THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Real-Time Spike Sorting of Neural Data

by

Michelle Sands

School of Electrical Engineering and Computer Science, University of Queensland.

Submitted for the degree of Bachelor of Engineering (Honours) in the division of Software Engineering.

04/11/2024

Michelle Sands

S4397121@student.uq.edu.au

04/11/2024

Prof Michael Brünig,
Head of School
School of Electrical Engineering and Computer Science
The University of Queensland
St Lucia QLD 4072

Dear Professor Brünig,

In accordance with the requirements of the Degree of Bachelor of Engineering (Honours) in the School of Electrical Engineering and Computer Science, I submit the following thesis entitled

“Real-Time Spike Sorting of Neural Data”

The thesis was performed under the supervision of Dr Clarissa Whitmire. I declare that the work submitted in the thesis is my own, except as acknowledged in the text and footnotes, and that it has not previously been submitted for a degree at the University of Queensland or any other institution.

Yours sincerely,

[illegible]

Michelle Sands

To …

Mum, Dad, Calum, and Pam – for your limitless love & support.
Daniel, Deon, Dylan, Ruby, Sophie, and Sophia – for making sure I remembered to eat.
The Lab – Clarissa, Phill, David, Audrey, Ryan, Ethan, and Alyssa – for fostering an incredible, supportive environment and making the development of this thesis far more fun than I could ever have hoped it would be.

Acknowledgements

Thank you to the contributions of:

> Dr Clarissa Whitmire, for her expert guidance in project direction, potential methodologies, presentation refinement, and data visualisation. I’m incredibly grateful for the time I’ve spent in her lab, which has been instructive in areas far beyond the technical.
>
> Dr Phillip Wisinski-Bokiniec, for his insight, patience, and willingness to explain basic biological concepts to a software engineering undergraduate student. Background research would have been a greater challenge without his generous input.
>
> David Lucha, for his help in getting started with the Bonsai suite and his consistent provision of excellent-quality coffee.

This thesis would not be possible without the substantial contribution of many open-source libraries and frameworks. Thank you in particular to:

> Gonçalo Lopes, David Maas, Bruno Cruz, and the Champalimaud Centre for the Unknown, for the creation of the Bonsai development suite and associated documentation.
>
> Marcus Pachitariu, Sashwat Shridhar, Jacob Pennington, and Carsen Stringer for the development of Kilosort, particularly Kilosort4, which is the offline sorter used in the final processing pipeline of this project.

Generative AI (ChatGPT) was used in some stages of the project for troubleshooting purposes and in cases where documentation was insufficient for my goals, but all code and documentation in the final submission, including this thesis, is my own work.

This thesis, and the project associated with it, was developed on the lands of the Yagara and Turrbal people. Sovereignty was never ceded. Australia always was, always will be, Aboriginal land.

Abstract

This document describes the process of developing a pipeline for online spike sorting using template matching. This technology has applications in experimental neuroscience and neuroengineering. Real-time processing of dense, noisy spatiotemporal data more broadly has applications in communication and signal processing, and the algorithms considered throughout the project have been applied to machine learning in many scenarios.

The approach to the project was twofold: first, select an offline sorter and validate its performance in order to extract the required templates. Second, use these templates perform template matching on neural data as it was generated. The development of an offline sorter from scratch was considered out of scope.

The project selected Kilosort4, a template matching algorithm officially published in early 2024, for the offline sorter and found that 5 minutes of training data was sufficient to generate templates of an acceptable quality. It then tested the online pipeline, implemented in the Bonsai graphical programming suite, against four representative units: one each for high activity, low activity, high amplitude, and low amplitude. it was found that template matching was able to achieve significantly improved performance (up to 4x improvement in F1 score) for some – but not all – combinations of amplitude and template similarity threshold.

Though further testing is required to achieve acceptable confidence in the results, initial outcomes are highly encouraging. This project contributes to the area of online spike sorting in a way that is transparent and reproducible. The implementation of the online sorter in Bonsai will hopefully allow for further customisation of the process without the need for programming expertise. Likewise, the limited coupling between the online and offline sorter allows users to adopt different offline sorters depending on their specific experimental requirements without the need to redevelop the entire pipeline.

Table of Contents

List of Figures

List of Tables

Glossary of abbreviations

API – Application Programming Interface
CAR – Common Average Referencing
CNN – Convolutional Neural Network
CWT – Continuous Wavelet Transform
DBC – Density-Based Clustering
DTW – Dynamic Time Warp
DWT – Discrete Wavelet Transform
EM- Expectation Maximisation
FCM – Fuzzy C-Means
FSDE – First- And Second- Derivative Extrema
GMM-EM – Gaussian Mixture Model Expectation Maximisation
GUI – Graphical User Interface
LDA – Linear Discriminant Analysis
LFP – Local Field Potential
MEA – Multi-Electrode Array
ML – Machine Learning
MLP – Multilayer Perceptron
NEO – Nonlinear Energy Operator
NN – Neural Network
PCA – Principal Component Analysis
RNN – Relational Neural Network
SNN – Spiking Neural Network
SPC – Superparamagnetic Clustering
STDP-SNN – Spike-Timing-Dependent Plasticity Neural Network
SVD – Singular Value Decomposition
TEO – Teager Energy Operator

1 Introduction

1.1 Motivation

Understanding the processes that underlie neural function can facilitate developments in both medicine and technology. Recent demonstrative advances in the field include brain-computer interfaces and deep brain stimulation for conditions including treatment-resistant depression [1] and Parkinson’s disease. The first step to understanding neural processes is collecting and processing neural data.

Methods for this, outlined in Figure 1, are chosen for different applications based on their invasiveness, depth of penetration, and spatial and temporal fidelity.

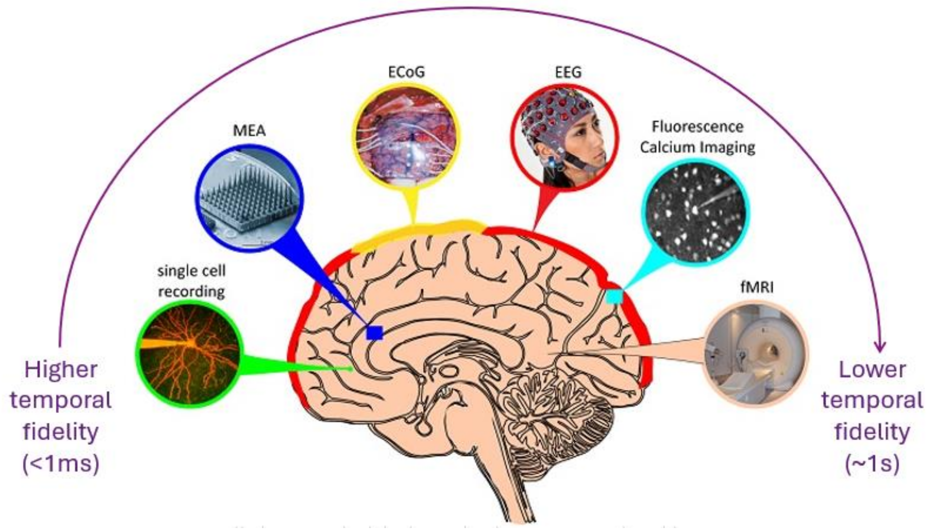

ECoG
EEG
MEA
Fluorescence Calcium Imaging
single cell recording
fMRI
Higher temporal fidelity (<1ms)
Lower temporal fidelity (~1s)

Figure 1: Methods of Neural Data Collection [2]

Electrophysiology, as used in multi-electrode arrays (MEAs) and single-cell patch clamping, takes advantage of the fact that biological cells exhibit voltage changes when active to measure cellular activity in the brain to a high level of spatial and temporal fidelity. Unlike patch clamps, electrodes are able to measure the activity of many cells at once, and can be applied to awake, performing subjects [3]. Electrodes are therefore able to provide a high level of insight across a population of neurons [4], provided the signal is appropriately processed.

Spike sorting is the process of analysing the data collected by neural electrodes to isolate the activity (‘spikes’) of individual neurons. Early versions of this process involved either time-consuming manual extraction by an expert or rudimentary computation methods like threshold crossing, in which a voltage threshold is set per channel, and all signals that pass that threshold are counted as a spike. Developments in hardware [5] leading to an increase in the number of channels each probe can support makes the former untenably time-consuming, and computation methods have grown far more sophisticated than the latter. Modern spike-sorting algorithms take advantage of some characteristics of electrophysiological activity, including spike shape variation between neurons, shown in

Figure 2, and neuronal refractory periods [6]. Other characteristics of electrophysiological data, however, such as the low signal-to-noise ratio (SNR), differences in neuron structure in different regions of the brain [7], and variations in neuron behaviour, complicate this process.

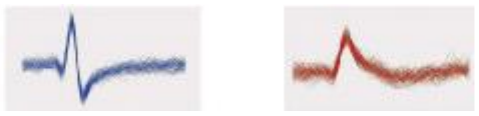


Figure 2: Differences in Spike Shape Between Neurons [8]

Many methods for automated offline spike sorting, in which the probe data is collected in its entirety before being processed, have been developed [6, 8-10]. Although some methods exist [11] to improve the accuracy of online sorters, in which data is processed as it is collected, threshold crossing remains widely used for online sorting. While this method is sufficient for some experimental applications, its accuracy is much lower than that of offline spike sorters, making it unsuitable for applications in which accuracy is important.

1.2 Statement of Purpose

This project aims to bridge the gap between online and offline spike sorters by developing an online spike sorter that achieves better accuracy than threshold crossing without sacrificing speed unnecessarily. It builds on existing work by Pachitariu et al. [6] and Lopes et al [12] for offline spike sorting and online data processing respectively. Although its intended purpose is in online spike sorting, specifically for use in closed-loop control experiments where neurons are stimulated in response to neural activity, the rapid and accurate processing of noisy data has many applications beyond the field of neuroscience.

1.3 Goals

The primary goal of creating an online spike sorter has sub goals outlined below:

1.3.1 Offline Spike Sorting

- Select or adapt an offline spike sorter that is:
  - Effective on thalamic data,
  - Compatible with lab hardware (ONIX [13] acquisition hardware and Neuropixels [5] probes),
  - Fast enough to extract templates within experimental time constraints.

1.3.2 Online Spike Sorting

- Develop an online sorter that:
  - Is compatible with lab hardware,
  - Works with less than 2.5ms latency from acquisition to detection,
  - Achieves higher accuracy than threshold crossing alone.

1.3.3 Validation and Benchmarking

- Both online and offline sorters should have their performance measured, particularly with regards to their:
  - Speed,
  - Accuracy.

2 Background

2.1 Electrophysiology Overview

Electrophysiology explores the electrical activity of living cells. Neural extracellular recording devices (typically probes) capture data on voltage, or local field potentials (LFPs), in the brain over time. This data typically comprises action potentials of nearby neurons, background noise created by the action potentials of more distant neurons, and noise from other sources [8]. Action potentials occur when external stimulus reaches a threshold voltage, causing voltage-gated ion channels in the cell membrane to open and ions to flow into and out of the cell [3]. This causes first a depolarisation and then a repolarisation of the cell potential [3]. Neuronal action potential signals differ depending on where and how they are observed: intracellular techniques can capture action potentials at a scale of ~50mV, while extracellular techniques capture spikes at a scale of ~0.5mV [14]. Neuronal action potentials are also differentiable by shape, which is determined by the neuron's morphology as well as its distance from and orientation relative to the recording device [8]. For example, a biphasic action potential signal indicates cell body activity, while a triphasic signal indicates axonal activity [14]. The relatively small amplitude of activation potentials recorded using extracellular methods means that only action potentials are detectable and sub-threshold activity, or voltage changes that do not lead to an action potential, are missed [3].

Once an action potential has occurred in a cell, that cell goes through a refractory period – typically 2.5ms – during which it will not activate again [8] as ions are 'pumped' back into the cell to restore its resting voltage [3]. Although a neuron will never violate its absolute refractory period, some types of neuron (e.g. bursting neurons) will fire at frequencies very close to that limit [8].

The properties of action potentials are used by spike-sorting algorithms to assign detected action potentials to particular computational units, and to validate those assignments. Theoretically, each unit detected by a spike sorting algorithm corresponds to a single neuron, but in practice many units comprise multi-unit activity from which individual neurons cannot be isolated. These units are often discarded. Complicating factors, such as those outlined below, mean that isolating the activity of individual neurons from the LFPs recorded by probes is a difficult task, but it is often crucial to know which action potential 'belongs' to which neuron. In the human hippocampus, for example, nearby neurons often fire in response to unrelated concepts [15], so knowing which neuron is active can be highly informative. Accurate spike sorting can also assist in identifying connectivity patterns and sparsely firing neurons, the latter of which are associated with memory processes [8, 15].

2.2 Complicating factors

Early setups for extracellular electrophysiology used single-unit electrodes and collected relatively small amounts of data, so it was feasible (if tedious) to sort or validate spikes manually [8]. Modern probes, however, have thousands of recording sites distributed over multiple shanks [5] and collect data at extremely high rates [16]. Spike sorting algorithms must therefore be capable of processing large amounts of data to a high standard. This task is made more complex by factors such as probe drift, variation in activation potential shapes and patterns across types of neurons, and overlapping spikes [8].

Once inserted, probes are likely to move up and down in the brain due to tissue relaxation and subject movement [6, 17]. This movement, or drift, can cause problems in classification if not addressed. Approaches to this problem include implementing adaptive algorithms that can handle small changes in probe location [11], and using statistical methods to model the movement of the electrode so it can be corrected [5].

Spikes with nonstandard activation shapes and patterns complicate the spike sorting process. Bursting neurons, for example, fire action potentials in clusters at high frequency which decrease in amplitude over time [8]. This behaviour may cause a unit that correlates with a bursting neuron to be mistakenly classified as non-refractory if not accounted for. The existence of nonstandard activation shapes and patterns is made more complex by the fact that activity looks different in different areas of the brain, as shown in Figure 3 [5].

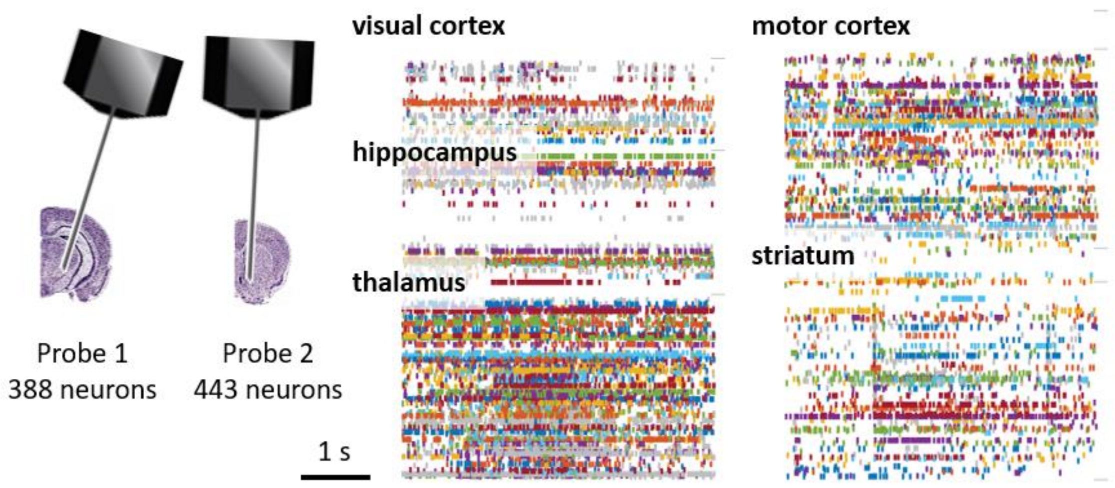


Figure 3: Neural Activity in Different Areas of the Brain [5]

Temporally overlapping spikes detected on the same electrode also present a challenge for cluster-based algorithms [8]. Many solutions to this problem, including template matching, statistical techniques, and specifically designed neural networks, involve iterative processes which require significant computational resources [11].

3 Prior Art

3.1 Spike Sorting Overview

Conventional spike sorting pipelines involve the following stages [8, 11]:

1. Preprocessing
2. Spike detection
3. Feature extraction
4. Clustering

Some algorithms do not conform to these broad steps: template-matching algorithms like Kilosort [6, 18], and neural network based models like SpikeDeeptector [19] and Yet Another Spike Sorter (YASS) [20] have also found success in recent years. Methods based on dictionary learning, basis pursuit, and independent component analysis have also been proposed [21].

Online spike sorting methods tend to follow a similar pipeline, but may use truncated versions of each step, or skip steps entirely, to improve computational complexity [11]. Many methods have been applied to each step to optimise for specific experimental applications. Factors such as computing time, performance on certain types of data, and ease of implementation are considered when choosing a pipeline, and there is currently no universal best choice for all applications. Among scientists, other factors which may be considered include compatibility with existing infrastructure (e.g. acquisition hardware, existing software pipelines), ease of access to source code, and ease of implementation and customisation. Bonsai [12], for example, is popular in the neuroscience field for its graphical implementation (shown in Section 6) and compatibility with Open Ephys [22] software and hardware.

3.2 Preprocessing

Data preprocessing aims to improve the effectiveness of later steps by reducing the impact of noise and non-neuronal activity on readings. Common preprocessing methods include:

- High- or band-pass filtering,
- Common average referencing (CAR),
- Data whitening, and
- Drift correction.

A typical pre-processing pipeline begins with running the data through a high-pass filter, typically with a cutoff of around 300Hz, to remove low-frequency noise from external sources [9]. Some pipelines [6] then implement common average referencing, subtracting the median across all recording sites at each instant of time and the mean across time for each channel. The former reduces the effect of artefacts shared across channels, and the latter reduces the impact of persistent local artefacts. Data whitening and drift correction, which reduce cross-channel correlations and adjust data to account for probe drift respectively, are also sometimes implemented [6]. Drift correction can also be implemented during processing, though the mechanism for doing so is slightly different [5, 6, 18]. Drift correction in the pre-processing stage can make the remainder of the process more robust, but if done incorrectly can corrupt the rest of the pipeline [11].

Online sorters may implement similar steps, though the specific details of implementation may differ slightly. The Bonsai DSP library, for example [12], implements filters in software by creating mini batches – buffers of 30-60 samples at a time which are filtered as they come in. Some purpose-built sorters implement hardware-based filtering [11],

3.3 Spike Detection

Spike detection methods typically implement some metric as a cutoff to separate spikes from white noise. The simplest of these methods is amplitude thresholding, where all spikes that surpass a given amplitude cutoff (typically determined by the local standard deviation) are assumed to be neuronal spikes [8]. Window discriminators work on a similar principle, defining temporal and spatial windows that a neuronal spike should cross [9, 20]. Window discriminators are still widely used, but often need to be set and reset manually [9], though some modern pipelines have made attempts to automate this process [20]. Nonlinear energy operators (NEOs) take advantage of observed changes in frequency during neuronal spikes to implement a threshold that takes signal amplitude and frequency into account [11]. Some modern pipelines use wavelet transform products to detect spikes [11] in a process roughly analogous to Fourier transforms or template matching (see section 3.6). This method, unlike the others addressed here, has the advantage of not assuming noise to be white noise, but comes at the cost of greater computational complexity [11].

3.4 Feature Extraction

Feature extraction reduces data dimensionality, with the goal of retaining features (or dimensions) that help with classification and discarding features that aren't informative [9]. If implemented correctly, this stage can improve the accuracy and decrease the computational complexity of the clustering algorithms used in the next stage. Principal component analysis (PCA) is a common method [18], and involves linearly projecting data into lower dimensions, the axes of which are chosen to maintain the most possible variance. Note that although variance and cluster separation are often analogous, they are not always directly correlated [11]. Singular value decomposition (SVD), which involves using matrix transformations to identify relationships within the data, is also often used [6]. Other methods include wavelet transforms which, as with wavelet transform products, are analogous to Fourier transforms; geometric- and derivative-based criteria, which have a lower computational complexity and are therefore preferred for online and on-site analysis; salient feature analysis, which aims to maximise discrimination between spike classes using normalised distances between classes as defined by their standard deviation, mean, and relative probability; and linear discriminant analysis (LDA), a linear projection technique that aims to minimise intra-class variance and maximise inter-class variance [11]. Some pipelines combine these methods using fuzzy logic or adaptive weighting to achieve better outcomes, though this results in increased computing cost [11].

3.5 Clustering

Clustering aims to group detected spikes based on extracted features to identify units, with the aim that each unit represents a single neuron's activity. By their nature, clustering methods require the data to be pre-collected, so must often be adapted or replaced in online applications.

K-means is a common initial clustering method, grouping data into k groups spread approximately evenly across space. When used alone, this method requires knowing the precise number of units to be identified, so it is commonly combined with other methods and group splitting and merging pipelines [6]. Superparamagnetic clustering (SPC), which involves simulating 'magnetic' attraction between points so that they form clusters [23], and expectation maximisation (EM) algorithms, typically involving Gaussian mixture models [11] are also often used [9]. EM algorithms, along with other algorithms like fuzzy C-means, provide probabilistic clustering, where each point is assigned a probability of belonging to each cluster rather than having a single binary assignment [11]. This can improve reliability but typically requires more computation. Another common method is density peak clustering [18] which uses density and distance metrics to assign clusters [24]. Some algorithms [6] use graph-based iterative neighbour clustering, where first graphs are constructed of neighbouring points, then points are assigned clusters based on their neighbours' assignments. An overview of the clustering methods investigated as a part of this project is shown in Table 1.

Table 1: Overview of Clustering Methods Used in Spike Sorting

Name	**Description**	**Ref**
K-Means	Split data into k groups: iteratively update the centroid of each cluster to be the mean of all points assigned to it. Handles new points reasonably well, but K must be manually set.	[11]
Superparamagnetic Clustering	Simulates 'magnetic' clustering between points. Does not handle new points particularly well.	
Expectation Maximisation	Iteratively find local maximum likelihood. Other methods like Gaussian Mixture Models can be applied.	
Fuzzy C-Means	Like K-means but with probability instead of a single assignment. More accurate, but higher computational complexity.	
Graph-Based Iterative Neighbour Clustering	Treats points as nodes on a graph and clusters based on the cluster assignment of a point's neighbours.	[25]
Manual Cluster Cutting	Experts use a visual interface to manually separate clusters.	[21]
Hierarchical Clustering	Starts with each datapoint in its own cluster, then repeatedly merges clusters until all points belong to the same cluster.	
Density-Based Clustering	Groups points in high-density areas together. Handles non-spherical data well.	
Valley-Seeking	Finds inter-cluster boundaries in low-density areas ("valleys").	
Gray Relational Analysis	Uses gray relational analysis to measure the similarity of points and groups similar points.	
Manual Amplitude Window	Experts manually set amplitude windows. If a spike falls within an amplitude window, it is assigned to that cluster.	
Gaussian Mixture Models	Uses a weighted sum of Gaussian distributions to model probability density functions for clusters.	
Watershed Algorithm	Typically applied to image segmentation. Treats brightness / amplitude as elevation, floods areas to segment.	

3.6 Template Matching

Template matching is an unconventional spike sorting method that involves matching data to a set of precomputed templates. First, data is matched to universal templates with arbitrary amplitude scaling, then the centroids of the clusters found with these matches are used to find learned templates. The contributions of these templates to overall waveforms are then iteratively subtracted via matching pursuit, which helps to isolate individual neuronal activity from overlapping spikes [6]. Templates can be precomputed, making this process fairly computationally efficient despite the iterative nature of matching pursuit; however, assumptions are necessarily made about the characteristics of the data, which may or may not apply to all neurological recordings.

Template matching is well-suited to online applications as it bypasses clustering algorithms' need for data to be pre-collected. Kilosort [6] uses template deconvolution to extract templates from the data, then graph-based clustering to find unit assignments offline [25]. Osort [27], an online template-matching algorithm, uses Euclidean distance to assess detected spikes' similarities to precomputed templates. Some other versions of Osort have been adapted to use metrics like correlation coefficient [28] to improve processing speed. Measures of similarity are not limited to the realm of spike sorting: Bandyopadhyay & Saha [29] outline angular separation (or cosine similarity), as well as Minkowski, Manhattan, Chebyshev, Canberra, Bray-Curtis, and Mahalnobis distance as alternate methods.

3.7 Neural Networks

Neural networks (NNs) have been applied to various stages of the spike-sorting pipeline, but rarely to the entire pipeline [11]. Yet Another Spike Sorter (YASS) uses a neural network in the spike extraction stage to extract clean – or non-overlapping – waveforms [20]. Convolutional, relational, and artificial neural networks have all been applied to various stages of the process, and spiking neural networks take advantage of the structural similarities between the biological neurons being observed and the artificial neurons in the network to reduce computational overhead [11]. The data-based approach used by neural networks has been shown to be effective, provided data is available [20]. Obtaining appropriate data to act as a ground truth remains an open problem in the spike-sorting domain, linked to issues both in training NN-based methods and in validating the outputs of spike-sorting algorithms.

3.8 Validation and Benchmarking

Output validation remains an open problem in spike sorting [8, 11]. With smaller datasets, spike sorter outputs could be done manually, but with the larger amounts of data generated by probes today this is no longer feasible [21]. Agreement rates for identified units are rarely 100% even using manual expert curation, and agreement between spike sorting algorithms is even lower [30], bringing into question the accuracy of these algorithms.

Ideally, datasets used for algorithmic validation have a known ground truth and algorithms can be evaluated based on how well their outputs match that ground truth. In practice, ground truth data where the activity of units is already known is difficult to obtain, requiring simultaneous intra- and extra-cellular recordings [21]. Datasets of this kind are limited both in number and scope – laboratories often do not have the time or resources to gather their own, and current methods for intracellular recording do not allow for recording of a full population, or even a meaningful sample. Synthetic or hybrid data, where data is generated from some combination of known neuronal activity and noise (which is either collated from data or generated randomly), has been used in some pipelines [6, 21], but this benchmarking method varies across labs and is often not reproducible [21]. It also necessarily makes assumptions about how data will look, which may or may not be applicable. Additionally, if synthetic data is generated by the same organisation making the algorithm, assumptions made in the algorithm are likely to be repeated in the synthetic data, artificially inflating benchmarked performance. Table 2 shows an overview of methods for generating datasets with known ground truths, along with their benefits and drawbacks

Table 2: Methods for Generating Datasets with Known Ground Truth

Method	**Benefits**	**Drawbacks**
Paired intra- / extra- cellular recordings	• Highly accurate for target neurons	• Resource-intensive • Few neurons can be monitored at once • Highly invasive
Simulated recordings	• Easy to generate	• May not accurately recreate real-world recordings • Reflect and reinforce creator assumptions • Often not transparent in how they were generated
Hybrid recordings	• Easy to generate • More likely than purely synthetic data to be true to real-world recordings	• Reflect and reinforce creator assumptions • Assumes initial classification of data is accurate • Often not transparent in how they were generated
Expert-curated ground truth recordings	• Experts can choose to be more or less sensitive based on experimental requirements	• Time- and cost-intensive to generate • Sensitive to the biases of human experts • Low agreement even among the same expert on different days

Many pipelines use biomechanical methods to evaluate data quality, taking advantage of known characteristics of action potentials – particularly the neuronal refractory period – to confirm the quality of outputs. Auto-correlograms can be used to identify non-refractory units, and cross-correlograms can help to identify oversplit clusters [6, 21]. Known information about the behaviour of specific areas of the brain can also be used for validation – place fields in hippocampal neurons, for example, can be used to help establish a ground truth for hippocampal data [10, 21].

Statistical methods have also been adopted to evaluate data quality [31]. Intracluster variance and cluster similarity measurements like the Rand and Jaccard indexes are often used to evaluate the quality of cluster separations [11]. Pairs of clusters can be confirmed as well-separated if their projection onto the regression axis of both clusters is bimodal [6]. Some pipelines also use other clustering algorithms to evaluate the effectiveness of their own clusters – D-prime (based on LDA) and nearest-neighbours metrics have both been used to assess the quality of output data [30]. Table 3 shows an overview of some statistical methods that have been applied to spike sorting. Note that many overall measures of cluster quality are not well suited to spike sorting, as data density varies significantly – per-cluster quality measures are typically the better choice.

Table 3: Methods for Spike Sorter Validation

Method	**Measures**	**Reference**
Signal-noise ratio (SNR)	Data quality Measures the power of the signal (spikes) vs noise (LFP etc.)	[30]
Presence ratio	Data quality Measures the proportion of discrete time bins in which at least one spike occurs	
Maximum & cumulative drift	Data quality Measures probe drift	
Isolation distance	Unit quality Measures the smallest ellipsoid that contains all points from a cluster and an equal number of points from other clusters	
L-ratio	Unit quality Measures average value of tail of distribution	
D-prime	Unit quality Measures classification accuracy, based on linear discrimination analysis (LDA)	
Nearest Neighbours	Unit quality Non-parametric estimation of unit contamination	
Silhouette score	Unit quality Measures similarity within clusters and separation between clusters	
Stability score	Sorter stability Measures how well a sorter reproduces its results if given the same data	[31]
Cross-validation	Sorter quality Measures how well a sorter is likely to perform on unseen data – prevents overfitting of given data	
Self-blurring	Unit quality Detects when cluster boundaries cross high-density areas by convolving classifications in an area around boundary lines	
Noise reversal	Unit quality Detects when cluster boundaries cross high-density areas by mirroring noise in the direction of the decision boundary	
Parameter sensitivity	Sorter stability Involves rerunning a sorter with slightly changed hyperparameters to test sensitivity to poorly chosen parameters	

Because most spike-sorting algorithms validate using some non-standardised combination of these methods, there has been a push in recent years to standardise benchmarking methods to better compare performance [21, 30]. SpikeForest [21] implemented a leaderboard-style set of benchmarks in 2020 which – perhaps unsurprisingly – found no clear victor in spike-sorting algorithms across all possible applications. SpikeInterface built on this work to propose a standardised framework for algorithm implementation and validation, with the goal of making it easier for laboratories to test supported algorithms and choose the best for their specific needs [30]. It has been found that in applications where computational complexity is not a concern, validating units through model agreement is extremely effective in improving the accuracy of units, though this method is currently infeasible for online applications without access to large amounts of computing power [30].

3.9 Online Processing

For some applications, like the development of neural prosthetics and brain-machine interfaces, spike sorting must be done in real time, and sometimes also on implantable hardware [11]. It is therefore often important to measure the performance of spike-sorting algorithms not only by accuracy, but also by computational complexity.

Some reviews have been conducted in this area [11], but as with algorithm selection and validation, the specific balance of performance and computational complexity required is highly dependent on the requirements of the application. By far the most common method for online spike sorting is threshold crossing [11], though threshold metrics have developed, as outlined in section 3.2.

Although Kilosort4 is purportedly fast enough to run online [25], it remains a batch processing algorithm, requiring all data to be available ahead of time. Osort has online processing built in [27], but uses the computationally complex metric of Euclidean distance to assess template similarity [27]. Various packages in the Bonsai suite [12] support real-time processing and threshold-based spike detection, but none yet exist that can perform fast, verifiable template-matching out of the box.

4 Methodology

4.1 Project Approach

The overarching approach to this project involved several key phases, adapted from the standard approach to software engineering as outlined by McConnell [32] and shown in Figure 4Figure 1.

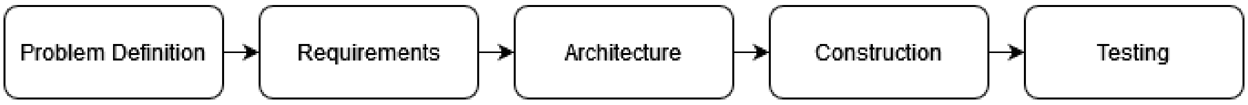


Figure 4: Standard Approach to Software Engineering

Because interfacing between the online and offline sorter was limited to the single point of contact of template transfer (i.e. templates generated by the offline sorter had to be readable by the online sorter), it was more or less possible to treat both sorters as ‘black boxes’ with regards to one another. This led to a large amount of freedom when choosing methodologies for both: the algorithm chosen for offline spike sorting did not need to be capable of ad-hoc spike sorting, which allowed algorithms that implemented methods like superparamagnetic clustering to be considered. It also allowed a parallel development approach, where the development pipeline for each section of the project could be split, and worked on asynchronously if necessary, as shown in Figure 5.

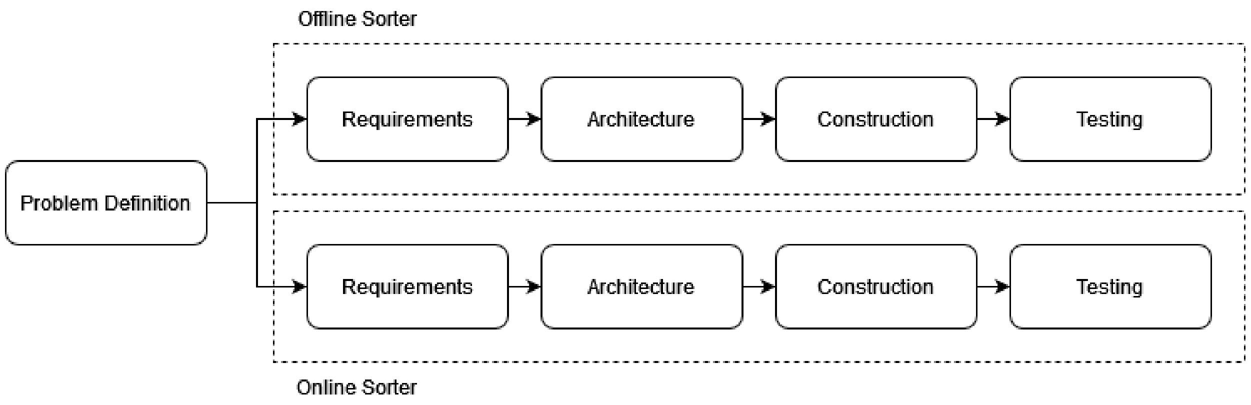


Figure 5: Adapted Project Approach

This interface-driven approach allowed focus on functionality first: once the format of the templates was agreed on, the two sorters could ‘communicate’ without any need for further wrapper functions or GUIs. This is fortunate because – as outlined in Section 6 – the APIs of some existing infrastructure did not allow for full programmatic execution, meaning that attempts for full pipeline integration were by necessity left for further work.

4.2 Project Timeline

Table 4 shows the final project timeline. Stretch goals were removed from the timeline initially presented in the project proposal, to allow for more time implementing and validating core features.

Table 4: Project Timeline

Type	Milestone	Start Date	Due Date
Goal	Initial research	19/02	21/03
Goal	Initial project planning	11/03	21/03
Goal	Write proposal draft	11/03	21/03
Assessment	**Proposal draft**	**21/03**	
Goal	Literature review • Characteristics of thalamic data • Existing spike sorting algorithms • Validation methods • Focus on computational efficiency • Focus on applicability to thalamic data	21/03	18/04
Calendar	*Midsemester break*	*29/03*	*08/04*
Goal	Implement feedback from draft literature review	11/04	18/04
Assessment	**Project proposal**		**18/04**
Goal	Testing data obtained • Supervised for benchmarking • Unsupervised for actual implementation	18/04	20/04
Goal	Decide on benchmarking methods	11/04	22/04
Goal	Set up benchmarking pipeline and test on labelled data	22/04	10/05
Goal	Prepare for progress seminar	01/05	10/05
Assessment	**Progress seminar**		**10/05**
Goal	Choose an offline pipeline	05/05	12/05
Goal	Adapt offline pipeline to Whitmire lab requirements	12/05	19/05
Goal	Implement pipeline on unlabelled data and validate resultss	17/05	25/05
Goal	Establish online methodology	25/05	29/07
Calendar	*Midyear break*	*15/06*	*22/07*
Goal	Online implementation	30/07	12/08
Goal	Evaluation and validation of full pipeline	12/08	30/09
Calendar	*Midsemester break*	*21/09*	*30/09*
Goal	Prepare poster and demonstration	30/09	18/10
Assessment	**Poster & demonstration**		**18/10**
Goal	Finish writing thesis	18/10	04/11
Assessment	**Final submission**		**04/11**

4.3 Ethical Considerations

Best efforts were made to ensure that all data used in this project was:

- Collected through experiments that had the approval of appropriate ethical and regulatory bodies, and
- Shared with permission from its creators.

4.4 Final Data Pipeline

It was decided that, as the only online method with the capacity to differentiate between units on the same channel, template matching would be the best choice for an online spike sorter. This meant that templates would need to be provided to the online sorter. This requirement led to the development of the final data pipeline, outlined in Figure 6:

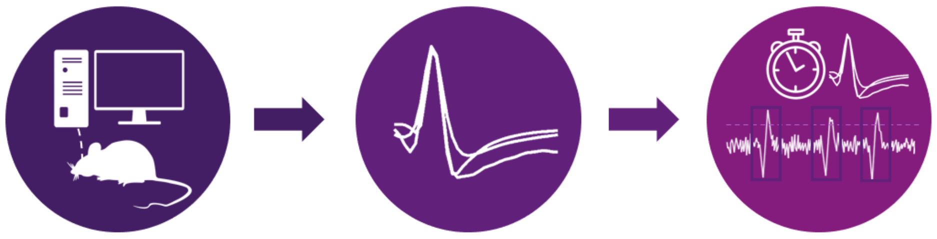


Figure 6: Graphical Representation of Final Pipeline

Essentially, the final data pipeline involved three stages:

1. Collection of data, either in real time from hardware or artificially generated from a pre-existing recording,
2. Offline template extraction, in which an offline sorter (in this case Kilosort4), extracts templates from collected data,
3. Online data processing, in which data is preprocessed before spikes are detected and compared to extracted templates.

Sections 5.2and 6.2 outline the methodology for each stage of the pipeline in further detail. A basic GUI for the full pipeline was developed in tkinter to improve the flow of execution during experiments and make the execution of the pipeline easier for non-programmers. Figure 7 shows this final GUI.

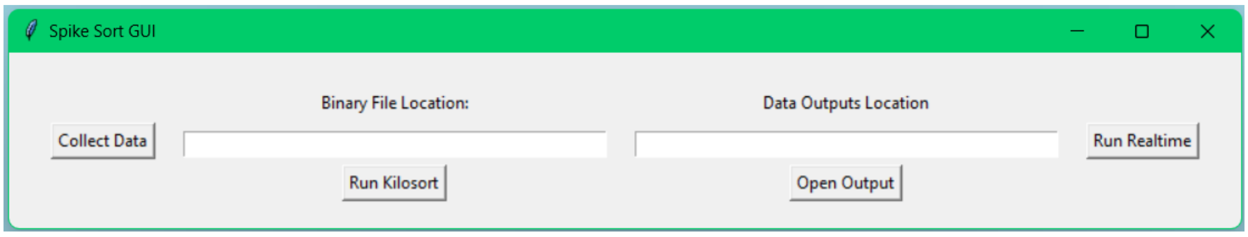


Figure 7: Pipeline GUI

The full codebase for the project can be found at github.com/oldhorizons/REIT4841. The forked code for Kilosort4 can be found at github.com/oldhorizons/Kilosort.

5 Study 1: Offline Sorter Selection

5.1 Aims

Because the creation of a bespoke offline spike sorting algorithm was outside the scope of this project, an algorithm needed to be selected from existing work in the field. The aim of this study was to select the sorter which best met the requirements of the lab, then implement it on lab data and tune any hyperparameters to suit the lab's specific experimental requirements. Table 5 shows an overview of these requirements, along with their reasoning.

Table 5: Offline Sorter Requirements

Requirement	**Description**	**Reasoning**
Speed	The chosen sorter must run within a ~45-minute experiment window, which includes the time required to: • Collect training data, • Run the sorter, • Select target units, and • Run the final online pipeline.	Experiments have time constraints related to experimenter and subject availability. 45 minutes was chosen as the target running time per the recommendation of the experimenter.
Accuracy	The chosen sorter must achieve accuracy comparable with other state-of-the-art sorters.	The online sorter requires templates to be provided. High quality offline sorting, leading to high-quality templates, sets the foundation for the rest of the pipeline.
Lab compatibility	The chosen sorter must be compatible with the lab's existing hardware and software environment: • The Open Ephys ONIX system [13] • Neuropixels probes [5] • The Bonsai graphical programming suite [12] • Windows 11	Any sorter not compatible with the lab's hardware cannot be used by the lab and is therefore not fit for purpose.
Multi-channel capability	The chosen sorter must be able to handle the information density generated by the lab's acquisition hardware: • 384 channels per probe, • Sampled at 30kHz.	Aside from the issue of compatibility with the lab hardware, single-channel sorters are more sensitive to probe drift (i.e. if a unit moves away from the channel it stops being detected at all). Some experiments also require monitoring of multiple channels and units at once, so a sorter without this capability is not fit for purpose.
Verifiability	The chosen sorter should have some verification method built in or have outputs compatible with existing verification platforms like SpikeInterface [30].	Without some validation metric able to be run by the lab, sorter accuracy cannot be assessed. Built-in validation metrics remove the need to build a bespoke validator, which saves time in the early stages of the project while keeping the option open to custom-build validators if time permits.

Requirement	**Description**	**Reasoning**
Drift & Overlap handling	Sorters were given preference if they were able to handle probe drift and spatiotemporally overlapping spikes from different units.	Well-implemented drift and overlap handling improve sorter accuracy. They were evaluated separately from accuracy to account for biases in test data for sorters which do not implement these features (i.e. if a sorter does not account for probe drift, it may be tested using data without drift).
Language familiarity	Sorters were given preference if they were written in a familiar language (e.g. Python, C, C#, C++) rather than an unfamiliar one.	Because the project had a strict time limit, becoming fluent in a new programming language was determined to add unnecessary complexity to the project. This was not a strict requirement, however, and standout sorters written in unfamiliar languages were still considered.
Open source	The chosen sorter should be open-source. Failing that, if a stand-out sorter is identified which is not open source, it should have its methodologies thoroughly outlined in a peer-reviewed paper.	Although open-source projects often carry complications with regards to documentation, they have the benefits of being: • Easily modified if needed, • Transparent in their methodologies, • Free to use and distribute.
Implementable without additional hardware purchase	Sorters which required the purchase of additional hardware (e.g. FPGAs) were not considered.	Adding hardware to the lab's existing setup would be costly and unnecessarily complicate any future changes.
Supported	Sorters were evaluated based on their documentation quality, repository activity, paper citations, and compatibility with existing frameworks like SpikeInterface, with preference given to sorters which scored well in these areas.	Poorly documented code, particularly in weakly typed languages, causes significant delays in development. In some cases, an active developer community and userbase can make up for poor documentation thanks to forum activity. Libraries which are not under active development and lack documentation and users are unlikely to be worth the trouble it takes to get them functioning.

5.2 Methods

5.2.1 Initial Search

First, a broad search was made of existing sorters, which were evaluated using a subset of the requirements outlined in 5.1. Appendix A.1: Full Offline Sorter Evaluations shows the initial search.

After the initial broad literature review, exclusion criteria were set. Sorters were excluded if they:

- Were not open-source,
- Had not been updated in over two years,
- Lacked documentation,
- Did not have sufficient activity to suggest support would be available ('sufficient activity' was defined as 200 combined paper citations and repository activity)

Sorters written in Matlab were also excluded, with the intention to re-include them if no Python-based sorters met the lab's needs. Applying these exclusion criteria left the following shortlist of sorters:

- Kilosort4,
- Mountainsort,
- SpyKING Circus.

These three sorters were then implemented on dummy data on a PC with specifications outlined in Table 6, and their time to run was informally tracked.

Table 6: Specifications for the Offline Benchmarking Computer

Component	**Specification**
CPU	Intel Core i5-4440 CPU @ 3.10GHz
GPU	ASUS Dual GTX1070-O8G
RAM	4 x 8GB DDR3-1600 PC3-12800 @ 12800MB/s

It should be noted that in actual experimental applications, pipelines will be implemented on a PC from the lab. The current lab setup has a PC with specs outlined in Table 7.

Table 7: Lab Computer Specifications

Component	**Specification**
CPU	Intel Core i7-12700F @ 2.10GHz
GPU	NVIDIA GeForce GTX 1650
RAM	4 x 8GB DDR3-1600 PC3-12800 @ 12800MB/s

Following this initial experiment, Kilosort4 was chosen as the best option, achieving high accuracy both by its own metrics and the metrics of standardization projects like SpikeForest [21] and SpikeInterface [30], and running in a similar timeframe to the other two sorters. Additional points in favour of Kilosort included its highly active development cycle and userbase, purported compatibility with wrapper libraries like SpikeInterface, and near out-of-the-box compatibility with the lab's existing data.

5.2.2 Output Validation

Next, synthetic data with known ground truth was run through the sorter to ensure its accuracy could be verified. The dataset used was a hybrid simulation of cortical data over 384 channels and 45 minutes, generated using a low-drift Neuropixels probe recording from the IBL dataset [33]. To validate the outputs, Kilosort's built-in validation function was used, which maps detected units to their nearest ground truth unit, then collates all a ground truth unit's associated detected unit to provide 3 major outputs:

- fPos: the proportion of timestamps incorrectly identified as spikes belonging to the ground truth cluster
- fMiss: the proportion of ground truth spike times missed by the sorter
- fMax: 1 – (fPos + fMax)

It also, for each ground truth unit, provides a list of the twenty best matching detected units. It is worth noting that the validation algorithm does not provide a strict 1:1 mapping of ground truth units. This means the false positive ratio is likely to be slightly deflated, as multiple detected spikes can be mapped to the same ground truth unit and denoted a true positive.

Minor changes – mostly bug fixes – were made to the built-in validation and benchmarking file to allow it to work with the provided data. A link to the forked repository is shown in Appendix B.2.

5.2.3 Hyperparameter Tuning

Ground truth data and outputs were then cropped, and sorter performance with different training times was evaluated empirically to find the optimal balance between training and testing time. Sorter runtime was also formally tracked using Python's built-in logging functions, to be accounted for in experimental applications.

Metrics chosen for evaluation were:

- Total time required to collect data and run the sorter,
- Sorter accuracy, and
- Template quality.

These metrics were found for each ground truth unit. Median unit activity (number of spikes over the course of the recording) and peak-peak amplitude were found. Performance was then assessed for all units, and for subsets of units that fell above and below these thresholds. Template quality was assessed by finding the similarity between ground truth templates and those created by the sorter. The similarity measures used were cosine and dynamic time warp (DTW) similarity – the two contenders for similarity measure in the initial implementation of the final online pipeline.

5.2.4 Final Offline Pipeline

Kilosort4 is fed binary data in a .bin or .dat file of 16-bit signed integers in C-order indexing (in which the last index changes fastest – in this case, data from all channels at time t=0 is written, then data from all channels at t=1, and so on). The Kilosort4 pipeline, described in detail in the original Kilosort4 paper [25], splits the data into batches, then follows the following steps:

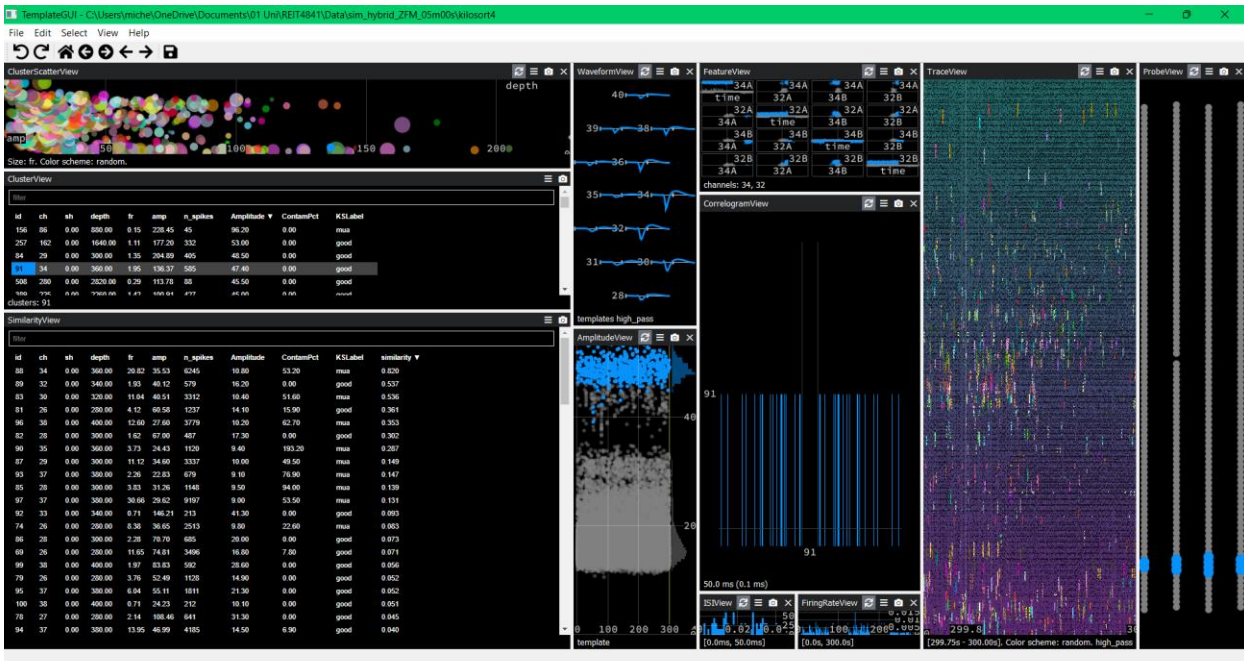
1. Preprocessing
    a. Common average referencing: remove first the mean across time, then the median across channels for each batch.
    b. Temporal filtering: a high-pass finite impulse response (FIR) filter at 300Hz is applied to each channel. The FIR filter emulates a 2nd-order Butterworth filter but has a faster GPU implementation.
    c. Channel whitening: a whitening vector is estimated for each channel using the zero-phase component (ZCA) transform for it and its nearest 32 neighbours. This is a linear operation.
    d. Drift correction: drift is detected in advance using rudimentary spike detection and linear interpolation, then data is aligned using a Gaussian kriging kernel. Data alignment is also a linear operation, so is combined with channel whitening to reduce computation time.
2. Template deconvolution
    a. Spikes are detected using template matching with arbitrary magnitude scaling
    b. Clustering is performed on detected spikes and default templates to learn spike templates
    c. Spikes are detected using template matching a second time, this time without arbitrary magnitude scaling, in an iterative/subtractive process called matching pursuit
    d. Principal components are extracted with background subtraction
3. Graph-based clustering
    a. Neighbour finding using an optimised brute-force algorithm
    b. Iterative neighbour assignment optimising for a modularity cost function:

$$\mathcal{H} = \frac{1}{2m}\sum_{c}\left(e_c - \gamma\frac{K_c^2}{2m}\right)$$ [25]

    c. Hierarchical merging if two points deemed spatially similar have a non-bimodal projection onto the regression axis between the two clusters or a highly refractory cross-correllogram.

By default, Kilosort outputs a set of .npy files that can be used for data visualisation and validation. In the validation pipeline, these outputs are then fed into the Kilosort4 validation library, methods for which are briefly outlined in section 5.2.2.

In the experimental pipeline, spike templates in .npy format are converted to .csv, to allow reading by arbitrary programs. The experimenter then uses the Phy GUI [34], shown in Figure 8, to select units to track with the online sorter. The unit numbers are passed into the online sorter, which is described in detail in section 6.


TemplateGUI - C:\Users\miche\OneDrive\Documents\01 Uni\REIT4841\Data\sim_hybrid_ZFM_05m00s\kilosort4

Size: fr. Color scheme: random.

ClusterView

id	ch	sh	depth	fr	amp	n_spikes	Amplitude ▼	ContamPct	KSLabel
156	86	0.00	880.00	0.15	228.45	45	96.20	0.00	mua
257	162	0.00	1640.00	1.11	177.20	332	53.00	0.00	good
84	29	0.00	300.00	1.35	204.89	405	48.50	0.00	good
91	34	0.00	360.00	1.95	136.37	585	47.40	0.00	good
508	280	0.00	2820.00	0.29	113.78	88	45.50	0.00	good

clusters: 91

SimilarityView

id	ch	sh	depth	fr	amp	n_spikes	Amplitude	ContamPct	KSLabel	similarity ▼
88	34	0.00	360.00	20.82	35.53	6245	10.80	53.20	mua	0.820
89	32	0.00	340.00	1.93	40.12	579	16.20	0.00	good	0.537
83	30	0.00	320.00	11.04	40.51	3312	10.40	51.60	mua	0.536
81	26	0.00	280.00	4.12	60.58	1237	14.10	15.90	good	0.361
96	38	0.00	400.00	12.60	27.60	3779	10.20	62.70	mua	0.353
82	28	0.00	300.00	1.62	67.00	487	17.30	0.00	good	0.302
90	35	0.00	360.00	3.73	24.43	1120	9.40	193.20	mua	0.287
87	29	0.00	300.00	11.12	34.60	3337	10.00	49.50	mua	0.149
93	37	0.00	380.00	2.26	22.83	679	9.10	76.90	mua	0.147
85	28	0.00	300.00	3.83	31.26	1148	9.50	94.00	mua	0.139
97	37	0.00	380.00	30.66	29.62	9197	9.00	53.50	mua	0.131
92	33	0.00	340.00	0.71	146.21	213	41.30	0.00	good	0.093
74	26	0.00	280.00	8.38	36.65	2513	9.80	22.60	mua	0.083
86	28	0.00	300.00	2.28	70.70	685	20.00	0.00	good	0.073
69	26	0.00	280.00	11.65	74.81	3496	16.80	7.80	good	0.071
99	38	0.00	400.00	1.97	83.83	592	28.60	0.00	good	0.056
79	26	0.00	280.00	3.76	52.49	1128	14.90	0.00	good	0.052
95	37	0.00	380.00	6.04	55.11	1811	21.30	0.00	good	0.052
100	38	0.00	400.00	0.71	24.23	212	10.10	0.00	good	0.051
78	27	0.00	280.00	2.14	108.46	641	31.30	0.00	good	0.045
94	37	0.00	380.00	13.95	46.99	4185	14.50	6.90	good	0.040

channels: 34, 32

50.0 ms (0.1 ms)

[0.0ms, 50.0ms]

[0.0s, 300.0s]

[299.75s - 300.00s]. Color scheme: random. high_pass

Figure 8: Example of the Phy GUI

5.3 Results

5.3.1 Time Taken

Time taken to sort the spikes offline was measured by logging the start and end times of the sorter and is shown in Figure 9. The precise measurements for time can be found in Appendix A.2: Study 1 Additional Tables.

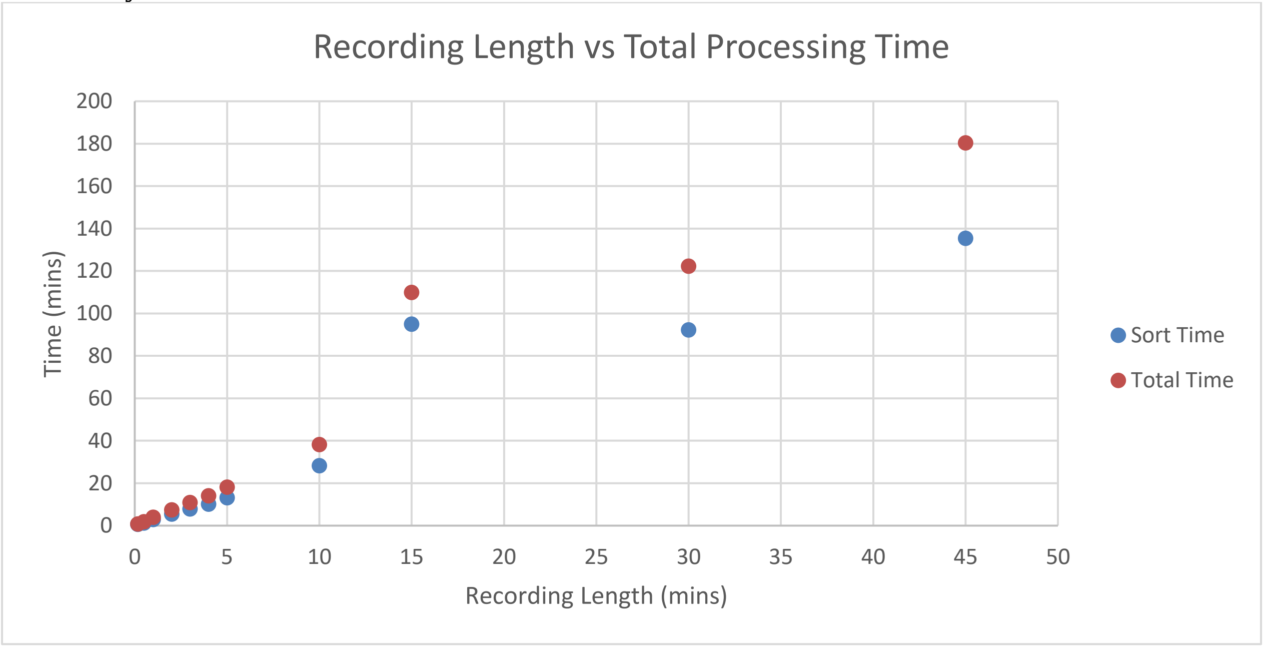


Figure 9: Total Processing Time vs Recording Length

The processing time for Kilosort4 on a computer with specifications shown in Table 6 increases the total time of stage 1 of the experimental pipeline by approximately 4 times. The roughly linear trend of the data is disrupted by the 15-minute recording. Although every effort was made to ensure an equitable testing environment, some scheduled background process may have run as the sorter was running, significantly slowing its processing time.

5.3.2 Accuracy and Template Quality

Figure 10 shows the accuracy of the sorter at different training times, measured using Kilosort's built-in validation function (described in Section 5.2.2).

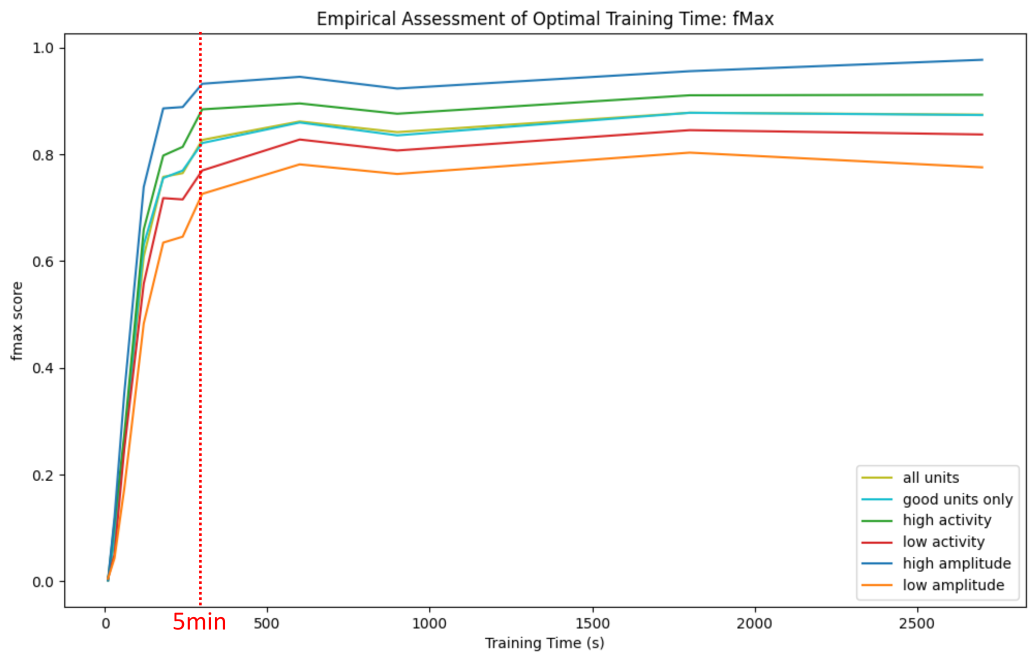


Figure 10: Kilosort4 fMax vs Training Time

Accuracy appears to plateau at the 5-minute mark for all units, indicating that increasing training time beyond 5 minutes may improve sorter accuracy, but by comparatively little. High-amplitude and high-activity units achieve higher accuracy than low-amplitude and low-activity units. The difference in performance between high- and low-amplitude units is greater than the difference between high- and low- activity units. Considering SNR is a major complication in the field of spike sorting, it makes intuitive sense that low-amplitude units would perform worst out of the unit types tested.

Figure 11 and Figure 12 show, respectively, the cosine and DTW similarity between ground truth units and their detected best matches. Cosine similarity appears to plateau in performance at approximately 5 minutes' recording time. DTW, however, reaches a plateau at approximately 3 minutes, and achieves very little increase in performance beyond that. This may indicate that DTW, while good for some applications, is unsuited for spike sorting applications as it struggles to differentiate between shapes once past a certain threshold of similarity.

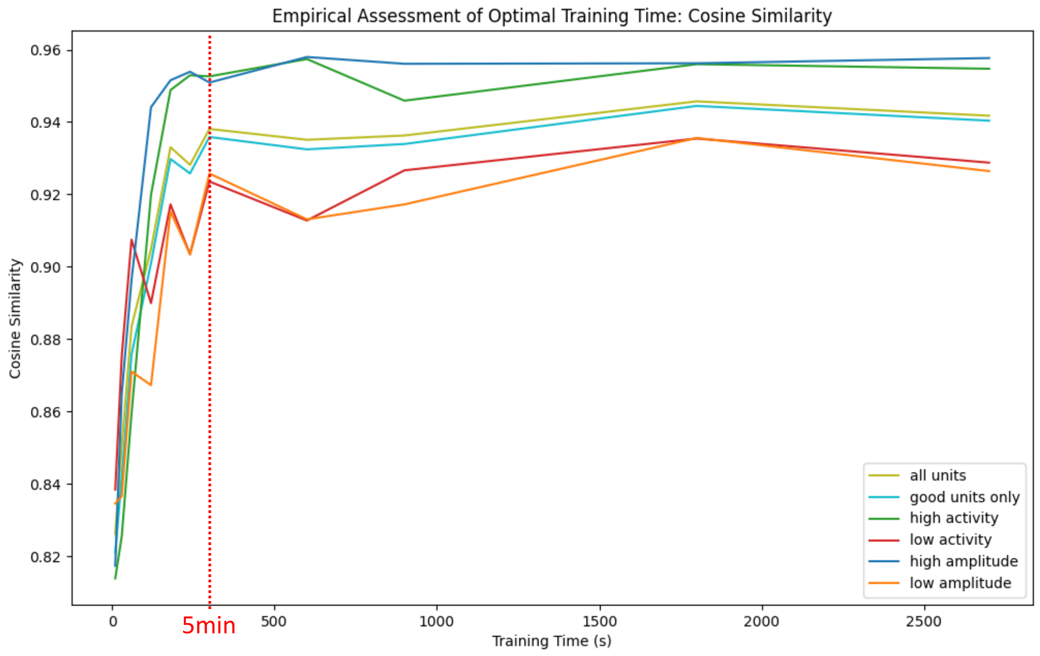


Figure 11: Template Cosine Similarity vs Training Time

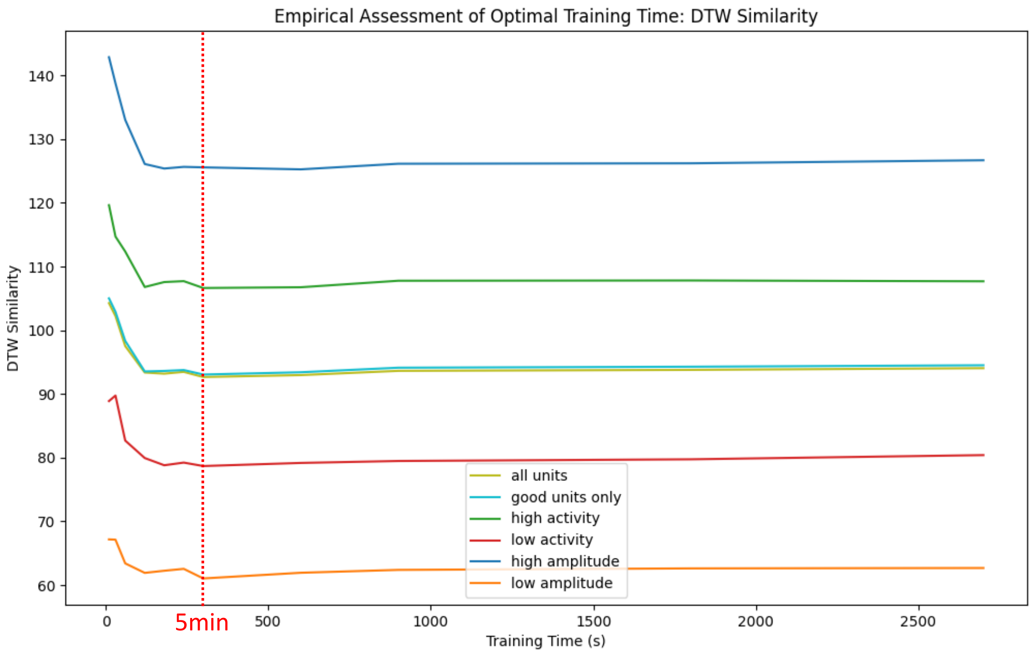


Figure 12: DTW Similarity vs Training Time

5.3.3 Number of Units Detected

As a point of interest, the number of units detected at different recording times was also measured. Figure 13 shows the number of units detected for different recording lengths of a piece of test data with 100 ground truth units. It shows that Kilosort4 tends to oversplit, and this tendency increases with the length of the recording.

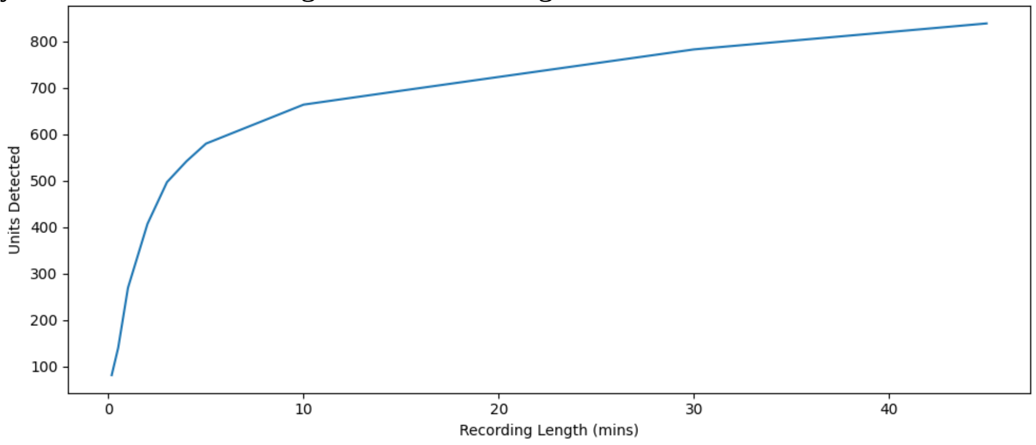


Figure 13: Number of Units Detected vs Training Time

5.4 Discussion

5.4.1 Outcomes

The final offline pipeline achieves comparable accuracy to other best-practice sorting algorithms, and was successfully implemented in a development environment on a personal computer. Although it did not quite achieve 'real-time' processing speeds on this computer (i.e. able to process 5 minutes of data in 5 minutes), it was acceptably fast, particularly considering the relatively low specifications of the computer used. Sorter output can be viewed using the Phy library [34] and experimenters can use this to manually select units for tracking.

In addition to this base functionality, wrapper functions were built to empirically find the optimal training time, which was determined to be ~5 minutes. Additionally, changes were made to the Kilosort4 library to improve its documentation and readability, as well as to fix some bugs in the benchmarking process.

Cosine and DTW similarity were evaluated as template matching methods, and although cosine similarity improved roughly in-line with the sorter's F1 score at different recording lengths, the apparent complete plateau of DTW indicated that it may not be an appropriate similarity measure for the online sorter, being insufficiently able to differentiate between units.

5.4.2 Changes in Direction

The original plan for testing and validation was to use SpikeInterface [30] to run data through all its supported pipelines, which included all the standout candidates from the initial search plus some others that would not have otherwise been tested. This plan had the additional benefit of standardizing and externalizing the validation process, preventing the biases encountered in (for example) Kilosort4's built-in benchmarking algorithm, in which the sorter's tendency to oversplit was generously accounted for. However, despite SpikeInterface's active development cycle and community, attempts to set it up with any of the pipelines proved excessively time-consuming and difficult to troubleshoot. This was compounded by Python's weak typing, which led to a general lack of transparency in the format data was expected to take. Additionally, at the beginning of the project, Kilosort4 – the standout candidate for the offline sorter – had not yet been formally published, so was not supported by SpikeInterface. Wrapping the offline sorter with SpikeInterface would vastly improve the modularity of the final pipeline, but after a little over a week of work in that area with little progress, this idea was ultimately abandoned due to time constraints.

In a similar vein, the original plan for validation data was to use simulated recordings from the SpikeForest project [21] to avoid artificially inflating accuracy scores by testing with data generated by the sorter itself. Once again, however, a lack of transparency in data format requirements, both in documentation and in the code itself, made implementation prohibitively time-intensive. This was not considered a major issue, however, as the goal with benchmarking was to find the comparative performance across different training times, rather than absolute and objective evaluation of the sorter's accuracy.

Although use of UQ's high performance computing system was considered to improve the speed of data processing, this was ruled out due to concerns with the experimental applicability of any data generated with regards to runtime. In an experiment, electrophysiological data would be processed on a PC in the lab (with specifications shown in Table 7), and the time spent processing on that computer would be time not able to be spent on the experiment. It was deemed a higher priority, therefore, to test the algorithm's running time on a computer with comparable specs to get an accurate measure.

5.4.3 Limitations & Further Work

It should be noted that measuring the temporal performance of software using a clock is highly variable, based on, among other things, the hardware and software specifications of the computer running it. A big-O analysis of the algorithm's time complexity would be ideal [35], but due to the algorithm's complexity plus time constraints this was ruled out. The measurement method was standardized as much as possible by using the same computer for all executions of the algorithm and refraining from running any other tasks on the computer while the sorter was running. However, it would still be best to test the sorter's runtime on target hardware in any application where execution time is important.

As discussed briefly in Section 5.3.3, Kilosort4 tends to oversplit units by over a factor of 8: a ground truth recording with 100 units was detected to have over 800 units by the sorter. Optimisation of this, or the ability to arbitrarily add postprocessing steps, is a clear contender for further work.

Additionally, although best efforts have been made to be accurate when specifying the functionality of Kilosort's code (particularly the benchmarking algorithm, which is not covered in the paper), it is possible mistakes have been made due to the lack of documentation and undescriptive document names. Kilosort is an excellent library for its intended purpose – spike sorting large amounts of electrophysiological data quickly and accurately – but difficult to parse if you want to make any changes to the algorithm yourself.

Additionally, although the provided GUIs for running Kilosort4 and viewing its outputs are excellent, integration of all GUIs into a single interface would significantly improve the flow for users of the pipeline.

6 Study 2: Online Sorter Development

6.1 Aims

This part of the project required the development of an online spike-sorter. This sorter had four major design requirements, outlined in Table 8.

Table 8: Online Sorter Requirements

Requirement	**Description**	**Reasoning**
Latency	The full data pipeline for the sorter must run within appropriate latency boundaries – in this case, 2.5ms.	In a closed loop design, feedback should be provided within one 'cycle.' Given that the refractory period of a neuron is ~2.5ms, this was seen as the absolute maximum allowable latency
Accuracy	The final sorter should achieve better accuracy than threshold crossing alone.	If the sorter does not achieve better accuracy than threshold crossing it provides no value.
Unit Tracking	The final sorter should have the ability to track specific units.	Experimenters expressed a desire to track the activity of specific units over time.
Lab compatibility	The final sorter must be compatible with the lab's existing hardware and software environment: • The Open Ephys ONIX system [13] • Neuropixels probes [5] • The Bonsai graphical programming suite [12] • Windows 11	Any sorter not compatible with the lab's hardware cannot be used by the lab and is therefore not fit for purpose.

Verification was a significant portion of the implementation process for this study. Unlike Kilosort4, Bonsai had no built-in verification, nor any way to run pre-recorded data through the pipeline. Verification was necessary to test the latency and accuracy of the sorter under different conditions.

6.2 Methods

6.2.1 Choosing a System

While selecting an offline sorter was a highly involved process, the choice of framework for the online sorter was straightforward, as the lab already had some pipelines set up for live data processing in Bonsai. The decision was therefore made to remain with Bonsai to maximise the maintainability of the project by others in the lab after the project was completed. Additionally, the modular nature of Bonsai meant that it would be fairly straightforward to combine the online spike sorting pipeline with other pipelines as desired, simply by dragging and dropping compatible nodes. Designed for use by non-programmers, Bonsai is highly user-friendly provided nodes' inputs and outputs are compatible, making it ideal for future maintainability.

6.2.2 Bonsai

The Bonsai graphical programming suite is written in C# 7.3 (.NET 4.7.2 framework). It has a set of built-in libraries developed by either the creators of Bonsai, hardware manufacturers like OpenEphys, or other members of the community. These libraries provide various functions which are represented in the program as nodes, which can be connected to one another to form a live data processing pipeline. Each function takes an observer of some data type, does some transform on the data within the observer, then returns an observer of another data type. Though parallelised data processing can be done in underlying libraries (typically communicating with C through some wrapper function, the top-layer flow of data between nodes is synchronous.

The libraries installed for this project are outlined in Table 9.

Table 9: Bonsai Libraries Installed for Online Sorter

Library	**Purpose**
OpenTK	Utility library for underlying mathematical functionality
OpenEphys.onix1	Firmware for interfacing with the OpenEphys ONIX hardware
Bonsai – System Library	IO functionality
Bonsai DSP	Digital signal processing - Butterworth filter - Spike detection - Scale conversion (bit depth conversion necessary for some matrix operations in OpenCV.NET)
Bonsai Visualisers	Signal visualisation
Bonsai – Scripting Library	Implementation of custom scripts
Bonsai – DSP Design Library	Visualisation for DSP library

6.2.3 Full Pipeline

The final pipeline, as visualised in Bonsai, can be seen in Figure 14.

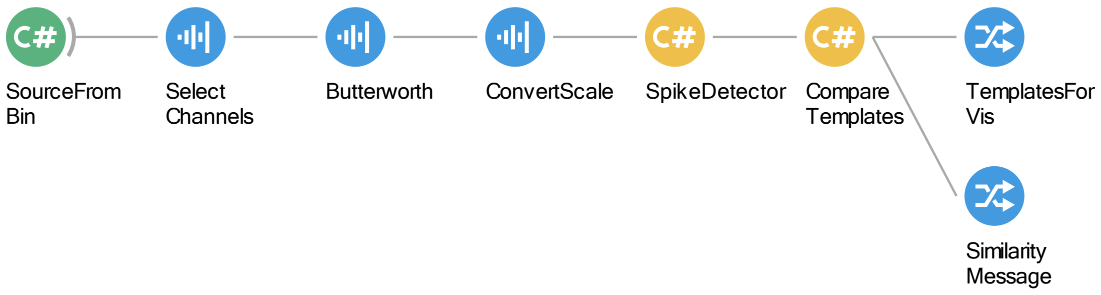
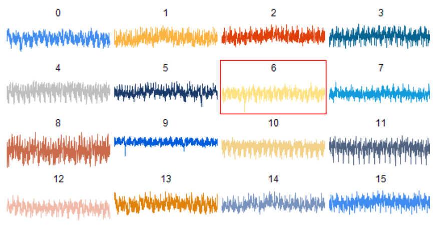
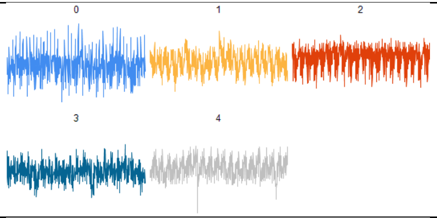
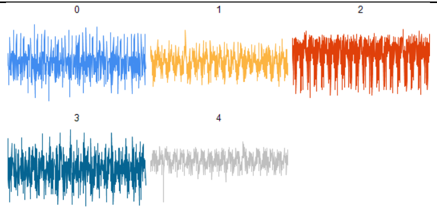
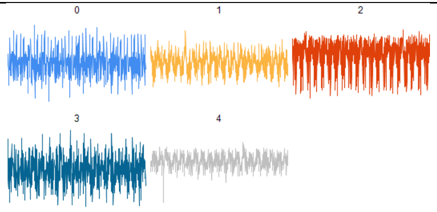
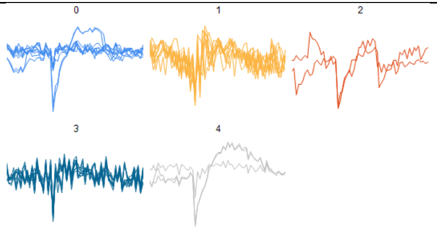
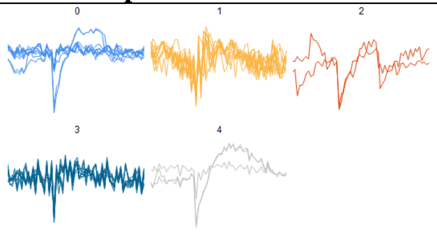
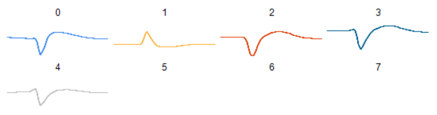


Figure 14: Final Online Pipeline Implemented in Bonsai

An overview of each step in the pipeline can be seen in Table 10.

Table 10: Pipeline Steps for Online Sorter

Node Name	Description	Data Output
SourceFrom Bin	Custom function Dynamically reads data from a .bin file at a given bit depth and feeds that data in batches of 30 to the next node. See Section 6.2.4. Designed to have the same outputs as the Open Ephys Onix interface library so the node can be swapped out when the experimental setup is ready.	
Select Channels	Library function Passes through only specific channels to the next node. Does not preserve original channel number. Improves processing speed of later steps by reducing the amount of data to process.	
Butterworth	Library function Collects the data in mini batches and applies a high-pass 3rd-order Butterworth filter at 300Hz. Batch size, filter order, filter type, and cutoff frequencies are all configurable within the Bonsai GUI.	
ConvertScale	Library function Converts bit depth of data from F16 to F32, with optional scale conversion. Necessary for compatibility with later library functions.	
SpikeDetector	Adapted library function Tracks activity in each channel in a buffer and outputs a single-channel 60-tick length spike waveform when a spike is detected (through amplitude threshold crossing), with the initial threshold crossing aligned to the 20th tick. Does not output a spike if the channel has been active in the last 2.5ms.	

Node Name	Description	Data Output
Compare Templates	Custom function Loads in target templates. When SpikeDetector passes through a detected spike, aligns template by peak then finds similarity. If a spike does not match the template's given channel, it is not compared to that template. See Section 6.2.5.	
TemplatesFor Vis	Custom function Visualisation for the templates that have been loaded in, as well as their ID within the pipeline.	
Similarity Message	Custom function Reports the similarity of each detected spike to its target template. Used for demo and validation purposes. In experiments, a signal is passed through if the similarity score passes some defined threshold.	N/A

6.2.4 Real-time synthetic data feed

The real-time synthetic data feed was built to emulate the data gathered during real experiments, but with synthetic data with a known ground truth to allow for validation of accuracy. The feed has four configurable attributes:

- FilePath, the path to the binary file to read,
- BufferSize, the number of ticks to output per cycle,
- NumChannels, the number of channels in the original data (384 for the ground truth dataset provided, 385 for the lab's acquired data), and
- Frequency, the target output frequency for the observer, in Hz.

The observer outputs OpenCV.net Mat objects of size $NumChannels \times BufferSize$, at a rate of $Frequency \div BufferSize$. It keeps a handle to the binary file loaded in memory and reads the file as needed. The data used for validation was sampled from the data used in the offline validation (see 5.2.2) offset by 10 minutes to simulate the temporal separation between training and testing data in the actual experiment.

6.2.5 Template Matching

The template matching algorithm has 6 configurable attributes:

- ComparisonMethod: the method used to compare the templates to detected spikes
- SourcePath: the location of the template .csv files,
- TemplatesToTrack: the templates to target for comparison,
- VisualisationOutput: a Boolean attribute that determines whether visualisation data is output from the node,
- DistanceThreshold: channels further than this threshold from a template's denoted channel will not be compared with that template.
- SimilarityThreshold: the similarity score above which a spike is said to be confirmed.

At initialisation, the templates defined by TemplatesToTrack are loaded in. Because of the way the built-in functions handle channel numbers, these templates must be entered in the order of their target channels, as entered in ChannelSelect. Once the templates are loaded in, the channel of maximum peak-peak amplitude for each template is found and extracted. These single-channel templates are then kept loaded in memory for the matching process, which takes the following steps for each detected spike:

1. Iterate through the list of tracked templates. If the template ID (internally determined by the order in which the template was loaded into memory) is not within the DistanceThreshold to the detected spike's channel, its similarity is set to -1 and the template is not compared.
2. If the channel matches, align the template and the spike by peak amplitude and crop the spike and template to the same length, then check the similarity of the template using the denoted comparison type.
3. If the similarity passes the SimilarityThreshold, a Boolean 'True' is output for use in further experimental values.

A computer-readable output of timestamp, spike channel, and similarity score for each template is also generated for use in validation and benchmarking.

ComparisonMethod currently supports use of cosine similarity, an amplitude-agnostic measure that takes two vectors, $\vec{A}$ and $\vec{B}$, and measures the angle between them using the formula:

$$\frac{A \cdot B}{\|A\| \times \|B\|}$$

6.2.6 Latency Benchmarking

Hardware was tested on a Lenovo Yoga 7, the specifications for which can be seen in Table 11.

Table 11: Online Benchmarking Computer Specifications

Component	**Specification**
OS	Windows 11 v.23H2, 64-bit
CPU	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
RAM	8.00GB
GPU	No dedicated GPU

Every stage was set up with a millisecond timestamp function written to CSV, as shown in Figure 15.

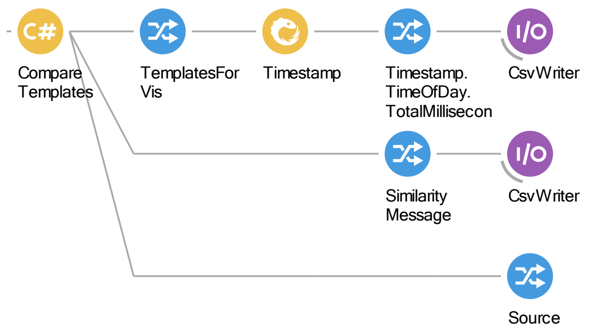


Figure 15: Excerpt of Bonsai Pipeline with Timestamping

These timestamps were then matched to one another and total latency could be found through subtraction.

6.2.7 Accuracy Validation

To validate accuracy, test data was first split into train and test sets: templates were extracted from the first 5 minutes of the dataset (recording length chosen per the process outlined in Section 5.2.3), then online performance was tested on 10 minutes from the 10 minute mark, to simulate the time spent extracting the templates. To test the pipeline's performance on different unit types, four representative units were chosen and tracked: one each for high activity, low activity, high amplitude, and low amplitude.

It was intended that Bonsai's built-in timestamp function would be sufficient to match tick times, but this was ultimately infeasible – timestamps were recorded every time the pipeline was run, not just every time a spike was detected, which made it impossible to match up the similarity scores for detected spikes with their appropriate ticks. The similarity scores were therefore timestamped, but these timestamps then needed to be matched with the data generation timestamps through the following steps:

1. Load in the detected timestamps from the SourceFromBin timestamped output
2. Convert each timestamp to a tick number using $t = i \times buffer$ where i is the index of the timestamp, t is the tick number, and $buffer$ is the number of ticks per observable.
3. Load in the detected timestamps from the SpikeDetector,
4. Find the first instance in the test data where it crosses the spike detection threshold and set that value as the preliminary offset
5. Alignment can now be treated as a series of bipartite matching problems, in which the minimal 1:1 mapping of the two timestamps is found. Scipy's linear_sum_assignment function is used for this.
6. Once a total bipartite cost has been found at the given offset, the data is shifted up and down by ~5000 ticks in both directions to check for a better minimal mapping. The offset is then set to the optimal value, and this value is treated as the true offset.
7. This offset is recorded and cross-checked with all channels.
8. Timestamps in the CompareTemplates set are then converted to tick numbers using their mapping to the SourceFromBin timestamps.
9. Detected spike tick numbers are then matched to ground truth spike ticks using bipartite mapping. If a ground truth tick has a corresponding detected tick within a tolerance of 45 ($1.5 \times bufferSize$), it is treated as a true positive

Once true positives, false negatives, and false positives are extracted, it is possible to calculate various performance metrics like F1 score.

6.3 Results

6.3.1 Optimal Threshold Setting

In the interests of making the final pipeline as automated as possible, amplitude thresholds were tested based on the metrics of the first 5 minutes of data and set at percentiles from 99.0 to 99.99999. Figure 16 and Figure 17 show the accuracy of the pipeline at each of these amplitude thresholds.

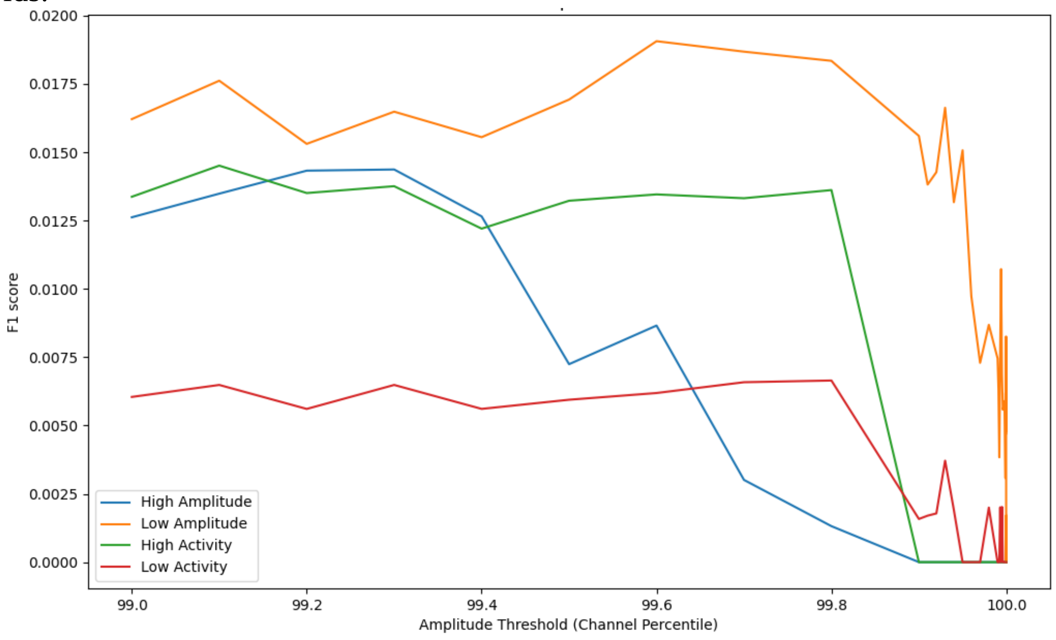


Figure 16: F1 Score vs Amplitude Threshold

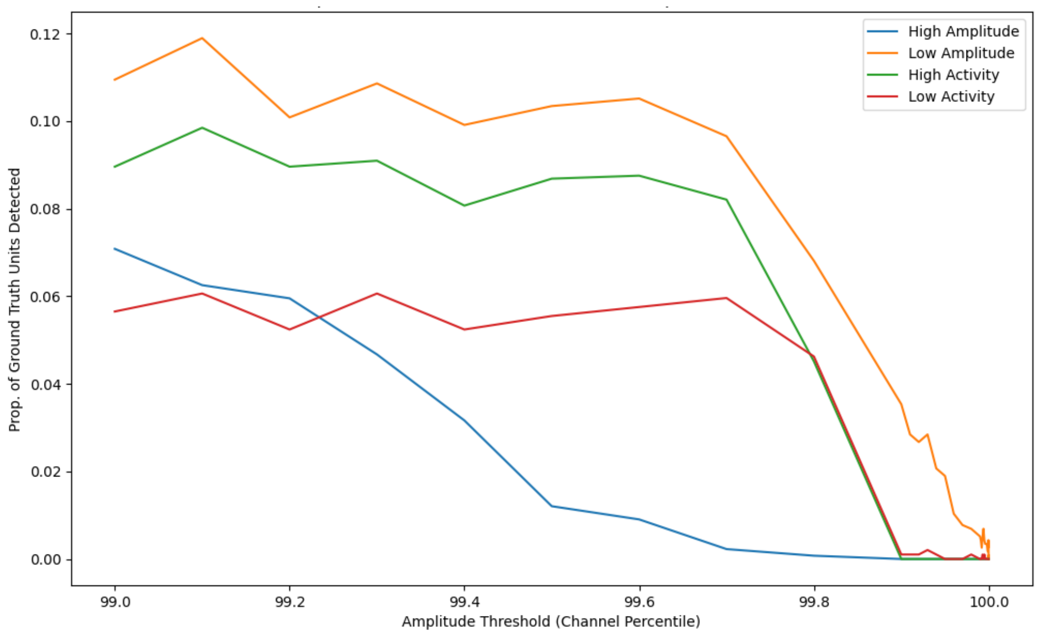


Figure 17: Proportion of Ground Truth Units Detected vs Amplitude Threshold

Figure 16 shows no clear universal best threshold. Performance as a rule appears to decrease as the threshold is raised.

Figure 17 shows a dramatic decrease in performance when the threshold is raised above the 97th percentile for both the high- and low-activity units, as well as the low amplitude unit. The high amplitude unit, interestingly, performed consistently worse as the percentile threshold increased. This may be attributed to its channel being relatively noisy, so the channel's SNR remained low despite the unit's high spiking amplitude.

6.3.2 Latency

Figure 18 shows the latency of the online sorter with differing numbers of tracked units. Latency was tested for a pipeline that had no template matching; second with template matching with no spatial overlap (i.e. each detected spike is compared only with the template associated with its channel of highest peak-peak amplitude); and third with template matching and high spatial overlap (i.e. each detected spike is compared with all tracked templates).

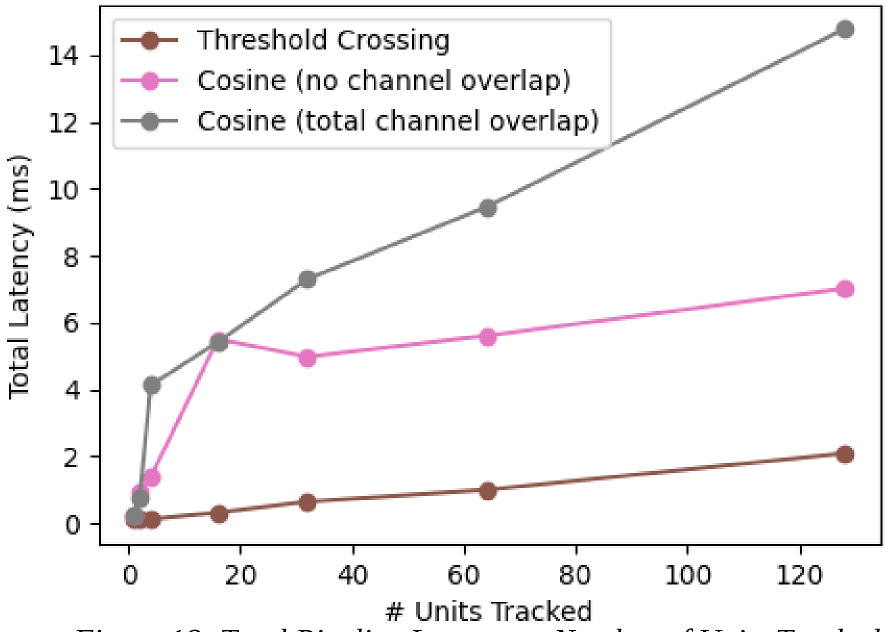


Figure 18: Total Pipeline Latency vs Number of Units Tracked

After a sharp initial increase, the latency of the no-overlap pipeline appears to increase at a comparable rate to the pipeline with threshold crossing alone. Predictably, the latency in the pipeline where all spikes are compared with all loaded templates increases far more rapidly, as the number of comparisons made per detected spike increases per unit added. On the benchmarking PC, 4 units are able to be tracked without exceeding the stated latency limit of 2.5ms.

6.3.3 Similarity Comparison

Figure 19 shows the change in F1 score per similarity threshold, with an amplitude threshold set at the 99th percentile of the data's amplitude. Dotted lines denote the performance of threshold crossing alone. This figure appears to show no improvement in accuracy from threshold crossing.

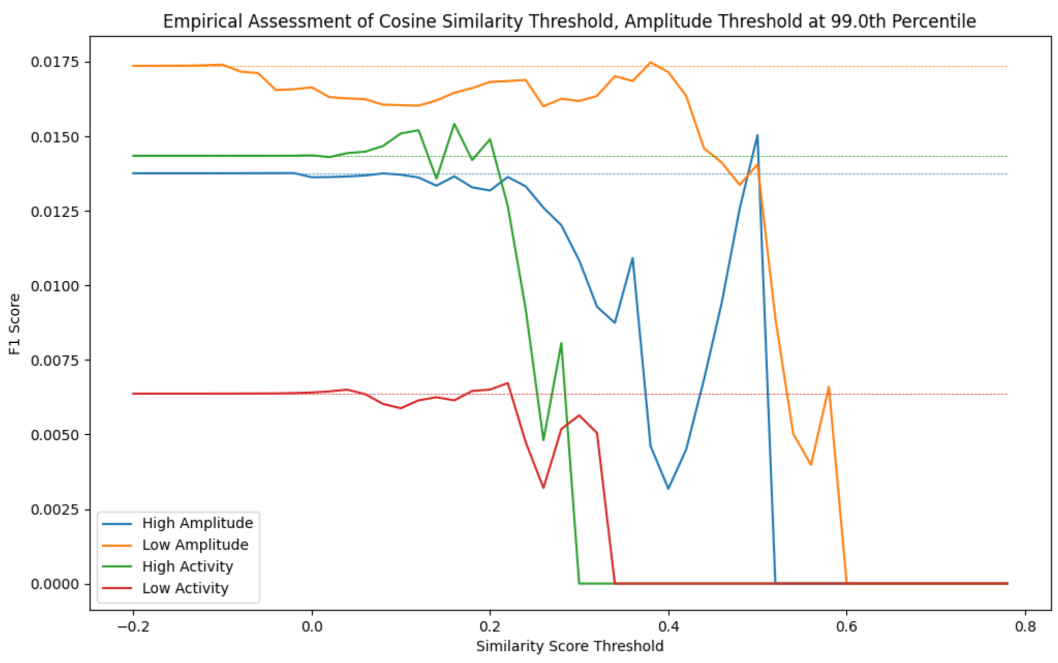


Figure 19: F1 Score vs Cosine Similarity Threshold at 99th Percentile Amplitude Threshold

However, Figure 20 shows a significant increase in F1 accuracy at a 99.6th percentile amplitude threshold for high-amplitude and low-activity units, at similarity thresholds of 0.5 and 0.4 respectively, and Figure 21 shows a comparable improvement in accuracy for high- and low-activity units at the 99.7th percentile amplitude threshold and similarity thresholds of 0.25 and 0.4 respectively. A small improvement in performance can also be seen for the low amplitude unit. Appendix C. Full Template Matching Results shows the full spread of F1 score at each threshold tested.

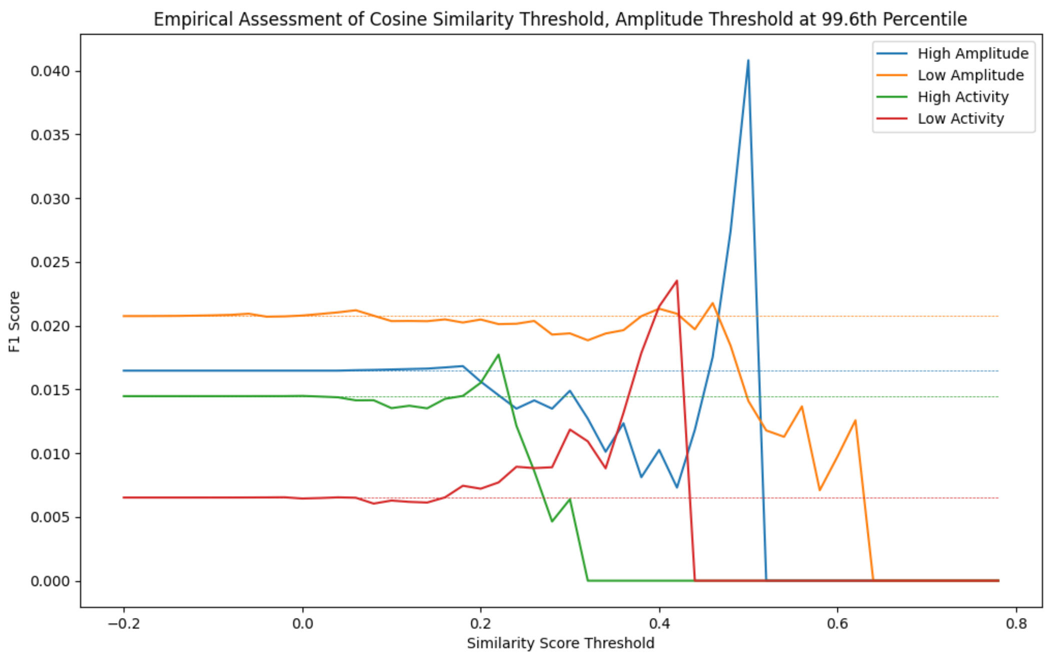


Figure 20: F1 Score vs Cosine Similarity Threshold at 99.6th Percentile Amplitude Threshold

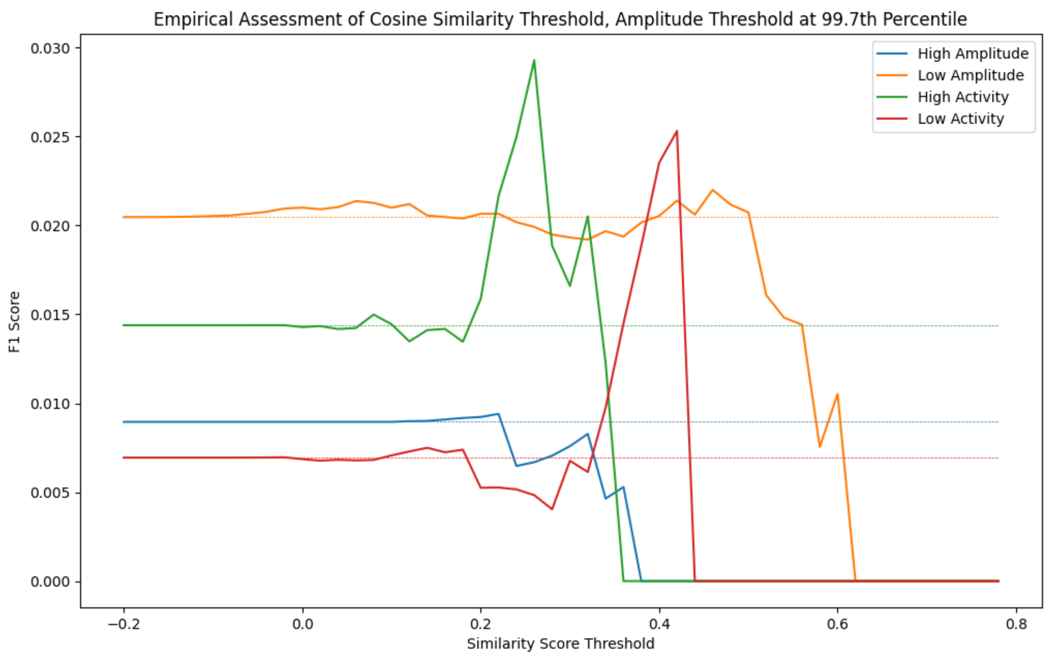


Figure 21: F1 Score vs Cosine Similarity Threshold at 99.7th Percentile Amplitude Threshold

Through the initial data generated, all unit types appear to have some combination of amplitude and similarity threshold that improves sorter accuracy, sometimes dramatically.

6.4 Discussion

6.4.1 Outcomes

An online template-matching spike sorter was implemented in Bonsai, able to track 4 units at once on the benchmarking PC. Hyperparameters, such as amplitude threshold, number of units tracked, and similarity score threshold, were tested to empirically find the optimal setup.

The lack of a clear best choice for amplitude threshold can largely be attributed to the low SNR of the data. The F1 score, as the harmonic mean of precision (the proportion of detected positives that are true positives) and recall (the proportion of true positives that are detected), is sensitive to proportional changes. If false positives outnumber true positives by a factor of ~10,000, as they did at nearly all tested thresholds, then a 10,000-fold decrease in false positives accompanied by the loss of one true positive will see no improvement in F1 performance. This high ratio of false to true positives may indicate a need for further preprocessing in the online pipeline to reduce the impact of LFP and other noise. For the pipeline as it is now, amplitudes are likely to continue to be manually set by experimenters based on their expert judgement and the specific requirements of the experiment.

Within the latency requirements set out by the experimenters, approximately 4 units can be tracked by the benchmarking PC before latency exceeds the stated limit of 2.5ms. It is important to note, however, that modern computers are able to achieve clock speeds more than double those reachable by the benchmarking PC. In addition, the process of dynamically reading synthetic data from a binary file is likely to have added latency that would not typically be found in an experiment. Although latency increased quickly to begin with, once the initial processing hurdle was passed, processing speed increased at a rate on par with threshold crossing alone. Although limiting tracking to four units may be sufficient for some experiments, experimenters should test latency on their own hardware before enforcing this four-unit limit.

Template matching achieved improved accuracy at some combinations of amplitude and similarity score threshold, but some other combinations showed limited improvement. Ultimately, more data must be collected before conclusions can be made about its experimental applicability. Although the data can be used to draw some preliminary conclusions, choosing a single representative unit for each neuron type naturally leads to an inability to control for factors like local noise level and spike overlap. Time limitations, combined with the inability to change hyperparameters and run the online pipeline programmatically, made the collection of more data untenable.

6.4.2 Synthetic data feed

Reading the file as the data was generated, rather than reading the entire .bin file into memory at once, had the minor drawback of increasing the time to generate each observable in the sequence. Practically, this meant that the maximum rate of generation on the computer used for benchmarking was in the order of 5-10kHz, so feeds with a buffer size smaller than ~5 were unable to feed data at 30kHz. However, it came with the greater benefit of allowing files of arbitrary length to be read without overloading the limited RAM of the benchmarking PC.

6.4.3 Template Matching Methods

Cosine similarity was chosen as the initial matching method as it was considered a good balance of processing time and accuracy. It was also a relatively novel approach – many existing sorters used some distance metric. Unfortunately, time limitations prevented a full exploration of the latency and accuracy of other measures (e.g. Euclidean distance, Manhattan distance), but the template comparison node is structured so that implementation of alternative measures is very straightforward.

As discussed in Section 5.2.3, dynamic time warping (DTW), a method commonly used in finding audio snippets in larger sections of audio, was also originally implemented. However, although it had comparable time complexity to cosine similarity, it was found in earlier testing to have a limited ability to differentiate between templates (i.e. while differences in training time made appreciable differences in cosine similarity between template spike shapes and their ground truths, but no appreciable difference for DTW similarity – see Section 5.2.3), so was ultimately scrapped.

6.4.4 Tick Matching for Validation

The process of validation for this portion of the pipeline was far more complex than originally anticipated. Timestamping within Bonsai appeared to have one of two options:

- Generate a timestamp for every time the full pipeline is called, giving no indication of whether or not a spike was detected at that timestamp, or
- Manually generating a timestamp along with the output data, which would be called earlier than the other timestamping functions.

The former was sufficient for latency benchmarking but not sufficient for accuracy. The latter meant that although the pipeline was executed synchronously, manually generated timestamps were all generated before any built-in timestamp generation. This created the unforeseen issue of needing to manually find the mapping between these timestamps before any validation could take place.

6.4.5 Further work

A major limitation to this study was the inability to programmatically execute Bonsai code, so any test cases needed to be implemented manually. This dramatically limited the scope of any benchmarking that could be done and should be the top priority for any further work on the online pipeline. Executing test cases became a major time sink towards the end of the study, and infringed on the capacity to implement and test, for example, multiple comparison methods to find the fastest and most effective one.

In that vein, a full grid search of potential preprocessing stages, including limited CAR where the running mean per channel is removed, PCA, and potentially also whitening, to find the optimal balance between accuracy and latency, would be a strong contender for further work. Implementation of other similarity measures like Euclidean distance (used in Osort) and Manhattan or Chebyshev distance (likely to be less accurate but much faster) would also be beneficial. It may also be interesting to test amplitude thresholds set by different metrics – for example, the standard deviation of a channel – and see if that improves the performance of the threshold crossing step.

In the interests of modularity, it would be useful to separate out the function that loads the tracked templates into its own node. Attempts were made in this area, but ultimately scrapped in the interests of time as the end of the project approached. Other areas of potential further work include parallelising the template matching process using underlying C libraries to increase processing speed and making refractory periods optional in the spike detector to allow for tracking of multiple units on one channel.

It would also potentially be useful to adapt the spike detector to allow for multiple units on one channel. The current spike detector tracks refractory channels and will not detect a spike if one has been detected in the last 2.5ms. This would be useful as a toggle-able feature, so experimenters can choose whether they want to track a single unit per channel, or multiple units.

Other areas of potential future development include the implementation of drift tracking, automatic suggestions for units to track, and – of course – integration with a full closed-loop control system.

7 General Discussion and Conclusions

7.1 Discussion

With the correct configuration of parameters, the final online spike sorter can achieve up to 5 times improved accuracy than threshold crossing alone, with latencies of <2.5ms for up to 4 units when run on the benchmarking PC. Limitations imposed primarily by time limits mean that the pipeline has only been tested in a limited capacity, but in that capacity it was shown to perform well given the correct parameter tuning, particularly for lower activity units. Ultimately, the application of this project to experimentation will be left to the discretion of the researchers involved and their assessment of experimental requirements. However, it shows great promise in the fast, accurate tracking of small numbers of units in control experiments.

The Cosine similarity, as opposed to some other similarity metric like Euclidean distance or correlation coefficient, was chosen because it was amplitude-agnostic, which was highly useful in a situation where the units of output templates were not clear or documented. It was also intended to help in dealing with overlapping spikes when they occurred. A stretch goal, and a clear area for further work, would be the implementation and systematic evaluation of additional comparison methods, to gain even greater confidence in the online pipeline.

A major drawback of Osort, the current best-in-class in online template matching, is that it was written in Matlab, which despite its ubiquity requires a license to use and is therefore inherently limited. Kilosort, while fast and written in Python, cannot handle data fed to it in real time. This project has been an attempt to marry the strengths of each of these state-of-the-art approaches in a way that is transparent and repeatable.

7.2 Conclusions

This project has resulted in the combination of an offline and online sorter which tracks units with low latency and, under the right parameters, a high level of accuracy compared to threshold crossing alone. It has shown promising initial results. It has been shown that 5 minutes of training time is sufficient for the generation of high-quality templates, while providing ample time for live experimentation, even in a short experimental window (45 minutes). Cosine similarity was evaluated to be the best of the considered template matching algorithms for online applications, though more should be evaluated for their accuracy before settling on this as a final result.

7.3 Future work

As tends to be the case with any project, the space for further work is vast. Much future work was discussed in sections 5.4.3 and 6.4.5, but an overview is presented below.

7.3.1 Offline Sorter

While optimising the offline sorter for speed and accuracy would be a worthy goal, the still-open field of spike sorting is unlikely to be solved by an undergraduate developer. Spike sorting will remain an open field far beyond the scope of this project. However, improvements that could be applied without first obtaining a PhD include modularity and adding external validation metrics. Modularity could be improved by wrapping the sorter in SpikeInterface, which allows a ‘plug and play’ approach to spike sorting. Going further, a bespoke wrapper could be created which packages the entire offline and online pipeline into one GUI. Although the internal validation metrics of Kilosort4 were sufficient for the purposes of this project, which only wanted comparative accuracy, adding validation metrics which do not take the assumptions of the sorter's creators for granted would improve confidence in the pipeline as a whole. Further potential future work is outlined in Section 5.4.3.

7.3.2 Online Sorter

The online sorter has the largest capacity for future work. Because it was developed after the offline sorter was implemented in the pipeline, time constraints were a far more pressing issue, so many 'nice to have' features were sacrificed in the interests of achieving a minimum viable product. Full integration with the lab's experimental setup is, of course, the primary contender for future work. Additional features that account for real-world experimental conditions like probe drift would also be worthwhile additions. Programmatic execution of the online pipeline, and by extension further testing of hyperparameters and types of tracked units, would be instrumental in improving confidence in the pipeline and its results. Implementation of the stretch goals outlined in the initial thesis proposal – implementation on lab hardware and application to closed-loop control experiments, are a clear area for further work. More areas for further work are outlined in Section 6.4.5.

7.3.3 Integration

Although a preliminary GUI was developed for this project, the development of a more cohesive GUI would dramatically improve the user experience of any experimenters intending to use the pipeline. The pipeline is designed to be integrated with the lab's existing hardware, but has not yet been tested on that hardware. This will be a natural next step in the project if it is used for experimental applications. Packaging the software up in a container using something like Docker would improve its capacity to be implemented in other laboratories without needing the help of a trained programmer.

7.3.4 Beyond The Project

A realiable, open-source online template-matching spike sorter has broad applications in neuroscience and neuroengineering, which has even broader applications in medicine, technology, and beyond. As the computational capacity of hardware increases, so does the sophistication of data processing algorithms – both online and offline. The ability to sort spikes in real time and thereby apply closed-loop control thinking to the circuits of the brain opens the door for an exciting range of experiments that could lead to a deeper understanding of the brain not just as a biological organ, but as an electrical system that can be observed, modeled, and analysed.

8 References

[1] H. S. Mayberg *et al.*, "Deep brain stimulation for treatment-resistant depression," (in eng), no. 0896-6273 (Print).

[2] "How to measure brain activity in people." Queensland Brain Institute. https://qbi.uq.edu.au/brain/brain-functions/how-measure-brain-activity-people (accessed.

[3] A. Tozer and I. Forsythe. "Electrophysiology Fundamentals, Membrane Potential and Electrophysiological Techniques." https://www.technologynetworks.com/neuroscience/articles/electrophysiology-fundamentals-membrane-potential-and-electrophysiological-techniques-359363 (accessed.

[4] J. Jacobs and M. J. Kahana, "Direct brain recordings fuel advances in cognitive electrophysiology," *Trends in Cognitive Sciences*, vol. 14, no. 4, pp. 162-171, 2010/04/01/ 2010, doi: https://doi.org/10.1016/j.tics.2010.01.005.

[5] N. A. Steinmetz *et al.*, "Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings," *Science*, vol. 372, no. 6539, p. eabf4588, 2021/04/16 2021, doi: 10.1126/science.abf4588.

[6] P. Marius, S. Shashwat, and S. Carsen, "Solving the spike sorting problem with Kilosort," *bioRxiv*, p. 2023.01.07.523036, 2023, doi: 10.1101/2023.01.07.523036.

[7] W. Martin Usrey, S. Murray Sherman, S. M. Sherman, and W. M. Usrey, "11Cell Types in the Thalamus and Cortex," in *Exploring Thalamocortical Interactions: Circuitry for Sensation, Action, and Cognition*: Oxford University Press, 2021, sec. Oxford Academic, p. 0.

[8] H. G. Rey, C. Pedreira, and R. Quian Quiroga, "Past, present and future of spike sorting techniques," *Brain Research Bulletin*, vol. 119, pp. 106-117, 2015/10/01/ 2015, doi: https://doi.org/10.1016/j.brainresbull.2015.04.007.

[9] R. Q. Quiroga, "Spike sorting," *Curr Biol*, vol. 22, no. 2, pp. R45-6, Jan 24 2012, doi: 10.1016/j.cub.2011.11.005.

[10] J. E. Chung *et al.*, "A Fully Automated Approach to Spike Sorting," *Neuron*, vol. 95, no. 6, pp. 1381-1394.e6, 2017/09/13/ 2017, doi: https://doi.org/10.1016/j.neuron.2017.08.030.

[11] T. Zhang, M. Rahimi Azghadi, C. Lammie, A. Amirsoleimani, and R. Genov, "Spike sorting algorithms and their efficient hardware implementation: a comprehensive survey," *Journal of Neural Engineering*, vol. 20, no. 2, p. 021001, 2023/04/14 2023, doi: 10.1088/1741-2552/acc7cc.

[12] G. Lopes *et al.*, "Bonsai: An event-based framework for processing and controlling data streams," *Front Neuroinform*, vol. 9, pp. 7-7, 2015, doi: 10.3389/fninf.2015.00007.

[13] J. P. Newman *et al.*, "A unified open-source platform for multimodal neural recording and perturbation during naturalistic behavior. LID - 2023.08.30.554672 [pii] LID - 10.1101/2023.08.30.554672 [doi]," (in eng).

[14] M. M. Heinricher, "2 Principles of Extracellular Single-Unit Recording," *Microelectrode Recording in Movement Disorder Surgery*Stuttgart: Georg Thieme Verlag KG, 2004. [Online]. Available: http://www.thieme-connect.de/products/ebooks/lookinside/10.1055/b-0034-56092

[15] H. G. Rey *et al.*, "Single-cell recordings in the human medial temporal lobe," *Journal of Anatomy*, vol. 227, no. 4, pp. 394-408, 2015/10/01 2015, doi: https://doi.org/10.1111/joa.12228.

[16] N. A. Steinmetz, C. Koch, K. D. Harris, and M. Carandini, "Challenges and opportunities for large-scale electrophysiology with Neuropixels probes," (in eng), *Curr*

Opin Neurobiol, vol. 50, pp. 92-100, Jun 2018, doi: 10.1016/j.conb.2018.01.009.
[17] A. P. Buccino, S. Garcia, and P. Yger, "Spike sorting: new trends and challenges of the era of high-density probes," *Progress in Biomedical Engineering*, vol. 4, no. 2, p. 022005, 2022/05/17 2022, doi: 10.1088/2516-1091/ac6b96.
[18] M. Pachitariu, N. Steinmetz, S. Kadir, M. Carandini, and H. Kenneth D, "Kilosort: realtime spike-sorting for extracellular electrophysiology with hundreds of channels," 2016, doi: 10.1101/061481.
[19] M. Saif-ur-Rehman *et al.*, "SpikeDeeptector: a deep-learning based method for detection of neural spiking activity," *Journal of Neural Engineering*, vol. 16, no. 5, p. 056003, 2019/07/23 2019, doi: 10.1088/1741-2552/ab1e63.
[20] J. H. Lee *et al.*, "YASS: Yet Another Spike Sorter," 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/1943102704f8f8f3302c2b730728e023-Paper.pdf.
[21] J. F. Magland *et al.*, "SpikeForest: reproducible web-facing ground-truth validation of automated neural spike sorters," *bioRxiv*, p. 2020.01.14.900688, 2020, doi: 10.1101/2020.01.14.900688.
[22] "Open Ephys: Open-Source Electrophysiology." open ephys. https://open-ephys.org/ (accessed 2024).
[23] E. Domany, "Superparamagnetic clustering of data — The definitive solution of an ill-posed problem," *Physica A: Statistical Mechanics and its Applications*, vol. 263, no. 1, pp. 158-169, 1999/02/01/ 1999, doi: https://doi.org/10.1016/S0378-4371(98)00494-4.
[24] T. Bangyu, "Density Peak Clustering Algorithm based on the Nearest Neighbor," in *Proceedings of the 3rd International Conference on Mechatronics Engineering and Information Technology (ICMEIT 2019)*, 2019/04 2019: Atlantis Press, pp. 665-670, doi: 10.2991/icmeit-19.2019.106. [Online]. Available: https://doi.org/10.2991/icmeit-19.2019.106
[25] M. Pachitariu, S. Sridhar, J. Pennington, and C. Stringer, "Spike sorting with Kilosort4," *Nature Methods*, vol. 21, no. 5, pp. 914-921, 2024/05/01 2024, doi: 10.1038/s41592-024-02232-7.
[26] G. Regalia, S. Coelli, E. Biffi, G. Ferrigno, and A. Pedrocchi, "A Framework for the Comparative Assessment of Neuronal Spike Sorting Algorithms towards More Accurate Off-Line and On-Line Microelectrode Arrays Data Analysis," (in eng), *Comput Intell Neurosci*, vol. 2016, p. 8416237, 2016, doi: 10.1155/2016/8416237.
[27] U. Rutishauser, E. M. Schuman, and A. N. Mamelak, "Online detection and sorting of extracellularly recorded action potentials in human medial temporal lobe recordings, in vivo," *J Neurosci Methods*, vol. 154, no. 1, pp. 204-224, 2006, doi: 10.1016/j.jneumeth.2005.12.033.
[28] Y. Wu *et al.*, "An unsupervised real-time spike sorting system based on optimized OSort," (in eng), *J Neural Eng*, vol. 20, no. 6, Nov 23 2023, doi: 10.1088/1741-2552/ad0d15.
[29] S. Bandyopadhyay and S. Saha, *Unsupervised Classification : Similarity Measures, Classical and Metaheuristic Approaches, and Applications*, 1st ed. Berlin, Heidelberg: Springer Berlin Heidelberg : Imprint: Springer, 2013.
[30] A. P. Buccino *et al.*, "SpikeInterface, a unified framework for spike sorting," (in eng), *Elife*, vol. 9, Nov 10 2020, doi: 10.7554/eLife.61834.
[31] A. H. Barnett, J. F. Magland, and L. F. Greengard, "Validation of neural spike sorting algorithms without ground-truth information," *Journal of Neuroscience Methods*, vol. 264, pp. 65-77, 2016/05/01/ 2016, doi: https://doi.org/10.1016/j.jneumeth.2016.02.022.
[32] S. McConnell, *Code Complete*, 2nd ed. Sebastopol: Microsoft Press, 2009.
[33] C. Stringer and M. Pachitariu, "Simulations from kilosort4 paper," ed: Janelia Research

Campus, 2024.

[34] *Phy: Interactive Visualization and Manual Spike Sorting of Large-Scale Ephys Data*. (2024). Cortex Lab. [Online]. Available: https://github.com/cortex-lab/phy

[35] H. C. Thomas, E. L. Charles, L. R. Ronald, and S. Clifford, *Introduction to Algorithms*. Cambridge, Mass: The MIT Press (in English), 2009.

[36] P. Yger *et al.*, "A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo," *eLife,* vol. 7, p. e34518, 2018/03/20 2018, doi: 10.7554/eLife.34518.

[37] C. Ekanadham, D. Tranchina, and E. P. Simoncelli, "A unified framework and method for automatic neural spike identification," *Journal of Neuroscience Methods,* vol. 222, pp. 47-55, 2014/01/30/ 2014, doi: https://doi.org/10.1016/j.jneumeth.2013.10.001.

[38] D. E. Carlson *et al.*, "Multichannel Electrophysiological Spike Sorting via Joint Dictionary Learning and Mixture Modeling," *IEEE Trans Biomed Eng,* vol. 61, no. 1, pp. 41-54, 2014, doi: 10.1109/TBME.2013.2275751.

[39] K. Q. Shan, E. V. Lubenov, and A. G. Siapas, "Model-based spike sorting with a mixture of drifting t-distributions," *Journal of Neuroscience Methods,* vol. 288, pp. 82-98, 2017/08/15/ 2017, doi: https://doi.org/10.1016/j.jneumeth.2017.06.017.

[40] M. Bernert and B. Yvert, "An Attention-Based Spiking Neural Network for Unsupervised Spike-Sorting," *International Journal of Neural Systems,* vol. 29, no. 08, p. 1850059, 2019/10/01 2018, doi: 10.1142/S0129065718500594.

[41] S. N. Kadir, D. F. M. Goodman, and K. D. Harris, "High-Dimensional Cluster Analysis with the Masked EM Algorithm," *Neural Computation,* vol. 26, no. 11, pp. 2379-2394, 2014, doi: 10.1162/NECO_a_00661.

[42] F. J. Chaure, H. G. Rey, and R. Quian Quiroga, "A novel and fully automatic spike-sorting implementation with variable number of features," *Journal of Neurophysiology,* vol. 120, no. 4, pp. 1859-1871, 2018/10/01 2018, doi: 10.1152/jn.00339.2018.

[43] J. Niediek, J. Boström, C. E. Elger, and F. Mormann, "Reliable Analysis of Single-Unit Recordings from the Human Brain under Noisy Conditions: Tracking Neurons over Hours," *PLOS ONE,* vol. 11, no. 12, p. e0166598, 2016, doi: 10.1371/journal.pone.0166598.

[44] N. V. Swindale and M. A. Spacek, "Spike sorting for polytrodes: a divide and conquer approach," *Frontiers in Systems Neuroscience,* Methods vol. 8, 2014. [Online]. Available: https://www.frontiersin.org/journals/systems-neuroscience/articles/10.3389/fnsys.2014.00006.

[45] G. Hilgen *et al.*, "Unsupervised Spike Sorting for Large-Scale, High-Density Multielectrode Arrays," *Cell Reports,* vol. 18, no. 10, pp. 2521-2532, 2017, doi: 10.1016/j.celrep.2017.02.038.

[46] J. J. Jun, C. Mitelut, C. Lai, S. L. Gratiy, C. A. Anastassiou, and T. D. Harris, "Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction," *bioRxiv,* p. 101030, 2017, doi: 10.1101/101030.

[47] R. Diggelmann, M. Fiscella, A. Hierlemann, and F. Franke, "Automatic spike sorting for high-density microelectrode arrays," *Journal of Neurophysiology,* vol. 120, no. 6, pp. 3155-3171, 2018/12/01 2018, doi: 10.1152/jn.00803.2017.

[48] K. Kyung Hwan and K. Sung June, "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,* vol. 47, no. 10, pp. 1406-1411, 2000, doi: 10.1109/10.871415.

Appendix A: Supplementary Tables

A.1: Full Offline Sorter Evaluations

In the interests of efficiency when beginning the search, any algorithm that did not specifically mention the ability to handle experimental conditions like probe drift and spike overlap was assumed not to. Activity is defined to be the combined paper citations and repository activity (stars and forks) at time of writing (Match 2024). Sorters with repositories hosted somewhere other than GitHub are denoted with a + next to their activity, as no measures for repository activity were available. Recency is defined to be the time of the most recent contribution (paper publication or repository commit) by developers at time of writing (March 2024).

Table 12: Evaluation of Alternative Method Sorters

Name	**Recent**	**Activity**	**Paper**	**Open-Source**	**Language**	**Method/s**	**Drift**	**Overlap**	**Online**
Osort	2021	373+	[27]	Y	Matlab	Template matching	N	N	Y
MountainSort	2023	512	[10]	Y	Python, C++	ISO-SPLIT, ISO-MERGE	N	N	N
KiloSort	2024	1633	[25]	Y	Python	Template matching	Y	Y	N
SpyKING CIRCUS	2023	434	[36]	Y	Python	Template matching	Y	Y	N
YASS	2022	143	[20]	Y	Python, C++	NN triage-cluster-pursuit	N	N	N
CBP Spike Demo	2014	113+	[37]	Y	Matlab	Continuous basis pursuit	Y	Y	N
FMM Spike Sorter	2014	64	[38]	Y	Matlab	Raw signal threshold, focused MM	N	N	N
MoDT	2019	51	[39]	Y	Matlab, CUDA, C++	PCA, drifting t-distribution	N	N	N
Spikedeeptector	2019	49	[19]	Y	Matlab	CNN	N	N	N
[unnamed]	2019	45+	[40]	N	N/A	STDP SNN	N	N	N

Table 13: Evaluation of Existing Conventional Three-Step Sorters

Name	**Recent**	**Activity**	**Paper**	**Open-Source**	**Language**	**Detection**	**Feature Extraction**	**Clustering**	**Drift**	**Overlap**	**Online**
Klusta	2015	400	[41]	Y	Python, C++	Amplitude threshold	PCA	GMM	N	N	N
WaveClus	2022	381	[42]	Y	Matlab	Amplitude threshold	DWT	SPC	N	N	N
bpSort	2014	13	N/A	Y	Matlab	Binary pursuit	PCA	T-distributed Mixture Model	Y	Y	N
Combinato	2023	151	[43]	Y	Python	Amplitude threshold	Wavelet Transforms	SPC, template matching	Y	N	N
Spyke	2022	107	[44]	Y	Python	Amplitude Threshold	PCA, ICA	Gradient Ascent	N	N	N
Ironclust	2022	43	N/A	Y	Python, Matlab	Savitzky-Golay filter	Covariance features	DPCLUS	N	N	N
Herding Spikes 2	2023	156	[45]	Y	Python	Amplitude Threshold	PCA	Mean Shift	N	N	Y
JRClust	2021	218	[46]	Y	Matlab, CUDA	Savitzky-Golay filter	Covariance features	DPCLUS	Y	N	Y?
HDSort	2021	36+	[47]	Y	Matlab	Amplitude Threshold	Prewhitened PCA	Mean Shift -> Bayes Optimal	N	N	Y
[unnamed]	2000	328+	[48]	N	N/A	N/A	N/A	3-layer MLP	N	N	N

NB Ironclust did not have its own paper as it was a Python port of JRclust, which did have a paper.

A.2: Study 1 Additional Tables

Table 14: Offline Processing Time vs Recording Time

Recording	**Sorting**	**Total Time**
00m 10s	00m 41s	00m 51s
00m 30s	01m 19s	01m 49s
01m 00s	02m 57s	03m 57s
02m 00s	05m 27s	07m 27s
03m 00s	07m 57s	10m 57s
04m 00s	10m 10s	14m 10s
05m 00s	13m 11s	18m 11s
10m 00s	28m 19s	38m 19s
15m 00s	01h 34m 56s	01h 49m 56s
30m 00s	01h 32m 18s	02h 02m 18s
45m 00s	02h 15m 30s	03h 00m 30s

Appendix B. GitHub Repositories

B.1 Full Original Work

The full codebase for this project can be found at github.com/oldhorizons/REIT4841

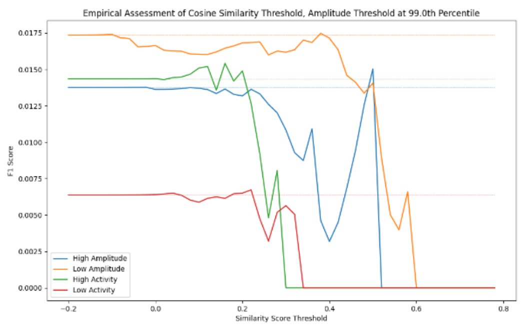
B.2 Kilosort4 Fork

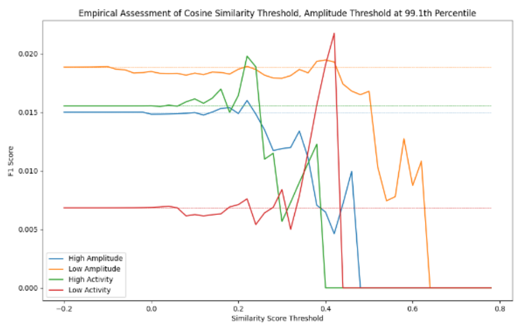
The Kilosort4 fork can be found at github.com/oldhorizons/Kilosort

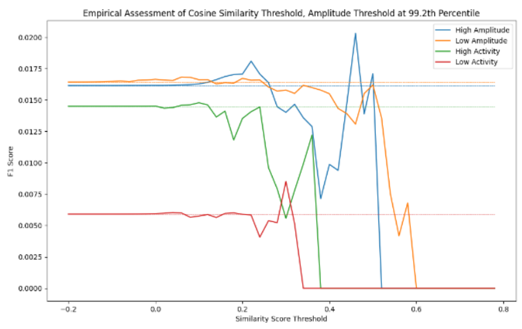
Appendix C. Full Template Matching Results

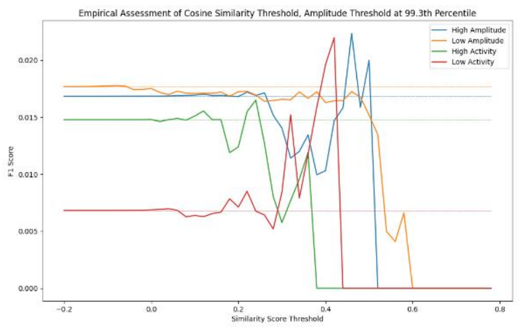
When a sorter detects no spikes or no true positive spikes, its F1 score is zero.

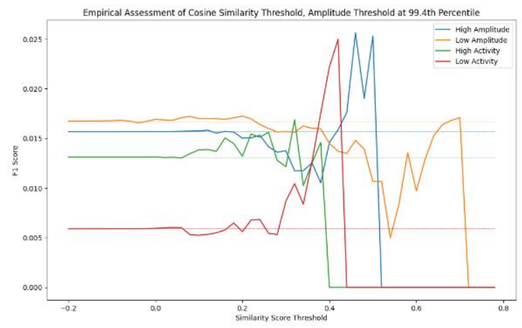
99.0 - 99.9%

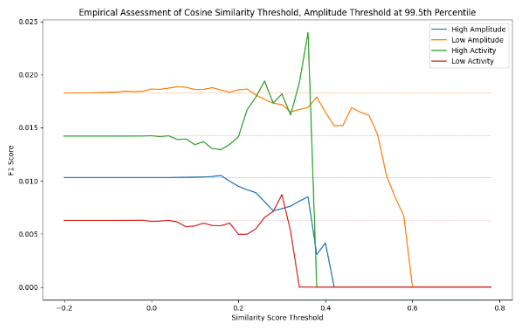


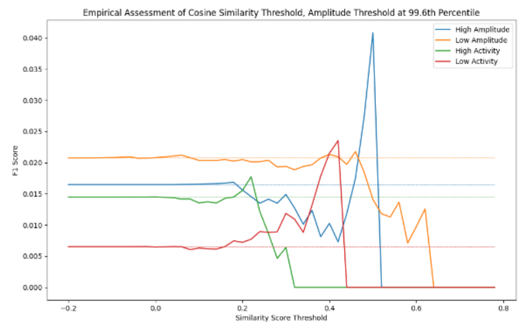


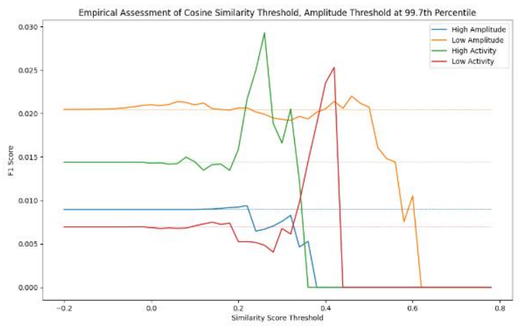


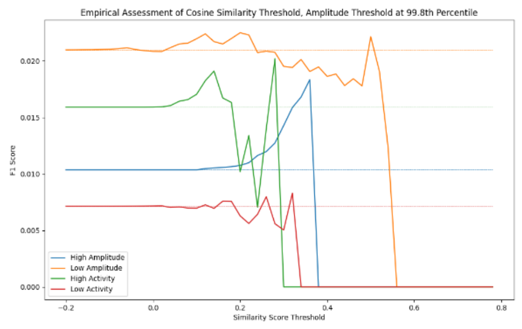


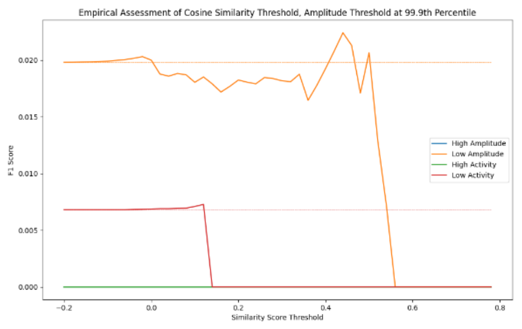




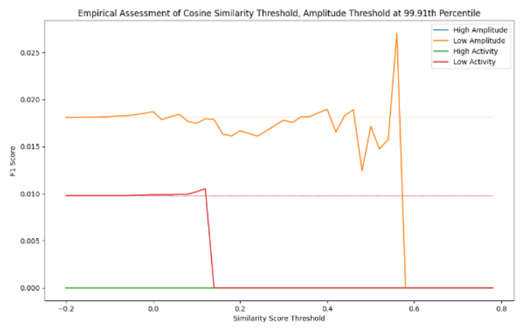


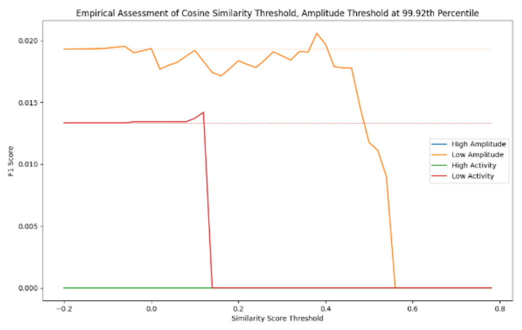


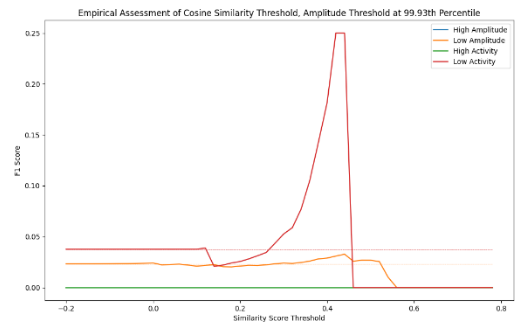

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.8th Percentile
F1 Score
Similarity Score Threshold
High Amplitude
Low Amplitude
High Activity
Low Activity

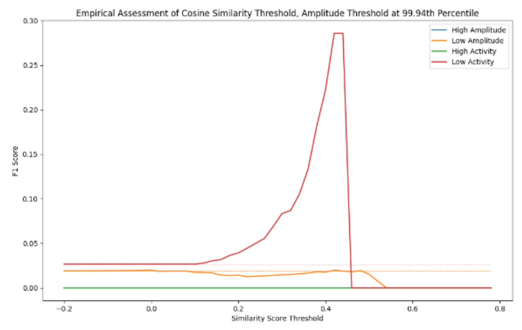

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.9th Percentile
F1 Score
Similarity Score Threshold
High Amplitude
Low Amplitude
High Activity
Low Activity

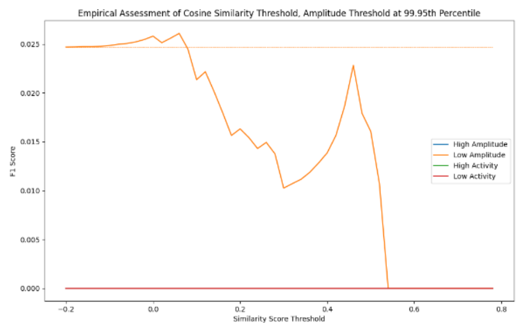
99.91 - 99.99%

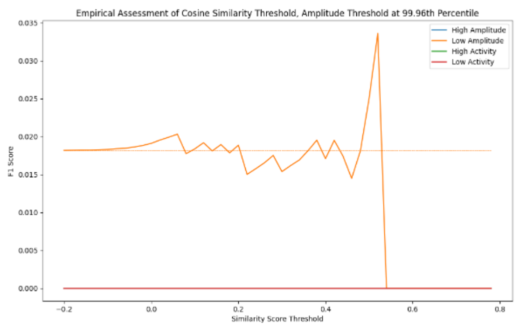

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.91th Percentile
F1 Score
Similarity Score Threshold
High Amplitude
Low Amplitude
High Activity
Low Activity

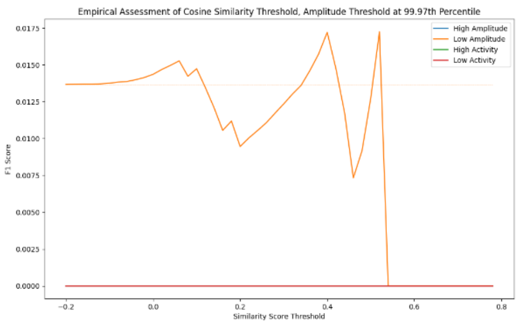

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.92th Percentile
F1 Score
Similarity Score Threshold
High Amplitude
Low Amplitude
High Activity
Low Activity

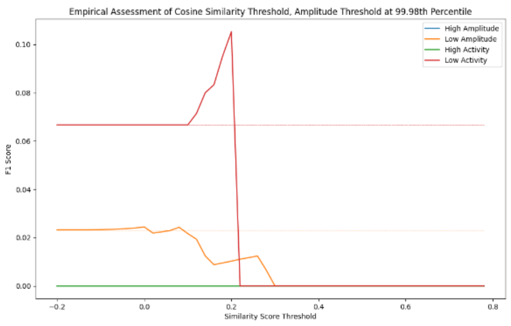

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.93th Percentile
F1 Score
Similarity Score Threshold
High Amplitude
Low Amplitude
High Activity
Low Activity

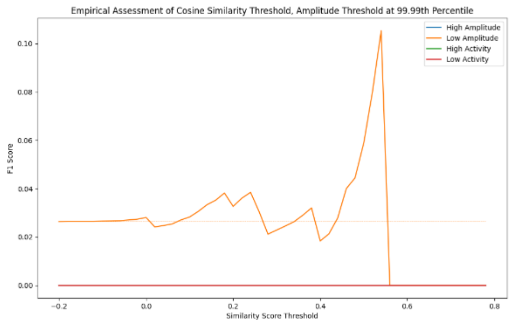

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.94th Percentile
F1 Score
Similarity Score Threshold
High Amplitude
Low Amplitude
High Activity
Low Activity


Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.95th Percentile
F1 Score
Similarity Score Threshold
High Amplitude
Low Amplitude
High Activity
Low Activity

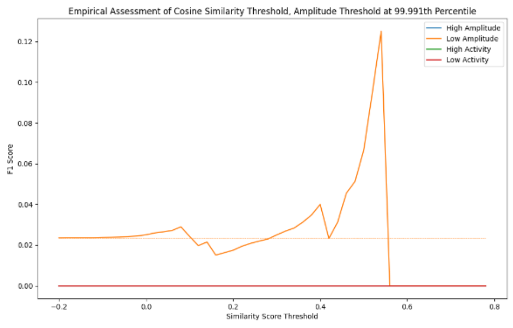

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.96th Percentile
F1 Score
Similarity Score Threshold
High Amplitude
Low Amplitude
High Activity
Low Activity

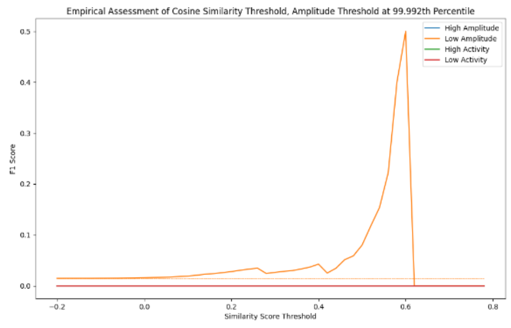

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.97th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold

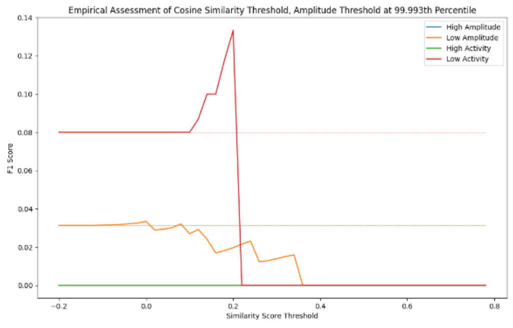

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.98th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold

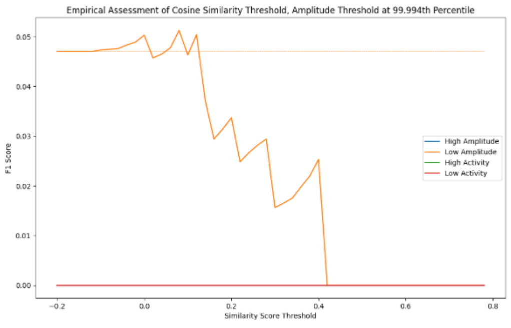

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.99th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold

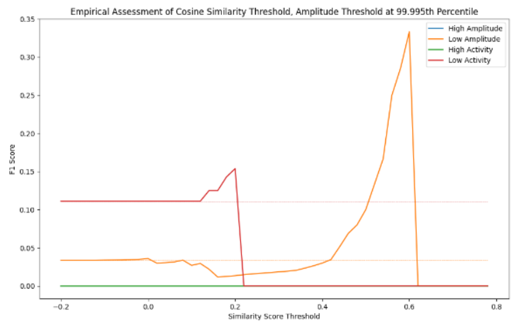
99.991 - 99.999%

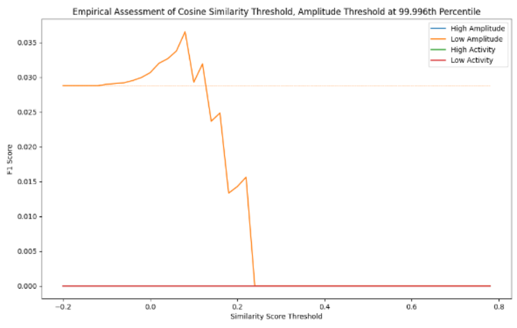

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.991th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold

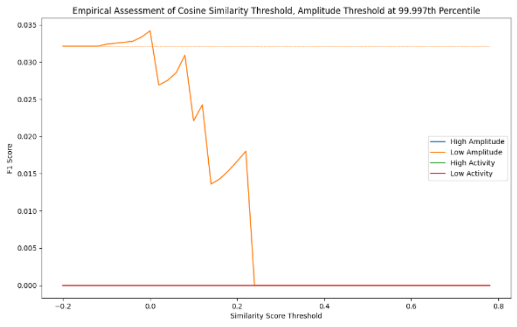

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.992th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold

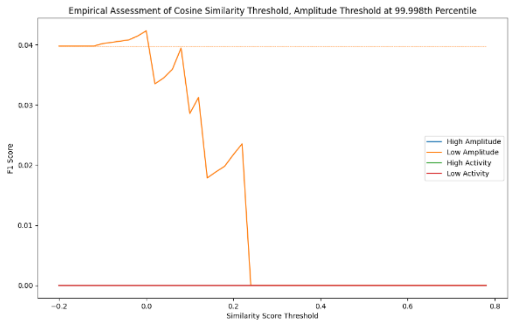

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.993th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold

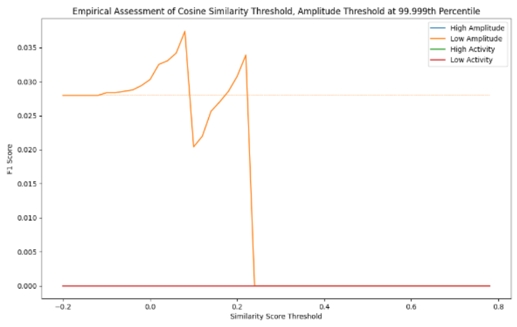

Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.994th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold


Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.995th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold


Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.996th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold


Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.997th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold


Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.998th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold


Empirical Assessment of Cosine Similarity Threshold, Amplitude Threshold at 99.999th Percentile
High Amplitude
Low Amplitude
High Activity
Low Activity
F1 Score
Similarity Score Threshold