



Edge computing on TPU for brain implant signal analysis

János Rokai^{a,b,*}, István Ulbert^{a,c,1}, Gergely Márton^{a,c,1}

^a Institute of Cognitive Neuroscience and Psychology, Research Centre for Natural Sciences, Magyar tudósok körútja 2, building Q2, H-1117 Budapest, Hungary

^b János Szentágothai Doctoral School of Neurosciences, Semmelweis University, Üllői út 26, H-1085 Budapest, Hungary

^c Faculty of Information Technology and Bionics, Pázmány Péter Catholic University, Práter utca 50/a, H-1083 Budapest, Hungary

ARTICLE INFO

Article history:

Received 10 October 2022

Received in revised form 18 January 2023

Accepted 23 February 2023

Available online 28 February 2023

Keywords:

Spike sorting

Deep learning

Brain–computer interface

Feature extraction

Edge device

Electrophysiology

ABSTRACT

The ever-increasing number of recording sites of silicon-based probes imposes a great challenge for detecting and evaluating single-unit activities in an accurate and efficient manner. Currently separate solutions are available for high precision offline evaluation and separate solutions for embedded systems where computational resources are more limited.

We propose a deep learning-based spike sorting system, that utilizes both unsupervised and supervised paradigms to learn a general feature embedding space and detect neural activity in raw data as well as predict the feature vectors for sorting. The unsupervised component uses contrastive learning to extract features from individual waveforms, while the supervised component is based on the MobileNetV2 architecture. One of the key advantages of our system is that it can be trained on multiple, diverse datasets simultaneously, resulting in greater generalizability than previous deep learning-based models.

We demonstrate that the proposed model does not only reaches the accuracy of current state-of-art offline spike sorting methods but has the unique potential to run on edge Tensor Processing Units (TPUs), specialized chips designed for artificial intelligence and edge computing. We compare our model performance with state of art solutions on paired datasets as well as on hybrid recordings as well. The herein demonstrated system paves the way to the integration of deep learning-based spike sorting algorithms into wearable electronic devices, which will be a crucial element of high-end brain–computer interfaces.

© 2023 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Extracellular recordings in the central nervous system (CNS) provide information on neural activity patterns that can be valuable both for researchers in the field of neuroscience and for developers in the brain–computer interface industry. In order to analyze these neural patterns, the sources of single neuronal activities (single-units, spikes) need to be identified and clustered (spike sorting). To increase the precision of spike sorting and the number of recorded extracellular activity (spikes) from neurons, high-density neural microelectrode arrays (MEAs) are used, which are implanted into the CNS (Fiáth et al., 2019; Steinmetz et al., 2021). The number of recording sites on the MEAs is growing rapidly (Berényi et al., 2014; Fiáth et al., 2018), by which the recorded data is also growing, making an automated, robust, input-source agnostic spike sorter an increasingly valuable asset.

Neural activities are usually recorded with sampling rates between 20–30 kHz (Bod et al., 2022). In order to remove local field potential, low- and high-frequency noises to more reliably identify single-cell activities, a bandpass frequency filter is applied to the recordings between 0.3–3 kHz (or 0.5–5 kHz).

Manual curation is no longer a viable option for interpreting raw data due to the increased number of channels, by which the time of the manual curation increases as well. Subjective bias is also present in manual curation based on the experience of the curator.

To automate the detection and clustering of the spikes, spike sorting algorithms are developed to speed up the processing of high-channel-number recordings. Conventional spike sorting algorithms comprise three main processes: spike detection, feature extraction, and clustering of the features. For spike detection a plethora of approaches are present already in the literature: thresholding, non-linear energy operators (Kim & Kim, 2000), Teager energy operator thresholding (Choi, Jung, & Kim, 2006), or wavelet decomposition (Quiroga, Nadasdy, & Ben-Shaul, 2004). For feature extraction some studies use principal component analysis (PCA) (Biffi, Ghezzi, Pedrocchi, & Ferrigno, 2008; Vargasi-Irwin & Donoghue, 2007; Wood, Fellows, Donoghue, & Black,

* Corresponding author at: Institute of Cognitive Neuroscience and Psychology, Research Centre for Natural Sciences, Magyar tudósok körútja 2, building Q2, H-1117 Budapest, Hungary.

E-mail address: rokai.janos@ttk.hu (J. Rokai).

¹ These authors contribute equally.

2004), Independent component analysis (Hill, Moore-Kochlacs, Vasireddi, Sejnowski, & Frost, 2010; Jäckel, Frey, Fiscella, Franke, & Hierlemann, 2012; Mamlouk, Sharp, Menne, Hofmann, & Martinetz, 2005; Takahashi, Anzai, & Sakurai, 2003), optimal wavelet transforms (Yang & Mason, 2017) or Laplacian eigenmaps (Chah et al., 2011). To cluster the so-generated features several clustering algorithms were proposed in the past: k-means-clustering (Dai & Luo, 2014; Takahashi et al., 2003; Vargas-Irwin & Donoghue, 2007), superparamagnetic clustering (Quiroga et al., 2004), spectral clustering (Huang, Gan, & Ling, 2021), Hdbscan (Yger et al., 2018) or Hdsort (Diggelmann, Fiscella, Hierlemann, & Franke, 2018).

Modern spike sorting methods also offer automated pipelines like the different versions of KiloSort 1 (Pachitariu, Steinmetz, Kadir, Carandini, & Kenneth, 2016), 2 (Stringer et al., 2019), 2.5 (Steinmetz et al., 2021), SpyKING CIRCUS (Yger et al., 2018) and MountainSort4 (Chung et al., 2017). Deep learning methods were also applied to different phases of spike sorting (Buccino, Garcia, & Yger, 2022). For the detection phase, architectures like LSTMs (Rácz et al., 2020), CNNs (Saif-Ur-Rehman et al., 2019) were suggested, but deep learning models were proposed for feature extraction (Eom et al., 2021; Moghaddasi, Aliyari Shoorehdeli, Fatahi, & Haghpour, 2020; Wouters, Kloosterman, & Bertr, 2021), clustering (Rácz et al., 2020; Yang, Wu, & Zeng, 2017), and for full spike sorting functionalities as well. Different approaches have been implemented such as a 1D-CNN-based architecture (Li, Wang, Zhang, & Li, 2020), an autoencoder-based model (Rokai, Rácz, Fiáth, Ulbert, & Márton, 2021) or deep contractive autoencoder (Radmanesh et al., 2022). Despite deep learning methods thriving in other research areas, spike sorting seems to be more challenging for them. The main difficulty of developing deep learning-based spike sorting is that the ground truth for a waveform is given in the form of its cluster assignment. Thus, using these ground truth labels as targets for a machine learning model will fail to generalize to new recordings, where the true clusters can differ in number and their properties. The cluster identity integrates not only the properties of the waveforms of the cluster but also the channel-wise position of the waveforms as well. An all-self-supervised deep-learning model can be an obvious answer to this problem, however current self-supervised architectures have limited performances compared to their supervised counterparts.

Another important aspect of spike sorting methods is the tradeoff between performance in speed and performance in accuracy. Modern state-of-the-art algorithms are performing offline spike sorting on a high-performance PC. However, a plethora of potential applications would benefit from on-site spike sorting. In order to evaluate the raw data on-site, allowing for building closed-loop systems, several hardware implementations were suggested to create an online embedded spike sorting system. These implementations can extract specific features, like the first and second derivative extrema (Paraskevopoulou, Barsakcioglu, Saberi, Eftekhari, & Constandinou, 2013), or can deliver spike sorting in its full spectrum (Cai, Gan, Wang, Zhang, & Han, 2019; Hao, Chen, Richardson, Van der Spiegel, & Aflatouni, 2021; Hwang, Lee, Lin, & Lai, 2013; Schaffer, Nagy, Kincses, Fiáth, & Ulbert, 2021; Seong, Lee, & Jeon, 2021; Valencia & Alimohammad, 2019; Wang et al., 2019; Xu et al., 2019), using methods like template matching (Wang et al., 2019) or autoencoders (Seong et al., 2021). The on-chip spike sorting solutions however sacrifice precision for speed, while they are also limited in the number of channels, they can efficiently process data from.

Spike sorting to be available on the different types of devices will be a real need potentially in the near future. For example, a proposed solution for brain-machine interfaces, Neuralink (Musk, 2019), uses a system that is made out of an implanted sensor chip communicating wirelessly with a mobile phone. Because of the

sensitivity aspect of the data, it is advisable to process the raw data locally. The model described in the paper is built upon a well-known and widely used library namely Tensorflow and Tensorflow Lite and is built to run on devices with smaller processing powers as well, like Tensor Processing Units (TPUs). TPUs are custom hardware, specialized in running deep learning models very efficiently. To be able to perform complex deep learning inferences, efficient implementation of the different operators was needed so only a limited number of architecture types are supported.

We present our system, where, by taking the advantage of both supervised and unsupervised worlds, a self-supervised model is trained to extract the waveform features, getting rid of the positional information from the cluster identity. The supervised model will detect the spikes present in a sample and at the same time will predict the previously learned features of the spikes (Fig. 1).

Our goal was to develop a deep learning-based spike sorting algorithm and demonstrate its efficiency on edge TPUs. We aimed to develop a system that can be easily scaled, so users could easily choose the best performance/speed suited for their needs.

2. Methods

2.1. Data description

For acquiring data for the training of the model, a popular database was chosen, namely the Spike Forest platform (Magl et al., 2020), which enables the access to a plethora of well-known spike sorting datasets through a standardized interface. Because the model was intended to be optimized to 128 channel recordings, only datasets with high channel count were considered. Because of its hybrid nature, the Hybrid Janelia dataset was chosen as the main data source. In this dataset, the waveform templates were recorded at 30 kHz as part of Kampff's Ultra Dense Extracellular Survey and based on these recorded templates, new hybrid recordings were generated with Kilosort2. From the Hybrid Janelia dataset, two recordings were chosen for training, namely the REC_64C_600S_11 and REC_32C_600S_31 recordings. For testing the model's performance, recording REC_64C_600S_12 (HS_64_12) and REC_32C_600S_32 (HS_32_32) was used.

Both training recordings were used to train the self-supervised and supervised models, while the results of the final model were generated on the test recordings.

To be able to compare the performance of the detection itself to other algorithms, recordings of two paired datasets were used: recordings from Boyden dataset (Allen et al., 2018) and from Yger dataset (Yger et al., 2018). In both studies extracellular and intracellular voltages were recorded simultaneously, where the ground-truth of the extracellular recording was established based on the intracellular recording. Despite these recordings provide high precision ground truth data of real electrophysiological data, the ground truth only considers spikes from a single cell.

All the recordings were padded to 128 channels.

In this study, we refer to the electrical activity of a single neuron on a single channel over a short period of time as a waveform. The term “samples” refers to individual inputs of data for either the supervised or unsupervised model. For the unsupervised model, the input data is a waveform, so the terms “sample” and “waveform” are used interchangeably. In the case of the supervised model, samples are snippets of electrical activity that may contain multiple waveforms.

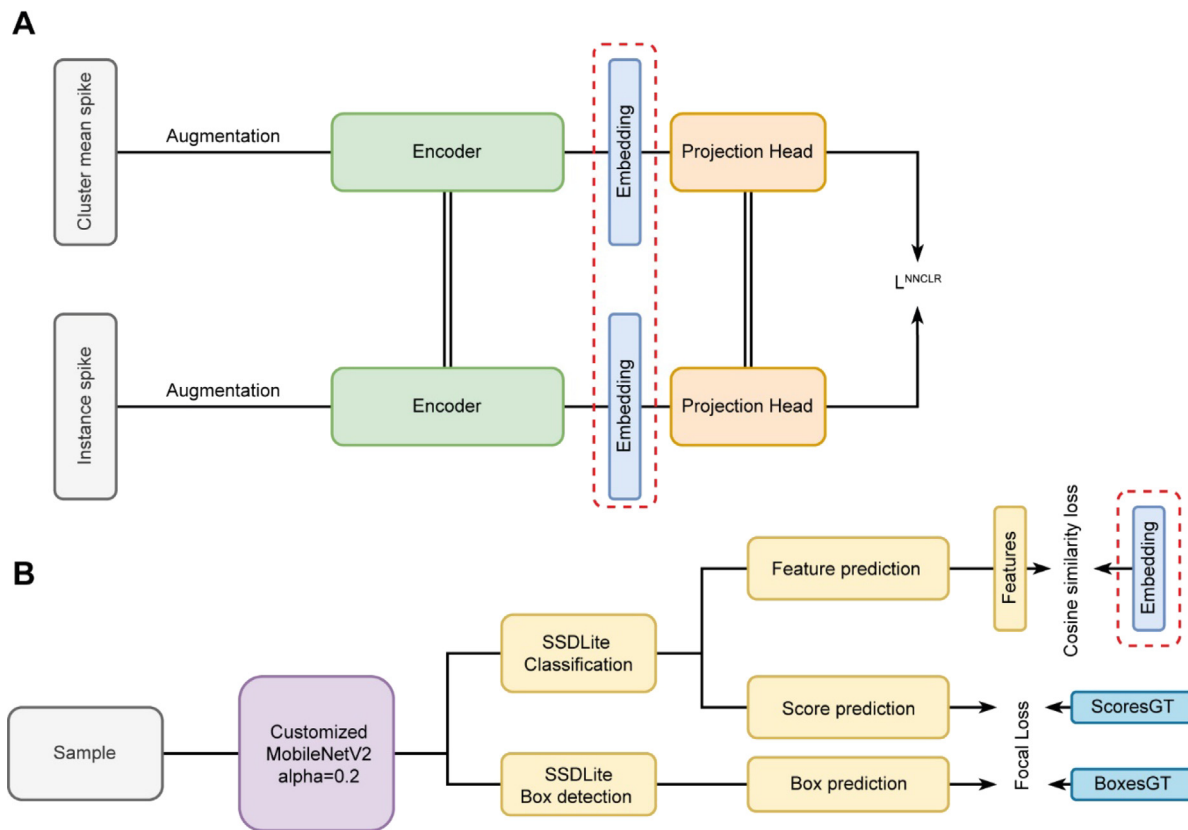


Fig. 1. Schematics of the training. In the top subfigure (A), the self-supervised model is depicted during training. Pairs of inputs are provided to the model and are processed by a shared encoder, which produces a feature vector. This feature vector is then passed through a projection head and the NNCLR 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.

2.2. Preprocessing

The raw data was minimally preprocessed in order to simulate the conditions of modern electrodes like Neuropixels (Jun, Steinmetz, et al., 2017), which have integrated bandpass filters. To further this goal, a bandpass filter between 300 and 3000 Hz was applied to the data. The main channel for different clusters of spikes was determined semi-automatically by averaging out windows containing spikes from a particular cluster and generating an automatic proposal for the main channel number, which was then manually reviewed by an expert.

2.3. Self-supervised learning for feature extraction

The self-supervised learning approach utilized in this study involves the use of unsupervised learning, specifically nearest-neighbor contrastive learning, to extract features from the given waveforms.

Unsupervised, or self-supervised learning has come a long way and the performance gap between unsupervised and supervised learning is closing. A handful of unsupervised methods offer promising results, however for the purpose of this study nearest-neighbor contrastive learning (NNCLR) was chosen.

During training, pairs of inputs are given to the model and fed through the same encoder, which produces a feature vector for each input. These feature vectors are then processed by a projection head, and the NNCLR loss is calculated based on the output of this projection head. The loss is then backpropagated through the model to update the weights of the encoder. At

inference time, only the encoder is used to extract features from new input samples. The generated embeddings are then used as labels in the supervised phase of training.

Nearest-neighbor contrastive learning (NNCLR) (Dwivedi, Aytar, Tompson, Sermanet, & Zisserman, 2021) 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 l_2 normalized). The other inputs in the mini-batch act as negative samples. The loss function, \mathcal{L}_i^{NNCLR} (Eq. (1)), 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

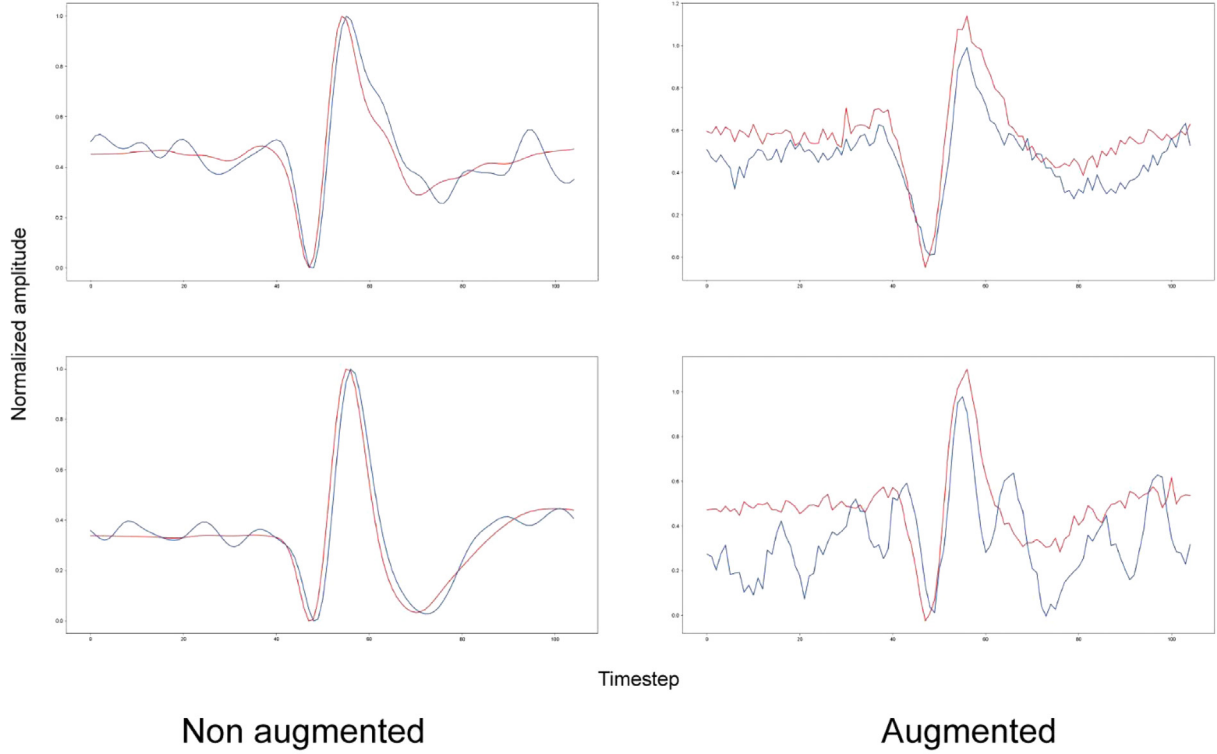


Fig. 2. Pairs of waveforms as inputs for self-supervised model. The pairs of inputs for the self-supervised model consists of an instance from a cluster and the mean template waveform of that cluster. The averaged waveform is showed in red, while the instance is depicted in blue. The normalization is 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.

augmentation (Eq. (2)) ($z_i = \theta(\text{aug}(\text{input}_i)), z_i^+ = \theta(\text{aug}(\text{input}_i))$). Instead of calculating the dot product, NNCLR uses NN to select a similar point ($\text{NN}(z_i, Q)$) 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.

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

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. (2)) and jittering (Eq. (3)), which introduced multiplicative and additive noise with a normal distribution, respectively (Eq. (4)).

$$\text{scaling}(x) = x * \text{random.normal}(\text{mean} = 1, \text{stddev} = 0.1) \quad (2)$$

$$\text{jittering}(x) = x + \text{random.normal}(\text{mean} = 0, \text{stddev} = 0.03) \quad (3)$$

$$\text{aug}(x) = \text{jittering}(\text{scaling}(x)) \quad (4)$$

Positive pairs were formed using an instance i waveform from a cluster k and the average waveform of the same cluster (Eq. (5), (6)) (Fig. 2).

$$\text{input}_{i,1}^k = \text{aug}(x_i^k) \quad (5)$$

$$\text{input}_2^k = \text{aug}(\frac{1}{N} \sum_{i=1}^N x_i^k) \quad (6)$$

The base model for NNCLR was constructed using Residual blocks, 1D convolution layers, Dense layers, and Batch normalization layers. (Fig. 3.) The input to the model is a 1D sample with shape (1×105) , and the output is a vector with 32 dimensions (1×32) . 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.

The NNCLR architecture includes a projection head during training, but this block is removed during inference. The remaining backbone model, depicted in Fig. 1 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.

2.4. Supervised model

The supervised model is the final model of the proposed system. As stated previously, the self-supervised model is used as an auxiliary model to generate embeddings for the spikes.

Samples were extracted from filtered data based on ground truth spike labeling. Each sample was 128 datapoints long (4.26 ms) and was formed by flattening the channels into a single axis and using the time axis as the other axis. Each sample was generated based on a specific spike, which was always placed in the middle of the sample, referred to as the “central spike”. This method of sample generation allows for natural augmentation of the data because multiple samples may contain the same spike,

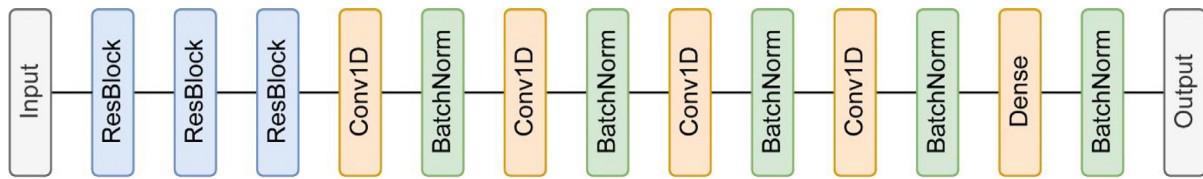


Fig. 3. Backbone model for embedding generation for NNCLR. The backbone of the NNCLR is a simple model made of customized 1D Residual Blocks, 1D convolution layers and Batch normalization. After every BatchNormalization layer a LeakyRelu activation layer is present as well, which is not depicted on this figure to increase clarity. The input is a 1-dimensional sample (of shape 1×105), while the output is a vector of 32 dimensions (1×32).

but with different central spikes and therefore different positions on the time axis. This not only allows for data augmentation, but also maximizes the positive to negative sample ratio by ensuring that every sample contains at least one spike. The samples were then augmented with 2D scaling and jittering (see Fig. 4) to improve the model's robustness against different noise levels and time-dependent noise.

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 $\sim 38 \mu\text{m}^2$), 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.

The SSD model was chosen based on the limitations of the edgeTPU hardware and the need for a simple yet efficient architecture. MobileNetV2 (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018) and EfficientDet (Tan, Pang, & Le, 2019) 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. The MobileNetV2 architecture uses Depthwise Separable Convolutions (Chollet, 2017) in inverted residual blocks, which give it a highly efficient and low-computational nature. 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, Focal-Loss (Lin, Goyal, Girshick, He, & Dollár, 2020) 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 a NMS is performed based on the box and score predictions and based on the results the predicted feature vectors are filtered as well.

2.5. Clustering

For the clustering of the post-processed data, two clustering algorithms were used. A more time-efficient choice was the Isosplit5 clustering algorithm, which was first introduced and used for PC-based real-time spike sorting (25). The other clustering algorithm chosen was the Accumulative clustering algorithm. The latter one is based on hierarchical clustering. While the former does not require any hyperparameters, the latter does. The hyperparameters were chosen by maximizing the clustering accuracy on a portion of the training dataset using hyperparameter search. The searched parameter for building the nearest neighbor graph was the number of neighbors ($n_neighbors = 5$) and for the agglomerative clustering was the distance threshold ($distance_threshold = 5.5$). The latter one determined the distance above which clusters were not merged.

To increase the speed and performance of these sorters, 10 principal components were extracted from the original features, with principal component analysis (PCA). The use of 10 principal

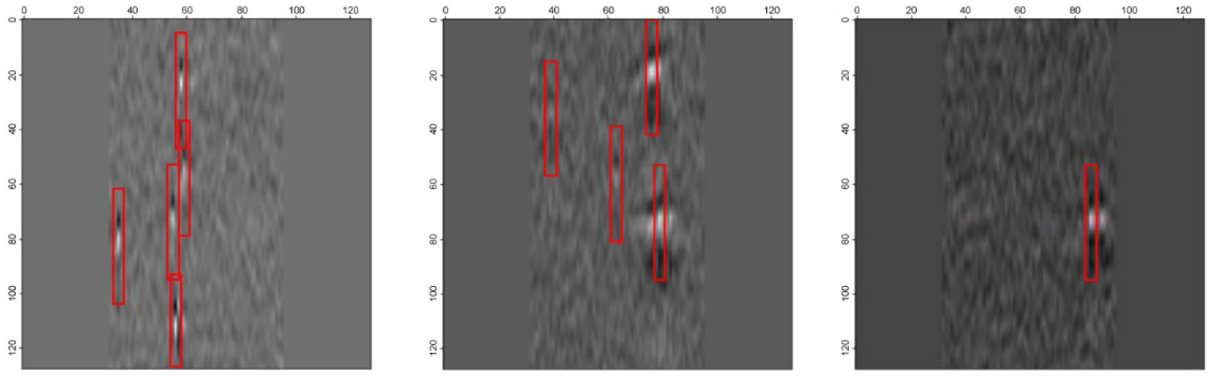


Fig. 4. Generated samples with boxes. Ground truth data is generated to be in 2D format, where X-axis is the channel axis, while Y-axis is the time axis. The boxes are formed so that they center the spikes both in length ($l = 32$) and channel-wise ($w = 5$). Because the 64-channel recordings are used, padding can be observed on both sides of the samples. The final input dimension thus being 128×128 .

components was chosen to provide flexibility for future recording sorting and to prevent overfitting (**Figure S1**). The so reduced features were then given as input to the clustering algorithms.

2.6. TPU device

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), 2 GB 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, 16 GB 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.

2.7. Evaluation

The evaluation of the self-supervised model was done with multiple similarity matrices. In order to build the Mean Embedding Similarity (MES) matrix, the self-supervised model's encoder (E) was applied to waveform samples ($x_{c,i}^w$, where c is the cluster identity index, i is the instance index) to generate feature vectors in the latent space. These feature vectors were then grouped by their ground truth cluster identity, and the Euclidean distance between them was calculated using Eq. (7). The resulting distance values were used to populate the similarity matrix s , which was then used to evaluate the separability of the different clusters in the latent space. To normalize the values in s , the minimum value ($\min(s)$) and the maximum value ($\max(s)$) of s are first determined. Then, for each value in s ($s_{i,j}$), the equation subtracts the minimum value from it, and divides the result by the difference between the maximum and minimum values. Finally, this result is subtracted from 1 to obtain the normalized value for MES (Eq. (8)).

$$s_{i,j} = \left\| \frac{1}{N} \sum_n E(x_{i,n}^w) - \frac{1}{M} \sum_m E(x_{j,m}^w) \right\|_2 \quad (7)$$

$$MES_{i,j} = 1 - \frac{s_{i,j} - \text{minimum}(s)}{\text{maximum}(s)} \quad (8)$$

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. (9), 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. (10). This resulted in the final distance matrix D , 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.

$$d_{i,j} = x_j^{ch} - x_i^{ch} \quad (9)$$

$$DBS_{i,j} = 1 - \frac{d_{i,j} - \text{minimum}(d)}{\text{channel_number}} \quad (10)$$

By normalizing the matrices S and D and combining them, a *Combined* matrix was created (Eq. (11)). This matrix combines both the positional difference between clusters on the channel axis and the similarity between their feature vectors, allowing for the separability of the clusters from each other to be investigated. The resulting *Combined* matrix reflects the overall similarity between pairs of clusters, taking into account both their positions

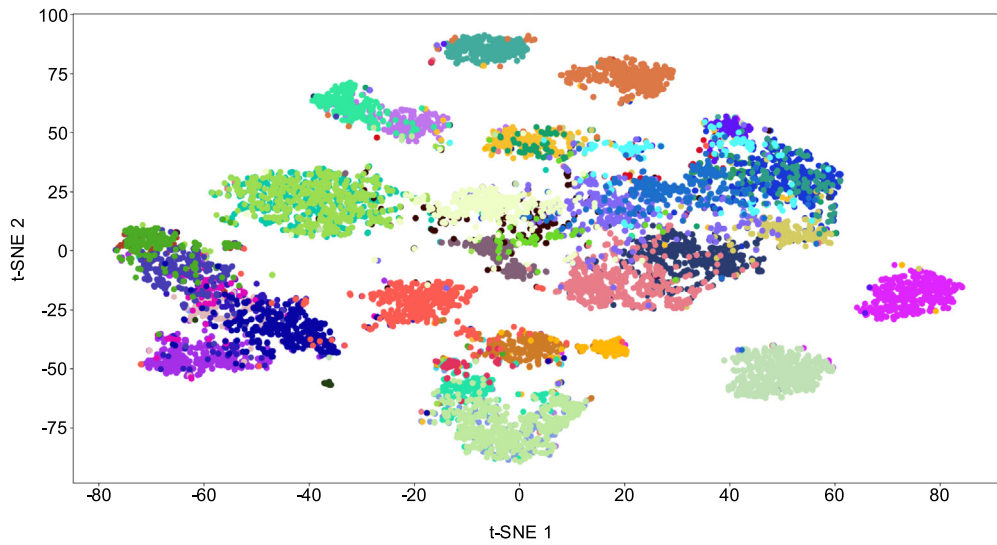


Fig. 5. Embedding of the waveforms after NNCLR training. Using the NNCLR method, a highly separable latent space is obtained. To visualize the high-dimensional space, t-SNE is applied for dimensionality reduction. In the figure, we can observe that different clusters overlap each other if their waveforms are similar while being separated from those that differ.

Table 1

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.

Algorithm	Datasets						Avg acc
	BOYDEN			YGER			
	1103_1_1	509_1_1	419_1_7	20170621	20170622_1	20170622_2	
HerdinSpikes2 Hilgen et al. (2017)	–	–	–	0,93	0,81	0,93	0,89*
IronClust Jun, Mitelut, et al. (2017)	0,84	0,76	0,74	0,84	0,66	0,94	0,8
JRClust Jun, Mitelut, et al. (2017)	0,92	0,53	0,88	–	–	0,94	0,82*
KiloSort Pachitariu et al. (2016)	0,96	0,05	0,75	0,97	0,97	0,94	0,77
KiloSort2 Stringer et al. (2019)	0,57	0,65	0,9	0,42	1	0,94	0,74
MountainSort4 Chung et al. (2017)	0,96	0,76	0,71	1	0,97	0,92	0,88
SpykingCircus Yger et al. (2018)	0,93	0,69	0,75	0,98	1	0,94	0,88
Tridesclous Pouzat and Garcia (2019)	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	0,80
Ours	0,96	0,73	0,88	1	1	1	0,93

and their feature vectors.

$$Combined = \frac{S \cdot D}{\text{maximum}(S \cdot D)} \quad (11)$$

The Template Embedding Similarity matrix (TES) (Eq. (12)) was built to investigate the distance between the embeddings of the cluster-wise averaged waveforms. In each point of the matrix, the distance is shown between the mean waveform embeddings and the embedding of the mean waveforms of a particular cluster.

$$TES_{i,j} = \left\| E \left(\frac{1}{N} \sum_n x_{i,n}^w \right) - E \left(\frac{1}{M} \sum_m x_{j,m}^w \right) \right\|_2 \quad (12)$$

The evaluation of the final supervised model was performed in two steps: the evaluation of the spike detection capability and the evaluation of the feature prediction quality. To evaluate the detection capability of the supervised model, a series of boxes were excluded from the computations: boxes that were on the edge of the samples in the time-axis were excluded because it is not optimal to detect and predict the feature set of halved spikes. The evaluation of the feature prediction was also made on the so filtered outputs.

Accuracy metric was used to evaluate the performance of the detection and sorting. The used accuracy metrics is identical to the one used on the Spikeforest database. This metric (Eq. (13)) takes into account the true positive (TP), false negative (FN) and

false positive (FP) elements as well:

$$Acc = \frac{TP}{TP + FN + FP} \quad (13)$$

The FP and FN boxes of the detector were excluded from sorting, but they were incorporated into the scores of the sorting metrics.

Scores were generated both for the stand-alone sorting step, as well as for the combination of the detection and score performance. The latter was used to evaluate the performance in comparison with other state-of-art offline methods.

3. Results

3.1. Results of the unsupervised part

See [Figs. 5 and 6](#).

3.2. Results of the supervised part

See [Tables 1–5](#).

4. Discussion

In this paper we introduced a new approach to deep-learning-based spike sorting, namely the training of a supervised network

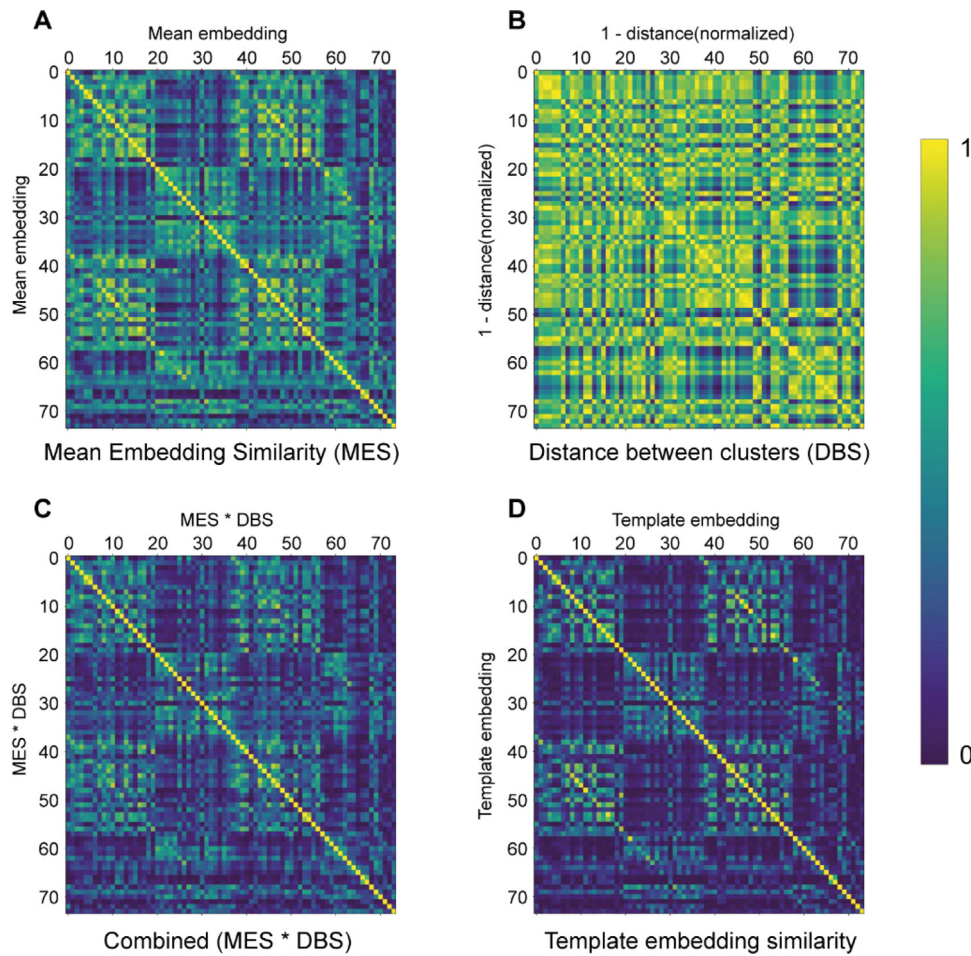


Fig. 6. Similarity matrices of spike clusters. Subfigure (A) shows similarity matrix between the different cluster-means. (B) depicts the normalized 1-distance between different clusters in the channel-axis. (C) subfigure is the combination of (A) and (B) both normalized between 0,1, are combined thus 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.

Table 2

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.

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

Table 3

Clustering performance. Results of the two clustering methods used for the sorting of the features given by our model.

Clustering algorithm	Datasets	
	HS_64_12	HS_32_32
ISOSplit5 (25)	74,60%	69,90%
Agglomerative clustering	89,85%	81,62%

Table 4

Spike sorting performance. Comparison of the different spike sorting algorithms on two of the hybrid datasets.

Algorithm	Accuracy	
	HS_64_12	HS_32_32
HerdSpikes2	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 5

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.

Setup	NMS incl.
PC CPU + GPU	3,95 ms
PC CPU + USB accel	5,32 ms
Coral DevBoard Mini	22,15 ms

in a two-staged process, using well-known architectures and concepts. The proposed model can be run on platforms such as TPUs, PCs, and mobile devices supported by the Tensorflow Lite library.

The self-supervised model is designed to learn the relevant features of the waveforms without being provided with any labels or specific cluster assignments. Instead, the model is trained to make sure that different views of the same waveform, which are produced through data augmentation, are mapped to similar

feature vectors in the embedding space. This is achieved through the use of the nearest-neighbor contrastive loss function (NNCLR), which measures the similarity between different views of the same waveform by selecting the nearest neighbor in the feature space for one of the feature vectors and calculating the dot product between the two. By training the model in this way, we can ensure that it learns a generalizable representation of the waveform, which is not specific to any particular dataset or recording. This improves the model's ability to generalize to new, unseen data and makes it more robust to variations in the data distribution.

The supervised model is then trained to detect spikes and predict the corresponding feature vectors in the embedding space. This is done through the use of an anchor-based single-shot detector (SSD), which uses a map of anchor boxes to identify possible detections. These boxes are designed to overlap heavily, enabling high-precision detection. To further increase precision, the model predicts the transformation parameters for the positions and sizes of the boxes, which allows it to detect shorter spike activities. Non-max suppression (NMS) is used to select the best box prediction from the overlapping boxes with different confidence scores.

The supervised model is trained in a similar way to other object detection models, by providing it with labeled data that specifies which boxes correspond to spikes and which do not. However, instead of predicting the cluster assignments for each box, the model is trained to predict the feature vectors for the boxes that contain spikes. This approach has the advantage of being more generalizable, as the model is not tied to any specific cluster assignments or electrode geometry. By training on multiple recordings at once, we can ensure that the model learns a more generalizable representation of the waveform features. This is important for the same reason that it was important for the self-supervised model: generalizability. By training on multiple recordings at once, we can ensure that the model is able to handle new recordings with different cluster assignments and/or electrode geometry. (Fig. 1.)

Importantly the model operates on pre-filtered input data, assuming a hardware-based bandpass frequency filter is already implemented when it comes to embedding systems (like Neuropixel probes (Jun, Steinmetz, et al., 2017)).

Because the input of the self-supervised model is the one-dimensional waveform itself, thus not relying on any geometrical structure of the electrode itself, we can state that the self-supervised model is input-source agnostic, thus having the advantage of training a single model with data from different sources, producing a more generalized embedding space for waveforms of all forms and shapes.

This appears to be a major benefit over other deep-learning-based spike sorting algorithms, where the system is either supervised, thus not source agnostic, or unsupervised with a lower accuracy and dependent on the geometrical structure of the electrode.

Supervised models, which are trained on input data from a single source, create an internal embedding of the given waveforms. This means that they represent the waveforms in a specific way in order to identify patterns and make predictions. However, when these models are applied to new recordings, they may struggle because the internal representation of the waveforms is not general enough. It only captures the variability of the properties of the waveforms seen during training and does not account for other types of waveforms. This is a common issue in supervised spike sorting, as the recordings used for training often only contain a limited number of waveforms, and new recordings may have different cluster assignments and/or a different electrode geometry.

In contrast, our self-supervised model is input-source agnostic, meaning it does not rely on the specific structure of the electrode in order to process the data. This allows us to train the model on multiple datasets simultaneously, resulting in a more generalized embedding of the waveform properties. By getting rid of channel-wise information, we are able to consider a broader range of waveform shapes and variations, improving the model's ability to generalize to new recordings.

The ability to train the final supervised model on multiple recordings at once is also important for improving its generalizability. In the supervised setup, this is achieved by assigning a binary class to the relevant boxes (containing neural activity) and predicting the features of those waveforms in the embedding space, rather than assigning cluster numbers. This approach differs from previous conventions in the field, where cluster numbers were typically assigned instead of predicting features. Overall, this two-staged process enables the development of a more robust and generalizable deep learning-based spike sorting model.

4.1. 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 Fig. 5. 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 (Fig. 6). 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.

4.2. 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 1, 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

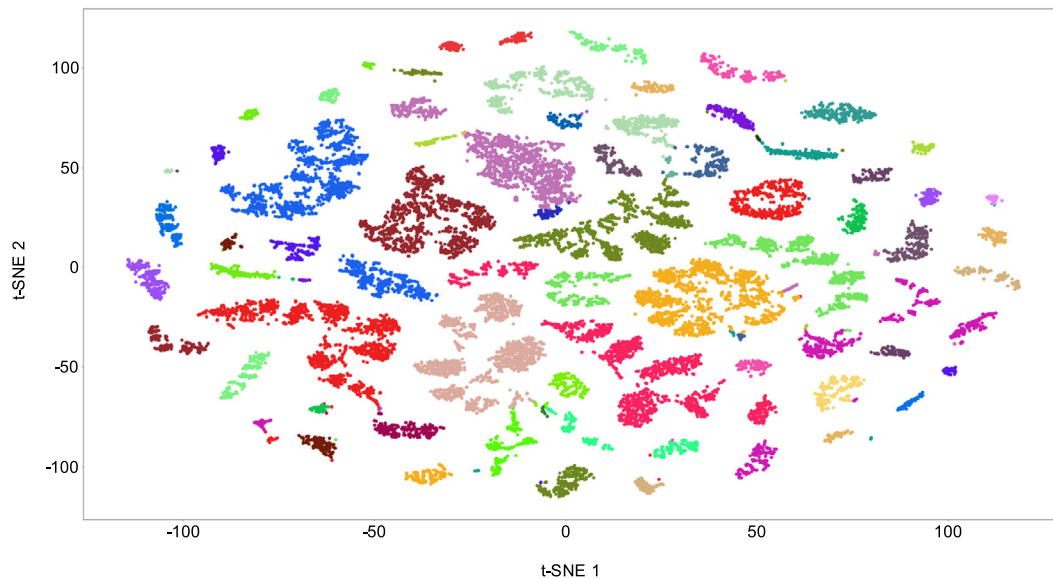


Fig. 7. Embedded features after clustering. The features generated by our model. To the features NMS is applied, PCA and the results of the latter are then given as inputs for t-SNE to visualize the original 32-dimensional feature space in 2 dimensions. The clustering was performed after PCA, the different coloring representing different ground truth clusters.

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 (Fig. 7). The detection performance has a quite large gap between the two recordings (Table 2.): 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 3.)

Table 4. 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.

4.3. 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 5, 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 6 kHz sampling rate.

4.4. Inside the black box

The supervised model was designed to identify and predict the relevant features of spike waveforms, and to do so, it utilizes a series of filters that are applied to the input data. These filters are designed to activate in response to specific patterns or features within the waveform data, and as such, they are able to identify and classify different types of spike waveforms. To better understand the features that the model is learning, a tool was used to extract the filters and their activations in response to a particular input sample. Fig. 8A shows a selection of these filters, and it can be seen that at the first layers, the filters are activated in response to any spike-like waveform. However, as the model progresses through the layers, the activations become more specific and are able to distinguish between different types of waveforms based on their form and characteristics. Fig. 8B shows the layer-specific activations averaged and superimposed on the original input data. The strong contrast in activation between the spikes and their surroundings suggests that the model is able to identify and predict the features of the spikes not only based on the waveforms themselves, but also based on their relationship with their surroundings and their appearances on other nearby channels.

4.5. 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.

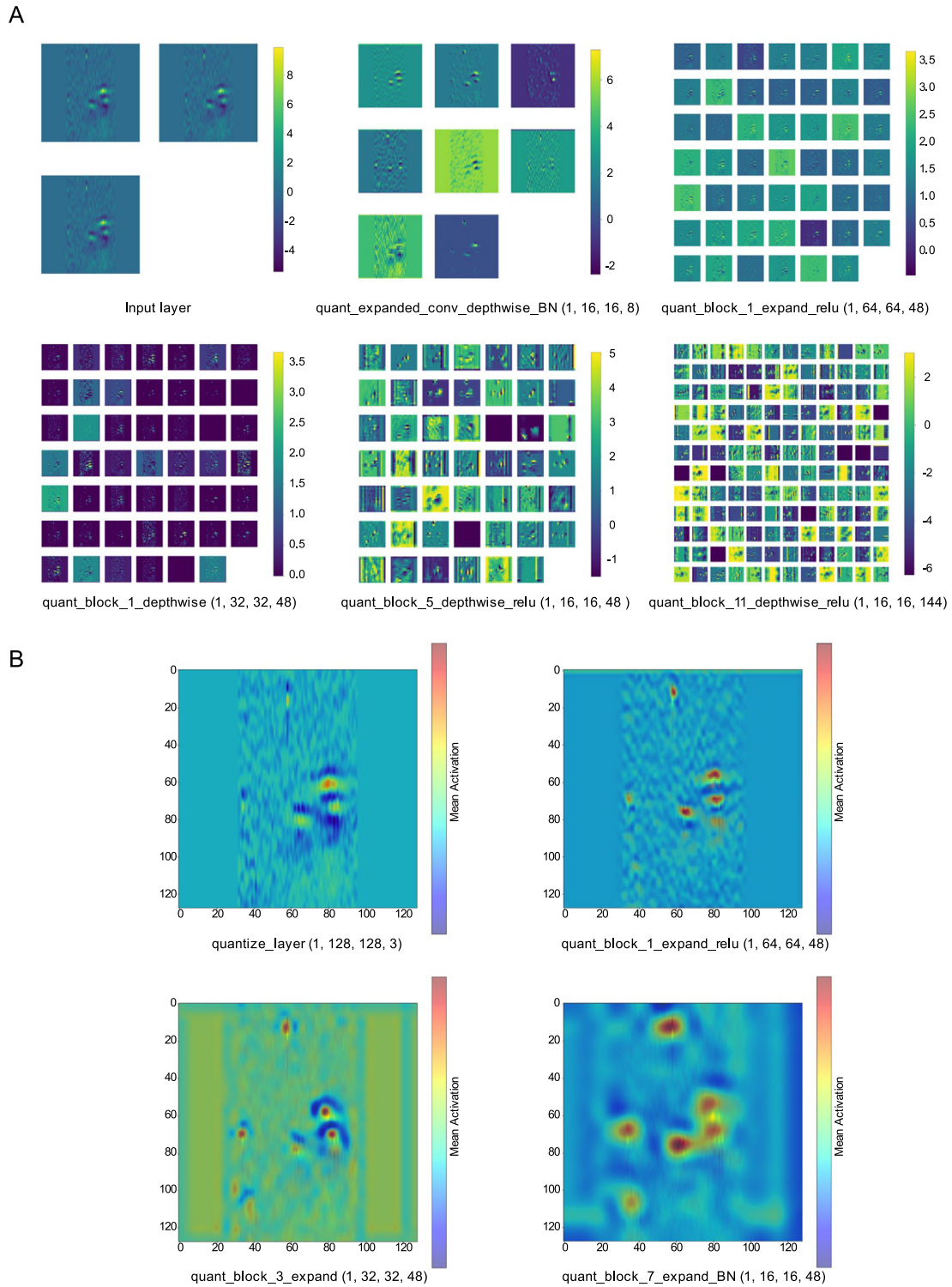


Fig. 8. Reverse engineering the model. Subfigure (A) depicts the activation of the filters on different levels of the model. The input layer shows the sample that the following activations were calculated on. The same input layer was used for subfigure (B) as well. On subfigure (B) four different heatmaps are superimposed on top of the original input. The heatmaps represent the mean activation of all the filters on the respective level.

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.

4.6. Limitations and future work

One of the limitations of the current system is that it may struggle with spatio-temporally highly overlapping spikes. This is because the current implementation of the Non-Maximum Suppression (NMS) postprocessing step is not optimized for separating spatio-temporally overlapping spikes. As a result, there is room for improvement in this area. As a future work, we intend to not only increase the separability between overlapping spikes, but also to migrate the NMS postprocessing to the Tensor Processing Unit (TPU) in order to reduce the computational time difference between using a PC with a USB accelerator and the Coral Dev Board. By addressing these issues, we hope to further improve the performance of the spike sorting pipeline.

5. Conclusion

To the best of our knowledge, our model is the first deep-learning-based model which can handle recordings with high number of channels, can be deployed to embedded systems (especially to TPUs) and can run real-time spike-sorting (depending on the configuration) on never-seen recordings with performance comparable to current state-of-the-art methods.

CRedit authorship contribution statement

János Rokai: Conceptualization, Methodology, Implemented the algorithms, Writing – original draft. **István Ulbert:** Writing – original draft, Supervision. **Gergely Márton:** Conceptualization, Methodology, Writing – original draft, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

Project no. FK132823 has been implemented with the support provided by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund, financed under the FK_19 funding scheme. This research was also funded by the Hungarian Brain Research Program (2017_1.2.1-NKP-2017-00002) and the TUDFO/51757-1/2019-ITM grant by the Hungarian National Research, Development and Innovation Office. Project no. RRF-2.3.1-21-2022-00015 has been implemented with the support provided by the European Union. JR is thankful to Semmelweis University for the EFOP-3.6.3-VEKOP-16-2017-00009 grant and to the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund for the ÚNKP-21-3-II-SE-1.

Code availability

The software is publicly available at: <https://github.com/roakai/jano/spike-on-edge>

Appendix A. Supplementary data

Supplementary material related to this article can be found online at <https://doi.org/10.1016/j.neunet.2023.02.036>.

References

- Allen, B. D., Moore-Kochlacs, C., Bernstein, J. G., Kinney, J. P., Scholvin, J., Seoane, L. F., et al. (2018). Automated in vivo patch-clamp evaluation of extracellular multielectrode array spike recording capability. *Journal of Neurophysiology*, 120(5), 2182–2200.
- Berényi, A., Somogyvári, Z., Nagy, A. J., Roux, L., Long, J. D., Fujisawa, S., et al. (2014). Large-scale, high-density (up to 512 channels) recording of local circuits in behaving animals. *Journal of Neurophysiology*, 111(5), 1132–1149.
- Biffi, E., Ghezzi, D., Pedrocchi, A., & Ferrigno, G. (2008). Spike detection algorithm improvement, spike waveforms projections with PCA and hierarchical classification. In *IET conf publ*, no. 540 CP.
- Bod, R. B., Rokai, J., Meszéna, D., Fiáth, R., Ulbert, I., & Márton, G. (2022). From end to end: Gaining, sorting, and employing high-density neural single unit recordings. *Frontiers in Neuroinformatics*, 16(June).
- Buccino, A. P., Garcia, S., & Yger, P. (2022). Spike sorting: new trends and challenges of the era of high-density probes. *Progress in Biomedical Engineering*, 4(2), 1–20.
- Cai, H., Gan, C., Wang, T., Zhang, Z., & Han, S. (2019). Once-for-all: Train one network and specialize it for efficient deployment. (pp. 1–15). Available from: <http://arxiv.org/abs/1908.09791>.
- Chah, E., Hok, V., Della-Chiesa, A., Miller, J. J. H., O'Mara, S. M., & Reilly, R. B. (2011). Automated spike sorting algorithm based on Laplacian eigenmaps and k-means clustering. *Journal of Neural Engineering [Internet]*, 8(1), Article 016006, Available from: <https://iopscience.iop.org/article/10.1088/1741-2560/8/1/016006>.
- Choi, J. H., Jung, H. K., & Kim, T. (2006). A new action potential detector using the MTEO and its effects on spike sorting systems at low signal-to-noise ratios. *IEEE Transactions on Biomedical Engineering*, 53(4), 738–746.
- Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *2017 IEEE conference on computer vision and pattern recognition, (CVPR) [Internet]* (pp. 1800–1807). IEEE, Available from: <http://ieeexplore.ieee.org/document/8099678/>.
- Chung, J. E., Magl, J. F., Barnett, A. H., Tolosa, V. M., Tooker, A. C., Lee, K. Y., et al. (2017). A fully automated approach to spike sorting. *Neuron [Internet]*, 95(6), 1381–1394.e6. <http://dx.doi.org/10.1016/j.neuron.2017.08.030>.
- Dai, M., & Luo, J. (2014). A robust method for spike sorting with overlap decomposition. *Journal of Computers*, 9(3), 1195–1198.
- Diggelmann, R., Fiscella, M., Hierlemann, A., & Franke, F. (2018). Automatic spike sorting for high-density microelectrode arrays. *Journal of Neurophysiology*, 120(6), 3155–3171.
- Dwibedi, D., Aytar, Y., Tompson, J., Sermanet, P., & Zisserman, A. (2021). With a little help from my friends: Nearest-neighbor contrastive learning of visual representations. In *Proc IEEE int conf comput vis*. (pp. 9568–9577).
- Eom, J., Park, I. Y., Kim, S., Jang, H., Park, S., Huh, Y., et al. (2021). Deep-learned spike representations and sorting via an ensemble of auto-encoders. *Neural Networks [Internet]*, 134, 131–142. <http://dx.doi.org/10.1016/j.neunet.2020.11.009>.
- Fiáth, R., Márton, A. L., Mátyás, F., Pinke, D., Márton, G., Tóth, K., et al. (2019). Slow insertion of silicon probes improves the quality of acute neuronal recordings. *Scientific Reports*, 9(1), 1–17.
- Fiáth, R., Raducanu, B. C., Musa, S., Andrei, A., Lopez, C. M., van Hoof, C., et al. (2018). A silicon-based neural probe with densely-packed low-impedance titanium nitride microelectrodes for ultrahigh-resolution in vivo recordings. *Biosensors and Bioelectronics [Internet]*, 106(2017), 86–92. <http://dx.doi.org/10.1016/j.bios.2018.01.060>.
- Hao, H., Chen, J., Richardson, A., Van der Spiegel, J., & Aflatouni, F. A. (2021). 10.8 μ W Neural signal recorder and processor with unsupervised analog classifier for spike sorting. *IEEE Transactions on Biomedical Circuits and Systems [Internet]*, 15(2), 351–364, Available from: <https://ieeexplore.ieee.org/document/9417633/>.
- Hilgen, G., Sorbaro, M., Pirmoradian, S., Muthmann, J. O., Kepiro, I. E., Ullo, S., et al. (2017). Unsupervised spike sorting for large-scale, high-density multi-electrode arrays. *Cell Reports [Internet]*, 18(10), 2521–2532. <http://dx.doi.org/10.1016/j.celrep.2017.02.038>.
- Hill, E. S., Moore-Kochlacs, C., Vasireddi, S. K., Sejnowski, T. J., & Frost, W. N. (2010). Validation of independent component analysis for rapid spike sorting of optical recording data. *Journal of Neurophysiology*, 104(6), 3721–3731.
- Huang, L., Gan, L., & Ling, B. W. K. (2021). A unified optimization model of feature extraction and clustering for spike sorting. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 29, 750–759.
- Hwang, W. J., Lee, W. H., Lin, S. J., & Lai, S. Y. (2013). Efficient architecture for spike sorting in reconfigurable hardware. *Sensors (Switzerland)*, 13(11), 14860–14887.

- Jäckel, D., Frey, U., Fiscella, M., Franke, F., & Hierlemann, A. (2012). Applicability of independent component analysis on high-density microelectrode array recordings. *Journal of Neurophysiology*, 108(1), 334–348.
- Jun, J. J., Mitelut, C., Lai, C., Gratiy, S. L., Anastassiou, C. A., & Harris, T. D. (2017). Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction. Article 101030, bioRxiv <https://doi.org/10.1101/101030>.
- Jun, J. J., Steinmetz, N. A., Siegle, J. H., Denman, D. J., Bauza, M., Barbarits, B., et al. (2017). Fully integrated silicon probes for high-density recording of neural activity. *Nature [Internet]*, 551(7679), 232–236. <http://dx.doi.org/10.1038/nature24636>.
- Kim, K. H., & Kim, S. J. (2000). Neural spike sorting under nearly 0-db signal-to-noise ratio using nonlinear energy operator and artificial neural-network classifier. *IEEE Transactions on Biomedical Engineering*, 47(10), 1406–1411.
- Li, Z., Wang, Y., Zhang, N., & Li, X. (2020). An accurate and robust method for spike sorting based on convolutional neural networks. *Brain Sciences*, 10(11), 1–16.
- Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollar, P. (2020). Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2), 318–327.
- Magl, J., Jun, J. J., Lovero, E., Morley, A. J., Hurwitz, C. L., Buccino, A. P., et al. (2020). SpikeForest, reproducible web-facing ground-truth validation of automated neural spike sorters. *Elife [Internet]*, 9, Article e55167, Available from: <https://elifesciences.org/articles/55167>.
- Mamlouk, A. M., Sharp, H., Menne, K. M. L., Hofmann, U. G., & Martinetz, T. (2005). Unsupervised spike sorting with ICA and its evaluation using GENESIS simulations. *Neurocomputing*, 65–66(SPEC. ISS.), 275–282.
- Moghaddasi, M., Aliyari Shoorehdeli, M., Fatahi, Z., & Haghparast, A. (2020). Unsupervised automatic online spike sorting using reward-based online clustering. *Biomedical Signal Processing and Control [Internet]*, 56, Article 101701. <http://dx.doi.org/10.1016/j.bspc.2019.101701>.
- Musk, E. (2019). An integrated brain-machine interface platform with thousands of channels. *Journal of Medical Internet Research*, 21(10), 1–14.
- Pachitariu, M., Steinmetz, N., Kadir, S., Carandini, M., & Kenneth, D. H. (2016). Kilosort: realtime spike-sorting for extracellular electrophysiology with hundreds of channels. Article 061481, bioRxiv <https://doi.org/10.1101/061481>.
- Paraskevopoulou, S. E., Barsakcioglu, D. Y., Saberi, M. R., Eftekhari, A., & Constantinou, T. G. (2013). Feature extraction using first and second derivative extrema (FSDE) for real-time and hardware-efficient spike sorting. *Journal of Neuroscience Methods [Internet]*, 215(1), 29–37. <http://dx.doi.org/10.1016/j.jneumeth.2013.01.012>.
- Pouzat, C., & Garcia, S. (2019). Tridesclous [internet]. [cited 2020 Jul 7]. Available from: <https://github.com/tridesclous/tridesclous>.
- Quiroga, R. Q., Nadasdy, Z., & Ben-Shaul, Y. (2004). Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural Computation*, 16(8), 1661–1687.
- Rácz, M., Liber, C., Németh, E., Fiáth, R., Rokai, J., Harmati, I., et al. (2020). Spike detection and sorting with deep learning. *Journal of Neural Engineering [Internet]*, 17(1), Article 016038, Available from: <https://iopscience.iop.org/article/10.1088/1741-2552/ab4896>.
- Radmanesh, M., Rezaei, A. A., Hashemi, A., Jalili, Mahdi, & Goudarzi, M. M. (2022). Online spike sorting via deep contractive autoencoder, *Neural Networks [Internet]* vol. 155. (pp. 39–49). Available from: <https://doi.org/10.1016/j.neunet.2022.08.001>.
- Rokai, J., Rácz, M., Fiáth, R., Ulbert, I., & Márton, G. (2021). Elvisort: Encoding latent variables for instant sorting, an artificial intelligence-based end-to-end solution. *Journal of Neural Engineering*, 18(4).
- Saif-Ur-Rehman, M., Lienk mper, R., Parpaley, Y., Wellmer, J., Liu, C., Lee, B., et al. (2019). SpikeDeeptector: A deep-learning based method for detection of neural spiking activity. *Journal of Neural Engineering*, 16(5).
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF conference on computer vision and pattern recognition [Internet]* (pp. 4510–4520). IEEE, Available from: <https://ieeexplore.ieee.org/document/8578572/>.
- Schaffer, L., Nagy, Z., Kincses, Z., Fiáth, R., & Ulbert, I. (2021). Spatial information based OSort for real-time spike sorting using FPGA. *IEEE Transactions on Biomedical Engineering*, 68(1), 99–108.
- Seong, C., Lee, W., & Jeon, D. (2021). A multi-channel spike sorting processor with accurate clustering algorithm using convolutional autoencoder. *IEEE Transactions on Biomedical Circuits and Systems*, 15(6), 1441–1453.
- Steinmetz, N. A., Aydin, C., Lebedeva, A., Okun, M., Pachitariu, M., Bauza, M., et al. (2021). Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings. *Science*(80-), 372(6539).
- Stringer, C., et al. (2019). Spontaneous behaviors drive multidimensional, brainwide activity. *Science*, 364, Article eaav7893. <http://dx.doi.org/10.1126/science.aav7893>.
- Takahashi, S., Anzai, Y., & Sakurai, Y. (2003). A new approach to spike sorting for multi-neuronal activities recorded with a tetrode - How ICA can be practical. *Neuroscience Research*, 46(3), 265–272.
- Tan, M., Pang, R., & Le, Q. V. (2019). EfficientDet: Scalable and efficient object detection. In *Proceedings of the IEEE computer society conference on computer vision and pattern recognition [Internet]* (pp. 10778–10787). Available from: <http://arxiv.org/abs/1911.09070>.
- Valencia, D., & Alimohammad, A. (2019). A real-time spike sorting system using parallel OSort clustering. *IEEE Transactions on Biomedical Circuits and Systems*, 13(6), 1700–1713.
- Vargas-Irwin, C., & Donoghue, J. P. (2007). Automated spike sorting using density grid contour clustering and subtractive waveform decomposition. *Journal of Neuroscience Methods*, 164(1), 1–18.
- Wang, P. K., Pun, S. H., Chen, C. H., McCullagh, E. A., Klug, A., Li, A., et al. (2019). Low-latency single channel real-time neural spike sorting system based on template matching. *PLoS One*, 14(11), 1–30.
- Wood, F., Fellows, M., Donoghue, J. P., & Black, M. J. (2004). Automatic spike sorting for neural decoding. In *Annu int conf IEEE eng med biol - proc., vol. 26, no. VI* (pp. 4009–4012).
- Wouters, J., Kloosterman, F., & Bertr, A. (2021). A data-driven spike sorting feature map for resolving spike overlap in the feature space. *Journal of Neural Engineering*, 18(4).
- Xu, H., Han, Y., Han, X., Xu, J., Lin, S., & Cheung, R. C. C. (2019). Unsupervised and real-time spike sorting chip for neural signal processing in hippocampal prosthesis. *Journal of Neuroscience Methods [Internet]*, 311(2018), 111–121. <http://dx.doi.org/10.1016/j.jneumeth.2018.10.019>.
- Yang, Y., & Mason, A. J. (2017). Frequency band separability feature extraction method with weighted Haar wavelet implementation for implantable spike sorting. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(6), 530–538.
- Yang, K., Wu, H., & Zeng, Y. (2017). A simple deep learning method for neuronal spike sorting. *Journal of Physics: Conference Series [Internet]*, 910(1), Article 012062, Available from: <https://iopscience.iop.org/article/10.1088/1742-6596/910/1/012062>.
- Yger, P., Spampinato, G. L. B., Esposito, E., Lefebvre, B., Deny, S., Gardella, C., et al. (2018). A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo. *Elife*, 7, 1–23.