SpikeInterface, a unified framework for spike sorting

Alessio P. Buccino[¶*1, 6], Cole L. Hurwitz[¶2], Samuel Garcia[3], Jeremy Magland[4], Joshua H. Siegle[5], Roger Hurwitz[7], and Matthias H. Hennig[2]

[1]Department of Biosystems Science and Engineering, ETH Zurich, Switzerland
[2]School of Informatics, University of Edinburgh, United Kingdom
[3]Centre de Recherche en Neuroscience de Lyon, CNRS, Lyon, France
[4]Flatiron Institute, New York City, NY, United States
[5]Allen Institute for Brain Science, Seattle, WA, United States
[6]Centre for Integrative Neuroplasticity (CINPLA), University of Oslo, Oslo, Norway
[7]Independent Researcher, Portland, Oregon, USA

[¶] These authors contributed equally to this work.

Abstract

Much development has been directed towards improving the performance and automation of spike sorting. This continuous development, while essential, has contributed to an over-saturation of new, incompatible tools that hinders rigorous benchmarking and complicates reproducible analysis. To address these limitations, we developed SpikeInterface, a Python framework designed to unify preexisting spike sorting technologies into a single codebase and to facilitate straightforward comparison and adoption of different approaches. With a few lines of code, researchers can reproducibly run, compare, and benchmark most modern spike sorting algorithms; pre-process, post-process, and visualize extracellular datasets; validate, curate, and export sorting outputs; and more. In this paper, we provide an overview of SpikeInterface and, with applications to real and simulated datasets, demonstrate how it can be utilized to reduce the burden of manual curation and to more comprehensively benchmark automated spike sorters.

1 Introduction

Extracellular recording is an indispensable tool in neuroscience for probing how single neurons and populations of neurons encode and transmit information. When analyzing extracellular recordings, most researchers are interested in the spiking activity of individual neurons, which must be extracted from the raw voltage traces through a process called *spike sorting*. Many laboratories perform spike sorting using fully manual techniques (e.g. XClust [53], SimpleClust [72], Plexon Offline Sorter [8]), but such approaches are nearly impossible to standardize due to inherent operator bias [73]. To alleviate this issue, spike sorting has seen decades of algorithmic and software improvements to increase both the accuracy and automation of the process [62]. This progress has accelerated in the past few years as high-density devices [27, 12, 28, 10, 55, 75, 46, 40, 24, 9], capable of recording from hundreds to thousands of neurons simultaneously have made manual intervention impractical, increasing the demand for both accurate and scalable spike sorting algorithms [65, 59, 44, 20, 74, 36, 39, 23].

Despite the development and widespread use of automatic spike sorters, there still exist no clear standards for how spike sorting should be performed or evaluated [62, 11, 18, 47]. Research labs

*alessio.buccino@bsse.ethz.ch

that are beginning to experiment with high-density extracellular recordings have to choose from a multitude of spike sorters, data processing algorithms, file formats, and curation tools just to analyze their first recording. As trying out multiple spike sorting pipelines is time-consuming and technically challenging, many labs choose one and stick to it as their de facto solution [47]. This has led to a fragmented software ecosystem which challenges reproducibility, benchmarking, and collaboration among different research labs.

Previous work to standardize the field has focused on developing open-source frameworks that make extracellular analysis and spike sorting more accessible [26, 16, 35, 29, 33, 13, 45, 14, 57, 43, 48, 15, 61, 76, 56]. While useful tools in their own right, these frameworks only implement a limited suite of spike sorting technologies since their main focus is to provide *entire* extracellular analysis pipelines (spike trains, LFPs, EEG, and more). Moreover, these tools do little to improve the evaluation and comparison of spike sorting performance which is still a relatively unsolved problem in electrophysiology. An exception to this is SpikeForest [47], a recently developed open-source software suite that benchmarks 10 automated spike sorting algorithms against an extensive database of ground truth recordings[1]. Despite these developments, there exists a need for an up-to-date spike sorting framework that can standardize the usage and evaluation of modern algorithms.

In this paper we introduce SpikeInterface, the first open-source, Python-based[2] framework exclusively designed to encapsulate all steps in the spike sorting pipeline. The goals of this software framework are five-fold.

1. To increase the accessibility and standardization of modern spike sorting technologies by providing users with a simple application programming interface (API) and graphical user interface (GUI) that exist within a continuously integrated code-base.

2. To make spike sorting pipelines fully reproducible by capturing the entire provenance of the data flow during run time.

3. To make data access and analysis both memory and computation-efficient by utilizing memory-mapping, parallelization, and high-performance computing platforms.

4. To encourage the sharing of datasets, results, and analysis pipelines by providing full compatibility with standardized file formats such as Neurodata Without Borders (NWB) [71, **?**] and the Neuroscience Information Exchange (NIX) Format [7].

5. To supply the most comprehensive suite of benchmarking capabilities available for spike sorting in order to guide future usage and development.

In the remainder of this article, we showcase the numerous capabilities of SpikeInterface by performing an in-depth meta-analysis of preexisting spike sorters. This analysis includes quantifying the agreement among 6 modern spike sorters for dense probe recordings, benchmarking each sorter on ground truth, and introducing a consensus-based technique to potentially improve performance and enable automated curation. Afterwards, we present an overview of the codebase and how its interconnected components can be utilized to build full spike sorting pipelines. Finally, we contrast SpikeInterface with preexisting analysis frameworks and outline future directions.

2 Results

In this section, we perform a meta-analysis of 6 modern spike sorters on real and simulated datasets. This meta-analysis includes quantifying agreement among the sorters, benchmarking each sorter on ground truth, and investigating whether it is possible to combine outputs from multiple spike sorters to improve overall performance and to reduce the burden of manual curation. All analysis is done

[1] SpikeForest makes use of SpikeInterface in many of its core capabilities (file IO, preprocessing, spike sorting).
[2] We utilize Python as it is open-source, free, and increasingly popular in the neuroscience community [54, 32].

with `spikeinterface` version 0.10.0 which is available on `PyPI`. The code to perform this analysis and produce all figures can be found at https://spikeinterface.github.io/ which also showcases other experiments performed using SpikeInterface. The datasets are publicly available in NWB format on the DANDI archive (https://gui.dandiarchive.org/ – ID: 00034).

2.1 Spike sorters show low agreement for the same high-density dataset

The dataset we use in this analysis is a Neuropixels recording from a head-fixed mouse acquired at the Allen Institute for Brain Science ([69] - Allen Brain Observatory Neuropixels dataset; ©2019 Allen Institute for Brain Science). The recording has a duration of 15 minutes, 246 active recording channels (the remaining of the 384 Neuropixels channels were either not inserted in the brain tissue or below a firing rate of 0.1 Hz), and a sampling frequency of 30 kHz. The probe records from part of the cortex (V1), the hippocampus (CA1), the dentate gyrus, and the thalamus (LP). During the experiment, the mouse was presented with a variety of visual stimuli while freely running on a rotating disk (for more details see [69]). An activity map of the probe and a 1-s snippet of the traces on 10 channels are shown in Figure 1A.

For this analysis, we select six different spike sorters: HerdingSpikes2 [36], Kilosort2 [58], IronClust [39], SpyKING Circus [74], Tridesclous [31], and HDSort [23][3]. As most of these algorithms have been tuned rigorously on multiple ground truth datasets (including the recent large-scale evaluation from [47]), we fix their parameters to default values to allow for straightforward comparison. We do not include Klusta [65], WaveClus [19], Kilosort [59], or MountainSort4 [20] in this analysis as Klusta can only handle up to 64 channels, WaveClus is designed low channel count probes, Kilosort is superseded by Kilosort2, and MountainSort4 does not scale well to recordings with high channel count (such as Neuropixels ones).

In Figure 1B, we show the number of units that each of the 6 sorters output. Immediately, we observe large variability among the sorters, with Tridesclous (TDC) finding the least units (187) and SpyKING Circus (SC) finding the most units (628). HerdingSpikes2 finds 210 units; Kilosort2 finds 446 units; IronClust finds 233 units; and HDSort finds 317 units. From this result, we can see that there is no clear consensus among the sorters on the number of neurons in the recording (without performing extensive manual curation).

Next, we compare the unit spike trains found by each sorter to determine the level of agreement among the different algorithms (see the SpikeComparison Section of the Methods for how this is done). In Figure 1C, we visualize the total number of units for which k sorters agree (unit agreement is defined as a 50% spike train match. The time window to consider spikes as matching is 0.4 ms). Of the 2031 total detected units, all 6 sorters agree on *just 33 of the units*. This is surprisingly low given the relatively undemanding criteria of a 50% spike train match. We also find that two or more sorters agree on just 263 of the total units. To further break down the disagreement between spike sorters, Figure 1D shows the number of units per sorter for which k other sorters agree. For most sorters, over 50% of the units that they find do not match with any other sorter (with the notable exceptions of Ironclust and Tridesclous).

The analysis performed on this dataset suggests that agreement among spike sorters is startlingly low. To corroborate this finding, we repeat the same analysis using different data sets including a Neuropixels recordings from another lab and several *in vitro* retinal recordings from a planar, high-density array. In both cases, we find similar disagreement among the sorters (Figures S2 and S3, the results can also be seen at https://spikeinterface.github.io/). This low agreement raises the following question: how many of the total outputted units actually correspond to real neurons? To explore this question, we turn to simulation where the ground truth spiking activity is known *a priori*.

[3]The versions for each spike sorter are as follows: SpyKING Circus==0.9.7, Tridesclous==1.6.0, hdsort==1.0.0, HerdingSpikes2==0.3.7, IronClust==5.9.8, Kilosort2==GitHub commit 48bf2b81d8ad, HDSort==1.0.1

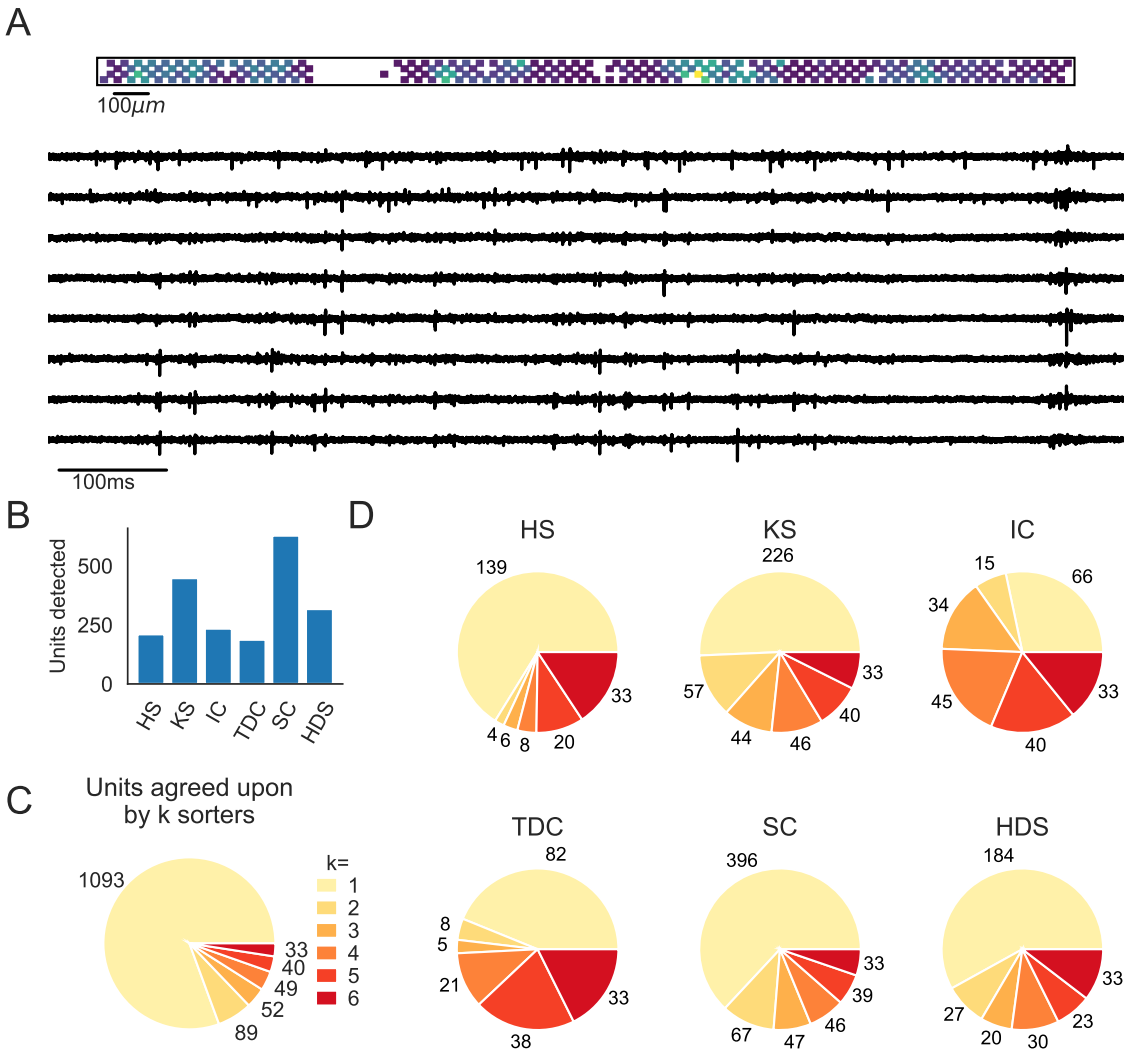

Figure 1: **Comparison of spike sorters on a real Neuropixels dataset**. **A)** A visualization of the activity on the Neuropixels array (top, color indicates spike rate estimated on each channel evaluated with threshold detection) and of traces from the Neuropixels recording (below). **B)** The number of detected units for each of the 6 spike sorters (HS=HerdingSpikes2, KS=Kilosort2, IC=IronClust, TDC=Tridesclous, SC=SpyKING Circus, HDS=HDSort). **C)** The total number of units for which k sorters agree (unit agreement is defined as 50% spike match). **D)** The number of units (per sorter) for which k sorters agree; Most sorters find many units that other sorters do not.

2.2 Evaluating spike sorters on a simulated dataset

In this analysis, we simulate a 10 minute Neuropixels recording using the MEArec Python package [17]. The recording contains the spiking activity of 250 biophysically detailed neurons (200 excitatory and 50 inhibitory cells from the Neocortical Micro Circuit Portal [60, 49]) that exhibit independent Poisson firing patterns. The recording also has an additive Gaussian noise with $10\mu V$ standard deviation. A visualization of the simulated activity map and extracellular traces from the Neuropixels probe is

shown in Figure 2A. A histogram of the signal-to-noise ratios (SNR) for the ground truth units is shown in Figure 2B.

We run the same six spike sorters on the simulated dataset, keeping the parameters the same as those used on the real Neuropixels dataset. We then utilize SpikeInterface to evaluate each spike sorter on the ground-truth dataset. Afterwards, we repeat the agreement analysis from the previous section to diagnose the low agreement among sorters.

The main result of the ground-truth evaluation is summarized in Figure 2. As can be seen in

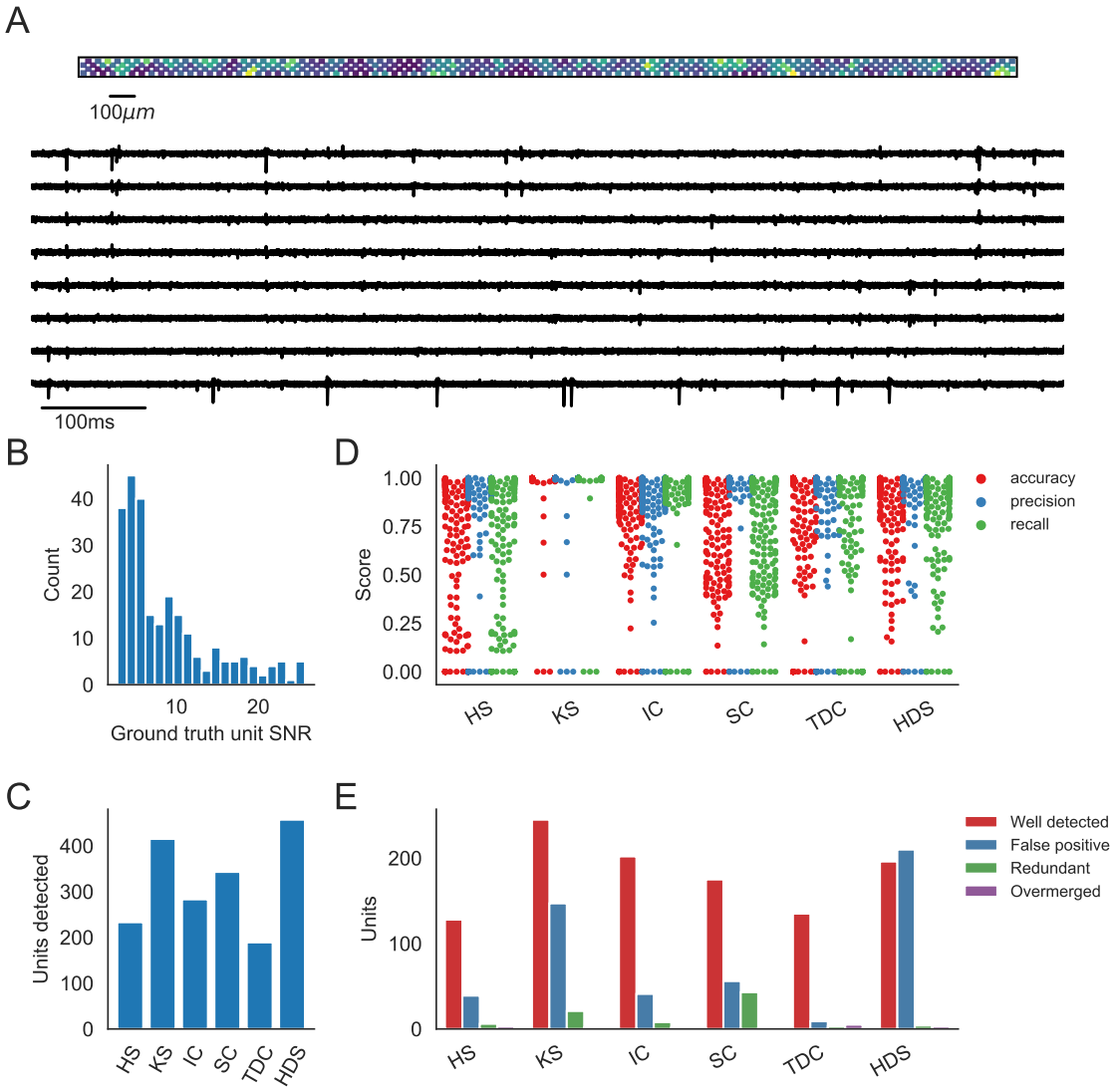

Figure 2: **Evaluation of spike sorters on a Simulated Neuropixels dataset**. **A)** A visualization of the activity on and traces from the simulated Neuropixels recording. **B)** The signal-to-noise ratios (SNR) for the ground truth units. **C)** The number of detected units for each of the 6 spike sorters (HS=HerdingSpikes2, KS=Kilosort2, IC=IronClust, TDC=Tridesclous, SC=SpyKING Circus, HDS=HDSort). **D)** The accuracy, precision, and recall of each sorter on the ground truth units. **E)** A breakdown of the detected units for each sorter (precise definitions of each unit type can be found in the SpikeComparison Section of the Methods).

Figure 2C, the sorters, again, have a large discrepancy in the number of detected units. The number of detected units range from the 189 units found by Tridesclous to the 458 units found by HDSort. HerdingSpikes2 finds 233 units; Kilosort2 finds 415 units; IronClust finds 283 units; and SpyKING Circus finds 343 units. We again see that there is no clear consensus among the sorters on the number of neurons in the simulated recording.

In Figure 2D, the accuracy, precision, and recall of all the ground truth units are plotted for each spike sorter. Some sorters tend to favor precision over recall while others do the opposite (Figure S4A). Moreover, the accuracy is modulated by the SNR of the ground-truth units for all spike sorters except Kilosort2 which achieves an almost perfect performance on the low-SNR units (Figure S4B). While most spike sorters have a wide range of scores for each metric, Kilosort2 attains significantly higher scores than the rest of the spike sorters for most ground-truth units.

Figure 2E shows the breakdown of detected units for each spike sorter. Each unit is classified as *well-detected*, *false positive*, *redundant*, and/or *overmerged* by SpikeInterface (the definitions of each unit type can be found in the SpikeComparison Section of the Methods). This plot, interestingly, may shed some light on the remarkable accuracy of Kilosort2. While Kilosort2 has the most well-detected units (245), this comes at the cost of a high percentage of false positive (147) and redundant (21) units[4]. Notably, Tridesclous detects very few false positive/redundant units while still finding many well-detected units. HDSort, on the flip side, finds many more false positive units than any other spike sorter. For a comprehensive comparison of spike sorter performance on both real and simulated datasets, we refer the reader to the related SpikeForest project (`https://spikeforest.flatironinstitute.org/`) [47].

2.3 Low-agreement units are mainly false positives

Similarly to the real Neuropixels dataset, we compare the agreement among the different spike sorters on the simulated dataset. Again, we observe a large disagreement among the spike sorting outputs with only 139 units of the 1921 total units (7.24%) being in agreement among all sorters (Figure 3A). We can break down the overall agreement by sorter (Figure 3B), highlighting that some sorters are more prone to finding low agreement units (HDSort, SpyKING Circus, Kilosort2) than other sorters (HerdingSpikes2, Ironclust, Tridesclous).

Given that we know the ground-truth spiking activity of the simulated recording, we can now investigate whether low-agreement units actually correspond to ground-truth units or if they are falsely detected (false positive) units. In Figure 3C, bar plots for each sorter show the number of matched ground-truth units (blue) and false positive units (red) in relation to the ensemble agreement (1 - no agreement, 6 - full agreement). The plots show that (almost) all false positive units are ones that are found by only a single sorter (not matched with any other sorters), while most real units are matched by more than one sorter. Furthermore, in Figure 3D we display the signal-to-noise ratio (SNR) as a function of the ensemble agreement. This shows, as expected, that higher SNR units have higher agreement among sorters. In other words, units with a large amplitude (high SNR) are easier to detect and more consistently found by many sorters. Taken together, these results suggest that the ensemble agreement among multiple sorters can be used to remove false positive units from each of the sorter outputs (or to at least inform the subsequent manual curation).

2.4 Consensus units highly overlap with manually curated ones

We next investigate the ensemble agreement among the sorters on the real Neuropixels recording presented in Figure 1. As there is no ground truth information in this setting to identify false positives,

[4]The high-rate of false positive/redundant units persists, but is alleviated, even when using Kilosort2's automated curation step which removes units that have >20% estimated contamination rate (computed from the refractory period violations). In that case the number of well-detected units is 241, false positives are 93, and redundant units are 18. In both cases 2 overmerged units are found.

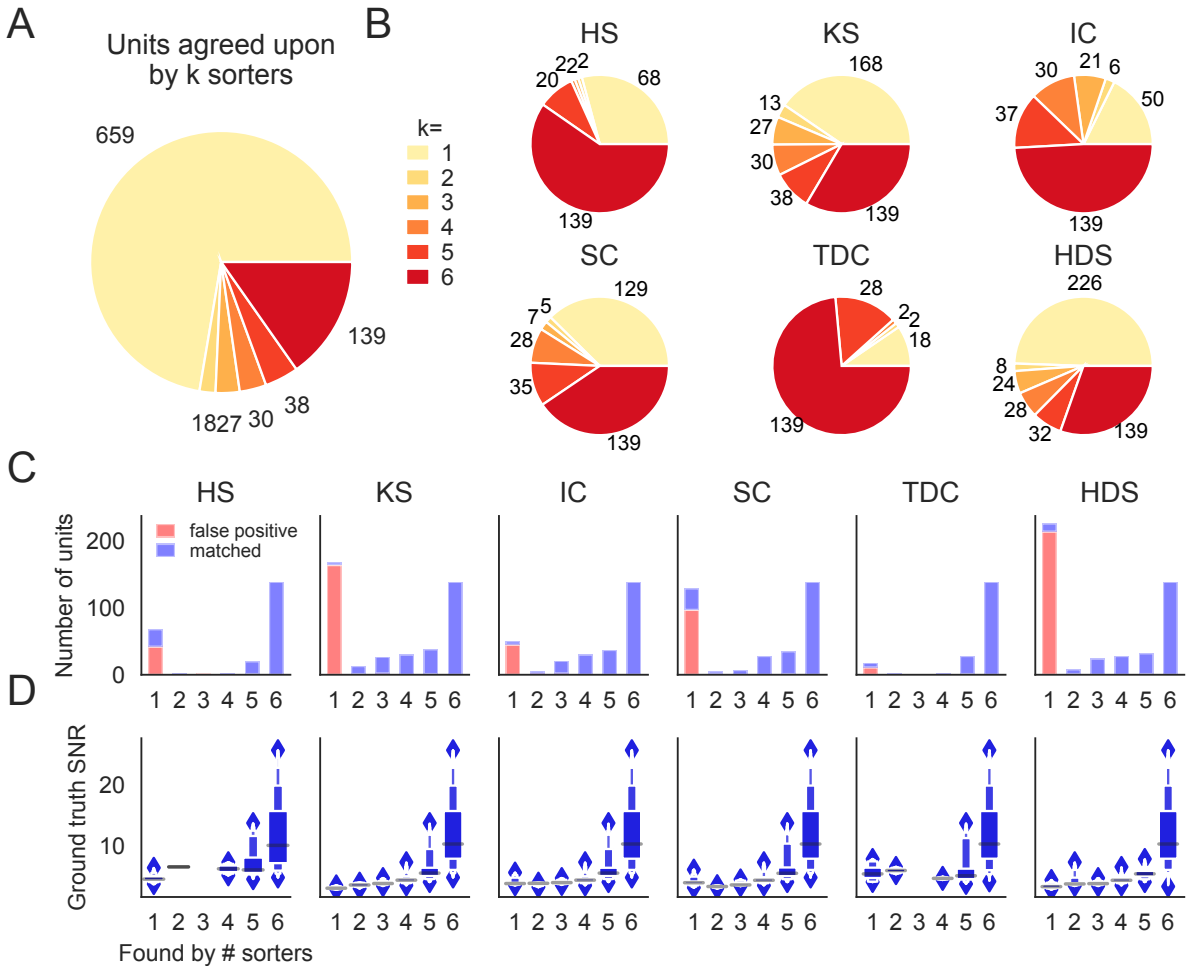

Figure 3: **Comparison of spike sorters on a simulated Neuropixels dataset**. **A)** The total number of units for which k sorters agree (unit agreement is defined as 50% spike match). **B)** The number of units (per sorter) for which k sorters agree; Most sorters find many units that other sorters do not. (HS=HerdingSpikes2, KS=Kilosort2, IC=IronClust, TDC=Tridesclous, SC=SpyKING Circus, HDS=HDSort) **C)** Number of matched ground-truth units (blue) and false positive units (red) found by each sorter on which k sorters agree upon. Most of the false positive units are only found by a single sorter. Number of false positive units found by $k \geq 2$ sorters: HS=4, KS=4, IC=4, SC=2, TDC=1, HDS=2. **D)** Signal-to-noise ratio (SNR) of ground-truth unit with respect to the number of k sorters agreement. Results are split by sorter.

we turn to manually curated sorting outputs. Two experts (which we will refer to as C1 and C2) manually curate the spike sorting output of Kilosort2 using the Phy software. During this curation step, the two experts label the sorted units as false positives or real units by rejecting, splitting, merging, or accepting units according to spike features [63].

Figure 4A shows the agreement between expert 1 (C1) and expert 2 (C2). While there are some discrepancies (as expected when manually curating spike sorting results [73]), most of the curated units (207 out of 351 - 64.2%) are agreed upon by both experts. Notably, 174 units found by Kilosort2 are discarded by both experts, indicating a large number of false positive units.

We then compare the output of each of the spike sorters to C1 and C2 and find that, in general, only a small percentage of units outputted by any single sorter is matched to the curated results (Figure 4).

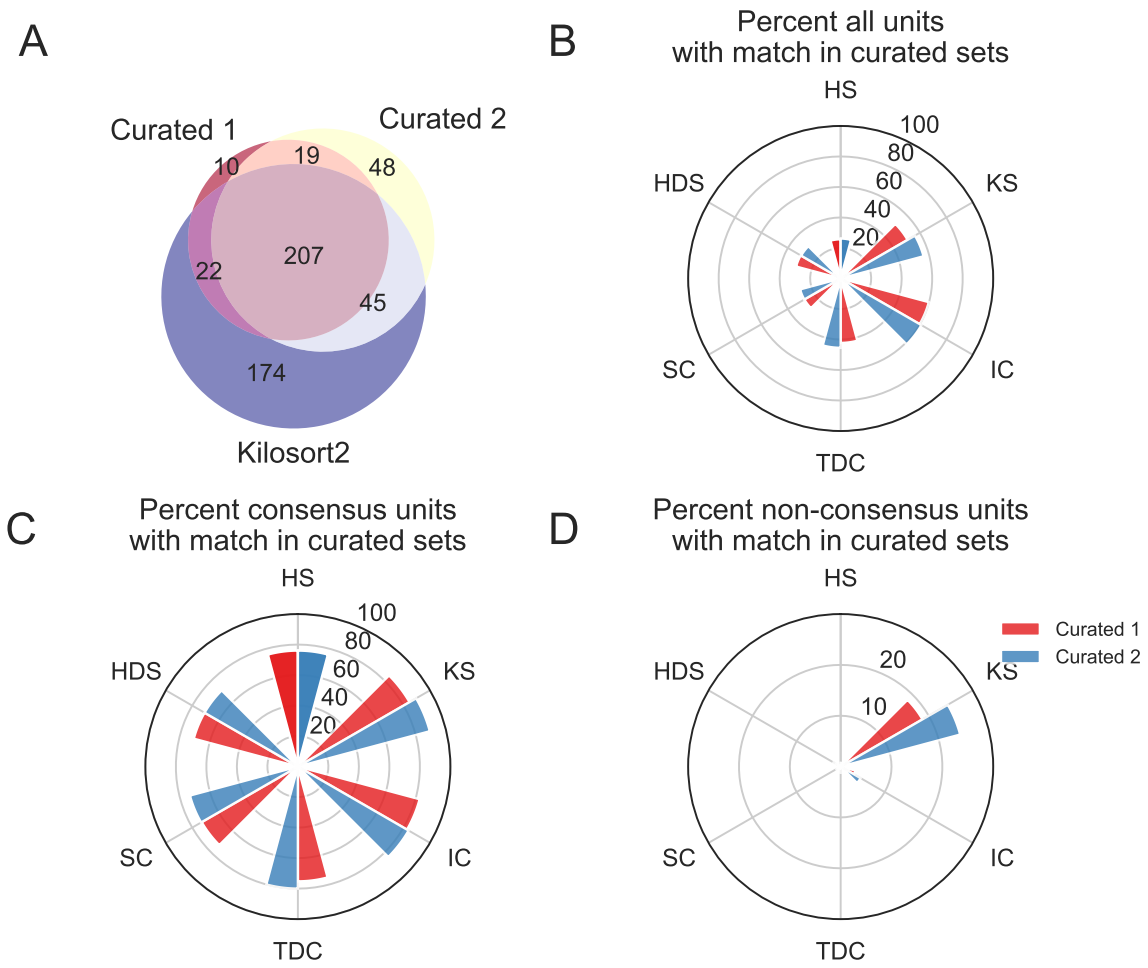

Figure 4: **Comparison between consensus and manually curated outputs**. **A)** Venn diagram showing the agreement between Curator 1 and 2. 174 units are discarded by both curators from the Kilsort2 output. **B)** Percent of matched units between the output of each sorter and C1 (red) and C2 (blue). Ironclust has the highest match with both curated datasets. **C)** Similar to **C**, but using the consensus units (units agreed upon by at least 2 sorters - $k \geq 2$). The percent of matching with curated datasets is now above 70% for all sorters, with Kilosort2 having the highest match ($\text{KS}_c \cap \text{C1} = 84.55\%$, $\text{KS}_c \cap \text{C2} = 89.55\%$), slightly higher than Ironclust ($\text{IC}_c \cap \text{C1} = 82.63\%$, $\text{IC}_c \cap \text{C2} = 83.83\%$). **D)** Percent of non-consensus units ($k = 1$) matched to curated datasets. The only significant overlap is between Curator 1 and Kilosort2, with a percent around 18% ($\text{KS}_{nc} \cap \text{C1} = 18.58\%$, $\text{KS}_{nc} \cap \text{C2} = 24.34\%$).

The highest percentage match is actually IronClust which is surprising given that the initial sorting output was curated from Kilosort2's output ($\text{IC} \cap \text{C1} = 59.83\%$, $\text{IC} \cap \text{C2} = 61.1\%$, $\text{KS} \cap \text{C1} = 50.67\%$, $\text{KS} \cap \text{C2} = 56.25\%$).

Next, for each sorter, we take all the units that are matched by at least one other sorter ($k \geq 2$ - *consensus units*) and all units that are found by only that sorter ($k = 1$ - *non-consensus units*). We refer to the consensus units of a sorter as Sorter_c and the non-consensus units of a sorter as Sorter_{nc}. In Figure 4C, we show the match percentage between consensus units and curated units. The average match percentage is above 70% for all sorters showing that there is a large agreement between the

manually curated outputs and the consensus-based output. Kilosort2 has the highest match ($KS_c \cap C1 = 84.55\%$, $KS_c \cap C2 = 89.55\%$), slightly higher than Ironclust ($IC_c \cap C1 = 82.63\%$, $IC_c \cap C2 = 83.83\%$). Conversely, the percentage of non-consensus units matched to curated units is very small (Figure 4D) for all sorters with the exception of C1 and the Kilosort2 output ($KS_{nc} \cap C1 = 18.58\%$, $KS_{nc} \cap C2 = 24.34\%$).

Overall, this analysis suggests that a consensus-based approach to curation could allow for identification of real neurons from spike sorted data. Despite differences among the sorters with respect to the number of detected neurons and the quality of their isolation (as demonstrated by the ground truth analysis), the consensus-based approach has excellent agreement with hand-curated data and appears to be less variable as illustrated by the small but significant disagreement between the two curators.

3 Methods

3.1 Overview of SpikeInterface

SpikeInterface consists of five main Python packages designed to handle different steps in the spike sorting pipeline: (i) `spikeextractors`, for extracellular recording, sorting output, and probe file I/O; (ii) `spiketoolkit` for low level processing such as pre-processing, post-processing, validation, curation; (iii) `spikesorters` for spike sorting algorithms and job launching functionality; (iv) `spikecomparison` for sorter comparison, ground-truth comparison, and ground-truth studies; and (v) `spikewidgets`, for data visualization.

These five packages can be installed and used through the `spikeinterface` metapackage, which contains stable versions of all five packages as internal modules (see Figure 5). With these five packages (or our meta-package), users can build, run, and evaluate full spike sorting pipelines in a reproducible and standardized way. In the following subsections, we present an overview of, and a code snippet for, each package.

3.1.1 SpikeExtractors

The `spikeextractors` package[5] is designed to alleviate issues of any file format incompatibility within spike sorting without creating additional file formats. To this end, `spikeextractors` contains two core Python objects that can directly and uniformly access all spike sorting related files: the `RecordingExtractor` and the `SortingExtractor`.

[5]https://github.com/SpikeInterface/spikeextractors

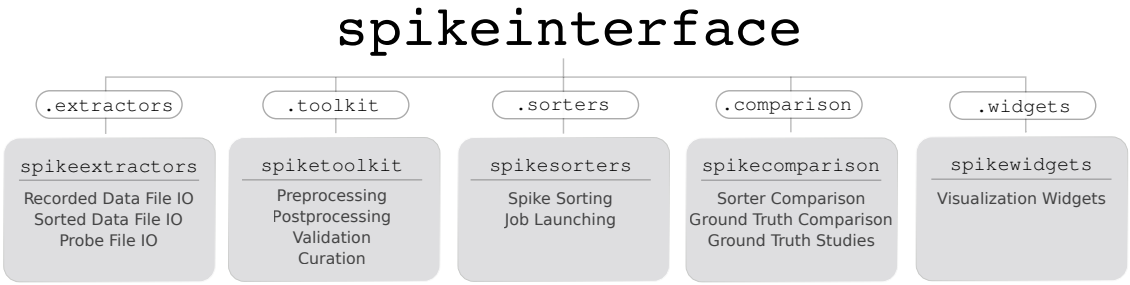

Figure 5: Overview of SpikeInterface's Python packages, their different functionalities, and how they can be accessed by our meta-package, `spikeinterface`.

The `RecordingExtractor` directly interfaces with an extracellular recording and can query it for four primary pieces of information: (i) the extracellular recorded traces; (ii) the sampling frequency; (iii) the number of frames, or duration, of the recording; and (iv) the channel indices of the recording electrodes. These data are shared across all extracellular recordings allowing for standardized retrieval functions. In addition, a `RecordingExtractor` may store extra information about the recording device as "channel properties" which are key–value pairs. This includes properties such as "location", "group", and "gain" which are either provided by certain extracellular file formats, loaded manually by the user, or loaded automatically with our built-in probe file (.prb or .csv) reader. Taken together, the `RecordingExtractor` is an object representation of an extracellular recording and the associated probe configuration.

The `SortingExtractor` directly interfaces with a sorting output and can query it for two primary pieces of information: (i) the unit indices; and (ii) the spike train of each unit. Again, these data are shared across all sorting outputs. A `SortingExtractor` may also store extra information about the sorting output as either "unit properties" or "unit spike features", key–value pairs which store information about the individual units or the individual spikes of each unit, respectively. This extra information is either loaded from the sorting output, loaded manually by the user, or loaded automatically with built-in post-processing tools (discussed in the SpikeToolkit Section). Taken together, the `SortingExtractor` is an object representation of a sorting output along with any associated post-processing.

Critically, both `Extractor` types can lazily query the underlying datasets for information as it is required, reducing their memory footprint and allowing their use for long, large-scale recordings. While this is the default operation mode, `Extractors` can also cache parts of the dataset in temporary binary files to enable faster downstream computations at the cost of higher memory usage. All extracted data is converted into either native Python data structures or into numpy arrays for immediate use in Python. Additionally, each `Extractor` can be dumped to and loaded from a `json` file, a `pickle` file, or a dictionary, ensuring full provenance and allowing for parallel processing.

The following code snippet illustrates `Extractors` can be used to retrieve raw traces from an extracellular recording and spike trains from a sorting output:

```
import spikeinterface.extractors as se
recording = se.MyFormatRecordingExtractor(file_path='myrecording')
sorting = se.MyFormatSortingExtractor(file_path='mysorting')
traces = recording.get_traces() #2D numpy array (channels x time)
spike_train = sorting.get_unit_spike_train(unit_id=1) #1D numpy array
```

Along with using `Extractors` for single files, it is possible to access data from multiple files or portions of files with the `MultiExtractors` and `SubExtractors`, respectively. Both have identical functionality to normal `Extractors` and can be used and treated in the same ways, simplifying, for instance, the combined analysis of a recording split into multiple files.

As of this moment, SpikeInterface supports 19 extracellular recording formats and 18 sorting output formats. The available file formats can be found in Table 1. Although this covers many popular formats in extracellular analysis (including Neurodata Without Borders [71] and NIX [7]), we expect the number of formats to grow with future versions as adding a new format is as simple as making a new `Extractor` subclass for it. We also have started to integrate NEO's [30] I/O system into `spikeextractors` which allow SpikeInterface to support many more open-source and proprietary file formats without changing any functionality. Already, two recording formats have been added through our NEO integration (Neuralynx [6] and Plexon [8]).

Raw Formats	Writable	Reference	Sorted Formats	Writable	Reference
Klusta	Yes	[65]	Klusta	Yes	[65]
Mountainsort	Yes	[40]	Mountainsort	Yes	[40]
Phy*	Yes	[63]	Phy*	Yes	[63]
Kilosort/Kilosort2	No	[59, 64]	Kilosort/Kilosort2	No	[59, 64]
SpyKING Circus	No	[74]	SpyKING Circus	Yes	[74]
Exdir	Yes	[25]	Exdir	Yes	[25]
MEArec	Yes	[17]	MEArec	Yes	[17]
Open Ephys	No	[70]	Open Ephys	No	[70]
Neurodata Without Borders	Yes	[71]	Neurodata Without Borders	Yes	[71]
NIX	Yes	[7]	NIX	Yes	[7]
Plexon	No	[8]	Plexon	No	[8]
Neuralynx	No	[6]	Neuralynx	No	[6]
SHYBRID	Yes	[?]	SHYBRID	Yes	[?]
Neuroscope	Yes	[35]	Neuroscope	Yes	[35]
SpikeGLX	No	[41]	HerdingSpikes2	Yes	[36]
Intan	No	[2]	JRCLUST	No	[39]
MCS H5	No	[5]	Wave Clus	No	[19]
Biocam HDF5	Yes	[1]	Tridesclous	No	[31]
MEA1k	Yes	[4]	NPZ (numpy zip)	Yes	N/A
MaxOne	No	[3]			
Binary	Yes	N/A			

Table 1: Currently available file formats in SpikeInterface and if they are writable. *The Phy writing method is implemented in `spiketoolkit` as the `export_to_phy` function (all other writing methods are implemented in `spikeextractors`).

3.1.2 SpikeToolkit

The `spiketoolkit` package[6] is designed for efficient pre-processing, post-processing, validation, and curation of extracellular datasets and sorting outputs. It contains four modules that encapsulate each of these functionalities: `preprocessing`, `postprocessing`, `validation`, and `curation`.

Pre-processing. The `preprocessing` module provides functions to process raw extracellular recordings before spike sorting. To pre-process an extracellular recording, the user passes a `RecordingExtractor` to a pre-processing function which returns a new "preprocessed" `RecordingExtractor`. This new `RecordingExtractor`, which can be used in exactly the same way as the original extractor, implements the preprocessing in a *lazy* fashion so that the actual computation is performed only when data

[6] https://github.com/SpikeInterface/spiketoolkit

is requested. As all pre-processing functions take in and return a `RecordingExtractor`, they can be naturally chained together to perform multiple pre-processing steps on the same recording.

Pre-processing functions range from commonly used operations, such as bandpass filtering, notch filtering, re-referencing signals, and removing channels, to more advanced procedures such as clipping traces depending on the amplitude, or removing artifacts arising, for example, from electrical stimulation. The following code snippet illustrates how to chain together a few common pre-processing functions to process a raw extracellular recording:

```
import spikeinterface.spiketoolkit as st
recording = st.preprocessing.bandpass_filter(recording, freq_min=300, freq_max=6000)
recording_1 = st.preprocessing.remove_bad_channels(recording, bad_channels=[5])
recording_2 = st.preprocessing.common_reference(recording_1, reference='median')
```

Post-processing. The `postprocessing` module provides functions to compute and store information about an extracellular recording given an associated sorting output. As such, post-processing functions are designed to take in both a `RecordingExtractor` and a `SortingExtractor`, using them in conjunction to compute the desired information. These functions include, but are not limited to: extracting unit waveforms and templates, computing principle component analysis projections, as well as calculating features from templates (e.g. peak to valley duration, full-width half maximum).

One essential feature of the `postprocessing` module is that it provides the functionality to export a `RecordingExtractor` / `SortingExtractor` pair into the `Phy` format for manual curation later. `Phy` [63, 65] is a popular manual curation GUI that allows users to visualize a sorting output with several views and to curate the results by manually merging or splitting clusters. `Phy` is already supported by several spike sorters (including `klusta`, `Kilosort`, `Kilosort2`, and `SpyKING Circus`) so our exporter function extends `Phy`'s functionality to all SpikeInterface-supported spike sorters. After manual curation is performed in `Phy`, the curated data can be re-imported into SpikeInterface using the `PhySortingExtractor` for further analysis. The following code snippet illustrates how to retrieve waveforms for each sorted unit, compute principal component analysis (PCA) features for each spike, and export to `Phy` using SpikeInterface:

```
import spikeinterface.toolkit as st
waveforms = st.postprocessing.get_unit_waveforms(recording, sorting)
pca_scores = st.postprocessing.compute_unit_pca_scores(recording, sorting, n_comp=3)
st.postprocessing.export_to_phy(recording, sorting_MS4, output_folder='phy_folder')
```

Validation. The `validation` module allows users to automatically evaluate spike sorting results in the absence of ground truth with a variety of quality metrics. The quality metrics currently available are a compilation of historical and modern approaches that were re-implemented by researchers at Allen Institute for Brain Science[7] and by the SpikeInterface team (see Table 2).

Each of SpikeInterface's quality metric functions internally utilize the `postprocessing` module to generate all data needed to compute the specified metric (amplitudes, principal components, etc.). The following code snippet demonstrates how to compute both a single quality metric (isolation distance) and also *all* the quality metrics with just two function calls:

```
import spikeinterface.toolkit as st
iso_metric = st.validation.compute_isolation_distances(sorting, recording)
all_metrics = st.validation.compute_quality_metrics(sorting, recording)
```

[7] https://github.com/AllenInstitute/ecephys_spike_sorting

Metric	Description	Reference
Signal-to-noise ratio	The signal-to-noise ratio computed on unit templates.	N/A
Firing rate	The average firing rate over a time period.	N/A
Presence ratio	The fraction of a time period in which spikes are present.	N/A
Amplitude Cutoff	An estimate of the miss rate based on an amplitude histogram.	N/A
Maximum drift	The maximum change in spike position (computed as the center of mass of the energy of the first principal component score) throughout a recording.	N/A
Cumulative drift	The cumulative change in spike position throughout a recording.	N/A
ISI violations	The rate of inter-spike-interval (ISI) refractory period violations.	[37]
Isolation Distance	Radius of the smallest ellipsoid that contains *all* the spikes from a cluster and an equal number of spikes from other clusters (centered on the specified cluster).	[34]
L-ratio	Assuming that the distribution of spike distances from a cluster center is multivariate normal, L-ratio is the average value of the tail distribution for non-member spikes of that cluster.	[68]
D-Prime	The classification accuracy between two units based on linear discriminant analysis (LDA)	[37]
Nearest-neighbors	A non-parametric estimate of unit contamination using nearest-neighbor classification.	[20]
Silhouette score	The ratio between cohesiveness of a cluster (distance between member spikes) and its separation from other clusters (distance to non-member spikes).	[66]

Table 2: Currently available quality metrics in Spikeinterface. Re-implemented by researchers at Allen Institute for Brain and by the SpikeInterface team.

Curation. The `curation` module allows users to quickly remove units from a `SortingExtractor` based on computed quality metrics. To curate a sorted dataset, the user passes a `SortingExtractor` to a curation function which returns a new "curated" `SortingExtractor` (similar to how preprocessing works). This new `SortingExtractor` can be used in exactly the same way as the original extractor. As all curation functions take in and return a `SortingExtractor`, they can be naturally chained together to perform multiple curation steps on the same sorting output.

Currently, all implemented curation functions are based on excluding units with respect to a user-defined threshold on a specified quality metric. These curation functions will compute the associated quality metric and then threshold the dataset accordingly. The following code snippet demonstrates how to chain together two curation functions that are based on different quality metrics and apply a "less" threshold to the underlying units (exclude all units below the given threshold):

```
import spikeinterface.toolkit as st
sorting_1 = st.curation.threshold_firing_rates(sorting, threshold=2.3,
                                               threshold_sign='less')
```

Name	Method	Notes	Reference
Klusta	DB	Python-based, semi-automatic, designed for low channel count, dense probes.	[65]
Mountainsort4	DB	Python-based, fully automatic, unique clustering method (isosplit), designed for low channel count, dense probes and tetrodes.	[20]
Kilosort	TM	MATLAB-based, GPU support, semi-automated final curation.	[59]
Kilosort2	TM	MATLAB-based, GPU support, semi-automated final curation, designed to correct for drift.	[58]
SpyKING Circus	TM	Python-based, fast and scalable with CPUs, designed to correct for drift.	[74]
HerdingSpikes2	DB + SL	Python-based, fast and scalable with CPUs, scales up to thousands of channels.	[36]
Tridesclous	TM	Python-based, graphical user interface, GPU support, multi-platform	[31]
IronClust	DB + SL	MATLAB-based, GPU support, designed to correct for drift.	[38]
Wave Clus	TM	Matlab-based, fully automatic, designed for single electrodes and tetrodes, multi-platform.	[19]
HDsort	TM	Matlab-based, fast and scalable, designed for large-scale, dense arrays.	[23]

Table 3: Currently available spike sorters in Spikeinterface. TM = Template Matching; SL = Spike Localisation; DB = Density-based clustering.

```
sorting_2 = st.curation.threshold_snrs(sorting_1, recording, threshold=10,
                                       threshold_sign='less')
```

3.1.3 SpikeSorters

The `spikesorters`[8] package provides a straightforward interface for running spike sorting algorithms supported by SpikeInterface. Modern spike sorting algorithms are built and deployed in a variety of programming languages including C, C++, MATLAB, and Python. Along with variability in the underlying program languages, each sorting algorithm may depend on external technologies like CUDA or command line interfaces (CLIs), complicating standardization. To unify these disparate algorithms into a single codebase, `spikesorters` provides Python-wrappers for each supported spike sorting algorithm. These spike sorting wrappers use a standard API for running the corresponding algorithms, internally handling intrinsic complexities such as automatic code generation for MATLAB- and CLI-based algorithms. Each spike sorting wrapper is implemented as a subclass of a `BaseSorter` class that contains all shared code for running the spike sorters.

[8] `https://github.com/SpikeInterface/spikesorters`

To run a specific spike sorting algorithm, users can pass a `RecordingExtractor` object to the associated function in `spikesorters` and overwrite any default parameters with new values (only essential parameters are exposed to the user for modification). Internally, each function initializes a spike sorting wrapper with the user-defined parameters. This wrapper then creates and modifies a new spike sorter configuration and runs the sorter on the dataset encapsulated by the `RecordingExtractor`. Once the spike sorting algorithm is finished, the sorting output is saved and a corresponding `SortingExtractor` is returned to the user. For each sorter, all available parameters and their descriptions can be retrieved using the `get_default_params()` and `get_params_description()` functions, respectively.

In the following code snippet, Mountainsort4 and Kilosort2 are used to sort an extracellular recording. Running each algorithm (and changing the default parameters) can be done as follows:

```
import spikeinterface.sorters as ss
sorting_MS4 = ss.run_mountainsort4(recording, adjacency_radius=50)
sorting_KS2 = ss.run_kilosort2(recording, detect_threshold=5)
```

Our spike sorting functions also allow for users to sort specific "groups" of channels in the recording separately (and in parallel, if specified). This can be very useful for multiple tetrode recordings where the data are all stored in one file, but the user wants to sort each tetrode separately. For large-scale analyses where the user wants to run many different spike sorters on many different datasets, `spikesorters` provides a launcher function which handles any internal complications associated with running multiple sorters and returns a nested dictionary of `SortingExtractor` objects corresponding to each sorting output. The launcher can be deployed on HPC platforms through the `multiprocessing` or `dask` engine [21]. Finally, and importantly, when running a spike sorting job the recording information and all the spike sorting parameters are saved in a log file, including the console output of the spike sorting run (which can be used to inspect errors). This provenance mechanism ensures full reproducibility of the spike sorting pipeline.

Currently, SpikeInterface supports 10 semi-automated spike sorters which are listed in Table 3. We encourage developers to contribute to this expanding list in future versions and we provide comprehensive documentation on how to do so[9].

3.1.4 SpikeComparison

The `spikecomparison` package[10] provides a variety of tools that allow users to compare and benchmark sorting outputs. Along with these comparison tools, `spikecomparison` also provides the functionality to run systematic performance comparisons of multiple spike sorters on multiple ground-truth recordings.

Within `spikecomparison`, there exist three core comparison functions:

1. `compare_two_sorters` - Compares two spike sorting outputs.
2. `compare_multiple_sorters` - Compares multiple spike sorting outputs.
3. `compare_sorter_with_ground_truth` - Compares a spike sorting output to ground truth.

Each of these comparison functions takes in multiple `SortingExtractor` objects and uses them to compute agreement scores among the underlying spike trains. The agreement score between two

[9] https://spikeinterface.readthedocs.io/en/latest/contribute.html
[10] https://github.com/SpikeInterface/spikecomparison

spike trains is defined as:

$$score = \frac{\#n_{matches}}{\#n_1 + \#n_2 - \#n_{matches}} \tag{1}$$

where $\#n_{matches}$ is the number of "matched" spikes between the two spike trains and $\#n_1$ and $\#n_2$ are the number of spikes in the first and second spike train, respectively. Two spikes from two different spike trains are "matched" when they occur within a certain time window of each other (this window length can be adjusted by the user and is 0.4 ms by default).

When comparing two sorting outputs (`compare_two_sorters`), a linear assignment based on the Hungarian method [42] is used. With this assignment method, each unit from the first sorting output can be matched to at most one other unit in the second sorting output. The final result of this comparison is then the list of matching units (given by the Hungarian method) and the agreement scores of the spike trains.

The multi-sorting comparison function (`compare_multiple_sorters`) can be used to compute the agreement among the units of many sorting outputs at once. Internally, pair-wise sorter comparisons are run for all of the sorting output pairs. A graph is then built with the sorted units as nodes and the agreement scores among the sorted units as edges. With this graph implementation, it is straightforward to query for units that are in agreement among multiple sorters. For example, if three sorting outputs are being compared, any units that are in agreement among all three sorters will be part of a subgraph with large weights.

For a ground-truth comparison (`compare_sorter_with_ground_truth`), either the Hungarian or the best-match method can be used. With the Hungarian method, each tested unit from the sorting output is matched to at most a single ground-truth unit. With the best-match method, a tested unit from the sorting output can be matched to multiple ground-truth units (above an adjustable agreement threshold) allowing for more in-depth characterizations of sorting failures. Note that in the SpikeForest benchmarking software suite [47], the best-match strategy is used.

Additionally, when comparing a sorting output to a ground-truth sorted result, each spike can be optionally labeled as:

- True positive (tp): Found both in the ground-truth spike train and tested spike train.
- False negative (fn): Found in the ground-truth spike train, but not in the tested spike train.
- False positive (fp): Found in the tested spike train, but not in the ground-truth spike train.

Using these labels, the following performance measures can be computed:

- Accuracy: $\frac{\#tp}{(\#tp + \#fn + \#fp)}$
- Recall: $\frac{\#tp}{(\#tp + \#fn)}$
- Precision: $\frac{\#tp}{(\#tp + \#fp)}$
- Miss rate: $\frac{\#fn}{(\#tp + \#fn)}$
- False discovery rate: $\frac{\#fp}{(\#tp + \#fp)}$

While previous metrics give a measure of individual spike train quality, we also propose metrics at a unit population level. Based on the matching results and the scores, the units of the sorting output are classified as *well-detected*, *false positive*, *redundant*, and *overmerged*. Well-detected units are matched

units with an agreement score above 0.8. False positive units are unmatched units or units which are matched with an agreement score below 0.2. Redundant units have agreement scores above 0.2 with only one ground-truth unit, but are not the best matched tested units (redundant units can either be oversplit or duplicate units). Overmerged units have an agreement score above 0.2 with two or more ground-truth units. All these agreement score thresholds are adjustable by the user. We would like to highlight to the reader that the unit classification proposed here is currently only based on agreement score (i.e. accuracy). More sophisticated classification rules could involve a combination of accuracy, precision, and recall values, which can be easily computed for each unit with the `spikecomparison` module.

The following code snippet shows how to perform all three types of spike sorter comparisons:

```
import spikeinterface.comparison as sc
comp_type_1 = sc.compare_two_sorters(sorting1, sorting2)
comp_type_2 = sc.compare_multiple_sorters([sorting1, sorting2, sorting3])
comp_type_3 = sc.compare_sorter_with_ground_truth(gt_sorting, tested_sorting)
```

Along with the three comparison functions, `spikecomparison` also includes a `GroundTruthStudy` class that allows for the systematic comparison of multiple spike sorters on multiple ground-truth datasets. With this class, users can set up a study folder (in which the recordings to be tested are saved), run several spike sorters and store their results in a compact way, perform systematic ground-truth comparisons, and aggregate the results in `pandas` dataframes [52].

3.1.5 SpikeWidgets

The `spikewidgets` package[11] implements a variety of widgets that allow for efficient visualization of different elements in a spike sorting pipeline.

There exist four categories of widgets in `spikewidgets`. The first category utilizes a `RecordingExtractor` for its visualization. This category includes widgets for visualizing time series data, electrode geometries, signal spectra, and spectrograms. The second category utilizes a `SortingExtractor` for its visualization. These widgets include displays for raster plots, auto-correlograms, cross-correlograms, and inter-spike-interval distributions. The third category utilizes both a `RecordingExtractor` and a `SortingExtractor` for its visualization. These widgets include visualizations of unit waveforms, amplitude distributions for each unit, amplitudes of each unit over time, and PCA features. The fourth category utlizes comparison objects from the `spikecomparison` package for its visualization. These widgets allow the user to visualize confusion matrices, agreement scores, spike sorting performance metrics (e.g. accuracy, precision, recall) with respect to a unit property (e.g. SNR), and the agreement between multiple sorting algorithms on the same dataset.

The following code snippet demonstrates how SpikeInterface can be used to visualize ten seconds of both the extracellular traces and the corresponding raster plot:

```
import spikeinterface.widgets as sw
sw.plot_timeseries(recording, channel_ids=[0,1,2,3], trange=[0,10])
sw.plot_rasters(sorting, unit_ids=[0,1,3], trange=[0,10])
```

3.2 Building a spike sorting pipeline

So far, we have given an overview of each of the main packages in isolation. In this section, we illustrate how these packages can be combined, using both the Python API and the `Spikely` GUI, to build

[11] https://github.com/SpikeInterface/spikewidgets

a robust spike sorting pipeline. The spike sorting pipeline that we construct using SpikeInterface is depicted in Figure 6A and consists of the following analysis steps:

1. Loading an Open Ephys recording [70].
2. Loading a probe file.
3. Applying a bandpass filter.
4. Applying common median referencing to reduce the common mode noise.
5. Spike sorting with `Mountainsort4`.
6. Removing clusters with less than 100 events.
7. Exporting the results to `Phy` for manual curation.

Traditionally, implementing this pipeline is challenging as the user has to load data from multiple file formats, interface with a probe file, memory-map all the processing functions, prepare the correct inputs for Mountainsort4, and understand how to export the results into `Phy`. Even if the user manages to implement all of the analysis steps on their own, it is difficult to verify their correctness or reuse them without proper unit testing and code reviewing.

3.2.1 Using the Python API

Using SpikeInterface's Python API to build the pipeline shown in Figure 6A is straightforward. Each of the seven steps is implemented with a single line of code (as shown in Figure 6B). Additionally, data visualizations can be added for each step of the pipeline using the appropriate widgets (as described in the SpikeWidgets Section). Unlike handmade scripts, SpikeInterface has a wide range of unit tests, employs continuous integration, and has been carefully developed by a team of researchers. Users, therefore, can have increased confidence that the pipelines they create are correct and reusable. Additionally, SpikeInterface tracks the entire provenance of the performed analysis, allowing other users (or the same user) to reproduce the analysis at a later date.

3.2.2 Using the spikely GUI

Along with our Python API, we also developed `spikely`[12], a PyQt-based GUI that allows for simple construction of complex spike sorting pipelines. With `spikely`, users can build workflows that include: (i) loading a recording and a probe file; (ii) performing pre-processing on the underlying recording with multiple processing steps; (iii) running any spike sorter supported by SpikeInterface on the processed recording; (iv) automatically curating the sorter's output; and (v) exporting the final result to a variety of file formats, including `Phy`. At its core, `spikely` utilizes SpikeInterface's Python API to run any constructed spike sorting workflow. This ensures that the functionality of `spikely` grows organically with that of SpikeInterface.

Figure 6C shows a screenshot from `spikely` where the pipeline in Figure 6A is constructed. Each stage of the pipeline is added using drop-down lists, and all the parameters (which were not left at their default values) are set in the right-hand panel. Once a pipeline is constructed in `spikely`, the user can save it using the built-in save functionality and then load it back into `spikely` at a later date. Since `spikely` is cross-platform and user-friendly, we believe it can be utilized to increase the accessibility and reproducibility of spike sorting.

[12] `https://github.com/SpikeInterface/spikely`

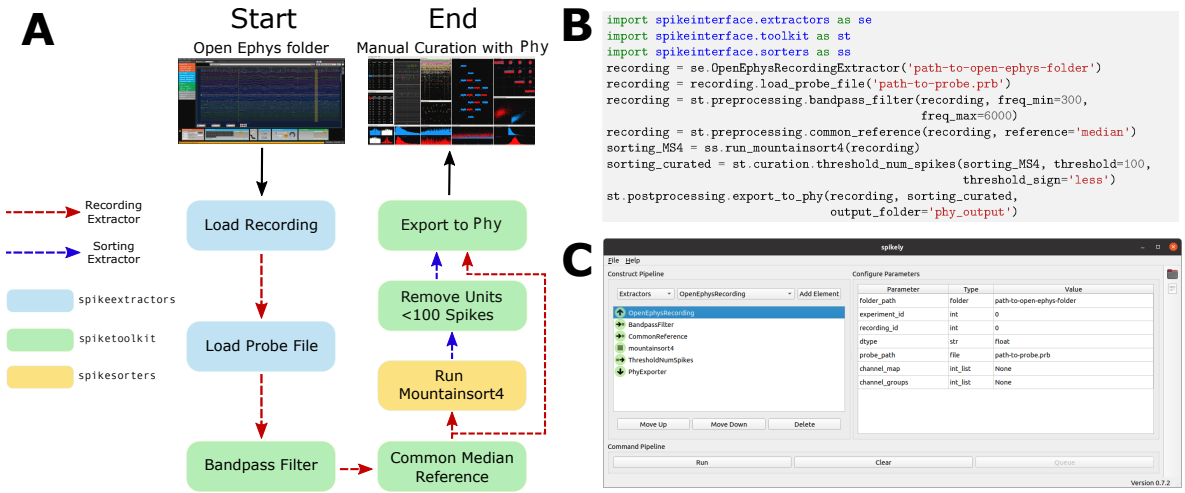

Figure 6: Sample spike sorting pipeline using SpikeInterface. (A) A diagram of a sample spike sorting pipeline. Each processing step is colored to represent the SpikeInterface package in which it is implemented and the dashed, colored arrows demonstrate how the `Extractors` are used in each processing step. (B) How to use the Python API to build the pipeline shown in (A). (C) How to use the GUI to build the pipeline shown in (A).

4 Discussion

In this paper, we introduced SpikeInterface, a Python framework designed to enhance the accessibility, reliability, efficiency, and reproducibility of spike sorting. To illustrate the use-cases and advantages of SpikeInterface, we performed a detailed meta-analysis that included: quantifying the agreement among 6 modern sorters on a real dataset, benchmarking each sorter on a simulated ground-truth recording, and investigating the performance of a consensus-based spike sorting and how it compares with manually curated results. To highlight the modular design of SpikeInterface, we then provided descriptions and code samples for each of the five main packages and showed how they could be chained together to construct flexible spike sorting workflows.

4.1 Ensemble spike sorting

Our analysis demonstrated that spike sorters not only differ in unit isolation quality, but can also return a significant number of false positive units. To identify true neurons and remove poorly sorted and noisy units, we combined the output of several spike sorters and found that although agreement between sorters is generally poor, units that are found by more than one sorter are likely true positives. This strategy, which we term consensus-based or ensemble spike sorting (a terminology borrowed from machine learning [22]) appears to be a viable alternative to manual curation which suffers from high-variability among different operators [73, 65]. Alternatives to manual curation are especially enticing as the density and number of simultaneously recording channels continue to increase rapidly.

We hypothesise that consensus-based spike sorting (or curation) can be utilized in a number of different ways. A first possibility is to choose a suitable spike sorter (for instance, based on the extensive ground truth comparison performed by SpikeForest [47]) and then to curate its output by retaining the units that are in agreement with other sorters. Alternatively, a more conservative approach is to simply record the agreement scores for all sorted units and then *hand-curate* only those units that have low agreement. A third method, already implemented in SpikeInterface, is to generate a consensus spike sorting by using, for each unit, the union of the two closest matching units from different sorters (matching spikes are only considered once). Although more work is needed to quantitatively assess the

advantages and disadvantages of each approach, our analysis indicates that agreement among sorters can be a useful tool for curating sorting results.

Although ensemble spike sorting is an exciting new direction to explore, there are other methods for curation that must be considered. One popular curation method is to accept or reject sorted units based on a variety of quality metrics (this is supported by SpikeInterface). Another method that is gaining more popularity is to use the large amount of available curated datasets to train classifiers that can automatically flag a unit as "good" or "noise" depending on some features, such as waveform shape. Finally, while manual curation is subjective and time consuming, it is the only method that allows for merging and splitting of units and, through powerful software tools such as `Phy` [64, 65], it allows for full control over the curation process. Future research into these different curation methods is required to determine which are appropriate for the new influx of high-density extracellular recording devices.

4.2 Comparison to other frameworks

As mentioned in the introduction, many software tools have attempted to improve the accessibility and reproducibility of spike sorting. Here, we review the four most recent tools that are in use (to our knowledge) and compare them to SpikeInterface.

`Nev2lkit` [15] is a cross-platform, C++-based GUI designed for the analysis of recordings from multi-shank multi-electrode arrays (Utah arrays). In this GUI, the spike sorting step consists of PCA for dimensionality reduction and then `klustakwik` for automatic clustering [65]. As `Nev2lkit` targets low-density probes where each channel is spike sorted separately, it is not suitable for the analysis of high-density recordings. Also, since it implements only one spike sorter, users cannot utilize any consensus-based curation or exploration of the data. The software is available online[13], but it lacks version-control and automated testing with continuous integration platforms.

`SigMate` [48] is a MATLAB-based toolkit built for the analysis of electrophysiological data. `SigMate` has a large scope of usage including the analysis of electroencephalograpy (EEG) signals, local field potentials (LFP), and spike trains. Despite its broad scope, or because of it, the spike sorting step in `SigMate` is limited to `Wave clus` [19], which is mainly designed for spike sorting recordings from a few channels. This means that both major limitations of `Nev2lkit` (as discussed above) also apply to `SigMate`. The software is available online[14], but again, it lacks version-control and automated testing with continuous integration platforms.

Regalia et al. [61] developed a spike sorting framework with an intuitive MATLAB-based GUI. The spike sorting functionality implemented in this framework includes 4 feature extraction methods, 3 clustering methods, and 1 template matching classifier (`O-Sort` [67]). These "building blocks" can be combined to construct new spike sorting pipelines. As this framework targets low-density probes where signals from separate electrodes are spike sorted separately, its usefulness for newly developed high-density recording technology is limited. Moreover, this framework only runs with a specific file format (MCD format from Multi Channel Systems [5]). The software is distributed upon request.

Most recently, Nasiotis et al. [56] implemented `IN-Brainstorm`, a MATLAB-based GUI designed for the analysis of invasive neurophysiology data. `IN-Brainstorm` allows users to run three spike sorting packages, (`Wave clus` [19], `UltraMegaSort2000` [37], and `Kilosort` [59]). Recordings can be loaded and analyzed from six different file formats: Blackrock, Ripple, Plexon, Intan, NWB, and Tucker Davis Technologies. `IN-Brainstorm` is available on GitHub[15] and its functionality is documented[16]. `IN-Brainstorm` does not include the latest spike sorting software [65, 74, 20, 39, 58, 36], however, and it does not cover any post-sorting analysis such as validation, curation, or sorting output comparison.

[13] `http://nev2lkit.sourceforge.net/`
[14] `https://sites.google.com/site/muftimahmud/codes`
[15] `https://github.com/brainstorm-tools/brainstorm3`
[16] `https://neuroimage.usc.edu/brainstorm/e-phys/Introduction`

4.3 Outlook

As it stands, spike sorting is still an open problem. No step in the spike sorting pipeline is completely solved and no spike sorter can be used for all applications. With SpikeInterface, researchers can quickly build, run, and evaluate many different spike sorting workflows on their specific datasets and applications, allowing them to determine which will work best for them. Once a researcher determines an ideal workflow for their specific problem, it is straightforward to share and re-use that workflow in other laboratories as the full provenance is automatically stored by SpikeInterface. We envision that many laboratories will use SpikeInterface to satisfy their spike sorting needs.

Along with its applications to extracellular analysis, SpikeInterface is also a powerful tool for developers looking to create new spike sorting algorithms and analysis tools. Developers can test their methods using our efficient and comprehensive comparison functions. Once satisfied with their performance, developers can integrate their work into SpikeInterface, allowing them access to a large-community of new users and providing them with automatic file I/O for many popular extracellular dataset formats. For developers who work on projects that utilize spike sorting, SpikeInterface can be used out-of-the-box, providing more reliability and functionality than lab-specific spike sorting scripts. We envision that many developers will be excited to use and integrate with SpikeInterface.

Already, SpikeInterface is being used in a variety of applications. The file IO, preprocessing, and spike sorting capabilities of SpikeInterface are an integral part of SpikeForest [47] which is an interactive website for benchmarking and tracking the accuracy of publicly available spike sorting algorithms. At present, this project includes ten spike sorting algorithms and more than 300 extracellular recordings with ground-truth firing information. SpikeInterface's ability to read and write to a multitude of extracellular file formats is also being utilized by Neurodata Without Borders [71] in their `nwb-conversion-tools` package. We hope to continue integrating SpikeInterface into cutting-edge extracellular analysis frameworks.

Competing interests

The authors declare no competing interests.

Ackowledgements

This work was supported by the Wellcome Trust grant 214431/Z/18/Z (MHH). APB is supported by an ETH Zurich Postdoctoral Fellowship 19-2 FEL-17, and by the Simula-UCSD-University of Oslo Research and PhD training (SUURPh) program, funded by the Norwegian Ministry of Education and Research. CLH is supported by the Thouron Award and by the Institute for Adaptive and Neural Computation, University of Edinburgh. JHS wishes to thank the Allen Institute founder, Paul G. Allen, for his vision, encouragement and support. We would also like to thank Shangmin Guo for his recent contributions to debugging and improving the codebase.

References

[1] Biocam. https://www.3brain.com/biocamx.html.

[2] Intan technologies. http://intantech.com/.

[3] MaxWell biosystems. https://www.mxwbio.com/.

[4] Mea1k. https://bsse.ethz.ch/bel/research/cmos-microsystems/microelectrode-systems.html.

[5] Multi channel systems. https://www.multichannelsystems.com/.

[6] Neuralynx. https://neuralynx.com/.

[7] Neuroscience information exchange format - nix. http://g-node.github.io/nix/.

[8] Plexon offline sorter. https://plexon.com/products/offline-sorter/.

[9] G. N. Angotzi, F. Boi, A. Lecomte, E. Miele, M. Malerba, S. Zucca, A. Casile, and L. Berdondini. Sinaps: An implantable active pixel sensor cmos-probe for simultaneous large-scale neural recordings. *Biosensors and Bioelectronics*, 126:355–364, 2019.

[10] M. Ballini, J. Müller, P. Livi, Y. Chen, U. Frey, A. Stettler, A. Shadmani, V. Viswam, I. L. Jones, D. Jäckel, et al. A 1024-channel cmos microelectrode array with 26,400 electrodes for recording and stimulation of electrogenic cells in vitro. *IEEE Journal of Solid-State Circuits*, 49(11):2705–2719, 2014.

[11] A. H. Barnett, J. F. Magland, and L. F. Greengard. Validation of neural spike sorting algorithms without ground-truth information. *Journal of neuroscience methods*, 264:65–77, 2016.

[12] L. Berdondini, P. Van Der Wal, O. Guenat, N. F. de Rooij, M. Koudelka-Hep, P. Seitz, R. Kaufmann, P. Metzler, N. Blanc, and S. Rohr. High-density electrode array for imaging in vitro electrophysiological activity. *Biosensors and bioelectronics*, 21(1):167–174, 2005.

[13] H. Bokil, P. Andrews, J. E. Kulkarni, S. Mehta, and P. P. Mitra. Chronux: a platform for analyzing neural signals. *Journal of neuroscience methods*, 192(1):146–151, 2010.

[14] L. L. Bologna, V. Pasquale, M. Garofalo, M. Gandolfo, P. L. Baljon, A. Maccione, S. Martinoia, and M. Chiappalone. Investigating neuronal activity by spycode multi-channel data analyzer. *Neural Networks*, 23(6):685–697, 2010.

[15] M. Bongard, D. Micol, and E. Fernandez. Nev2lkit: a new open source tool for handling neuronal event files from multi-electrode recordings. *International journal of neural systems*, 24(04):1450009, 2014.

[16] M. P. Bonomini, J. M. Ferrandez, J. A. Bolea, and E. Fernandez. Data-means: an open source tool for the classification and management of neural ensemble recordings. *Journal of neuroscience methods*, 148(2):137–146, 2005.

[17] A. P. Buccino and G. T. Einevoll. Mearec: a fast and customizable testbench simulator for ground-truth extracellular spiking activity. *Neuroinformatics*, pages 1–20, 2020.

[18] D. Carlson and L. Carin. Continuing progress of spike sorting in the era of big data. *Current opinion in neurobiology*, 55:90–96, 2019.

[19] 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*, 120(4):1859–1871, 2018.

[20] J. E. Chung, J. F. Magland, A. H. Barnett, et al. A fully automated approach to spike sorting. *Neuron*, 95(6):1381–1394, 2017.

[21] Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016.

[22] T. G. Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[23] R. Diggelmann, M. Fiscella, A. Hierlemann, and F. Franke. Automatic spike sorting for high-density microelectrode arrays. *Journal of neurophysiology*, 120(6):3155–3171, 2018.

[24] G. Dimitriadis, J. P. Neto, A. Aarts, A. Alexandru, M. Ballini, F. Battaglia, L. Calcaterra, F. David, R. Fiath, J. Frazao, et al. Why not record from every channel with a cmos scanning probe? *bioRxiv*, page 275818, 2018.

[25] S.-A. Dragly, M. Hobbi Mobarhan, M. E. Lepperød, S. Tennøe, M. Fyhn, T. Hafting, and A. Malthe-Sørenssen. Experimental directory structure (exdir): An alternative to hdf5 without introducing a new file format. *Frontiers in neuroinformatics*, 12:16, 2018.

[26] U. Egert, T. Knott, C. Schwarz, M. Nawrot, A. Brandt, S. Rotter, and M. Diesmann. Meatools: an open source toolbox for the analysis of multi-electrode data with matlab. *Journal of neuroscience methods*, 117(1):33–42, 2002.

[27] B. Eversmann, M. Jenkner, F. Hofmann, C. Paulus, R. Brederlow, B. Holzapfl, P. Fromherz, M. Merz, M. Brenner, M. Schreiter, et al. A 128× 128 cmos biosensor array for extracellular recording of neural activity. *IEEE Journal of Solid-State Circuits*, 38(12):2306–2317, 2003.

[28] U. Frey, J. Sedivy, F. Heer, R. Pedron, M. Ballini, J. Mueller, D. Bakkum, S. Hafizovic, F. D. Faraci, F. Greve, et al. Switch-matrix-based high-density microelectrode array in cmos technology. *IEEE Journal of Solid-State Circuits*, 45(2):467–482, 2010.

[29] S. Garcia and N. Fourcaud-Trocmé. Openelectrophy: an electrophysiological data-and analysis-sharing framework. *Frontiers in neuroinformatics*, 3:14, 2009.

[30] S. Garcia, D. Guarino, F. Jaillet, T. R. Jennings, R. Pröpper, P. L. Rautenberg, C. Rodgers, A. Sobolev, T. Wachtler, P. Yger, et al. Neo: an object model for handling electrophysiology data in multiple formats. *Frontiers in neuroinformatics*, 8:10, 2014.

[31] S. Garcia and C. Pouzat. Tridesclous. `https://github.com/tridesclous/tridesclous`.

[32] P. Gleeson, A. P. Davison, R. A. Silver, and G. A. Ascoli. A commitment to open source in neuroscience. *Neuron*, 96(5):964–965, 2017.

[33] D. H. Goldberg, J. D. Victor, E. P. Gardner, and D. Gardner. Spike train analysis toolkit: enabling wider application of information-theoretic techniques to neurophysiology. *Neuroinformatics*, 7(3):165–178, 2009.

[34] K. D. Harris, H. Hirase, X. Leinekugel, D. A. Henze, and G. Buzsáki. Temporal interaction between single spikes and complex spike bursts in hippocampal pyramidal cells. *Neuron*, 32(1):141–149, 2001.

[35] L. Hazan, M. Zugaro, and G. Buzsáki. Klusters, neuroscope, ndmanager: a free software suite for neurophysiological data processing and visualization. *Journal of neuroscience methods*, 155(2):207–216, 2006.

[36] G. Hilgen, M. Sorbaro, S. Pirmoradian, J.-O. Muthmann, I. E. Kepiro, S. Ullo, C. J. Ramirez, A. P. Encinas, A. Maccione, L. Berdondini, et al. Unsupervised spike sorting for large-scale, high-density multielectrode arrays. *Cell reports*, 18(10):2521–2532, 2017.

[37] D. N. Hill, S. B. Mehta, and D. Kleinfeld. Quality metrics to accompany spike sorting of extracellular signals. *Journal of Neuroscience*, 31(24):8699–8705, 2011.

[38] J. J. Jun, J. F. Magland, C. Mitelut, and A. H. Barnett. Ironclust: Scalable and drift-resistant spike sorting for long-duration, high-channel count recordings, 2020. in preparation.

[39] J. J. Jun, C. Mitelut, C. Lai, S. Gratiy, C. Anastassiou, and T. D. Harris. Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction. *bioRxiv*, page 101030, 2017.

[40] J. J. Jun, N. A. Steinmetz, J. H. Siegle, D. J. Denman, M. Bauza, B. Barbarits, A. K. Lee, C. A. Anastassiou, A. Andrei, Ç. Aydın, et al. Fully integrated silicon probes for high-density recording of neural activity. *Nature*, 551(7679):232, 2017.

[41] B. Karsh. SpikeGLX. `https://billkarsh.github.io/SpikeGLX/`.

[42] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[43] K. Y. Kwon, S. Eldawlatly, and K. Oweiss. Neuroquest: a comprehensive analysis tool for extracellular neural ensemble recordings. *Journal of neuroscience methods*, 204(1):189–201, 2012.

[44] J. H. Lee, D. E. Carlson, H. S. Razaghi, W. Yao, G. A. Goetz, E. Hagen, E. Batty, E. Chichilnisky, G. T. Einevoll, and L. Paninski. Yass: yet another spike sorter. In *Advances in Neural Information Processing Systems*, pages 4002–4012, 2017.

[45] X.-q. Liu, X. Wu, and C. Liu. Spktool: An open source toolbox for electrophysiological data processing. In *2011 4th International Conference on Biomedical Engineering and Informatics (BMEI)*, volume 2, pages 854–857. IEEE, 2011.

[46] C. M. Lopez, S. Mitra, J. Putzeys, B. Raducanu, M. Ballini, A. Andrei, S. Severi, M. Welkenhuysen, C. Van Hoof, S. Musa, et al. 22.7 a 966-electrode neural probe with 384 configurable channels in 0.13 μm soi cmos. In *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*, pages 392–393. IEEE, 2016.

[47] J. F. Magland, J. J. Jun, E. Lovero, A. J. Morley, C. L. Hurwitz, A. P. Buccino, S. Garcia, and A. H. Barnett. Spikeforest: reproducible web-facing ground-truth validation of automated neural spike sorters. *bioRxiv*, 2020.

[48] M. Mahmud, A. Bertoldo, S. Girardi, M. Maschietto, and S. Vassanelli. Sigmate: a matlab-based automated tool for extracellular neuronal signal processing and analysis. *Journal of neuroscience methods*, 207(1):97–112, 2012.

[49] H. Markram, E. Muller, S. Ramaswamy, et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015.

[50] A. Marques-Smith, J. P. Neto, G. Lopes, J. Nogueira, L. Calcaterra, J. Frazão, D. Kim, M. G. Phillips, G. Dimitriadis, and A. Kampff. Simultaneous patch-clamp and dense cmos probe extracellular recordings from the same cortical neuron in anaesthetized rats. data available from `http://dx.doi.org/10.6080/K0J67F4T`.

[51] A. Marques-Smith, J. P. Neto, G. Lopes, J. Nogueira, L. Calcaterra, J. Frazão, D. Kim, M. G. Phillips, G. Dimitriadis, and A. Kampff. Recording from the same neuron with high-density cmos probes and patch-clamp: a ground-truth dataset and an experiment in collaboration. *bioRxiv*, page 370080, 2018.

[52] W. McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[53] H.-J. Mucha. Xclust: clustering in an interactive way. In *XploRe: an Interactive Statistical Computing Environment*, pages 141–168. Springer, 1995.

[54] E. Muller, J. A. Bednar, M. Diesmann, M.-O. Gewaltig, M. Hines, and A. P. Davison. Python in neuroscience. *Frontiers in neuroinformatics*, 9:11, 2015.

[55] J. Müller, M. Ballini, P. Livi, Y. Chen, M. Radivojevic, A. Shadmani, V. Viswam, I. L. Jones, M. Fiscella, R. Diggelmann, et al. High-resolution cmos mea platform to study neurons at subcellular, cellular, and network levels. *Lab on a Chip*, 15(13):2767–2780, 2015.

[56] K. Nasiotis, M. Cousineau, F. Tadel, A. Peyrache, R. M. Leahy, C. C. Pack, and S. Baillet. Integrated open-source software for multiscale electrophysiology. *BioRxiv*, page 584185, 2019.

[57] R. Oostenveld, P. Fries, E. Maris, and J.-M. Schoffelen. Fieldtrip: open source software for advanced analysis of meg, eeg, and invasive electrophysiological data. *Computational intelligence and neuroscience*, 2011:1, 2011.

[58] M. Pachitariu, N. A. Steinmetz, and J. Colonell. Kilosort2. `https://github.com/MouseLand/Kilosort2`.

[59] M. Pachitariu, N. A. Steinmetz, S. N. Kadir, et al. Fast and accurate spike sorting of high-channel count probes with kilosort. In *Advances in Neural Information Processing Systems*, pages 4448–4456, 2016.

[60] S. Ramaswamy, J. Courcol, M. Abdellah, et al. The neocortical microcircuit collaboration portal: a resource for rat somatosensory cortex. *Front Neural Circuits*, 9, 2015.

[61] 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. *Computational intelligence and neuroscience*, 2016, 2016.

[62] H. G. Rey, C. Pedreira, and R. Q. Quiroga. Past, present and future of spike sorting techniques. *Brain research bulletin*, 119:106–117, 2015.

[63] C. Rossant and K. D. Harris. Hardware-accelerated interactive data visualization for neuroscience in python. *Frontiers in neuroinformatics*, 7:36, 2013.

[64] C. Rossant, S. Kadir, D. Goodman, M. Hunter, and K. Harris. Phy. `https://github.com/cortex-lab/phy`.

[65] C. Rossant, S. N. Kadir, D. F. Goodman, J. Schulman, M. L. Hunter, A. B. Saleem, A. Grosmark, M. Belluscio, G. H. Denfield, A. S. Ecker, et al. Spike sorting for large, dense electrode arrays. *Nature neuroscience*, 19(4):634, 2016.

[66] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[67] 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. *Journal of neuroscience methods*, 154(1-2):204–224, 2006.

[68] N. Schmitzer-Torbert and A. D. Redish. Neuronal activity in the rodent dorsal striatum in sequential navigation: separation of spatial and reward responses on the multiple t task. *Journal of neurophysiology*, 91(5):2259–2272, 2004.

[69] J. H. Siegle, X. Jia, S. Durand, S. Gale, C. Bennett, N. Graddis, G. Heller, T. K. Ramirez, H. Choi, J. A. Luviano, et al. A survey of spiking activity reveals a functional hierarchy of mouse corticothalamic visual areas. *Biorxiv*, page 805010, 2019.

[70] J. H. Siegle, A. C. López, Y. A. Patel, K. Abramov, S. Ohayon, and J. Voigts. Open ephys: an open-source, plugin-based platform for multichannel electrophysiology. *Journal of neural engineering*, 14(4):045003, 2017.

[71] J. L. Teeters, K. Godfrey, R. Young, C. Dang, C. Friedsam, B. Wark, H. Asari, S. Peron, N. Li, A. Peyrache, et al. Neurodata without borders: creating a common data format for neurophysiology. *Neuron*, 88(4):629–634, 2015.

[72] J. Voigts. Simpleclust. `https://jvoigts.scripts.mit.edu/blog/simpleclust-manual-spike-sorting-in-matlab/`.

[73] F. Wood, M. J. Black, C. Vargas-Irwin, M. Fellows, and J. P. Donoghue. On the variability of manual spike sorting. *IEEE Transactions on Biomedical Engineering*, 51(6):912–918, 2004.

[74] P. Yger, G. L. Spampinato, E. Esposito, B. Lefebvre, S. Deny, C. Gardella, M. Stimberg, F. Jetter, G. Zeck, S. Picaud, et al. A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo. *Elife*, 7:e34518, 2018.

[75] X. Yuan, S. Kim, J. Juyon, M. D'Urbino, T. Bullmann, Y. Chen, A. Stettler, A. Hierlemann, and U. Frey. A microelectrode array with 8,640 electrodes enabling simultaneous full-frame readout at 6.5 kfps and 112-channel switch-matrix readout at 20 ks/s. In *VLSI Circuits (VLSI-Circuits), 2016 IEEE Symposium on*, pages 1–2. IEEE, 2016.

[76] B. Zhang, J. Dai, and T. Zhang. Neoanalysis: A python-based toolbox for quick electrophysiological data processing and analysis. *Biomedical engineering online*, 16(1):129, 2017.

5 Supplementary figures

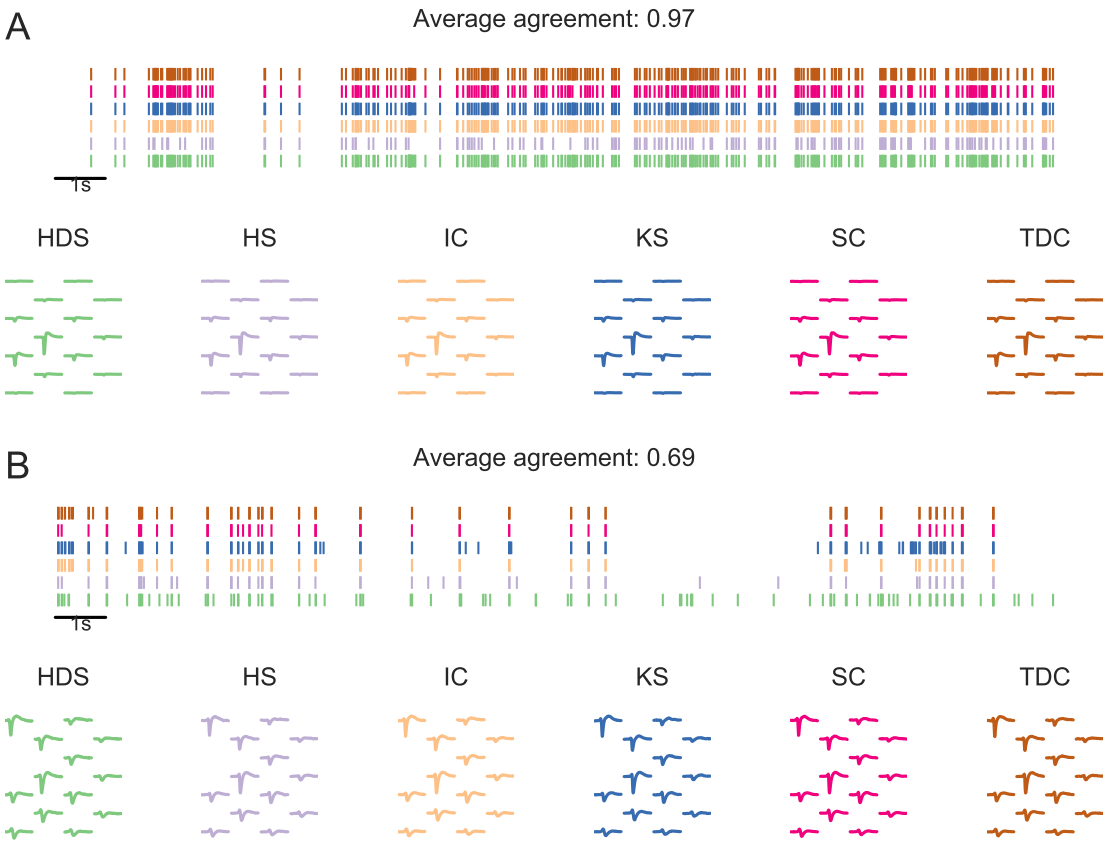

Figure S1: **Examples of agreement units**. **A)** Spike trains and templates of a unit with a high agreement score (0.97). **B)** Spike trains and templates of a unit with a lower agreement score (0.69).

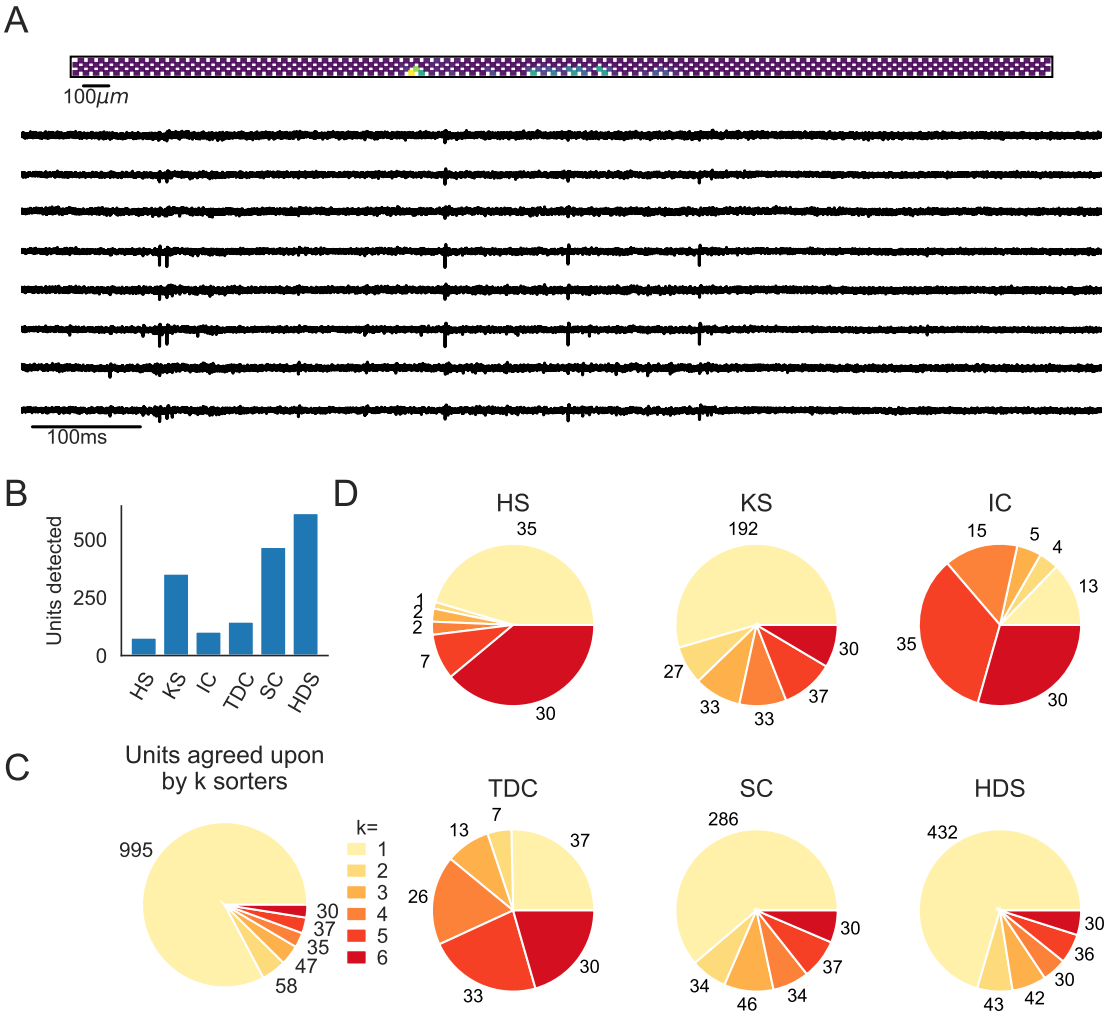

Figure S2: **Comparison of spike sorters on a Neuropixels recording.** This data set contains spontaneous neural activity from the rat cortex (motor and somatosensory areas) by Marques-Smith et al. [51, 50] (data set spe-c1). **A)** A visualization of the activity on the Neuropixels array (top, color indicates spike rate estimated on each channel evaluated with threshold detection) and of traces from the Neuropixels recording (below). **B)** The number of detected units for each of the 6 spike sorters (HS=HerdingSpikes2, KS=Kilosort2, IC=IronClust, TDC=Tridesclous, SC=SpyKING Circus, HDS=HDSort). **C)** The total number of units for which k sorters agree (unit agreement is defined as 50% spike match). **D)** The number of units (per sorter) for which k sorters agree; Most sorters find many units that other sorters do not.

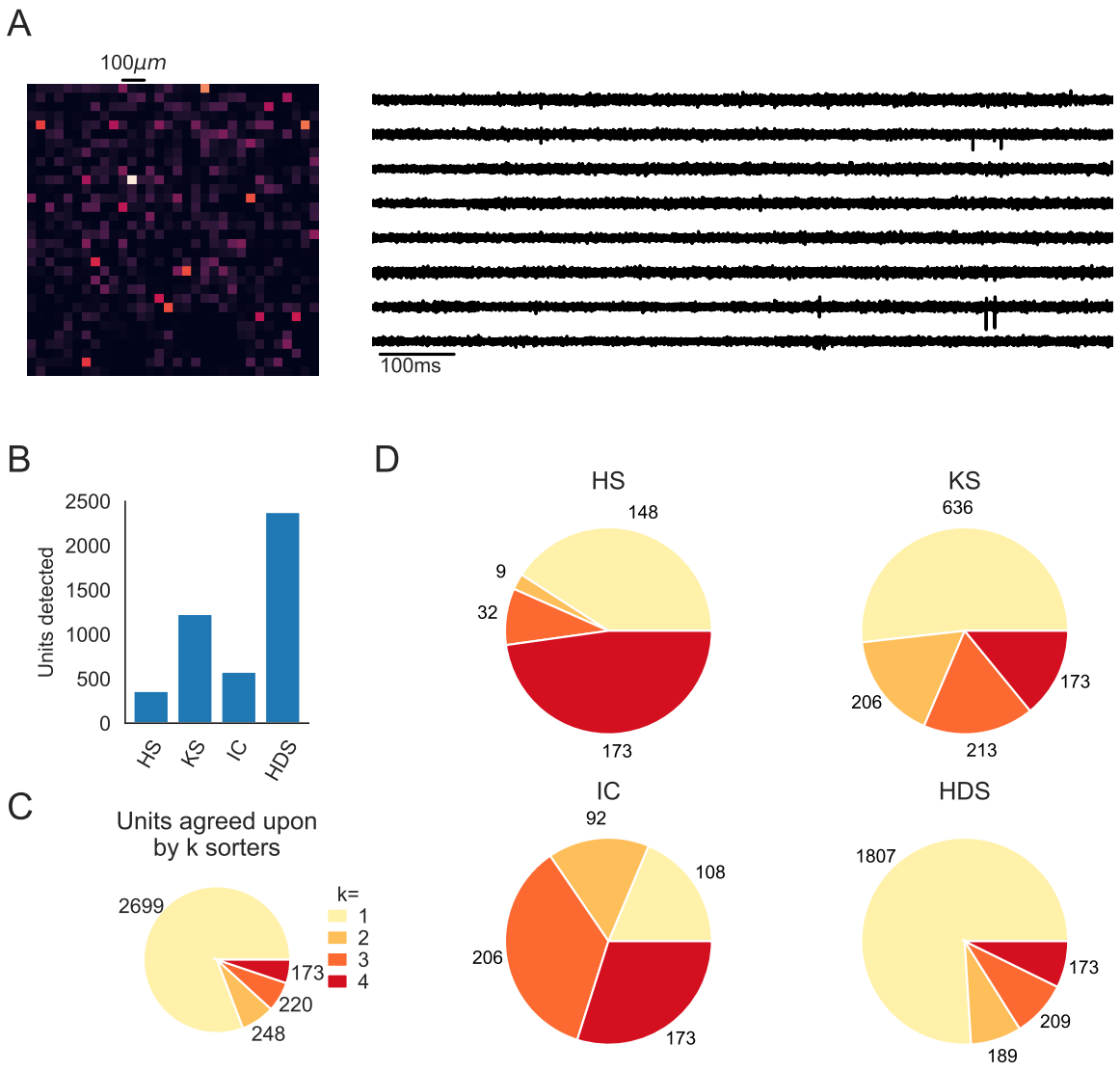

Figure S3: **Comparison of spike sorters on a Biocam recording from a mouse retina.** The recording was performed *in vitro* during full field stimulation by Gerrit Hilgen et al. [36] (data shown in Figure 1 of the source paper, 1,024 channels in square configuration, 23199kHz sampling frequency). Only four spike sorters were capable of processing this data set (HS=HerdingSpikes2, KS=Kilosort2, IC=IronClust, HDS=HDSort). **A)** A visualization of the activity on the Biocam array (top, color indicates spike rate estimated on each channel evaluated with threshold detection) and of traces from the recording (below). **B)** The number of detected units for each of the 4 spike sorters. **C)** The total number of units for which k sorters agree (unit agreement is defined as 50% spike match). **D)** The number of units (per sorter) for which k sorters agree; Most sorters find many units that other sorters do not.

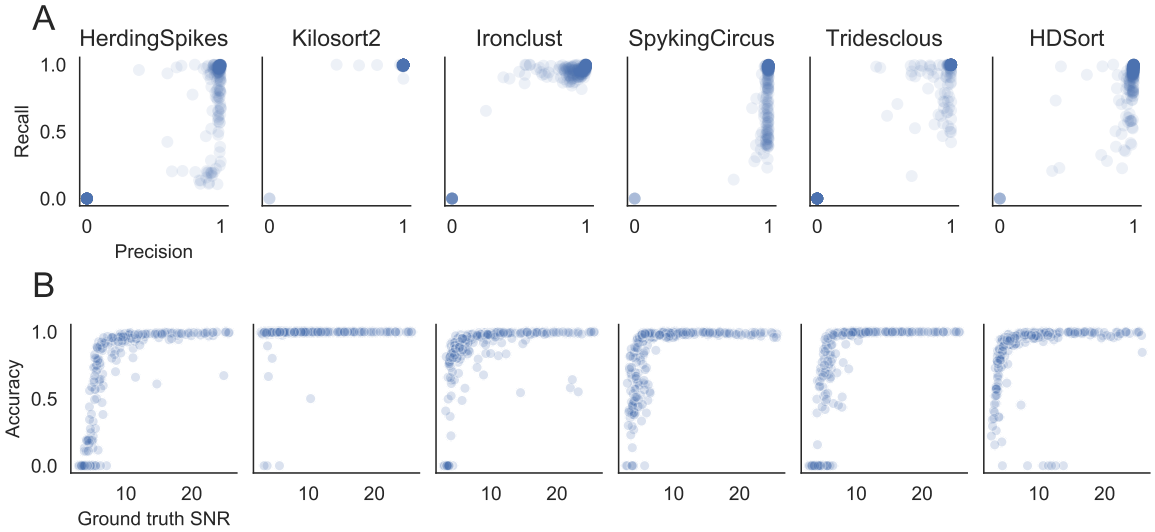

Figure S4: (A) Precision versus recall for the ground-truth comparison the simulated dataset. Some sorters seem to favor precision (HerdingSpikes, SpyKING Circus, HDSort), others instead have higher recall (Kilosort2, Ironclust). Tridesclous does not show a bias towards precision or recall. (B) Accuracy versus SNR. All the spike sorters (except Kilosort2) show a strong dependence of performance with respect to the SNR of the ground-truth units. Kilosort2, remarkably, is capable of achieving a high accuracy also for low-SNR units.