channels (there are 128 of them for all the datasets), and snippet timespan (64 for (b) and (c) and 32 for (a)), respectively. Despite the use of a high β value (β = 15), which according to studies makes reconstruction more challenging, vital information is preserved accurately while a fine noise reduction on the reconstructed snippets can be observed. All the snippets are normalized individually prior to processing and batch-wise using a batch normalization layer embedded in the model (due to this, color ranges of different pairs may differ). (90)

4.1.1. Results from the Kampff dataset

ELVISort was trained and tested on the Kampff dataset (2015_09_03_Cell.9.0), but similarly to state-of-the-art spike detectors and sorters, it managed to produce good results from one recording only. This is due to the low signal-to-noise ratio (SNR) of the majority of the recordings: spikes of the patched neurons have an average peak-to-peak (P2P) amplitude smaller than 30.8 μV for the majority of the recordings because ground truth neurons are mostly located quite distantly from the MEA.

It is worthwhile to note that the description of other spike sorting algorithms featuring the same dataset only report the one recording for which ELVISort gave good results. ELVISort identified the ground truth spikes from the Kampff recording *2015_09_03_Cell.9.0* with an F_1 score of 0.964 and an accuracy of 95% (for details, see **Table 1)**.

Table 1. Results of different algorithms from the SpikeForest website for recording 2015_09_03_Cell.9.0 of the Kampff dataset. Recall and precision parameters are available on the SpikeForest website, F_1 score for each algorithm was calculated by us. (90)

	Recall	**Precision**	**F_1 score**
HerdingSpikes2	1.00	0.95	0.97
IronClust	1.00	0.95	0.97
JRClust	0.97	0.99	0.98
KiloSort	1.00	0.96	0.98
KiloSort2	0.95	0.94	0.94
MountainSort4	0.55	0.93	0.69
SpykingCircus	1.00	0.95	0.97
Tridesclous	0.61	0.99	0.75
ELVISort	0.95	0.98	0.96

4.1.2. Results from the Fiath dataset

In order to test the detection performance on the Fiath dataset, the clusters were repartitioned (all the spikes were put in one group). ELVISort yielded results comparable to the ones given by the existing supervised learning methods. Having trained it separately on every recording, an average F_1 score of 85.55% was produced for the validation and 82.42% for the test set (see **Figure 7a**).

To test classification capability, the non-spike cluster was excluded and ELVISort was trained on each recording separately. It performed very well in general, providing an average F_1 score of 0.77 across the datasets (see **Figure 7b**). Only one of the 9 recordings yielded poor results (0.51 F_1 score).

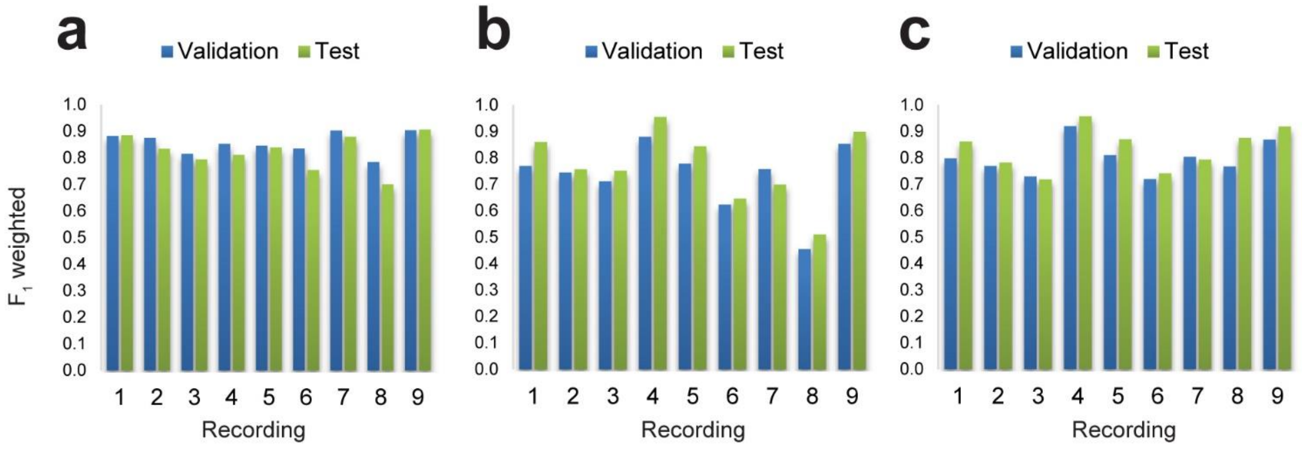
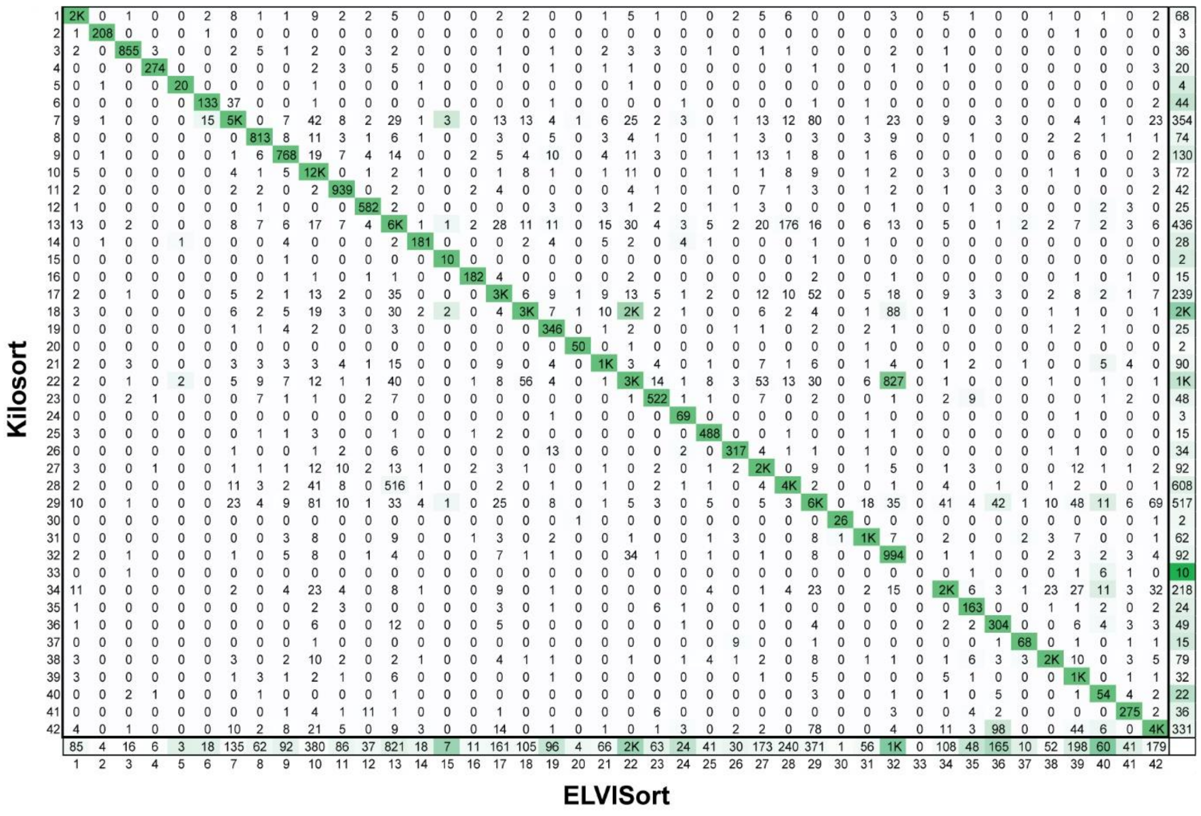


Figure 7. Results from the Fiath dataset. Performance (F_1 score) of ELVISort for the different Fiath recordings (1–9) aiming spike detection (a), classification (b) and both (c). For the supervised part, labels generated by KiloSort followed by manual correction were used. In the cases where classification is applied (subfigure b and c) a higher value of F1 weighted score can be observed for the test datasets relative to the validation datasets. This seeming contradiction is overcome when one understands that the test datasets were longer recordings compared to validation. This latter fact means that frequent clusters outweigh those lower frequency clusters for which the model has a lower F1 score (possibly also due to their relatively low representation in the training data). (90)

Finally, ELVISort was trained to perform the combined task (i.e. the detection and classification at the same time). It performed well, maintaining the peak performance of the previous "clustering only" phase (weighted F_1 score: 0.96) while producing better results for the recording that had yielded poor performance previously (weighted F_1 score: 0.87). The average score has also improved providing an average of 0.84 F_1 score over the different recordings (see **Figure 7c**; the classification performance of the system for a representative recording is illustrated in **Figure 8**).



Kilosort

1	2K	0	1	0	0	2	8	1	1	9	2	2	5	0	0	0	2	2	0	0	1	5	1	0	0	2	5	6	0	0	0	3	0	5	1	0	0	1	0	1	0	2	68
2	1	208	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	3
3	2	0	855	3	0	0	2	5	1	2	0	3	2	0	0	0	1	0	1	0	2	3	3	0	1	0	1	1	0	0	0	2	0	1	0	0	0	0	0	0	0	0	36
4	0	0	0	274	0	0	0	0	0	2	3	0	5	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	3	20
5	0	1	0	0	20	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
6	0	0	0	0	0	133	37	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	2	44
7	9	1	0	0	0	15	5K	0	7	42	8	2	29	1	3	0	13	13	4	1	6	25	2	3	0	1	13	12	80	0	1	23	0	9	0	3	0	0	4	1	0	23	354
8	0	0	0	0	0	0	0	813	8	11	3	1	6	1	0	0	3	0	5	0	3	4	1	0	1	1	3	0	3	0	3	9	0	0	1	0	0	2	2	1	1	1	74
9	0	1	0	0	0	0	1	6	768	19	7	4	14	0	0	2	5	4	10	0	4	11	3	0	1	1	13	1	8	0	1	6	0	0	0	0	0	0	6	0	0	2	130
10	5	0	0	0	0	0	4	1	5	12K	0	1	2	1	0	0	1	8	1	0	1	11	0	0	1	1	1	8	9	0	1	2	0	3	0	0	0	1	1	0	0	3	72
11	2	0	0	0	0	0	2	2	0	2	939	0	2	0	0	2	4	0	0	0	0	4	1	0	1	0	7	1	3	0	1	2	0	1	0	3	0	0	0	0	0	2	42
12	1	0	0	0	0	0	0	1	0	0	0	582	2	0	0	0	0	0	3	0	3	1	2	0	1	1	3	0	0	0	0	1	0	0	1	0	0	0	0	2	3	0	25
13	13	0	2	0	0	0	8	7	6	17	7	4	6K	1	1	2	28	11	11	0	15	30	4	3	5	2	20	176	16	0	6	13	0	5	0	1	2	2	7	2	3	6	436
14	0	1	0	0	1	0	0	0	4	0	0	0	2	181	0	0	0	2	4	0	5	2	0	4	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	28
15	0	0	0	0	0	0	0	0	1	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2
16	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	182	4	0	0	0	0	2	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0	1	0	1	0	15
17	2	0	1	0	0	0	5	2	1	13	2	0	35	0	0	0	3K	6	9	1	9	13	5	1	2	0	12	10	52	0	5	18	0	9	3	3	0	2	8	2	1	7	239
18	3	0	0	0	0	0	6	2	5	19	3	0	30	2	2	0	4	3K	7	1	10	2K	2	1	0	0	6	2	4	0	1	88	0	1	0	0	0	1	1	0	0	1	2K
19	0	0	0	0	0	0	1	1	4	2	0	0	3	0	0	0	0	0	346	0	1	2	0	0	0	1	1	0	2	0	2	1	0	0	0	0	0	1	2	1	0	0	25
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	2
21	2	0	3	0	0	0	3	3	3	3	4	1	15	0	0	0	9	0	4	0	1K	3	4	0	1	0	7	1	6	0	1	4	0	1	2	0	1	0	0	5	4	0	90
22	2	0	1	0	2	0	5	9	7	12	1	1	40	0	0	1	8	56	4	0	1	3K	14	1	8	3	53	13	30	0	6	827	0	1	0	0	0	1	0	1	0	1	1K
23	0	0	2	1	0	0	0	7	1	1	0	2	7	0	0	0	0	0	0	0	0	1	522	1	1	0	7	0	2	0	0	1	0	2	9	0	0	0	0	1	2	0	48
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	69	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	3
25	3	0	0	0	0	0	0	1	1	3	0	0	1	0	0	1	2	0	0	0	0	0	0	0	488	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	15
26	0	0	0	0	0	0	1	0	0	1	2	0	6	0	0	0	0	0	13	0	0	0	0	2	0	317	4	1	1	0	1	0	0	0	0	0	0	1	1	0	0	0	34
27	3	0	0	1	0	0	1	1	1	12	10	2	13	1	0	2	3	1	0	0	1	0	2	0	1	2	2K	0	9	0	1	5	0	1	3	0	0	0	12	1	1	2	92
28	2	0	0	0	0	0	11	3	2	41	8	0	516	1	0	0	2	0	1	0	1	0	2	1	1	0	4	4K	2	0	0	1	0	4	0	1	0	1	2	0	0	1	608
29	10	0	1	0	0	0	23	4	9	81	10	1	33	4	1	0	25	0	8	0	1	5	3	0	5	0	5	3	6K	0	18	35	0	41	4	42	1	10	48	11	6	69	517
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	26	0	0	0	0	0	0	0	0	0	0	0	1	2
31	0	0	0	0	0	0	0	0	3	8	0	0	9	0	0	1	3	0	2	0	0	1	0	0	1	3	0	0	8	1	1K	7	0	2	0	0	2	3	7	0	0	1	62
32	2	0	1	0	0	0	1	0	5	8	0	1	4	0	0	0	7	1	1	0	0	34	1	0	1	0	1	0	8	0	0	994	0	1	1	0	0	2	3	2	3	4	92
33	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	6	1	0	10
34	11	0	0	0	0	0	2	0	4	23	4	0	8	1	0	0	9	0	1	0	0	0	0	0	4	0	1	4	23	0	2	15	0	2K	6	3	1	23	27	11	3	32	218
35	1	0	0	0	0	0	0	0	0	2	3	0	0	0	0	0	3	0	1	0	0	0	6	1	0	0	1	0	0	0	0	0	0	0	163	0	0	1	1	2	0	2	24
36	1	0	0	0	0	0	0	0	0	6	0	0	12	0	0	0	5	0	0	0	0	0	0	1	0	0	0	0	4	0	0	0	0	2	2	304	0	0	6	4	3	3	49
37	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	1	0	0	0	0	0	0	1	68	0	1	0	1	1	15
38	3	0	0	0	0	0	3	0	2	10	2	0	2	1	0	0	4	1	1	0	0	1	0	1	4	1	2	0	8	0	1	1	0	1	6	3	3	2K	10	0	3	5	79
39	3	0	0	0	0	0	1	3	1	2	1	0	6	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	5	0	0	0	0	5	1	0	0	0	1K	0	1	1	32
40	0	0	2	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	1	0	1	0	5	0	0	1	54	4	2	22
41	0	0	0	0	0	0	0	0	1	4	1	11	1	0	0	0	1	0	0	0	0	0	6	0	0	0	0	0	0	0	0	3	0	0	4	2	0	0	0	0	275	2	36
42	4	0	1	0	0	0	10	2	8	21	5	0	9	3	0	0	14	0	1	0	1	0	1	3	0	2	2	0	78	0	0	4	0	11	3	98	0	0	44	6	0	4K	331
	85	4	16	6	3	18	135	62	92	380	86	37	821	18	7	11	161	105	96	4	66	2K	63	24	41	30	173	240	371	1	56	1K	0	108	48	165	10	52	198	60	41	179	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	

ELVISort

Figure 8. Comparison of the classification results of ELVISort and KiloSort. The comparison was made using a representative recording from the Fiath dataset. Non-spike clusters were excluded for getting a more straightforward comparison. Each row corresponds to a cluster determined by the Kilosort algorithm, while the columns present the clusters determined by ELVISort. The cluster identifiers are shown at the very left and bottom of the table. The numbers in the matrix correspond to the number of matching events. The last column represents the number of spikes assigned to each cluster by KiloSort but not by ELVISort. The numbers in the second last row correspond to the events assigned to each cluster by ELVISort, but not by KiloSort. Our algorithm performs similar to KiloSort, sorting spikes in similar clusters as the compared algorithm. Cluster nr. 33 was not identified by our algorithm, possibly due to the small number of occurrences (n=10, shown in the last column). (90)

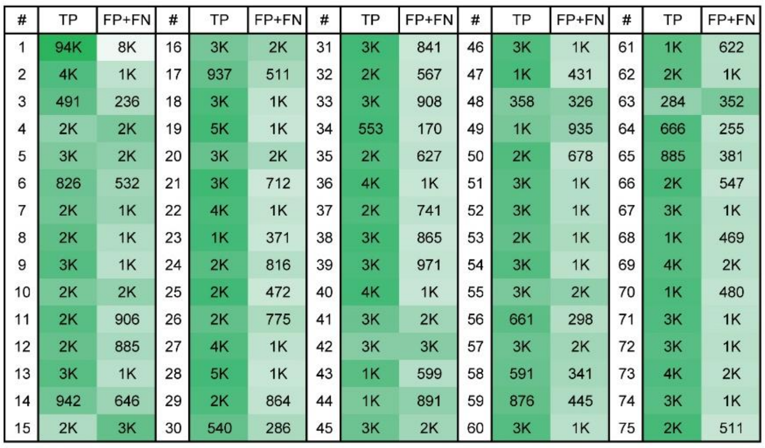
4.1.3. Results from the Hybrid Janelia dataset

The same combined (detector and sorter) model architecture that had been developed for the Fiath dataset was trained and tested on the Hybrid Janelia dataset. The performance was assessed with and without the non-spike cluster and no major differences were observed (see **Table 2**). The performance per cluster was also inspected, with respect to matched (true positives, TP) versus falsely matched (false positives + false negatives, FP+FN) snippets **(Figure 9a)**. The performance of ELVISort in relation to maximum peak values of the different clusters has also been evaluated (**Figure 9b)**.

Table 2. Results of ELVISort for the Hybrid Janelia dataset. The results were the same for the model performing sorting only (S) and the model performing detection and sorting simultaneously (D+S). Values are calculated according to the methods described on the SpikeForest website (i.e. F_1 score is determined without weighting cluster scores by cluster size). (90)

	F_1 score	**Accuracy**
HerdingSpikes2	0.62	0.44
IronClust	0.83	0.71
JRClust	0.63	0.46
KiloSort	0.81	0.66
KiloSort2	0.83	0.74
MountainSort4	0.67	0.49
SpykingCircus	0.83	0.70
Tridesclous	0.74	0.59
ELVISort (S)	0.81	0.67
ELVISort (D+S)	0.81	0.67

a



#	TP	FP+FN	#	TP	FP+FN	#	TP	FP+FN	#	TP	FP+FN	#	TP	FP+FN
1	94K	8K	16	3K	2K	31	3K	841	46	3K	1K	61	1K	622
2	4K	1K	17	937	511	32	2K	567	47	1K	431	62	2K	1K
3	491	236	18	3K	1K	33	3K	908	48	358	326	63	284	352
4	2K	2K	19	5K	1K	34	553	170	49	1K	935	64	666	255
5	3K	2K	20	3K	2K	35	2K	627	50	2K	678	65	885	381
6	826	532	21	3K	712	36	4K	1K	51	3K	1K	66	2K	547
7	2K	1K	22	4K	1K	37	2K	741	52	3K	1K	67	3K	1K
8	2K	1K	23	1K	371	38	3K	865	53	2K	1K	68	1K	469
9	3K	1K	24	2K	816	39	3K	971	54	3K	1K	69	4K	2K
10	2K	2K	25	2K	472	40	4K	1K	55	3K	2K	70	1K	480
11	2K	906	26	2K	775	41	3K	2K	56	661	298	71	3K	1K
12	2K	885	27	4K	1K	42	3K	3K	57	3K	2K	72	3K	1K
13	3K	1K	28	5K	1K	43	1K	599	58	591	341	73	4K	2K
14	942	646	29	2K	864	44	1K	891	59	876	445	74	3K	1K
15	2K	3K	30	540	286	45	3K	2K	60	3K	1K	75	2K	511

b

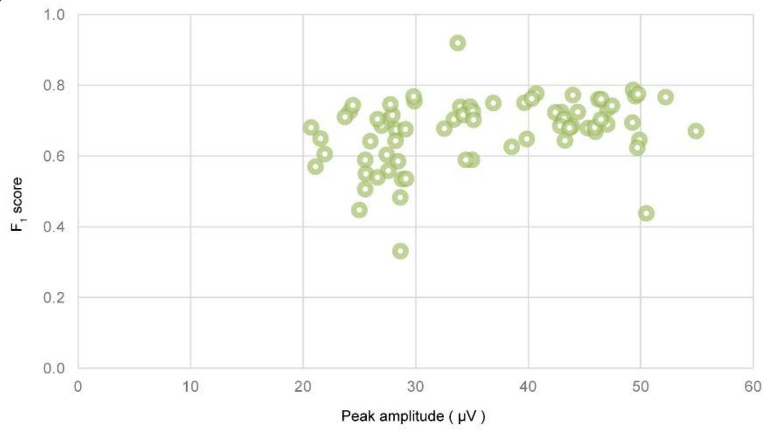


Figure 9. The performance of ELVISort on the Hybrid Janelia dataset.
(a) Detailed results per cluster. Cluster no. 1 represents the non-spike snippets. # – Cluster number, TP – True Positive, FP – False Positive, FN – False Negative.
(b) Accuracies for different clusters plotted against the peak amplitude.
Peak amplitude was computed averaging the maximum absolute spike amplitude from snippets where no other spike was present. (90)

4.1.4. Assessment of Sorting Efficiency

The training and testing for ELVISort was perfomed on a Windows PC with i9-7920X with 64 GB RAM and a GeForce RTX 2080 Ti GPU. We relied on the fast implementations of basic layers and arithmetics from the Numpy, Scipy and Tensorflow2 libraries.

ELVISort was tested with respect to computational speed as well. During the test, the decoder module was removed (which was only necessary for the training phase). In order to avoid falsely optimistic results, data caching was disabled. Since CPU to GPU and GPU to CPU data transfers are one of the most time-consuming stages of GPU data processing, caching (i.e. storing data in GPU memory for further use) can make algorithms run faster. In real-time processing scenarios, however, caching is not an option.

As test data, one of the recordings from the Hybrid Janelia (REC_64C_600S_12) was used with 64 channels. For other algorithms Docker containers were used, provided by the Spikeforest framework.

Efficiency tests were dispatched on a Windows PC with i9-7920X with 64 GB RAM and an NVIDIA GeForce RTX 2080 Ti GPU. The batch of data, that can be processed depends heavily on the GPU memory capacity, a larger memory enabling a larger batch size of instances (batch size x instances) to be sorted. ELVISort could evaluate the 600 s long recording in 38.17 s. This corresponds to an execution speed which is 15.71 times faster than required to perform real-time operation, surpassing other solutions (see **Table 3**) ELVISort was measured in a way where every single incoming instance was evaluated, this can be reduced significantly with simple yet efficient online filtering methods, thus further reducing the sorting time of the whole measurement. In spite real-time data comes at the sampling rate, for future real-time multiple simultaneous measurement evaluation, faster-than-sampling-rate algorithms are needed. A faster-than-sampling-rate algorithm also has the potential to be run on smaller devices, where the performance is more limited, thus efficiency becomes a key aspect.

To assess the computational efficiency of the model, the number of floating-point operations (FLOP) were determined. In order to process input data in real-time, a minimum computational performance of ~259 GFLOP/sec is needed.

We illustrate that ELVISort is the most efficient compared to other state-of-art methods in **Figure 10.**, having a shorter runtime, while maintaining similar sorting performance.

Table 3. Performance comparison of different spike sorting algorithms. The duration of evaluation of different algorithms based on the Hybrid Janelia recording: REC_64C_600S_12. Sorting time is represented in seconds, while xSpeed is described in eq.12. Despite ELVISort treats the 64 channel recording as a 128 channel recording by padding it, it has the least running duration among the compared. All but ELVISort were run from Docker containers obtained from the Spikeforest framework. To evaluate the speed compared to real-time, the duration of the recording (600 seconds) was divided by the sorting time of each algorithm. ELVISort was able to run 15.71 times faster than real-time on the Hybrid Janelia dataset, achieving the fastest speed among the compared algorithms. (90).

	Sorting time	**xSpeed**
Herdinspikes2	192.97	3.10
IronClust	84.38	7.11
JRClust	153.51	3.90
KiloSort2	86.41	6.94
MountainSort4	299.49	2.00
SpykingCircus	2521.78	0.23
ELVISort	38.17	15.71

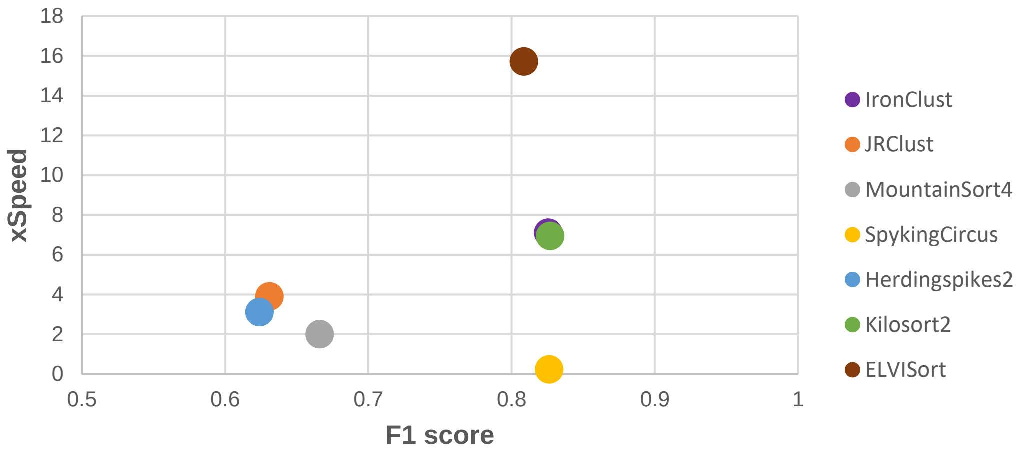


Figure 10. Sorting efficiency of different solutions regarding F1 score and xSpeed. The X-axis represents the F1 scores, which are listed in Table 2, while the Y-axis represents the xSpeed of each solution (Table 3). xSpeed is a factor without upper boundary, while the F1 score is a normalized score between 0 and 1. The upper-right corner represents the ideal spike sorting solution: having a fast algorithm while maintaining a perfect F1 score (=1). Based on the examined algorithms, ELVISort proves to be the most efficient: compromising minimal sorting performance, yet achieving an xSpeed higher than any other compared algorithm. (90)

4.2. Results concerning to developing a deep learning solution for edge devices

4.2.1. Results of the unsupervised part

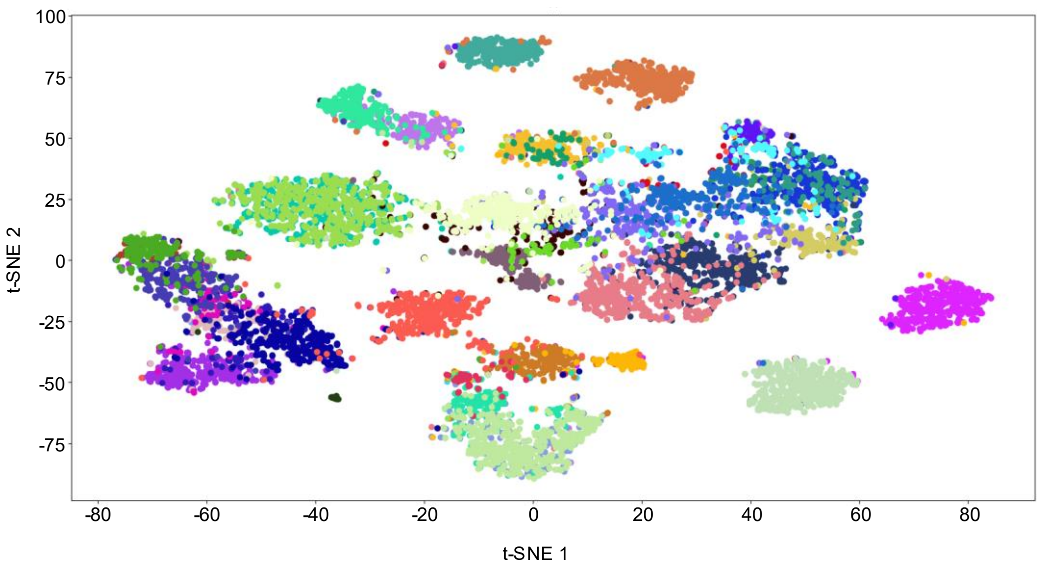


Figure 11. Embedding of the waveforms after NNCLR training. Using the NNCLR method, a highly separable latent space is obtained. To visualize the high-dimensional space, t-SNE is applied for dimensionality reduction. In the figure, we can observe that different clusters overlap each other if their waveforms are similar while being separated from those that differ. (96)

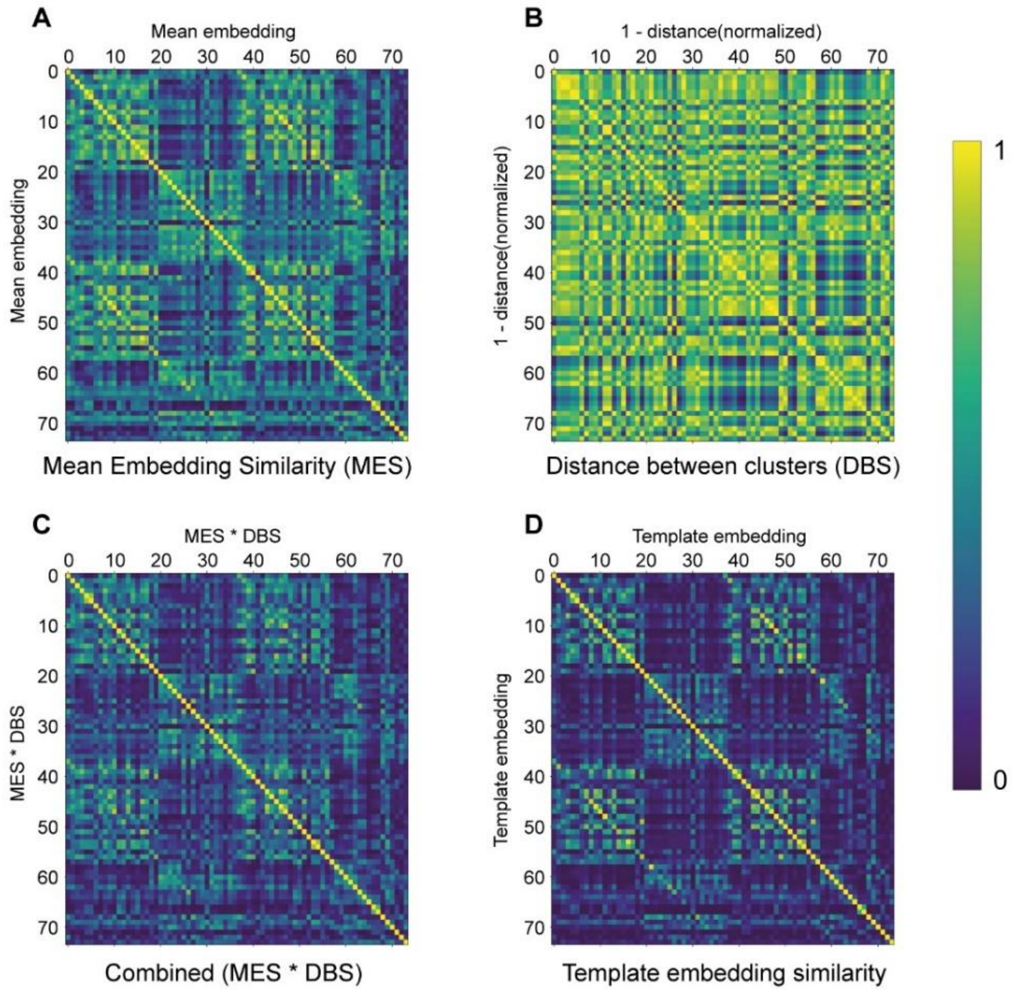


Figure 12. Similarity matrices of spike clusters. Subfigure (A) shows similarity matrix between the different cluster-means. (B) depicts the normalized 1-distance between different clusters in the channel-axis. (C) subfigure is the combination of (A) and (B) both normalized between 0,1, are combined thus a similarity matrix is built which considers both distance and mean embedding between clusters. In subfigure (D) the normalized similarity matrix between the cluster template embeddings can be seen. (96)

4.2.2. Results of the supervised part

Table 4. Detection accuracy comparison. Results of different algorithms on different, paired datasets. Two main datasets were used to compare our model to state-of-art spike sorting algorithms. (96)

Algorithm	Datasets						Avg acc
	BOYDEN			**YGER**			
	1103_1_1	*509_1_1*	*419_1_7*	*20170621*	*20170622_1*	*20170622_2*	
HerdingSpikes2 (76)	-	-	-	0,93	0,81	0,93	89*
IronClust (77)	0,84	0,76	0,74	0,84	0,66	0,94	79.67
JRClust (77)	0,92	0,53	0,88	-	-	0,94	0,82*
KiloSort (56)	0,96	0,05	0,75	0,97	0,97	0,94	0,77
KiloSort2	0,57	0,65	0,9	0,42	1	0,94	0,74
MountainSort4 (45)	0,96	0,76	0,71	1	0,97	0,92	0,88
SpykingCircus (58)	0,93	0,69	0,75	0,98	1	0,94	0,88
Tridesclous (102)	0,89	0	0,71	0,98	0,96	0,94	0,74
Average acc	0,86	0,49	0,77	0,87	0,91	0,93	80.50
Ours	0,96	0,73	0,88	1	1	1	0,93

Table 5. Detection performance on hybrid data. Results of average accuracies on the two hybrid data with multiple ground truth. The hybrid datasets are from the Hybrid Janelia dataset. (96)

Datasets	**Average detection accuracy**
HS_64_12	95,69%
HS_32_32	50,05%

Table 6. Clustering performance. Results of the two clustering methods used for the sorting of the features given by our model. (96)

Clustering algorithm	Datasets	
	HS_64_12	**HS_32_32**
ISOSplit5 (45)	74,60%	69,90%
Agglomerative clustering	89,85%	81,62%

Table 7. Spike sorting performance. Comparison of the different spike sorting algorithms on two of the hybrid datasets. (96)

Algorithm	Accuracy	
	HS_64_12	**HS_32_32**
HerdingSpikes2	0,79	0,47
IronClust	0,86	0,52
IRClust	0,89	0,53
KiloSort	0,94	0,51
KiloSort2	0,84	0,53
MountainSort4	0,82	0,48
SpykingCircus	0,91	0,54
Tridesclous	0,87	0,54
Mean	0,86	0,51
Ours	0,86	0,42

Table 8. Inference speed performance. Comparison of the inference speed of the different types of setups. Inference speed is composed of the model's computation time for making predictions and the time required for the non-maximum suppression (NMS) step. (96)

Setup	**NMS incl.**
PC CPU + GPU	3,95 msec
PC CPU + USB accel	5.32 msec
Coral DevBoard Mini	22,15 msec

5. Discussion

5.1. Applying semi-supervised deep learning methods to spike sorting

Despite multiple deep-learning based spike classification and/or sorting algorithms are present in the literature, to our knowledge, ELVISort is the first system that unifies spike detection and sorting with unsupervised deep learning architecture (i.e. β-VAE), and proved to be successful in performing the addressed tasks.

It was demonstrated that a method that applies both supervised and unsupervised learning paradigms performs well compared to commonly used state-of-the-art methods. It surpasses many conventional spike sorting systems that are featured on the SpikeForest platform.

The main objectives during training were the amelioration of the reconstruction capability of ELVISort and the disentanglement of its latent space; these two having a negative influence on each other meant that the effects of the parameter changes on these factors constantly had to be monitored. The reconstruction capability of ELVISort was kept very strong, despite the value of β was increased to 15 (the original and reconstructed snippets can be compared using **Figure 3**). The latter phenomenon was investigated previously and it was demonstrated, that β-VAE models with higher β value are significantly more robust to adversarial attacks and noisy data, compared to models with lower β value or Vanilla VAE (103).

During the training of an unsupervised clustering autoencoder the so-called "feature randomness" can occur, which is resulting from the usage of pseudo-labels during training. Pseudo-labels are based on hypothetical similarities and in the process of generating these pseudo-labels, a significant portion of true labels are substituted by random ones, producing latent spaces that underfit the semanticity of natural datasets (104). In order to alleviate the impact of this phenomenon, another branch was used to regularize the latent space (the supervised/detector branch), especially in the early training phases, where feature randomness can have a bigger impact on the performance of the trained model.

Another important phenomenon, the "feature drift" can deteriorate the performance of the final model as well. The feature drift occurs during model optimization when multiple loss functions are present and they strongly compete with each other, which can lead to an underperforming model (104). In order to weight each loss component the most optimal way, 9-fold cross-validation on the Fiath dataset was performed using several different reconstruction/detection loss rates.

The latent space was inspected during training. To compress information as much as possible, its disentanglement had to be facilitated. As previously noted, this was done by choosing a larger β value for the KL loss.

As it had been anticipated, data mostly concentrated around the origin, although snippets belonging to different clusters formed separate distributions in the parameter space (see **Figure 5** for details). Data were compressed without significant performance loss to a total of 40 latent variables, representing only 0.488% of the input. By being able to both reconstruct snippets previously unknown for ELVISort and segment the spikes with high accuracy using the latent space, it can be concluded that the construction of an end-to-end deep learning model capable of extracting spike-relevant information from $64 \times 128 = 8192$ dimension inputs was successful. While in our model the unsupervised part had a good generalization capability, the supervised classifier branch showed a varying performance. We assume this is partly due to the great diversity of SNRs across the recordings. To test whether the performance of the classifier branch can be improved, the classifier of a pre-trained instance of ELVISort was fine-tuned (having its β-VAE part frozen) for a small number of epochs ($n = 10$) on a dataset unknown to the model. A great improvement in the F1 score was observed, from an initial 50.03% to 83.92%. This leads us to assume that applying few-shot learning methods to the classifier (like Reptile or First Order Model Agnostic Meta Learning) would enable us to efficiently fine-tune classifiers in the future. In its current form, for unsupervised use on a dataset where the number of clusters is unknown, we propose the usage of the sigmoid activation function on the classifier branch to assign soft labels to snippets and use a traditional clustering algorithm on these soft labels. Because the unsupervised part of the model is robust we can use the same parameters for different recordings and only the supervised classification branch has to be fine-tuned.

Other real-time spike sorting systems offer integrated solutions like (47), however their performance regarding multi-channel recordings, is unknown. While not fully unsupervised as the previously mentioned solutions, EVLISort offers similar sorting performance to offline algorithms while reducing runtime significantly.

5.2. Developing a deep learning solution for edge devices

5.2.1. Self-supervised model

One of the limitations of some unsupervised learning algorithms is that they can be biased by the inclusion of labeled data or assumptions about the data. In the case of the self-supervised model used in this paper, the inclusion of cluster-wise-averaged waveforms as one of the pairs for the contrastive learning process could be seen as introducing a supervised bias. However, to mitigate this potential bias, augmentation was applied to the cluster average waveforms as well. Additionally, using different clusters with similar average waveforms can also help to alleviate the impact of this bias, as shown in **Figure 11**. In this figure, different clusters tend to overlap because the waveforms are very similar, which helps to avoid unnecessary cluster separation. Overall, these measures help to ensure that the self-supervised model is able to learn more generalized feature representations, rather than being overly influenced by any labeled data or assumptions about the data.

To demonstrate the effectiveness of our approach in creating a general embedding space, and the overlapping clusters can be indeed resolved by using channel information, we generated similarity matrices to analyze the distinguishability of various clusters (**Figure 12**). These matrices were generated by training our model on two different datasets simultaneously, which allowed us to observe the separability of the different waveforms within these datasets. In order to further examine the separability of the clusters, we also included channel-distance information between the clusters in our analysis. This was necessary because the hybrid recordings we used to train our model contained similar waveforms that were used to generate different clusters on different channels. The combination of both types of information resulted in a highly separable matrix, demonstrating the ability of our model to create a general embedding space that is able to effectively separate different waveforms.

5.2.2. Supervised model

Feature prediction and the detection of the individual spikes were assessed separately as well. To assess the performance of the detection of our model, we used paired recordings. This allowed us to compare the results to those of other existing solutions. The results, shown in **Table 4.**, demonstrate that our model performs very well in terms of spike detection and is able to generalize to new recordings with different electrode parameters and waveform types. In fact, the results show that our model performs better and more consistently than current state-of-the-art methods, even though it is specifically designed for use on embedded systems. These results suggest that our model is a promising solution for accurate and reliable spike sorting in a variety of settings.

A separate assessment was made for the two hybrid datasets, where detection, sorting and the combination of the two was considered (**Figure 13.**). The detection performance has a quite large gap between the two recordings (**Table 5.**): one of the probable explanations for this is that for the HS_64_12 recording cluster with the smallest SNR has an SNR value of 4.38, while for the HS_32_32 the minimum SNR is 0.34. The sorting of the found spikes show a more robust performance: while the Isosplit5 algorithm provides a faster sorting, the agglomerative clustering has a better performance on the generated feature space, however being the slower one. (**Table 6.**)

Table 7. compares the spike sorting performance of our system with other methods. We demonstrate that our compact model can reach the performance of some of the offline sorters and comparable with other the state-of-the-art methods.

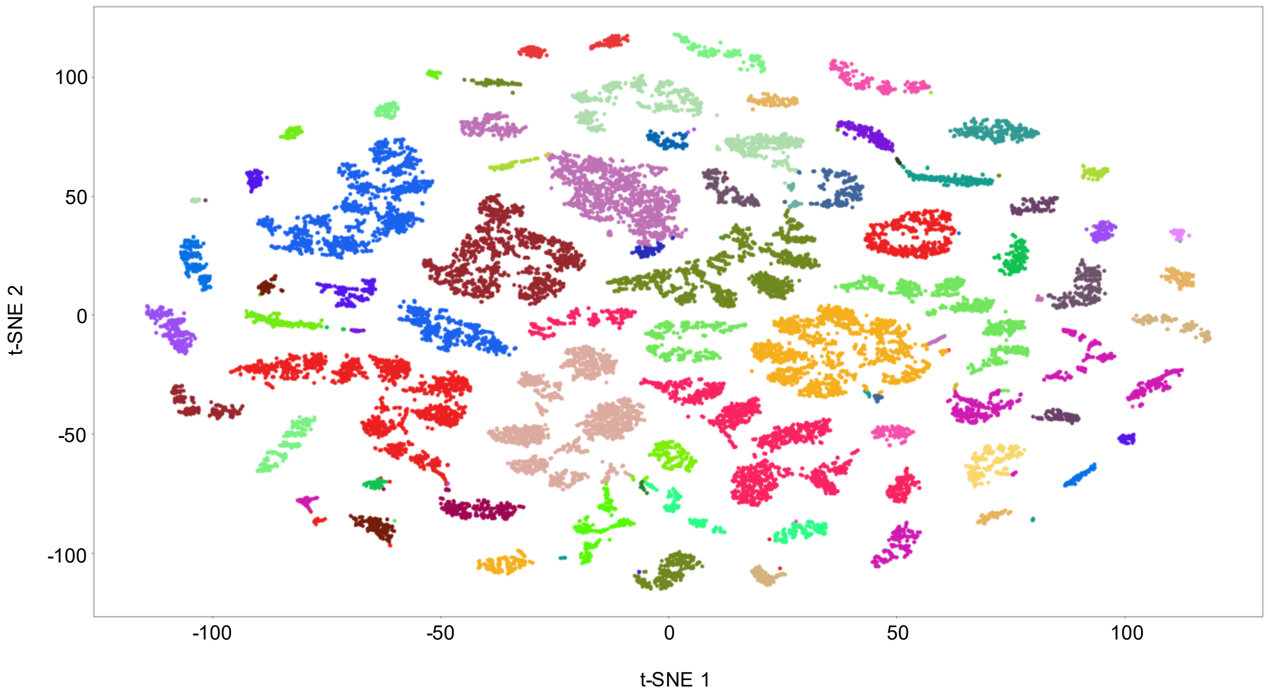


Figure 13. Embedded features after clustering. The features generated by our model. To the features NMS is applied, PCA and the results of the latter are then given as inputs for t-SNE to visualize the original 32-dimensional feature space in 2 dimensions. The clustering was performed after PCA, the different coloring representing different ground truth clusters. (96)

5.3. TPU inference

We tested the inference speed of our model on 128-channel samples in three different scenarios: a completely PC-based setup, where high performance CPU and GPU is available; a hybrid setup where high performance CPU is coupled with a TPU-based USB Accelerator, and a development-board-based setup, where a lower performance CPU is coupled with a TPU. The first setup was obviously the fastest, while the last one was the slowest one. The exported TFLite model is also heavily influenced by the speed of the CPU, because of the integrated NMS which runs on the CPU. The difference in latency time seen between the different setups in the *Postprocessing included* column in **Table 8.**, consists of the different processing speeds of the NMS node in the model. The CPU-TPU setup can achieve real-time inference speed, being able to process recordings with sampling rate up to 24 kHz. In contrast, the DevBoard-setup has a slower inference speed, because of the lower CPU performance: in an online matter it can handle data with 6kHz sampling rate.

5.3.1. Scalability

The presented system can also be scaled both data-wise as architecture wise. One of the benefits of such system is that it can be trained on more datasets at once but also the backbone architecture of both the unsupervised and supervised part can be improved according to the state-of-the-art methods.

The presented system is designed to be scalable in two key ways. First, it can be trained on multiple datasets simultaneously, allowing it to learn a more generalized representation of the waveform properties. This is achieved by using a self-supervised model to extract relevant features from the waveforms, and a supervised object-detection-based model to detect spikes and predict the feature-vectors in the embedding-space learned by the self-supervised model. This approach allows the system to be input-source agnostic, meaning it can be trained on data from different sources without requiring any prior knowledge of the recording conditions or electrode geometry.

Second, the system can also be scaled in terms of architecture by using state-of-the-art methods to improve both the self-supervised and supervised components of the model. This means that the system can be updated to reflect advances in machine learning techniques, without requiring any hardware changes. The so created embedding system then can be updated without any hardware changes, effectively converting the problem of spike sorting from a hardware problem to a software problem at the level of embedded systems. This makes the problem of spike sorting more flexible and adaptable, allowing for more efficient and accurate spike sorting on various platforms and devices.

Overall, the scalable nature of the system makes it a powerful tool for addressing the challenge of spike sorting in a variety of different contexts.

6. Conclusions

The application of semi-supervised deep learning methods to spike sorting has yielded promising results. The proposed deep learning model, ELVISort, leverages the β-VAE architecture to efficiently detect and sort spikes. ELVISort successfully reduces the input data size to less than 0.5% of its original dimensions, thereby achieving notable gains in memory and time efficiency during the clustering process. The model's performance was rigorously assessed using publicly available datasets, demonstrating commendable F1 scores on both the Hybrid Janelia and Kampff datasets. Furthermore, ELVISort showcased its viability for real-time applications, accomplishing spike processing within a substantially reduced timespan. These findings underscore the potential of ELVISort as a valuable tool in the development of memory and time-efficient brain-computer interfaces in the future.

In the realm of deep learning solutions for edge devices, our model distinguishes itself as the pioneering deep learning-based solution capable of accommodating recordings with a high number of channels, while being deployable on embedded systems, particularly on TPUs and at the same time, being able to exhibit a performance similar to existing state-of-art methods on unseen recordings.

7. Summary

We are faced with the challenge of accurately detecting and evaluating neural cell activities due to the increasing number of recording sites on silicon-based probes. To overcome this challenge, we have developed highly automated signal processing tools. Our focus is on deep learning-based spike sorting systems that aim to achieve high efficiency and performance comparable to offline spike sorting methods, while enabling real-time processing.

In our approach, we have developed ELVISort, a spike sorting model that combines the detection and clustering of different action potentials using deep learning. We have tested ELVISort on multiple independent datasets and achieved high average F1 scores. Notably, ELVISort exhibits real-time processing capabilities, enabling it to execute computations for a given sample length significantly faster than the actual duration.

Additionally, we have proposed another deep learning-based spike sorting system that combines unsupervised and supervised paradigms. Our system learns a general feature embedding space and effectively detects neural activity in raw data. An advantage of our system is its ability to be trained on multiple diverse datasets simultaneously, resulting in greater generalizability compared to previous models. We have demonstrated that our proposed system achieves accuracy on par with state-of-the-art offline spike sorting methods. Moreover, our system has the potential to run on edge TPUs, which opens up possibilities for integrating it into wearable electronic devices for advanced brain-computer interfaces.

By developing these deep learning-based spike sorting systems, we aim to address the challenge of accurately detecting and evaluating neural cell activities. Our systems offer high performance, efficiency, and real-time processing capabilities. They also have the potential to leverage AI-specific hardware, such as GPUs and TPUs, for parallel processing.

8. References

1. Buzsáki G, Anastassiou C a, Koch C. The origin of extracellular fields and currents--EEG, ECoG, LFP and spikes. Nat Rev Neurosci [Internet]. 2012;13(6):407–20. Available from: http://www.ncbi.nlm.nih.gov/pubmed/22595786

2. Hodgkin AL, Huxley AF. A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol [Internet]. 1952 Aug 28;117(4):500–44. Available from: https://onlinelibrary.wiley.com/doi/10.1113/jphysiol.1952.sp004764

3. Hodgkin AL, Huxley AF. The dual effect of membrane potential on sodium conductance in the giant axon of Loligo. J Physiol [Internet]. 1952 Apr 28;116(4):497–506. Available from: https://onlinelibrary.wiley.com/doi/10.1113/jphysiol.1952.sp004719

4. Hodgkin AL, Huxley AF. Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo. J Physiol [Internet]. 1952 Apr 28;116(4):449–72. Available from: https://onlinelibrary.wiley.com/doi/10.1113/jphysiol.1952.sp004717

5. O'Keefe J, Conway DH. Hippocampal place units in the freely moving rat: Why they fire where they fire. Exp Brain Res [Internet]. 1978 Apr;31(4). Available from: http://link.springer.com/10.1007/BF00239813

6. O'Keefe J, Dostrovsky J. The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. Brain Res [Internet]. 1971 Nov;34(1):171–5. Available from: https://linkinghub.elsevier.com/retrieve/pii/0006899371903581

7. Hafting T, Fyhn M, Molden S, Moser M-B, Moser EI. Microstructure of a spatial map in the entorhinal cortex. Nature [Internet]. 2005 Aug 19;436(7052):801–6. Available from: https://www.nature.com/articles/nature03721

8. Moser EI, Kropff E, Moser M-B. Place Cells, Grid Cells, and the Brain's Spatial Representation System. Annu Rev Neurosci [Internet]. 2008 Jul 1;31(1):69–89.

Available from: https://www.annualreviews.org/doi/10.1146/annurev.neuro.31.061307.090723

9. Hochberg LR, Serruya MD, Friehs GM, Mukand JA, Saleh M, Caplan AH, et al. Neuronal ensemble control of prosthetic devices by a human with tetraplegia. Nature [Internet]. 2006 Jul;442(7099):164–71. Available from: https://www.nature.com/articles/nature04970

10. Chaudhary U, Birbaumer N, Ramos-Murguialday A. Brain–computer interfaces for communication and rehabilitation. Nat Rev Neurol [Internet]. 2016 Sep 19;12(9):513–25. Available from: https://www.nature.com/articles/nrneurol.2016.113

11. Homer ML, Nurmikko A V., Donoghue JP, Hochberg LR. Sensors and Decoding for Intracortical Brain Computer Interfaces. Annu Rev Biomed Eng [Internet]. 2013 Jul 11;15(1):383–405. Available from: https://www.annualreviews.org/doi/10.1146/annurev-bioeng-071910-124640

12. Rey HG, Pedreira C, Quian Quiroga R. Past, present and future of spike sorting techniques. Brain Res Bull [Internet]. 2015;119:106–17. Available from: http://dx.doi.org/10.1016/j.brainresbull.2015.04.007

13. Lewicki MS. A review of methods for spike sorting: The detection and classification of neural action potentials. Netw Comput Neural Syst. 1998;9(4):R53–78.

14. Vargas-Irwin C, Donoghue JP. Automated spike sorting using density grid contour clustering and subtractive waveform decomposition. J Neurosci Methods. 2007;164(1):1–18.

15. Fee MS, Mitra PP, Kleinfeld D. Automatic sorting of multiple unit neuronal signals in the presence of anisotropic and non-Gaussian variability. J Neurosci Methods [Internet]. 1996;69(2):175–88. Available from: http://ovidsp.ovid.com/ovidweb.cgi?T=JS&CSC=Y&NEWS=N&PAGE=fulltext&D=med4&AN=8946321%5Cnhttp://sirius.library.unsw.edu.au:9003/sfx_local?sid=OVID:medline&id=pmid:8946321&id=doi:&issn=0165-0270&isbn=&volume=69&issue=2&spage=175&pages=175-

88&date=1996&title

16. Chah E, Hok V, Della-Chiesa A, Miller JJH, O'Mara SM, Reilly RB. Automated spike sorting algorithm based on Laplacian eigenmaps and k -means clustering. J Neural Eng [Internet]. 2011 Feb 1;8(1):016006. Available from: https://iopscience.iop.org/article/10.1088/1741-2560/8/1/016006

17. Kim KH, Kim SJ. Neural spike sorting under nearly 0-dB signal-to-noise ratio using nonlinear energy operator and artificial neural-network classifier. IEEE Trans Biomed Eng. 2000;47(10):1406–11.

18. Choi JH, Jung HK, Kim T. A new action potential detector using the MTEO and its effects on spike sorting systems at low signal-to-noise ratios. IEEE Trans Biomed Eng. 2006;53(4):738–46.

19. Tambaro M, Bisio M, Maschietto M, Leparulo A, Vassanelli S. FPGA Design Integration of a 32-Microelectrodes Low-Latency Spike Detector in a Commercial System for Intracortical Recordings. Digital. 2021;1(1):34–53.

20. Malik MH, Saeed M, Kamboh AM. Automatic threshold optimization in nonlinear energy operator based spike detection. In: 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) [Internet]. IEEE; 2016. p. 774–7. Available from: http://ieeexplore.ieee.org/document/7590816/

21. Quiroga RQ, Nadasdy Z, Ben-Shaul Y. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. Neural Comput. 2004;16(8):1661–87.

22. Semmaoui H, Drolet J, Lakhssassi A, Sawan M. Setting Adaptive Spike Detection Threshold for Smoothed TEO Based on Robust Statistics Theory. IEEE Trans Biomed Eng [Internet]. 2012 Feb;59(2):474–82. Available from: http://ieeexplore.ieee.org/document/6070974/

23. Kim S, McNames J. Automatic spike detection based on adaptive template matching for extracellular neural recordings. J Neurosci Methods [Internet]. 2007 Sep;165(2):165–74. Available from:

https://linkinghub.elsevier.com/retrieve/pii/S0165027007002634

24. Laboy-Juárez KJ, Ahn S, Feldman DE. A normalized template matching method for improving spike detection in extracellular voltage recordings. Sci Rep. 2019;9(1):1–12.

25. Barabino G, Baldazzi G, Sulas E, Carboni C, Raffo L, Pani D. Comparative evaluation of different wavelet thresholding methods for neural signal processing. Proc Annu Int Conf IEEE Eng Med Biol Soc EMBS. 2017;(1):1042–5.

26. Quotb A, Bornat Y, Renaud S. Wavelet Transform for Real-Time Detection of Action Potentials in Neural Signals. Front Neuroeng [Internet]. 2011;4(JULY):1–10. Available from: http://journal.frontiersin.org/article/10.3389/fneng.2011.00007/abstract

27. Issar D, Williamson RC, Khanna SB, Smith MA. A neural network for online spike classification that improves decoding accuracy. J Neurophysiol [Internet]. 2020;123(4):1472–85. Available from: https://www.biorxiv.org/content/10.1101/722934v1.abstract?%3Fcollection=

28. Rácz M, Liber C, Németh E, Fiáth R, Rokai J, Harmati I, et al. Spike detection and sorting with deep learning. J Neural Eng [Internet]. 2020 Jan 24;17(1):016038. Available from: https://iopscience.iop.org/article/10.1088/1741-2552/ab4896

29. Metcalfe BW, Clarke CT, Donaldson N, Taylor J. A New Method for Neural Spike Alignment: The Centroid Filter. IEEE Trans Neural Syst Rehabil Eng [Internet]. 2017 Nov;25(11):1988–97. Available from: https://ieeexplore.ieee.org/document/7951023/

30. Valencia D, Alimohammad A. Neural Spike Sorting Using Binarized Neural Networks. IEEE Trans Neural Syst Rehabil Eng [Internet]. 2021;29:206–14. Available from: https://ieeexplore.ieee.org/document/9288743/

31. Valencia D, Alimohammad A. An Efficient Hardware Architecture for Template Matching-Based Spike Sorting. IEEE Trans Biomed Circuits Syst [Internet]. 2019 Jun;13(3):481–92. Available from: https://ieeexplore.ieee.org/document/8675427/

32. Lee JH, Carlson D, Shokri H, Yao W, Goetz G, Hagen E, et al. YASS: Yet another

spike sorter. Adv Neural Inf Process Syst. 2017;2017-Decem(Nips):4003–13.

33. Salmasi M, Büttner U, Glasauer S. Fractal dimension analysis for spike detection in low SNR extracellular signals. J Neural Eng [Internet]. 2016 Jun 1;13(3):036004. Available from: http://dx.doi.org/10.1088/1741-2560/13/3/036004

34. Chen K, Jiang Y, Wu Z, Zheng N, Wang H, Hong H. HTsort: Enabling Fast and Accurate Spike Sorting on Multi-Electrode Arrays. Front Comput Neurosci [Internet]. 2021 Jun 21;15(June):1–13. Available from: https://www.frontiersin.org/articles/10.3389/fncom.2021.657151/full

35. Yang K, Wu H, Zeng Y. A Simple Deep Learning Method for Neuronal Spike Sorting. J Phys Conf Ser [Internet]. 2017 Oct;910(1):012062. Available from: https://iopscience.iop.org/article/10.1088/1742-6596/910/1/012062

36. Wood F, Fellows M, Donoghue JP, Black MJ. Automatic spike sorting for neural decoding. Annu Int Conf IEEE Eng Med Biol - Proc. 2004;26 VI:4009–12.

37. Dai M, Luo J. A Robust Method for Spike Sorting with Overlap Decomposition. J Comput. 2014;9(3):1195–8.

38. Biffi E, Ghezzi D, Pedrocchi A, Ferrigno G. Spike detection algorithm improvement, spike waveforms projections with PCA and hierarchical classification. IET Conf Publ. 2008;(540 CP).

39. Buccino AP, Hagen E, Einevoll GT, Hafliger PD, Cauwenberghs G. Independent Component Analysis for Fully Automated Multi-Electrode Array Spike Sorting. In: 2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) [Internet]. IEEE; 2018. p. 2627–30. Available from: https://ieeexplore.ieee.org/document/8512788/

40. Leibig C, Wachtler T, Zeck G. Unsupervised neural spike sorting for high-density microelectrode arrays with convolutive independent component analysis. J Neurosci Methods [Internet]. 2016 Sep;271:1–13. Available from: http://dx.doi.org/10.1016/j.jneumeth.2016.06.006

41. Jäckel D, Frey U, Fiscella M, Franke F, Hierlemann A. Applicability of

independent component analysis on high-density microelectrode array recordings. J Neurophysiol. 2012;108(1):334–48.

42. Yang Y, Mason AJ. Frequency Band Separability Feature Extraction Method with Weighted Haar Wavelet Implementation for Implantable Spike Sorting. IEEE Trans Neural Syst Rehabil Eng. 2017;25(6):530–8.

43. Oweiss KG, Anderson DJ. Spike sorting: A novel shift and amplitude invariant technique. Neurocomputing. 2002;44–46:1133–9.

44. Mahallati S, Bezdek JC, Popovic MR, Valiante TA. Cluster tendency assessment in neuronal spike data. Cymbalyuk G, editor. PLoS One [Internet]. 2019 Nov 12;14(11):e0224547. Available from: http://dx.doi.org/10.1371/journal.pone.0224547

45. Chung JE, Magland JF, Barnett AH, Tolosa VM, Tooker AC, Lee KY, et al. A Fully Automated Approach to Spike Sorting. Neuron [Internet]. 2017;95(6):1381-1394.e6. Available from: https://doi.org/10.1016/j.neuron.2017.08.030

46. Gibson S, Judy JW, Marković D. An FPGA-based platform for accelerated offline spike sorting. J Neurosci Methods. 2013;215(1):1–11.

47. Valencia D, Alimohammad A. A Real-Time Spike Sorting System Using Parallel OSort Clustering. IEEE Trans Biomed Circuits Syst [Internet]. 2019 Dec;13(6):1700–13. Available from: https://ieeexplore.ieee.org/document/8869918/

48. Schaffer L, Nagy Z, Kineses Z, Fiath R. FPGA-based neural probe positioning to improve spike sorting with OSort algorithm. In: 2017 IEEE International Symposium on Circuits and Systems (ISCAS) [Internet]. IEEE; 2017. p. 1–4. Available from: http://ieeexplore.ieee.org/document/8050608/

49. Schaffer L, Nagy Z, Kincses Z, Fiath R, Ulbert I. Spatial Information Based OSort for Real-Time Spike Sorting Using FPGA. IEEE Trans Biomed Eng [Internet]. 2021 Jan;68(1):99–108. Available from: https://ieeexplore.ieee.org/document/9103059/

50. Zhang B, Jiang Z, Wang Q, Seo J-S, Seok M. A neuromorphic neural spike

clustering processor for deep-brain sensing and stimulation systems. In: 2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED) [Internet]. IEEE; 2015. p. 91–7. Available from: http://ieeexplore.ieee.org/document/7273496/

51. Chen Y-L, Hwang W-J, Ke C-E. An Efficient VLSI Architecture for Multi-Channel Spike Sorting Using a Generalized Hebbian Algorithm. Sensors [Internet]. 2015 Aug 13;15(8):19830–51. Available from: http://www.mdpi.com/1424-8220/15/8/19830

52. Chen T-C, Wentai Liu, Chen L-G. 128-channel spike sorting processor with a parallel-folding structure in 90nm process. In: 2009 IEEE International Symposium on Circuits and Systems [Internet]. IEEE; 2009. p. 1253–6. Available from: http://ieeexplore.ieee.org/document/5117990/

53. Yu B, Mak T, Li X, Xia F, Yakovlev A, Sun Y, et al. Real-Time FPGA-Based Multichannel Spike Sorting Using Hebbian Eigenfilters. IEEE J Emerg Sel Top Circuits Syst [Internet]. 2011 Dec;1(4):502–15. Available from: http://ieeexplore.ieee.org/document/6132381/

54. Hwang WJ, Lee WH, Lin SJ, Lai SY. Efficient architecture for spike sorting in reconfigurable hardware. Sensors (Switzerland). 2013;13(11):14860–87.

55. Zhang T, Rahimi Azghadi M, Lammie C, Amirsoleimani A, Genov R. Spike sorting algorithms and their efficient hardware implementation: a comprehensive survey. J Neural Eng [Internet]. 2023 Apr 1;20(2):021001. Available from: https://iopscience.iop.org/article/10.1088/1741-2552/acc7cc

56. Pachitariu M, Steinmetz N, Kadir S, Carandini M, Kenneth D. H. Kilosort: realtime spike-sorting for extracellular electrophysiology with hundreds of channels. bioRxiv [Internet]. 2016;061481. Available from: http://biorxiv.org/lookup/doi/10.1101/061481

57. Pachitariu M, Sridhar S, Stringer C. Solving the spike sorting problem with Kilosort. bioRxiv [Internet]. 2023;2023.01.07.523036. Available from: https://www.biorxiv.org/content/10.1101/2023.01.07.523036v1%0Ahttps://www.biorxiv.org/content/10.1101/2023.01.07.523036v1.abstract%0Ahttps://www.bior

xiv.org/content/10.1101/2023.01.07.523036v1%0Ahttps://www.biorxiv.org/content/10.1101/2023.01.07.523036v1.

58. Yger P, Spampinato GLB, Esposito E, Lefebvre B, Deny S, Gardella C, et al. A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo. Elife. 2018;7:1–23.

59. Pouzat C, Mazor O, Laurent G. Using noise signature to optimize spike-sorting and to assess neuronal classification quality. J Neurosci Methods [Internet]. 2002 Dec;122(1):43–57. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0165027002002765

60. Bod RB, Rokai J, Meszéna D, Fiáth R, Ulbert I, Márton G. From End to End: Gaining, Sorting, and Employing High-Density Neural Single Unit Recordings. Front Neuroinform. 2022;16(June).

61. Li Z, Wang Y, Zhang N, Li X. An Accurate and Robust Method for Spike Sorting Based on Convolutional Neural Networks. Brain Sci [Internet]. 2020 Nov 11;10(11):835. Available from: https://www.mdpi.com/2076-3425/10/11/835

62. Mokri Y, Salazar RF, Goodell B, Baker J, Gray CM, Yen S-C. Sorting Overlapping Spike Waveforms from Electrode and Tetrode Recordings. Front Neuroinform [Internet]. 2017 Aug 17;11(August):1–15. Available from: http://journal.frontiersin.org/article/10.3389/fninf.2017.00053/full

63. Chiarion G, Mesin L. Resolution of spike overlapping by biogeography-based optimization. Electron. 2021;10(12).

64. Wouters J, Kloosterman F, Bertrand A. A data-driven spike sorting feature map for resolving spike overlap in the feature space. J Neural Eng. 2021;18(4).

65. Huang L, Ling BW-K, Cai R, Zeng Y, He J, Chen Y. WMsorting: Wavelet Packets' Decomposition and Mutual Information-Based Spike Sorting Method. IEEE Trans Nanobioscience [Internet]. 2019 Jul;18(3):283–95. Available from: https://ieeexplore.ieee.org/document/8681441/

66. Liu M, Feng J, Wang Y, Li Z. Classification of overlapping spikes using convolutional neural networks and long short term memory. Comput Biol Med

[Internet]. 2022;148(May):105888. Available from: https://doi.org/10.1016/j.compbiomed.2022.105888

67. Gong W, Senčar J, Bakkum DJ, Jäckel D, Obien MEJ, Radivojevic M, et al. Multiple Single-Unit Long-Term Tracking on Organotypic Hippocampal Slices Using High-Density Microelectrode Arrays. Front Neurosci [Internet]. 2016 Nov 22;10(NOV):1–16. Available from: http://journal.frontiersin.org/article/10.3389/fnins.2016.00537/full

68. Harris KD, Quiroga RQ, Freeman J, Smith SL. Improving data quality in neuronal population recordings. Nat Neurosci [Internet]. 2016 Sep 26;19(9):1165–74. Available from: https://www.nature.com/articles/nn.4365

69. Chaure FJ, Rey HG, Quian Quiroga R. A novel and fully automatic spike-sorting implementation with variable number of features. J Neurophysiol [Internet]. 2018 Oct 1;120(4):1859–71. Available from: https://www.physiology.org/doi/10.1152/jn.00339.2018

70. Buccino AP, Garcia S, Yger P. Spike sorting: new trends and challenges of the era of high-density probes. Prog Biomed Eng. 2022;4(2):1–20.

71. Greenberg DS, Kerr JND. Automated correction of fast motion artifacts for two-photon imaging of awake animals. J Neurosci Methods [Internet]. 2009 Jan;176(1):1–15. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0165027008004913

72. Steinmetz NA, Aydin C, Lebedeva A, Okun M, Pachitariu M, Bauza M, et al. Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings. Science (80-). 2021;372(6539).

73. Varol E, Boussard J, Dethe N, Winter O, Urai A, Brain Laboratory TI, et al. Decentralized Motion Inference and Registration of Neuropixel Data. In: ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) [Internet]. IEEE; 2021. p. 1085–9. Available from: https://ieeexplore.ieee.org/document/9414145/

74. Hurwitz CL, Xu K, Srivastava A, Buccino AP, Hennig MH. Scalable Spike Source

Localization in Extracellular Recordings using Amortized Variational Inference. arXiv [Internet]. 2019 May 29;(NeurIPS 2019). Available from: http://arxiv.org/abs/1905.12375

75. Boussard J, Varol E, Lee HD, Dethe N, Paninski L. Three-dimensional spike localization and improved motion correction for Neuropixels recordings. Adv Neural Inf Process Syst. 2021;27(NeurIPS):22095–105.

76. Hilgen G, Sorbaro M, Pirmoradian S, Muthmann JO, Kepiro IE, Ullo S, et al. Unsupervised Spike Sorting for Large-Scale, High-Density Multielectrode Arrays. Cell Rep [Internet]. 2017 Mar 7;18(10):2521–32. Available from: http://dx.doi.org/10.1016/j.celrep.2017.02.038

77. Jun JJ, Mitelut C, Lai C, Gratiy SL, Anastassiou CA, Harris TD. Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction. bioRxiv [Internet]. 2017;101030. Available from: https://www.biorxiv.org/content/10.1101/101030v1

78. Hurwitz C, Kudryashova N, Onken A, Hennig MH. Building population models for large-scale neural recordings: Opportunities and pitfalls. Curr Opin Neurobiol [Internet]. 2021;70:64–73. Available from: https://doi.org/10.1016/j.conb.2021.07.003

79. Magland J, Jun JJ, Lovero E, Morley AJ, Hurwitz CL, Buccino AP, et al. SpikeForest, reproducible web-facing ground-truth validation of automated neural spike sorters. Elife [Internet]. 2020 May 19;9:e55167. Available from: https://elifesciences.org/articles/55167

80. Pedreira C, Martinez J, Ison MJ, Quian Quiroga R. How many neurons can we see with current spike sorting algorithms? J Neurosci Methods [Internet]. 2012 Oct;211(1):58–65. Available from: https://linkinghub.elsevier.com/retrieve/pii/S0165027012002749

81. Henze D a, Borhegyi Z, Csicsvari J, Mamiya a, Harris KD, Buzsáki G. Intracellular features predicted by extracellular recordings in the hippocampus in vivo. J Neurophysiol [Internet]. 2000;84(1):390–400. Available from: http://www.ncbi.nlm.nih.gov/pubmed/10899213

82. Neto JP, Lopes G, Frazão J, Nogueira J, Lacerda P, Baião P, et al. Validating silicon polytrodes with paired juxtacellular recordings: Method and dataset. J Neurophysiol. 2016;116(2):892–903.

83. Fiáth R, Márton AL, Mátyás F, Pinke D, Márton G, Tóth K, et al. Slow insertion of silicon probes improves the quality of acute neuronal recordings. Sci Rep. 2019;9(1):1–17.

84. Allen BD, Moore-Kochlacs C, Bernstein JG, Kinney JP, Scholvin J, Seoane LF, et al. Automated in vivo patch-clamp evaluation of extracellular multielectrode array spike recording capability. J Neurophysiol. 2018;120(5):2182–200.

85. Rumelhart DE, Hinton GE, Williams RJ. Learning internal representations by error propagation. Parallel Distrib Process Explor Microstruct Cogn [Internet]. 1986;567. Available from: https://web.stanford.edu/class/psych209a/ReadingsByDate/02_06/PDPVolIChapter8.pdf

86. Hochreiter S, Schmidhuber J. Long Short-Term Memory. Neural Comput. 1997;9(8):1735–80.

87. Graves A, Schmidhuber J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks. 2005;18(5–6):602–10.

88. Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol Cybern. 1980;36(4):193–202.

89. Ciregan D, Meier U, Schmidhuber J. Multi-column deep neural networks for image classification. Proc IEEE Comput Soc Conf Comput Vis Pattern Recognit. 2012;3642–9.

90. Rokai J, Rácz M, Fiáth R, Ulbert I, Márton G. Elvisort: Encoding latent variables for instant sorting, an artificial intelligence-based end-to-end solution. J Neural Eng. 2021;18(4).

91. Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper

with convolutions. Proc IEEE Comput Soc Conf Comput Vis Pattern Recognit. 2015;07-12-June:1–9.

92. Lin H, Jia W, Sun Y, You Y. Spatial-temporal self-attention network for flow prediction. arXiv [Internet]. 2019; Available from: http://arxiv.org/abs/1912.07663

93. Razavi A, Van Den Oord A, Vinyals O. Generating diverse high-fidelity images with VQ-VAE-2. arXiv [Internet]. 2019; Available from: http://arxiv.org/abs/1906.00446

94. García-Alonso CR, Pérez-Naranjo LM, Fernández-Caballero JC. Multiobjective evolutionary algorithms to identify highly autocorrelated areas: The case of spatial distribution in financially compromised farms. Ann Oper Res. 2014;219(1):187–202.

95. Stephen Butterworth. On the Theory of Filter Amplifiers. Vol. 7, Experimental Wireless and the Wireless Engineer. 1930. p. 536–541.

96. Rokai J, Ulbert I, Márton G. Edge computing on TPU for brain implant signal analysis. Neural Networks [Internet]. 2023 Feb; Available from: https://linkinghub.elsevier.com/retrieve/pii/S0893608023001089

97. Dwibedi D, Aytar Y, Tompson J, Sermanet P, Zisserman A. With a Little Help from My Friends: Nearest-Neighbor Contrastive Learning of Visual Representations. Proc IEEE Int Conf Comput Vis. 2021;9568–77.

98. Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L-C. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition [Internet]. IEEE; 2018. p. 4510–20. Available from: https://ieeexplore.ieee.org/document/8578572/

99. Tan M, Pang R, Le Q V. EfficientDet: Scalable and Efficient Object Detection. Proc IEEE Comput Soc Conf Comput Vis Pattern Recognit [Internet]. 2019 Nov 20;10778–87. Available from: http://arxiv.org/abs/1911.09070

100. Chollet F. Xception: Deep Learning with Depthwise Separable Convolutions. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

[Internet]. IEEE; 2017. p. 1800–7. Available from: http://ieeexplore.ieee.org/document/8099678/

101. Lin TY, Goyal P, Girshick R, He K, Dollar P. Focal Loss for Dense Object Detection. IEEE Trans Pattern Anal Mach Intell. 2020;42(2):318–27.

102. Pouzat C, Garcia S. Tridesclous [Internet]. 2019 [cited 2020 Jul 7]. Available from: https://github.com/tridesclous/tridesclous

103. Willetts M, Camuto A, Rainforth T, Roberts S, Holmes C. Improving VAEs' Robustness to Adversarial Attack. 2019;1–36. Available from: http://arxiv.org/abs/1906.00230

104. Mrabah N, Bouguessa M, Ksantini R. Adversarial Deep Embedded Clustering: on a better trade-off between Feature Randomness and Feature Drift. IEEE Trans Knowl Data Eng [Internet]. 2020;1–1. Available from: https://ieeexplore.ieee.org/document/9099972/

9. Bibliography of the candidate`s publications

9.1. Papers closely related to the PhD dissertation

Rokai J, Ulbert I, Márton G. Edge computing on TPU for Brain Implant Signal Analysis. Neural Networks. 2023;162:212–24. doi:10.1016/j.neunet.2023.02.036

Bod RB, Rokai J, Meszéna D, Fiáth R, Ulbert I, Márton G. From end to end: Gaining, sorting, and employing high-density neural single unit recordings. Frontiers in Neuroinformatics. 2022;16. doi:10.3389/fninf.2022.851024

Rokai J, Rácz M, Fiáth R, Ulbert I, Márton G. ELVISort: Encoding latent variables for instant sorting, an artificial intelligence-based end-to-end solution. SSRN Electronic Journal. 2020; doi:10.2139/ssrn.3699796

Rácz M, Liber C, Németh E, Fiáth R, Rokai J, Harmati I, et al. Spike detection and sorting with Deep Learning. Journal of Neural Engineering. 2020;17(1):016038. doi:10.1088/1741-2552/ab4896

10. Acknowledgements

I express sincere gratitude for the unwavering support and seamless coordination provided by my supervisor, Dr. Márton Gergely. His guidance and mentorship have been instrumental in shaping the trajectory of my work. I extend my heartfelt appreciation to my colleagues for fostering an intellectually stimulating environment. The exchange of ideas and collaborative spirit within our professional community has undeniably enriched my perspective and contributed to the depth of my research. Furthermore, I would like to express my sincere appreciation to my friends, whose unwavering encouragement and companionship have provided a constant source of inspiration.

Last but certainly not least, I owe a debt of gratitude to my family. Their steadfast support, belief in my abilities, and unwavering encouragement have been the cornerstone of my accomplishments.