Rokai et al., J. Neural Eng. 18(4): 046033 (2021)
DOI: 10.1088/1741-2552/abf521
Manuscript as accepted

ELVISort: Encoding Latent Variables for Instant Sorting, an Artificial Intelligence-Based End-to-End Solution

János Rokai[1,2], Melinda Rácz[1,2], Richárd Fiáth[1,3], István Ulbert[1,3,*], Gergely Márton[1,3]

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

[2] School of Ph.D. Studies, Semmelweis University, Üllői út 26, H - 1085 Budapest, Hungary

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

[*] Corresponding author

E-mail: ulbert.istvan@ttk.hu

Abstract

Objective. The growing number of recording sites of silicon-based probes means that an increasing amount of neural cell activities can be recorded simultaneously, facilitating the investigation of underlying complex neural dynamics. In order to overcome the challenges generated by the increasing number of channels, highly automated signal processing tools are needed. Our goal was to build a spike sorting model that can perform as well as offline solutions while maintaining high efficiency, enabling high-performance online sorting.

Approach. In this paper we present ELVISort, a deep learning method that combines the detection and clustering of different action potentials in an end-to-end fashion.

Main results. The performance of ELVISort is comparable with other spike sorting methods that use manual or semi-manual techniques, while exceeding the methods which use an automatic approach: ELVISort has been tested on three independent datasets and yielded average F_1 scores of 0.96, 0.82 and 0.81, which comparable with the results of state-of-the-art algorithms on the same data. We show that despite the good performance, ELVISort is capable to process data in real-time: the time it needs to execute the necessary computations for a sample of given length is only 1/15.71 of its actual duration (i.e. the sampling time multipled by the number of the sampling points).

Significance. ELVISort, because of its end-to-end nature, can exploit the massively parallel processing capabilities of GPUs via deep learning frameworks by processing multiple batches in parallel, with the potential to be used on other cutting-edge AI-specific hardware such as TPUs, enabling the development of integrated, portable and real-time spike sorting systems with similar performance to offline sorters.

Keywords: spike sorting, deep learning, variational autoencoder

1. Introduction

The utilization of high-density neural microelectrode arrays (MEA) with recording sites in the order of hundreds and thousands increases rapidly (1–3). This trend creates a need for fast, reliable and highly automated data processing algorithms. Using extracellularly implanted MEAs, raw data is typically recorded at a high sampling rate (20–30 kHz) in order to capture temporal features of individual action potentials of nearby neurons (single-unit activities, spikes). To replace the slow and time-inefficient human evaluation of these data, spike detecting and sorting algorithms have emerged to automate the process. Although the fine spatial resolution and the high number of recording channels of modern MEAs can make the detection and sorting of spikes more precise, the utilization of conventional methods for these tasks become more difficult and time consuming. Many approaches that work well on few-channel MEAs cannot be applied with such efficiency to MEAs with a larger number of channels (4). As will be discussed later, among the several methods which were implemented for this problem, the ones which are based on the rapidly emerging deep learning techniques are still underrepresented. This cannot be stated for other areas of electrophysiology, like motor imaginary task classification based on EEG, where a plethora of deep learning techniques was applied (5). One of the main reasons might be the demand for supervised learning for high-quality labeled data. Recently several unsupervised deep learning methods were revisited and promising results were produced in relation to different problems such as extracting visual concepts from images or image generation (6–8).

A further aspect to be considered in the development of spike detection and sorting algorithms is their applicability in systems that require real-time processing, for example, brain-computer interfaces (BCIs). It will be shown that the solution presented in this paper allows the detection and sorting of single unit activities 15 times faster than real time on a regular computer equipped with an NVIDIA GeForce 2080Ti GPU. Modern artificial intelligence accelerator application-specific integrated circuits (ASICs) such as tensor processing units (TPUs) (9) allow the inference of deep convolutional neural networks (CNNs) on wearable devices. To maximize the efficiency offered by these devices, an end-to-end solution is needed.

Traditional spike sorting algorithms usually consist of three separate phases: detection, feature extraction, and clustering. Different approaches are combined in multiple ways, our aim for this section is not to give an exhaustive review on all of the possible systems, but to present the general methodology.

Action potential detection usually starts with filtering in the frequency domain in order to clean wideband data from local field potential signals, low and high-frequency noises. This is typically done between 300 and 3000 Hz (4,10). The following step implementing the actual detection is approached in different ways: in some researches (11) (12) (13) a simple threshold-based method was applied. This threshold is computed relatively to the estimated standard deviation of the noise in the particular data. Other approaches to spike detection include non-linear energy operator thresholding (14), Teager energy operator thresholding (15), and wavelet decomposition (16). Approaches similar to ours use deep learning methods, e.g. a single dense layer for spike detection (17) or LSTM layers to evaluate recordings per data point (18).

Whilst a good detection module is necessary for a satisfactory spike sorting system, having a good feature extraction algorithm is not less important: having distinctive features extracted from the filtered data is essential to any clustering algorithm to operate well. The challenges to overcome during this phase include but are not limited to overlapping spike waveforms, bursts with changing amplitude, and electrode drift. To properly cluster spikes even in these cases, a robust feature extractor is needed. For this, several methods have been proposed, such as feature extraction based on principal component analysis (PCA) (19–21), wavelet coefficient (22), wavelet packet coefficient (16), and wavelet packet decomposition used with support vector machine (23).

The third phase, i.e. the segmentation of extracted features, has a wide range of approaches as well. The most widely applied semiautomatic method is k-means clustering, which is featured in many research papers (11,20,24), combined with other methods. Other approaches such as Bayesian classification (10), superparamagnetic clustering (16), support vector machine classification (25), and independent component analysis (24) are also used in the segmentation phase of spike sorting. Deep learning-based solutions for this problem have also been applied lately, such as PCANet (26) which performs similarly to conventional methods. In another study, high precision classification with the utilization of convolutional neural networks (CNNs) (18) was achieved. However, this system relies on supervised learning.

Modern solutions such as KiloSort (27), SpyKING CIRCUS (28), and MountainSort (29) aim to automate the process of spike sorting. Although the different phases of spike sorting are usually organized in an automatic pipeline, hyperparameter adjustments and manual curation are still required for KiloSort and SpyKING CIRCUS. MountainSort4 offers a solution without the need of hyperparameter fine-tuning, at the cost of performance (29). The ideal spike sorting algorithm is fully unsupervised. Autoencoders exploit the advantages of deep learning but are capable of fulfilling this condition (30). Supervised models outperform conventional approaches in many different areas, whilst unsupervised deep learning methods were not applied to problems with great success until recently. Several autoencoder variants were proposed in recent years, e.g. variational autoencoders (VAEs) (31), which offer a highly robust generalization capability while producing a modified and regulated latent variable space. Autoencoder-based spike processing methods were also proposed recently: using a deep compressive autoencoder (32) managed to compress data of APs for a more efficient transfer; (33) used a VAE-based model to determine the position of the neurons in an extracellular recording.

In this study, a β-VAE based deep learning architecture called ELVISort will be presented. The artificial neural network completes the detection, feature extraction and sorting phases, which could be advantageous in wearable and implantable closed-loop systems. ELVISort was trained by applying the supervised paradigm to the detection/clustering and the unsupervised paradigm to the reconstruction part of the model.

2. Methods

2.1 Autoencoders

Autoencoder models are trained in an unsupervised way, aiming at the reproduction of the input data while having a feature reduction process in-between. Relevant features of the input data are extracted by a bottleneck (latent) layer in the network with (usually much) fewer nodes than the input. The idea of the autoencoders first appeared in publication in 1986 (30).

Autoencoders can be defined using the pair of formulas below:

$$\begin{aligned} \boldsymbol{z} &= E(\boldsymbol{x}), \\ \boldsymbol{x}' &= D(\boldsymbol{z}), \end{aligned} \quad (1)$$

where encoder E and decoder D can be any differentiable function, any deep learning structure. Latent layer $\boldsymbol{z}$ serves as a compressed representation of input snippet $\boldsymbol{x}$. $\boldsymbol{x}'$ is the result of the reconstruction.

A general principle in building autoencoder architectures is that the decoder is a mirrored encoder. In this work, this principle will be followed less strictly, which will be discussed later. The autoencoder is trained so that reconstruction $\boldsymbol{x}'$ should be as close to the original input $\boldsymbol{x}$ as possible. The difference is called reconstruction error, which the model is trained to minimize.

The reconstruction loss (L) is given by the expression

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

where $\boldsymbol{N}$ is the length of the input $\boldsymbol{x}$.

2.2 Variational autoencoders

Variational autoencoders (VAE) are generative models, taking a stochastic approach to data reconstruction. In paper (31) a variational model based on the probabilistic concept of Bayesian inference is described. The generative process starts at the level of latent layer $\boldsymbol{z}$. The conditional probability of data $\boldsymbol{x}$ given latent variable $\boldsymbol{z}$, $p(\boldsymbol{x}|\boldsymbol{z})$ can be described as:

$$p(\boldsymbol{x}|\boldsymbol{z}) = \frac{q(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{z})}\,.\,(3)$$

Likelihood $q(\boldsymbol{z}|\boldsymbol{x})$ in VAE is approximated by the probabilistic encoder q, assuming that $q(\boldsymbol{z}|\boldsymbol{x})$ is a Gaussian distribution. The probabilistic encoder outputs the prediction of the mean and variance of normal distribution $q(\boldsymbol{z}|\boldsymbol{x})$: $z_i \sim \mathrm{N}(\mu_i(\boldsymbol{x}), \sigma_i(\boldsymbol{x}))$.

The VAE model is trained using a cost function composed of reconstruction and Kullback-Leibler divergence (KL) losses. The reconstruction loss guarantees that the autoencoder will be trained to reconstruct the input data, while the KL loss regularizes the latent space. In the VAE, $p(\boldsymbol{z})$ is considered to be a normal distribution. The KL loss guarantees that the latent distribution is kept as close to a normal distribution as possible.

The loss function of the VAE (L_{VAE}) is defined as

$$L_{VAE}(\theta, \varphi; \beta) = \mathrm{E}_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta\,(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{D_{KL}}\left(q_\varphi(\boldsymbol{z}|\boldsymbol{x}) \;||\; p(\boldsymbol{z})\right).\,(4)$$

To make backpropagation applicable, the reparametrization trick is used (31).

2.3 β -VAE

β-VAE is an extended version of the original VAE model: it introduces a new hyperparameter β to the original loss function acting as a weighting variable (34).

The modified loss function of the β-VAE ($L_{\beta-VAE}$) is defined as

$$L_{\beta-VAE}(\theta, \varphi; \beta) = \mathrm{E}_{q_\varphi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta\,(\boldsymbol{x}|\boldsymbol{z})] - \beta\mathrm{D_{KL}}\left(q_\varphi(\boldsymbol{z}|\boldsymbol{x}) \;||\; p(\boldsymbol{z})\right).\,(5)$$

Having β equal to 1, we get the standard VAE model. By increasing it, a more disentangled representation can be extracted from the input (at the bottleneck layer level), but the reconstruction capability of the network will be diminished (35). A loss function with a great β causes the latent space being heavily centered around zero (which, as mentioned, facilitates the disentanglement of the latent space). This phenomenon along with the potentially high dimensionality of the data has a negative effect on the clustering capability of conventional unsupervised clustering algorithms, such as k-means. Throughout the datasets, a fixed β value of β = 15 was used in our model.

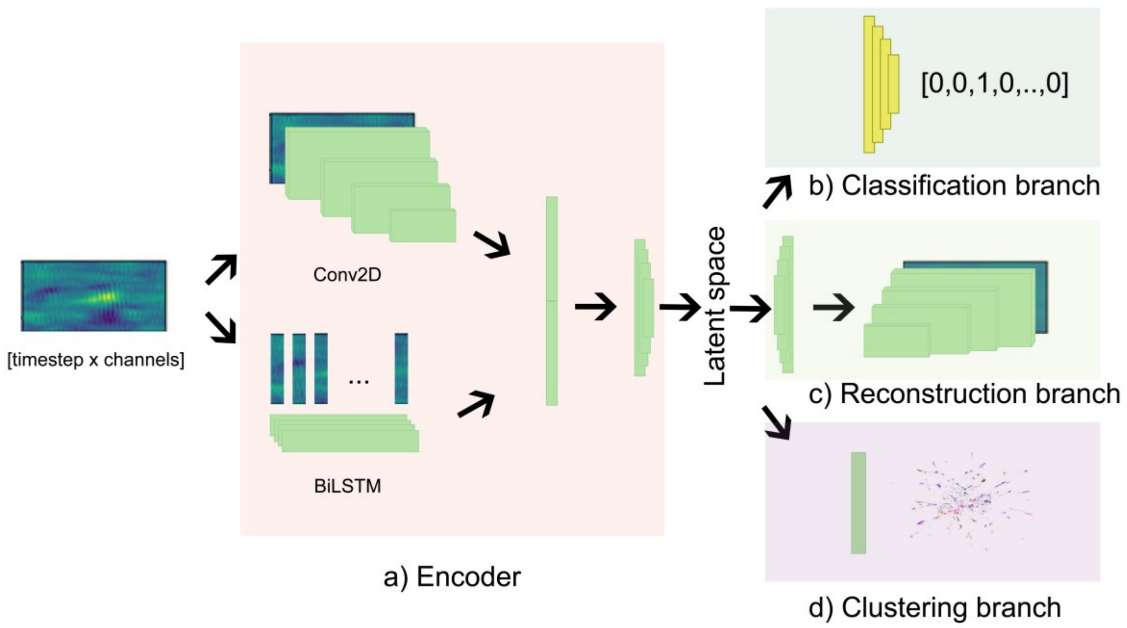
2.4 The architecture of ELVISort

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) (40), bidirectional LSTM (Bi-LSTM) (41) and 2D convolutional layers. (42,43).

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 (depicted in **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 transformation 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.

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 (44) 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 (45). 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 in order to improve clustering, finally a size of 32 was chosen. To further compress information, a hierarchical latent layout was used (8), 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) (46) and depicted in **Figure 2**. ELVISort clustering software is publically available at: www.github.com/rokaijano/elvisort.

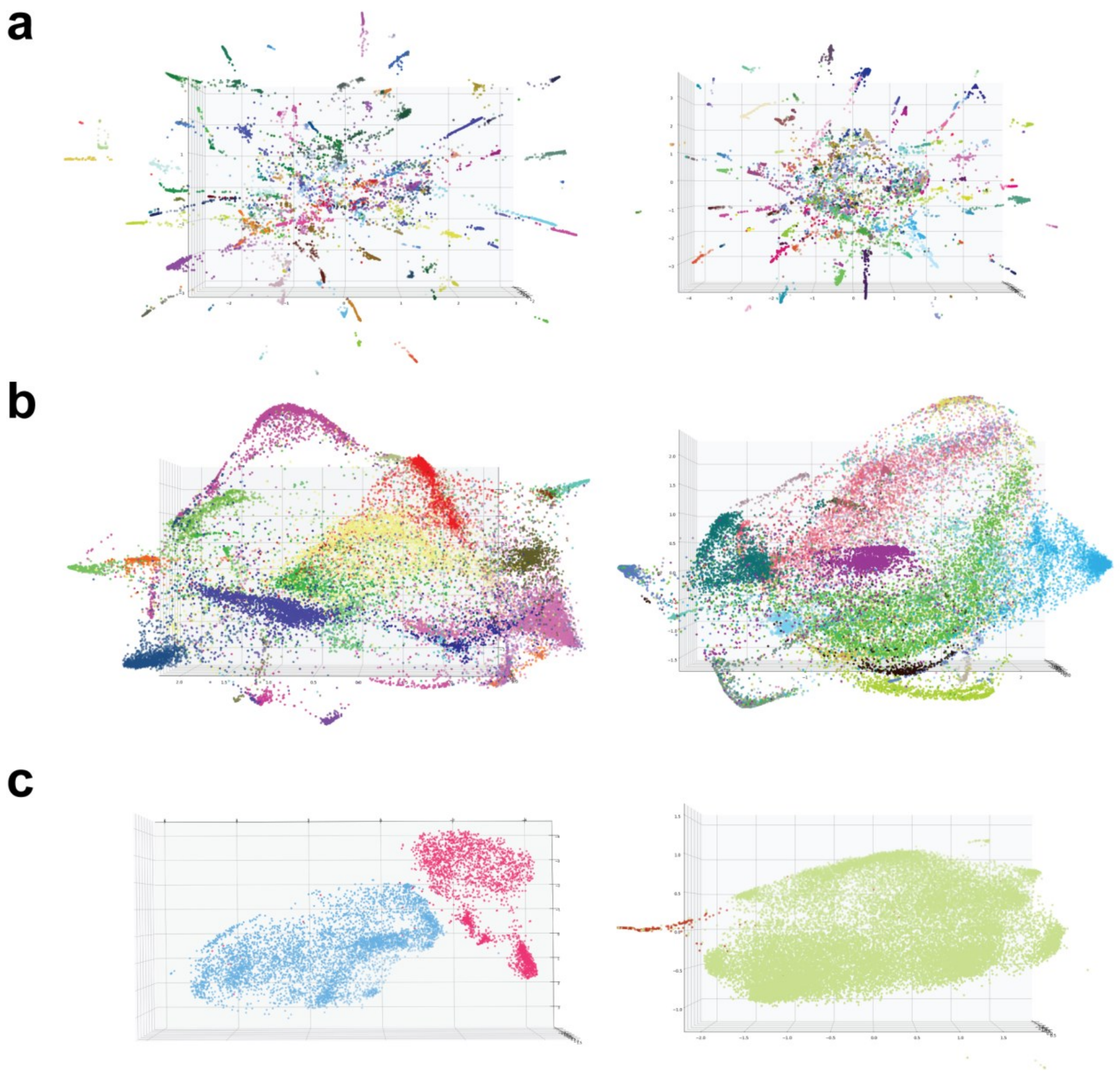


Figure 2. Latent spaces of train and test data for different datasets.

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).

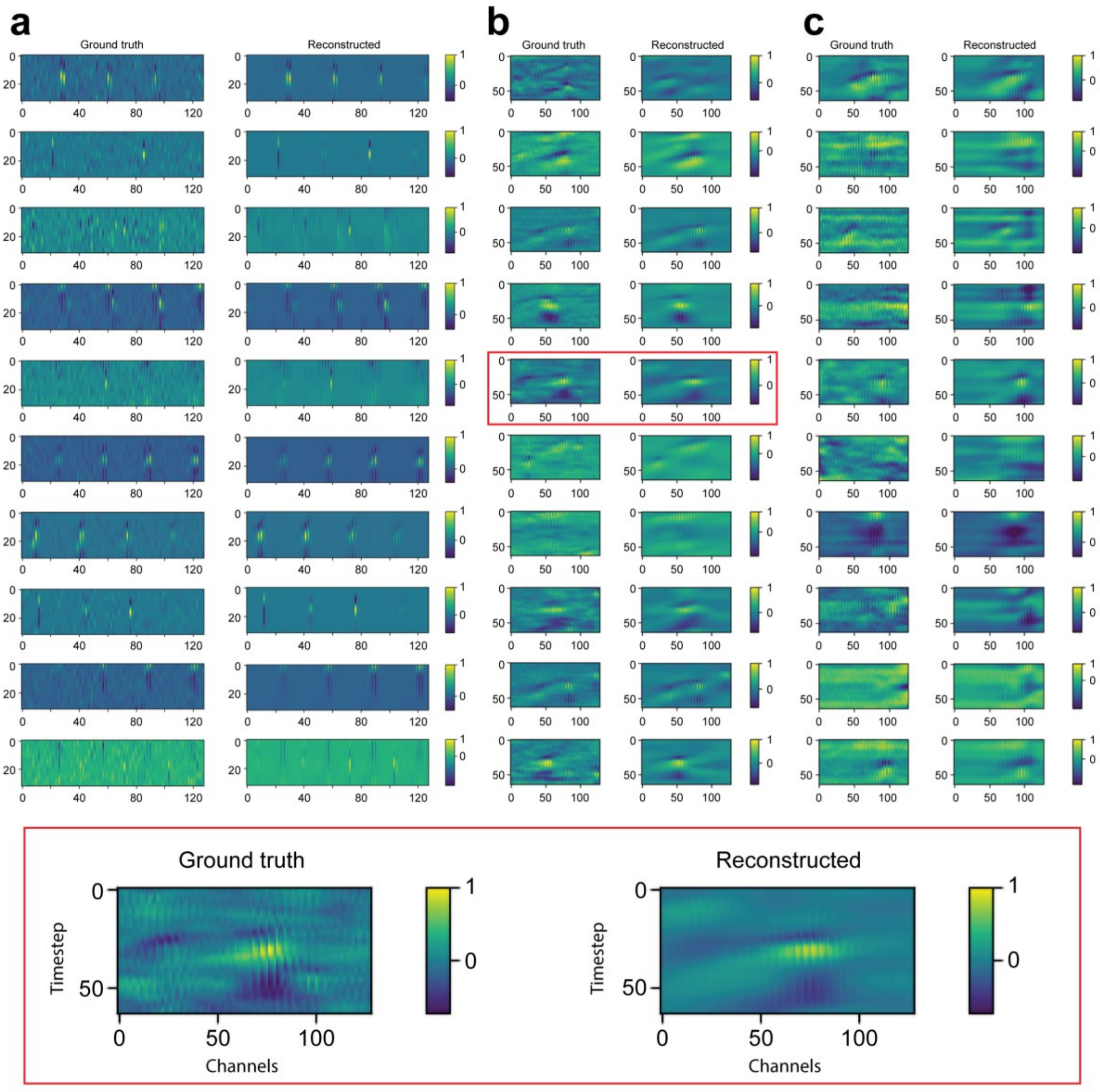


Figure 3. Reconstruction performance of ELVISort.
Snippet pairs are from Hybrid Janelia (a), Fiath (b) and Kampff (c) test datasets. Each row corresponds to an individual snippet with the original instance on the left and the reconstructed on the right. The horizontal and vertical axes represent channels (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).

To fine-tune the training according to our needs, regularizing branches have been included in the network. One of them was a classifier branch which consisted of several dense layers having softmax output activation during training on the Fiath and Kampff datasets and sigmoid for Hybrid Janelia signal snippets. The output of this branch was a vector of size 2 for spike detection, while for spike sorting the size was determined by the number of clusters in the processed dataset. For testing the combined detection and clustering capability of the model, the size of the output vector was determined by the number of clusters with an additional element included to represent the non-spike cluster. This branch was the only supervised part of the model. For the clustering branch, the approach suggested by Junyuan Xie (47) was followed, using Student's t-distribution (48) for assigning soft labels during training and k-means (49) for cluster center initialization. In our experience, the use of the clustering branch improved the overall performance of the model, however, increasing its loss weight above a certain value was not beneficial therefore it was kept relatively low. During training and architecture selection, reconstruction capability was considered a high priority (**Figure 3**).

2.5 Fiath dataset

The Fiath dataset contains 9 different recordings produced using 128-channel high-density silicon MEAs (the square-shaped recording sites form a 32 × 4 sensor array) with data recorded at 20 kHz. The data is recorded from different necortical areas like somatosensory cortex, parietal association cortex, motor cortex or the cingulate cortex. For each recording, spike labels were generated using KiloSort (36). After sorting the neural activity a manual revision was made on the clusters, with Phy and custom-written MATLAB scripts. During manual revision, the initial results of the Kilosort algorithm were corrected by applying different methods, such as: cluster merging, cluster separation based on FeatureView. Additionally clusters containing less than 100 elements were discarded and considered as noise. Clusters were considered as of good quality if their inter-unit waveform variability was low and they had a clear refractory period (in their autocorrelogram). Additional custom scripts were applied to filter the results further, eliminating low-amplitude waveforms identified as units (< 60 µV peak-to-peak amplitude). For further technical details, regarding the used MEAs, please refer to the paper (1). Note that because spikes were labeled using another automatic algorithm, results based on the Fiath dataset reflect the performance of ELVISort relatively to the performance of the KiloSort. In the absence of a genuine ground truth, other datasets are to be evaluated to get a more objective picture on the performance of ELVISort.

2.6 Kampff dataset

Dataset provided by the Kampff Lab (37) was produced using two probes, an extracellular silicon MEA and a glass micro-pipette monitoring the potential inside the cell body of a singular neuron, at the same location, at a range of inter-probe distances, 800 to 1800 µm deep in cortex. Measurements were taken at 30 kHz from the cortex of anesthetized rats. Juxtacellular signals were analyzed by us using a custom script based on thresholding and, as a result, a binary label was generated for every 128-channel recording of the dataset.

2.7 Hybrid Janelia dataset

Since ground truth data is not available for clustering real-world data, the performance assessment of different methods is an ambiguous task. To alleviate this problem, the SpikeForest platform features a collection of the most popular spike sorting methods along with openly available datasets and offers a standardized interface to access them (38). To test the performance of ELVISort on recordings with a high number of channels, the *Hybrid Janelia* dataset was chosen with simulated probe drifting generated using 2D interpolation, where the waveform templates were recorded at 30 kHz using 5 µm-spaced electrodes as part of Kampff's Ultra Dense Extracellular Survey and based on that a hybrid recording was generated using Kilosort2 with added noise and drift. From the *hybrid_drift_siprobe* study, two recordings: *REC_64C_600S_11* and *REC_64C_600S_12* were used. The first one was used as a train and validation dataset while the latter was used as test data for the trained model. To keep the input feasible for a better inter-dataset comparison, the 64 channels were padded to match the channel number (128) of the other datasets.

2.8 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 (4)(10) using a Butterworth filter (39) of order 5. After this, a threshold is computed 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).

$$T^c = \frac{\text{median}(D^c)}{0.6745} + \theta \cdot \text{std}(D^c), (6)$$

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 $F_t = \begin{cases} 1, & if\ \sum_{i=0}^{C} abs(D_t^i) - Tr^i > 0 \\ 0, & if\ \sum_{i=0}^{C} abs(D_t^i) - Tr^i \le 0 \end{cases}$
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 (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.

2.9 Evaluation Metrics

To assess the performance of ELVISort, several metrics were applied. For detection, F_1 weighted score was used to take label imbalance into account. This score is calculated from the recall (R) and precision (P) performance metrics, which are based on the number of true positives (TP), false positives (FP) and false negatives (FN):

$$R = \frac{TP}{TP + FN}, \quad P = \frac{TP}{TP + FP}, \quad F_1 = 2 \cdot \frac{R \cdot P}{R + P}. (7)$$

For multi-categorical problems, F_1 score can be calculated multiple ways:

a) F_1 micro score (F_1^m) takes TP, FP and FN globally across categories (C):

$$R_m = \frac{\sum_{j=1}^{C} TP_j}{\sum_{j=1}^{C}(TP_j + FN_j)}, \quad P_m = \frac{\sum_{j=1}^{C} TP_j}{\sum_{j=1}^{C}(TP_j + FP_j)}, \quad F_1^m = 2 \cdot \frac{R_m \cdot P_m}{R_m + P_m}. \ (8)$$

b) F_1 macro score (F_1^M) takes the score for each category and an unweighted mean is calculated from these values:

$$F_1^M = \frac{1}{C}\sum_{j=1}^{C} F_{1_j}. (9)$$

c) F_1 weighted score (F_1^W) takes category imbalance into consideration. During averaging the category score is weighted relative to the number of snippets (n) in the respective category:

$$F_1^W = \frac{\sum_{j=1}^{C} n_j \cdot F_{1_j}}{\sum_{j=1}^{C} n_j}. (10)$$

To consider label imbalance, weighted F_1 score was used to evaluate the results for the Kampff and Fiath datasets while for the Hybrid Janelia dataset, micro F_1 score was calculated to compare the performance of ELVISort to other methods available at the SpikeForest platform. In order to make further comparisons, the accuracy (A) of the model was calculated according to SpikeForest:

$$A = \frac{TP}{TP + FN + FP}. (11)$$

3. Results

3.1 Training and testing on three different datasets

Throughout this paper, we will refer to data obtained from a single measurement as a "recording", and the collection of similar recordings as a "dataset". Three different datasets have been utilized: the Kampff (37), the Fiath (36) and the Hybrid Janelia (38); 128-channel recordings were used from the Kampff and Fiath datasets and 64-channel recordings (with simulated electrode drift) from the Hybrid Janelia (See Methods for further details).

In the preprocessing phase, data were filtered between 300 and 3000 Hz (4,10) with a Butterworth filter (39) of order 5. The recordings were sliced into snippets of the same sizes and basic thresholding was applied to generate spike candidate snippets (See Methods for details). This procedure resulted in a significant number of non-spikes within the spike candidate snippets, which allowed us to perform detection, classification and their combination on the same dataset.

Our aim was to build a model where the different spike sorting phases are merged and performed in an end-to-end fashion, but at the same time, we were interested in how the combination of the phases affects the initial performance of a particular phase. Therefore, ELVISort was tested on separate phases first.

Firstly, the spike detection capability of ELVISort was inspected having only a spike and a non-spike cluster, where the classification branch had to regularize the latent space to maximize the separation between the latent variables of spike and non-spike snippets. At this point, the clustering branch was not used.

Then, input data consisting of snippets containing spikes only was fed into ELVISort, thus its ability to form natural clusters and recognize them using its classification branch could be tested adequately. From this point, only Fiath and Hybrid Janelia datasets were used due to the nature of the Kampff dataset where the spiking events of only the patched neuron are known.

Finally, detection and clustering were combined. This time the input dataset contained both spike and non-spike snippets, maintaining the distinction between spike clusters.

3.2 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)**.

	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

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.

3.3 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 4a**).

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 4b**). Only one of the 9 recordings yielded poor results (0.51 F_1 score).

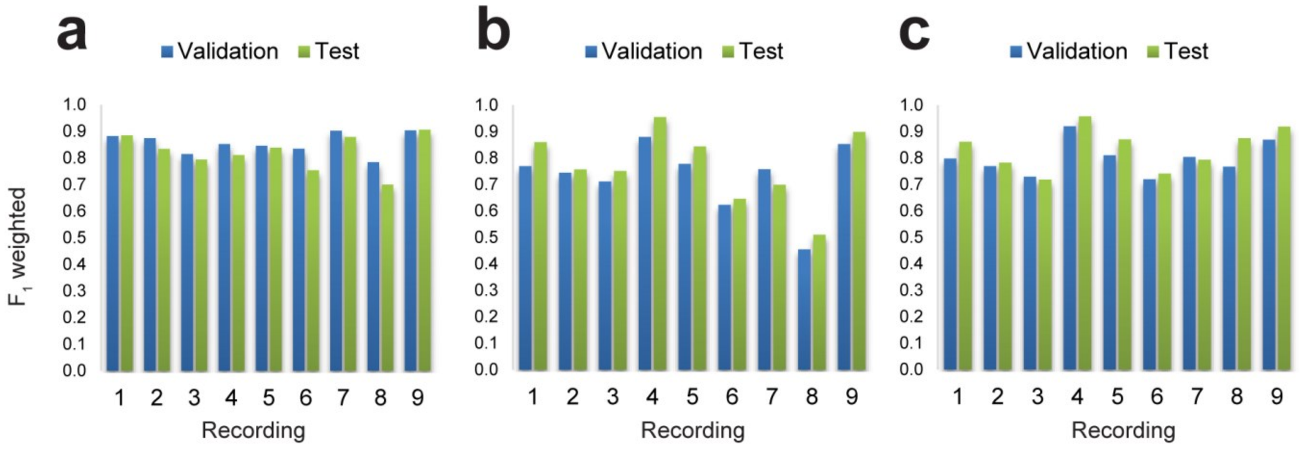
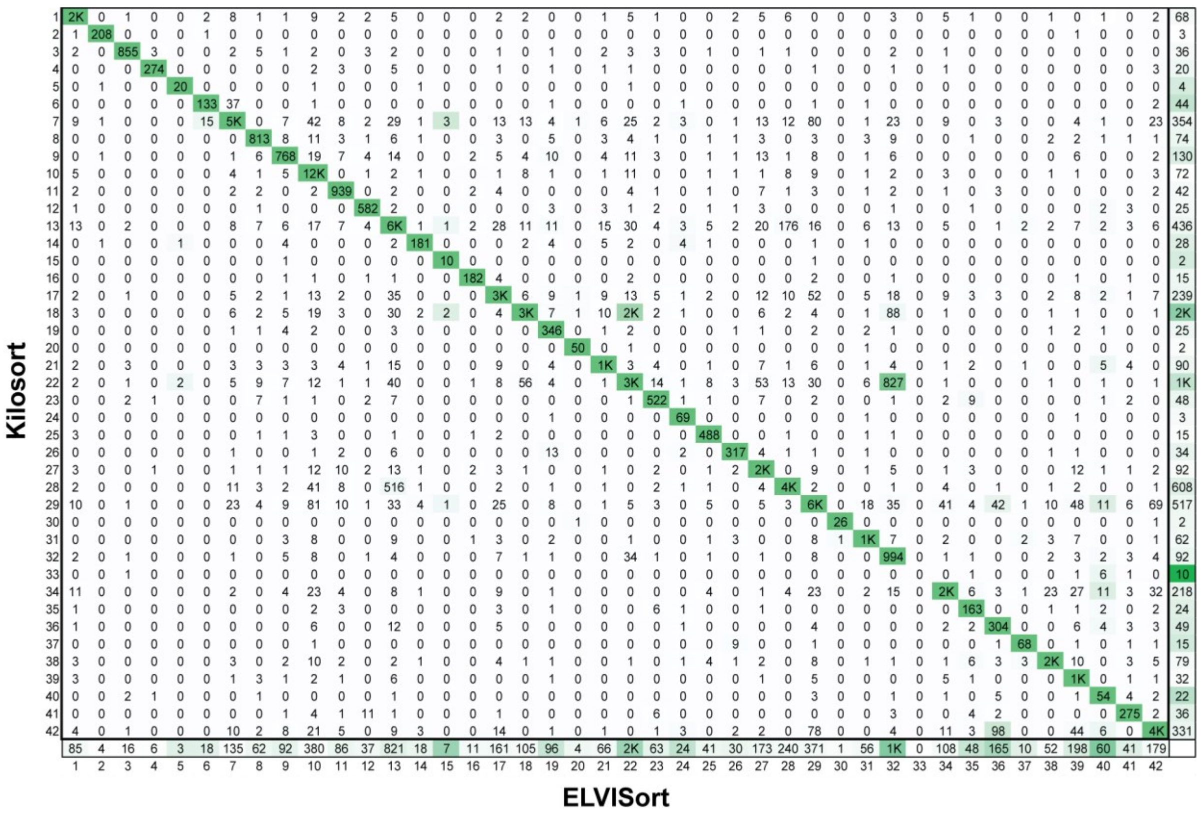


Figure 4. Results from the Fiath dataset.
Performance (F_1 score) of ELVISort for the different Fiath recordings (1–9) aiming spike detection (a), classification (b) and both (c). For the supervised part, labels generated by KiloSort followed by manual correction were used. In the cases where classification 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).

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 4c**; the classification performance of the system for a representative recording is illustrated in **Figure 5**).


Kilosort
ELVISort

	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	
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	

Figure 5. 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).

3.4 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 6a)**). The performance of ELVISort in relation to maximum peak values of the different clusters has also been evaluated (**Figure 6b)**.

	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

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).

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

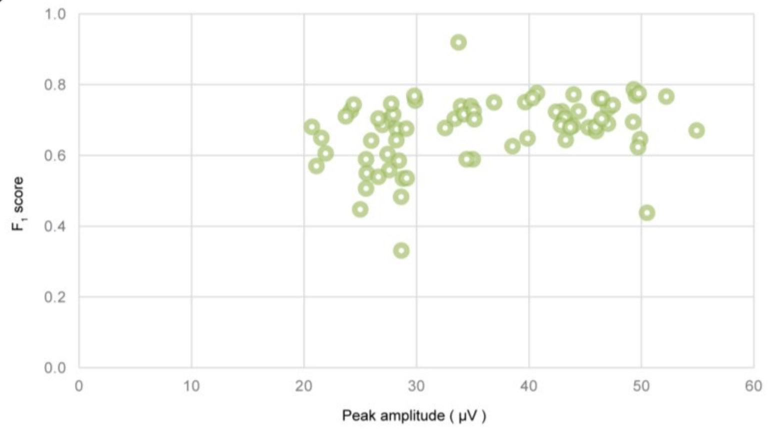


Figure 6. The performance of ELVISort on the Hybrid Janelia dataset.
(a) Detailed results per cluster. Cluster no. 1 represents the non-spike snippets. # – Cluster number, TP – True Positive, FP – False Positive, FN – False Negative.
(b) Accuracies for different clusters plotted against the peak amplitude.
Peak amplitude was computed averaging the maximum absolute spike amplitude from snippets where no other spike was present.

3.5 Assessment of Sorting Efficiency

All the training and test were run 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.

In **Figure 7**, we illustrate that ELVISort is the most efficient compared to other state-of-art methods, having a shorter runtime, while maintaining similar sorting performance.

We define xSpeed as the factor by which the sorting speed is faster compared to the actual recording:

$$xSpeed = \frac{Record\ duration}{Sorting\ time}. \quad (12)$$

	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

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.

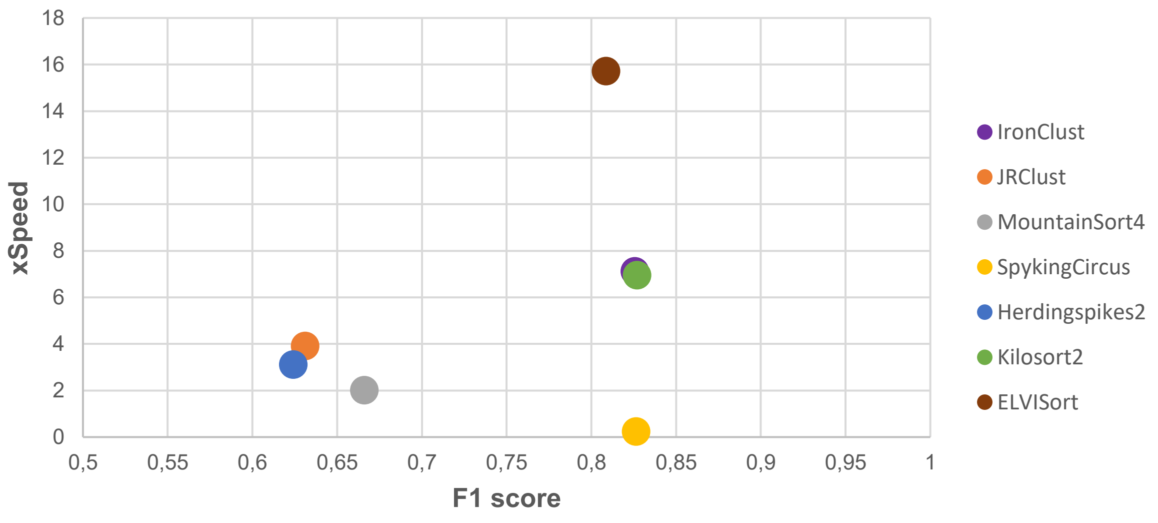


Figure 7. Sorting efficiency of different solutions regarding F_1 score and xSpeed.
The X-axis represents the F1 scores, which are listed in Table 2, while the Y-axis represents the xSpeed of each solution (Table 3). xSpeed is a factor without upper boundary, while the F1 score is a normalized score 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.

4. Discussion

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. (50)

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. (51) 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. (51) 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 (**Figure S1**).

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 2** 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 F_1 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 (52) or (53), 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. Integrating ELVISort into a TPU system would make runtime comparison to online sorting systems feasible.

5. Conclusion

In this paper, a deep learning model, ELVISort, was proposed for simultaneous spike detection and sorting. ELVISort utilizes variational autoencoder (i.e. β-VAE) architecture that compresses input data to less than half (0.488) percent of its original size making the clustering process memory and time-efficient.

The capabilities of ELVISort were demonstrated using three datasets; two of them (Kampff and Hybrid Janelia) were publicly available making the comparison of performance between ELVISort and other spike sorting methods feasible. We showed that our model can be a good choice in terms of both computational and runtime performances: ELVISort yielded F_1 scores of 0.81 for Hybrid Janelia (being surpassed by three sorters out of 8) and 0.96 for Kampff (comparably to the performance of the other methods) and proved to be applicable for real-time applications, requiring 1/36 of the original data timespan for processing. For the third dataset (Fiath), ELVISort gave an average F_1 score of 0.87 across the nine recordings containing activities from 23–46 single units.

The significance of our work is that we have shown that achieving a near state-of-the-art performance using deep learning architecture is possible, while we also demonstrated the efficiency of our model regarding the processing speed. We assume that ELVISort, after further investigation and development, can provide a basis of memory and time efficient brain-computer interfaces in the future.

Further investigations can be made in the future to replace the supervised, classification branch of the current model with an unsupervised one. We believe that the future of spike sorting models will tend towards end-to-end models, similar to ELVISort. End-to-end models enable different modern technologies (like TPUs) to be integrated into a real-time spike sorting system, providing precision comparable to offline sorters with high processing speeds, enabling a higher number of channels to be analyzed simultaneously. We believe, that the existence of end-to-end spike sorting models will give rise to small, integrated, portable spike sorting devices in the future.

Acknowledgements

G.M. and R.F. are thankful to the Hungarian National Research, Development and Innovation Office (FK132823 and PD124175) grants. This research has been partially funded by the Hungarian Brain Research Program (2017_1.2.1-NKP-2017-00002).

References

1. Fiáth R, Raducanu BC, Musa S, Andrei A, Lopez CM, van Hoof C, et al. A silicon-based neural probe with densely-packed low-impedance titanium nitride microelectrodes for ultrahigh-resolution in vivo recordings. Biosens Bioelectron [Internet]. 2018;106(October 2017):86–92. Available from: https://doi.org/10.1016/j.bios.2018.01.060

2. Berényi A, Somogyvári Z, Nagy AJ, Roux L, Long JD, Fujisawa S, et al. Large-scale, high-density (up to 512 channels) recording of local circuits in behaving animals. J Neurophysiol. 2014;111(5):1132–49.

3. Jun JJ, Steinmetz NA, Siegle JH, Denman DJ, Bauza M, Barbarits B, et al. Fully integrated silicon probes for high-density recording of neural activity. Nature [Internet]. 2017;551(7679):232–6. Available from: http://dx.doi.org/10.1038/nature24636

4. 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

5. Craik A, He Y, Contreras-Vidal JL. Deep learning for electroencephalogram (EEG) classification tasks: A review. J Neural Eng. 2019;16(3).

6. Higgins I, Sonnerat N, Matthey L, Pal A, Burgess CP, Bošnjak M, et al. SCAN: Learning hierarchical compositional visual concepts. arXiv [Internet]. 2017;1–24. Available from: http://arxiv.org/abs/1707.03389

7. Dizaji KG, Herandi A, Deng C, Cai W, Huang H. Deep Clustering via Joint Convolutional Autoencoder Embedding and Relative Entropy Minimization. Proc IEEE Int Conf Comput Vis.

2017;2017-Octob:5747–56.

8. 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

9. Chen J, Ran X. Deep Learning With Edge Computing: A Review. Proc IEEE. 2019;107(8).

10. 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.

11. 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.

12. 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

13. 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. 2011;8(1).

14. 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.

15. 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.

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

17. Issar D, Williamson RC, Khanna SB, Smith MA. A neural network for online spike classification that improves decoding accuracy. J Neurophysiol. 2020;123(4):1472–85.

18. Rácz M, Liber C, Németh E, Fiáth R, Rokai J, Harmati I, et al. Spike detection and sorting with deep learning. J Neural Eng. 2020;17(1).

19. 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.

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

21. 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).

22. Chen YY, Tsai YM, Chen LG. Algorithm and implementation of multi-channel spike sorting using GPU in a home-care surveillance system. Proc - IEEE Int Conf Multimed Expo. 2011;(June 2014).

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

24. Takahashi S, Anzai Y, Sakurai Y. A new approach to spike sorting for multi-neuronal activities recorded with a tetrode - How ICA can be practical. Neurosci Res. 2003;46(3):265–72.

25. Vogelstein RJ, Murari K, Thakur PH, Diehl C, Chakrabartty S, Cauwenberghs G. Spike sorting with support vector machines. Proc 26th Annu Int Conf IEEE EMBS. 2004;546–9.

26. Yang K, Wu H, Zeng Y. A Simple Deep Learning Method for Neuronal Spike Sorting. J Phys Conf Ser. 2017;910(1).

27. 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

28. 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.

29. 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

30. 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

31. Kingma DP, Welling M. Auto-encoding variational bayes. 2nd Int Conf Learn Represent ICLR 2014 - Conf Track Proc. 2014;(Ml):1–14.

32. Wu T, Zhao W, Keefer E, Yang Z. Deep compressive autoencoder for action potential compression in large-scale neural recording. J Neural Eng. 2018;15(6).

33. Hurwitz CL, Xu K, Srivastava A, Buccino AP, Hennig M. Scalable spike source localization in extracellular recordings using amortized variational inference. arXiv [Internet]. 2019;(NeurIPS 2019). Available from: http://arxiv.org/abs/1905.12375

34. Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed AL. Β-Vae: Learning Basic Visual Concepts With a Constrained Variational Framework. 2000;(July):1–14. Available from: https://openreview.net/references/pdf?id=Sy2fzU9gl

35. Sikka H, Zhong W, Yin J, Pehlevant C. A Closer Look at Disentangling in β-VAE. Conf Rec - Asilomar Conf Signals, Syst Comput [Internet]. 2019;2019-Novem:888–95. Available from: http://arxiv.org/abs/1912.05127

36. 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.

37. 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.

38. 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. 2020;9.

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

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

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

42. 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.

43. 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.

44. 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.

45. 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

46. 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.

47. Xie J, Girshick R, Farhadi A. Unsupervised deep embedding for clustering analysis. 33rd Int

Conf Mach Learn ICML 2016. 2016;1:740–9.

48. Van Der Maaten L. Learning a parametric embedding by preserving local structure. J Mach Learn Res. 2009;5:384–91.

49. Macqueen J. Some methods for classification and analysis of multivariate observations. 5-th Berkeley Symp Math Stat Probab. 1967;281–97.

50. 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

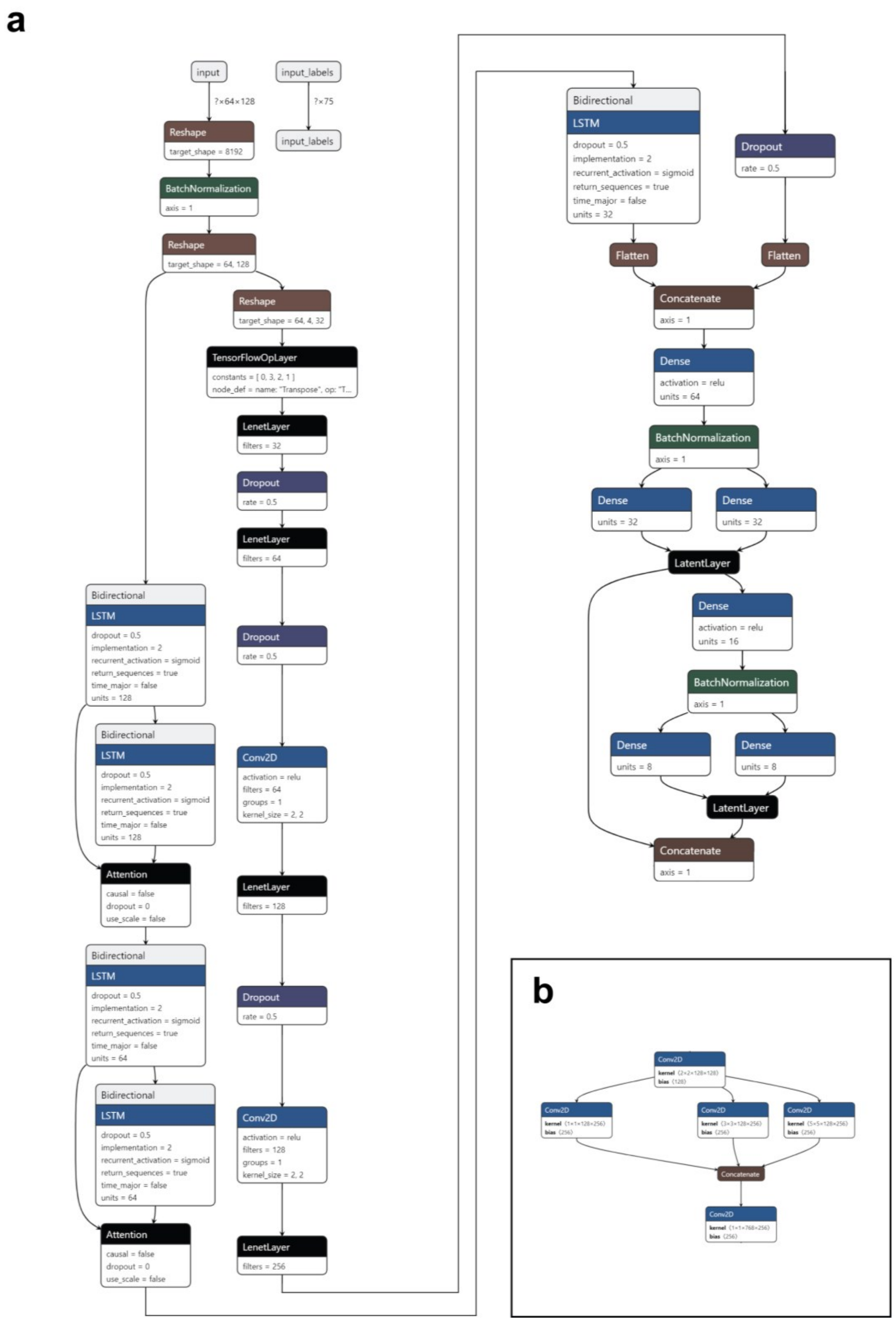
51. Mrabah N, Bouguessa M, Ksantini R. Adversarial deep embedded clustering: On a better trade-off between feature randomness and feature drift. IEEE Trans Knowl Data Eng. 2020;1–1.

52. Valencia D, Alimohammad A. A Real-Time Spike Sorting System Using Parallel OSort Clustering. IEEE Trans Biomed Circuits Syst. 2019;13(6):1700–13.

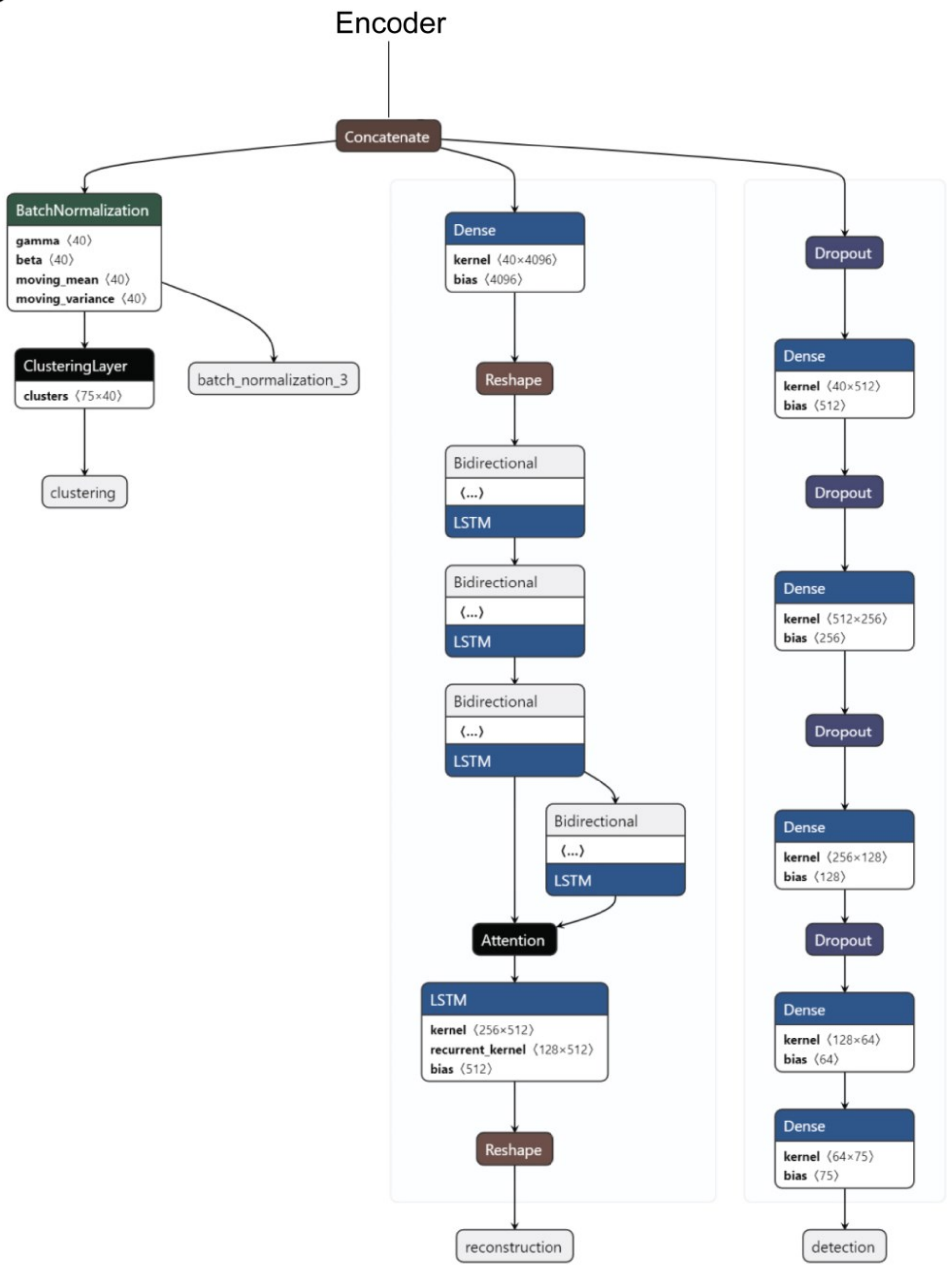
53. Mohammadi Z, Kincaid JM, Pun SH, Klug A, Liu C, Lei TC. Computationally inexpensive enhanced growing neural gas algorithm for real-time adaptive neural spike clustering. J Neural Eng. 2019;16(5).

Supplemental Information titles and legends

Figure S1. The detailed architecture of the model. An overall look at the architecture of the model. Details of each layer can be found also in the repository. We split the model for a better overview. On subfigure a) the details of the encoder can be seen. In subfigure b) a modified inception module is presented called LenetLayer with 256 filters. On the c) subfigure the different branches can be observed. The last concatenation layer of the encoder on subfigure a) is equivalent with the first concatenation layer on subfigure (c).



c



Encoder
Concatenate
BatchNormalization
gamma ⟨40⟩
beta ⟨40⟩
moving_mean ⟨40⟩
moving_variance ⟨40⟩
ClusteringLayer
clusters ⟨75×40⟩
clustering
batch_normalization_3
Dense
kernel ⟨40×4096⟩
bias ⟨4096⟩
Reshape
Bidirectional
⟨...⟩
LSTM
Bidirectional
⟨...⟩
LSTM
Bidirectional
⟨...⟩
LSTM
Bidirectional
⟨...⟩
LSTM
Attention
LSTM
kernel ⟨256×512⟩
recurrent_kernel ⟨128×512⟩
bias ⟨512⟩
Reshape
reconstruction
Dropout
Dense
kernel ⟨40×512⟩
bias ⟨512⟩
Dropout
Dense
kernel ⟨512×256⟩
bias ⟨256⟩
Dropout
Dense
kernel ⟨256×128⟩
bias ⟨128⟩
Dropout
Dense
kernel ⟨128×64⟩
bias ⟨64⟩
Dense
kernel ⟨64×75⟩
bias ⟨75⟩
detection

Figure S2. The k-fold cross-validation hyperparameter tuning. Several (n=15) k-fold cross-validations were evaluated on the recordings of the Fiath dataset with different reconstruction and detection loss ratios. The model was trained for a fixed number of epochs. For the parameters for each cross-validation see Table S1.

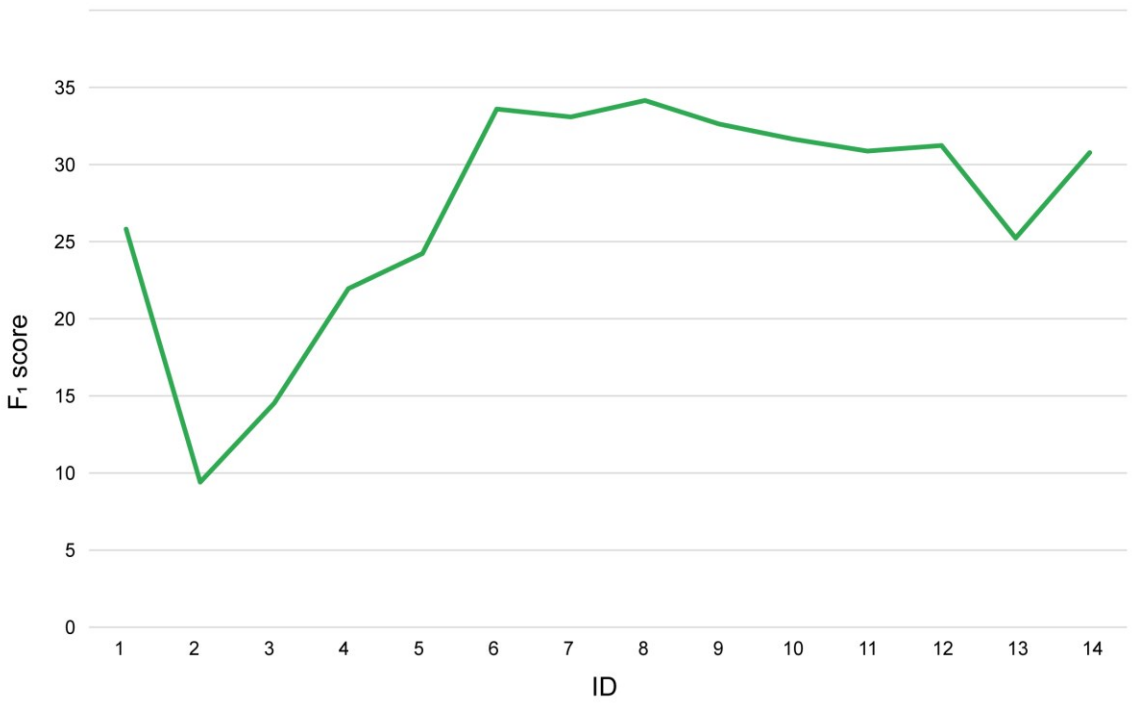
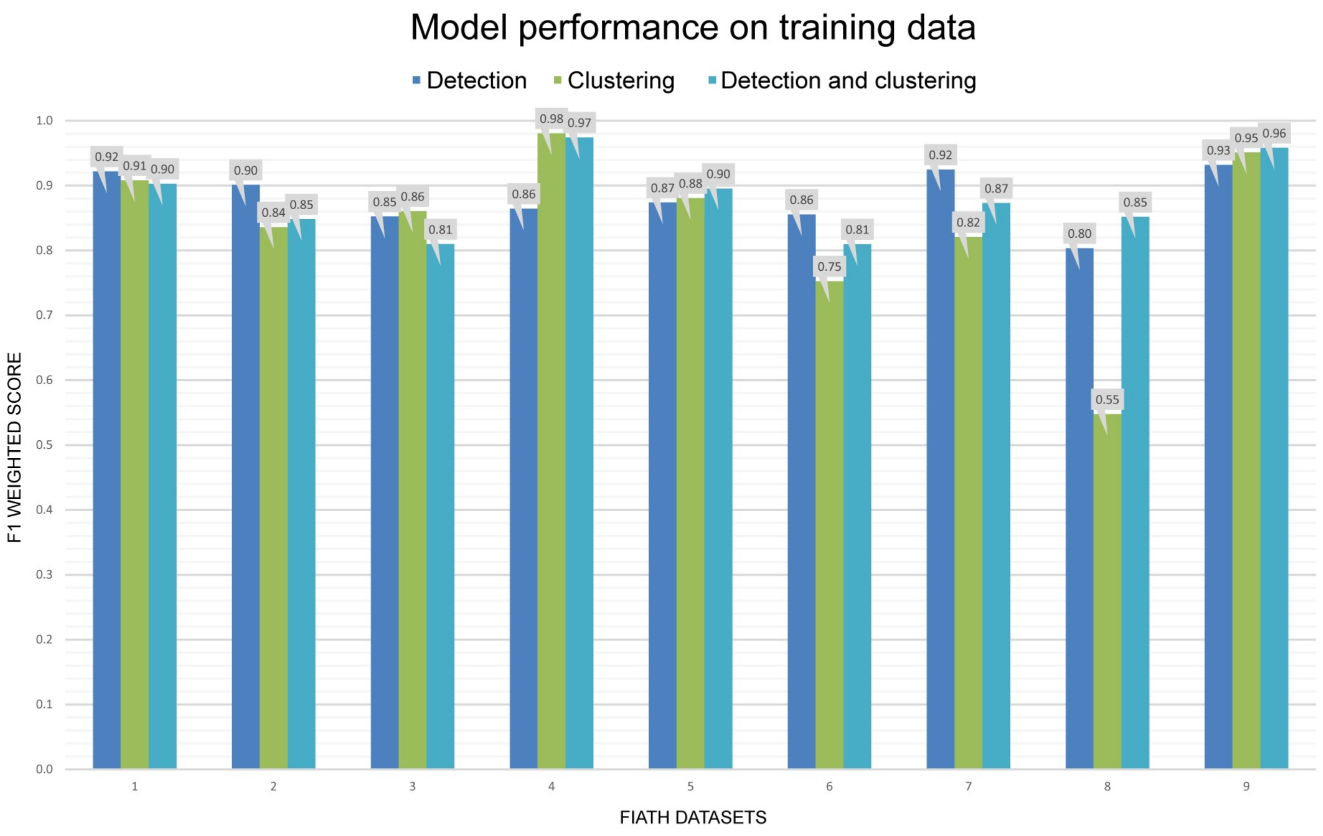


Table S1. The different reconstruction and detection loss ratios used in the k-fold cross-validations. A series of combinations were tested to find the best performing hyperparameter-pair. The reconstruction loss was taken from a larger interval: from 0 to 4096, while the detection loss was considered between 0.1 and 1.3.

ID	Reconstruction loss	Detection loss
1	4096	0.1
2	2048	0.2
3	1024	0.3
4	512	0.4
5	256	0.5
6	128	0.6
7	64	0.7
8	32	0.8
9	16	0.9
10	8	1
11	4	1.1
12	2	1.2
13	1	1.3
14	0	1

Figure S3. Training results on Fiath dataset. The performance of each model on its training data. For the three modes (detection, clustering, detection and clustering) a separate model was trained to compare the different approaches while also be able to inspect the performance drop of the model when it's dealt with the combined detection and clustering problem.


Model performance on training data
Detection
Clustering
Detection and clustering
F1 WEIGHTED SCORE
FIATH DATASETS