SpikeInterface, a unified framework for spike sorting

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

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

[¶] These authors contributed equally to this work.

Abstract

Given the importance of understanding single-neuron activity, much development has been directed towards improving the performance and automation of spike sorting. These developments, however, introduce new challenges, such as file format incompatibility and reduced interoperability, that hinder benchmarking and preclude 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 standardize extracellular data file operations. With a few lines of code and regardless of the underlying data format, researchers can: 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 both real and simulated extracellular datasets, demonstrate how it can improve the accessibility, reliability, and reproducibility of spike sorting in preparation for the widespread use of large-scale electrophysiology.

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 [49], MClust [65], SimpleClust [71], Plexon Offline Sorter [7]), 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 [58]. This progress has accelerated in the past few years as high-density devices [24, 11, 25, 9, 51, 75, 42, 36, 21, 8], capable of recording

*cole.hurwitz@ed.ac.uk

from hundreds to thousands of neurons simultaneously, have made manual intervention impractical, increasing the demand for both accurate and scalable spike sorting algorithms [61, 55, 40, 19, 74, 33, 35].

Along with these exciting advances, however, come unintended complications. Over the years, dozens of new file formats have been introduced, a multitude of data processing and evaluation methods have been developed, and an enormous amount of software, written in a variety of different programming languages, has been made available for general-use. In an ideal world, standards and best-practices for spike sorting would naturally arise from using these tools in experimental and clinical settings. However, due to the high complexity of spike sorting and a lack of interoperability among its corresponding technologies, no clear standards exist for how it should be performed or evaluated [58, 10, 17]. Furthermore, the importance of publication for career development rewards proof-of-concept breakthroughs at the expense of long-term maintenance of software tools. The lack of best practices, the scarcity of well-maintained code, the dearth of rigorous benchmarking, and the high barrier to entry of using a new spike sorter or file format all contribute to many laboratories either continuing to use manual spike sorting techniques or arbitrarily settling on one automated algorithm and its corresponding suite of analysis tools and supported file formats. Reproducibility, data provenance, and data sharing become increasingly difficult as different laboratories adopt different spike sorting solutions [20].

Recent work to alleviate these issues has focused on tackling file format incompatibilities in electrophysiology. This has led to progress in creating a common description of neurophysiological data both with new software tools and file formats [72, 27, 70, 69, 67, 68]. Despite progress in defining a common standard, many different file formats are still widely used in electrophysiology, with more being developed continuously [22]. Along with attempts to define common standards, much work has been put into creating open-source analysis tools that make extracellular analysis and spike sorting more accessible [23, 15, 32, 26, 30, 12, 41, 13, 53, 39, 44, 14, 57, 76, 52]. These software frameworks, while valuable tools in electrophysiology, implement spike sorting as a small step in a larger extracellular analysis pipeline, leading to undersupported, incomplete, and outdated spike sorting functionality. Given the ever-increasing amount of spike sorting software and file formats, there is an urgent need for an open-source analysis framework that is up-to-date with modern spike sorting methods, is agnostic to the underlying file formats of the extracellular datasets, and is extendable to new technologies.

In this paper, we introduce SpikeInterface, the first open-source, Python[1] framework designed exclusively to encapsulate all steps in the spike sorting pipeline. SpikeInterface overcomes both file format incompatibilities and software interoperability in spike sorting with a intuitive application program interface (API), and with a unified, extendable codebase of modern analysis tools. Using SpikeInterface, researchers can: run, compare, and benchmark most modern spike sorters; pre-process, post-process, and visualize extracellular datasets; validate, curate, and export sorting outputs; and more. This can all be done regardless of the underlying data format as SpikeInterface addresses file format compatibility issues within spike sorting pipelines without creating yet another file format. For developers, SpikeInterface enables easy integration and evaluation of their spike sorting software, allowing for accelerated development and a constantly expanding codebase. We also introduce a graphical user interface (GUI) based on SpikeInterface that allows for straightforward construction of spike sorting pipelines without any Python programming knowledge. To illustrate the advantages of a unified framework for spike sorting, we utilize SpikeInterface's Python API and GUI to build a complex spike sorting pipeline. We also use SpikeInterface to run, compare, and evaluate six modern spike sorters on both a sample Neuropixels recording and a simulated, ground-truth recording. With these three use cases, we demonstrate how SpikeInterface can help alleviate long-standing challenges in spike sorting. To clarify, the main contribution of this work is a novel framework for running and comparing spike sorting pipelines, not an exhaustive comparison of current spike sorting algorithms. All code for SpikeInterface is open-source and can be found on GitHub[2].

[1] We utilize Python as it is open-source, free, and increasingly popular in the neuroscience community [50, 29].
[2] https://github.com/SpikeInterface

2 Design Principles

SpikeInterface is designed to efficiently encapsulate all aspects of a spike sorting pipeline. To this end, we apply a set of design principles. These principles inform both the project's overall structure and the implementation of specific functionalities.

Focused. SpikeInterface was designed to unify all operations related to the spike sorting of extracellular recordings. Therefore, we did not attempt to incorporate any metadata from the underlying experiments (stimulus information, behavioral readouts, etc.) as such a task is beyond the scope of our problem statement. Also, we did not incorporate any analysis steps unrelated to spike sorting and did not attempt to handle any other electrophysiological data such as intracellular or electroencephalographic recordings. Keeping this narrow focus makes SpikeInterface light-weight, scalable, easy to use, and extendable.

Comprehensive. To do justice to years of research and development into spike sorting, we incorporated many existing extracellular file formats and the most current, semi-automatic spike sorters into SpikeInterface. We also incorporated common pre- and post-processing methods, quality metrics, evaluation and curation tools, and data visualization widgets. The broad range of methods and technologies that are supported makes SpikeInterface the most expansive spike sorting toolbox currently available by a wide margin. For an overview of the current file formats and spike sorters that are supported in SpikeInterface, see Table 1.

Modularized. The SpikeInterface codebase is separated into multiple, distinct modules which encapsulate individual processing steps shared across *all* spike sorting pipelines. In Section 3, we explain this modularized and conceptualized structure and show how it can be utilized to build robust and flexible spike sorting workflows. This design also makes SpikeInterface easily extendable, allowing new formats, methods, and tools to be added rapidly. We encourage the community to contribute to its further development.

Efficient. As we aim to support the analysis of large-scale extracellular recordings, much consideration has been put into making SpikeInterface as memory- and computation-efficient as possible. For instance, file input/output (I/O) operations are generally memory-mapped (only the data needed for a computation are loaded into memory) and processing is parallelized where feasible. We also made sure that running a spike sorter *in* our framework adds little to no extra computational cost in comparison to running the same spike sorter *outside* of our framework.

Reproducible. Although spike sorting is an essential step in extracellular analysis, it is often difficult to reproduce due to the variety of complex (and sometimes stochastic) processing steps performed on the underlying dataset. We designed SpikeInterface to make spike sorting and all associated computation as reproducible as possible with a unified codebase, fixed random seeds, a standard API, and a careful version control system. Laboratories using SpikeInterface can share and process extracellular datasets with a guarantee that they get identical results for the same functions.

3 Overview of SpikeInterface

SpikeInterface consists of five main Python packages designed to handle different aspects of 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; (v) `spikecomparison`

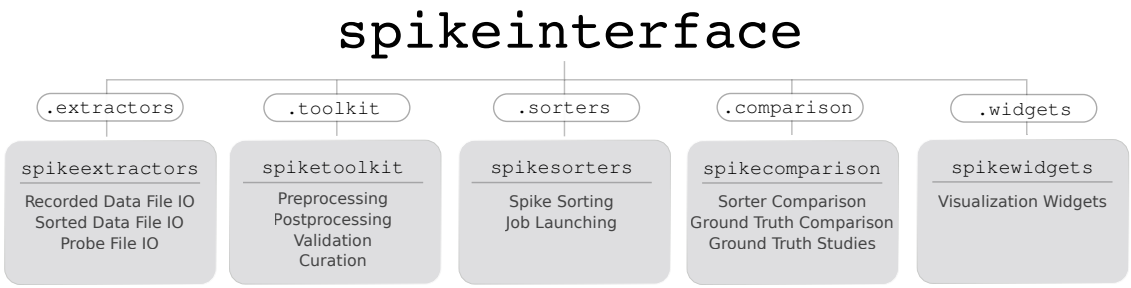

Figure 1: Overview of SpikeInterface's Python packages, their different functionalities, and how they can be accessed by our meta-package, `spikeinterface`.

for sorter comparison, ground-truth comparison, and ground-truth studies; and (iv) `spikewidgets`, for data visualization.

These five packages can be used individually or installed and used together with the `spikeinterface` metapackage, which contains stable versions of all five packages as internal modules (see Figure 1). 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 main package.

3.1 SpikeExtractors

The `spikeextractors` package[3] is designed to solve issues of file format incompatibility within spike sorting without creating yet another file format. This goal is met by standardizing data retrieval rather than data storage. By standardizing access to data from *all* spike sorting related files, whether extracellular recordings or sorting outputs, we eliminate the need for shared file formats and can allow for new tools and packages to directly interface with our framework instead. We distinguish between three data files in spike sorting: the extracellular recording, the sorting output, and the probe configuration. Being able to efficiently and easily interface with these three data file types is essential for running and evaluating any spike sorting pipeline. To this end, we developed two Python objects that can provide all the functionality required to access these data: the `RecordingExtractor` and the `SortingExtractor`.

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 standardized access. 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

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

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 Section 3.2). In sum, the `SortingExtractor` is an object representation of a sorting output along with any associated post-processing.

Critically, `Extractors` only query the underlying datasets for information as it is required, reducing their memory footprint and allowing their use for long, large-scale recordings. All extracted data is converted into either native Python data structures or into numpy arrays for immediate use in Python.

The following code snippet illustrates how to return a 2D numpy array of raw data (channels×time) from an extracellular recording and a list of unit indices 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()
unit_ids = sorting.get_unit_ids()
```

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, we support 15 extracellular recording formats, 11 sorting output formats, and 2 probe file formats. Although this covers many popular formats in extracellular analysis, we expect this number to grow with future versions as supporting a new format is as simple as making a new `Extractor` subclass for it. Also, we plan to integrate Neo's [27] I/O system into spikeextractors which would allow our framework to support many more open-source and proprietary file formats in extracellular electrophysiology without changing any functionality.

3.2 SpikeToolkit

The `spiketoolkit` package[4] 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`.

3.2.1 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 "processed" `RecordingExtractor`. This new `RecordingExtractor`, which can be used in exactly the same way as the original extractor, implements the processing in a *lazy* fashion so that the actual computation is performed only when data 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.

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

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')
```

In this code snippet, `recording_2` is still a `RecordingExtractor`. However, the extracted data when using `recording_2` will have channel 5 removed and the underlying extracellular traces bandpass filtered and common median referenced to remove noise.

Overall, the pre-processing functions in SpikeInterface represent a wide range of tools that are used in modern spike sorting applications and, since implementing a new pre-processor is straightforward, we expect more to be added in future versions.

3.2.2 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, as well as computing principle component analysis projections.

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` [59, 61] 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')
```

3.2.3 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[5] and by the SpikeInterface team. All quality metrics can be computed for the entire duration of the recording or for specific time periods (epochs) specified by the user.

The quality metrics that have been implemented so far include:

- spike count: the total spike count for a sorted unit.
- SNR: the signal-to-noise ratio (SNR) of the sorted units.
- firing rate: the average firing rate in a time period.
- presence ratio: the fraction of a time period in which spikes are present.
- ISI violations: the rate of inter-spike-interval (ISI) refractory period violations
- amplitude cutoff: an estimate of the miss rate based on an amplitude histogram.
- isolation distance: the Mahalanobis distance from a specified unit within as many spikes belong to the specified unit as to other units [31].
- L-ratio: the Mahalanobis distance and χ^2 inverse cumulative density function (under the assumption that the spikes in the unit distribute normally in each dimension) are used to find the probability of unit membership for each spike [64].
- d': the classification accuracy between units based on linear discriminant analysis (LDA) [34].
- nearest-neighbors: a non-parametric estimate of unit contamination using nearest-neighbor classification [19].
- silhouette score: a standard metric for quantifying cluster overlap [62].
- maximum drift: the maximum change in spike position throughout a recording.
- cumulative drift: The cumulative change in spike position throughout a recording.

To compute quality metrics with SpikeInterface, the user can instantiate and use a `MetricCalculator` object. The `MetricCalculator` utilizes a `RecordingExtractor`/`SortingExtractor` pair to generate and cache all data needed to run the quality metrics (amplitudes, principal components, etc.) and also to calculate any (or all) of the quality metrics. We also allow for quality metrics to be computed with a functional interface. All quality metric function calls internally utilize a `MetricCalculator`, concealing the object representation from the user.

The following code snippet demonstrates how to compute a single quality metric (SNR) and all quality metrics with two function calls:

```
import spikeinterface.toolkit as st
snr_metric = st.validation.compute_snrs(sorting, recording)
all_metrics = st.validation.compute_metrics(sorting, recording)
```

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

3.2.4 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 pre-processing 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 given a threshold that is specified by the user (we provide sensible default values). If passed a `MetricCalculator`, curation functions will threshold units based on the *cached* quality metrics that are stored in the `MetricCalculator`. Otherwise, curation functions will recompute 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 than" threshold to the underlying units:

```
import spikeinterface.toolkit as st
sorting = st.curation.threshold_firing_rate(sorting, threshold=2.3,
                                            threshold_sign='less')
sorting_1 = st.curation.threshold_snr(sorting, recording, threshold=10,
                                      threshold_sign='less')
```

In this code snippet, `sorting_1` is still a `SortingExtractor`. However, when queried about the underlying units, `sorting_1` will return only the units that had firing rates higher than 2.3 Hz and SNRs greater than 10.

As of this moment, we support thresholding of all basic quality metrics including: spike count; SNR; firing rate; presence ratio; and ISI violations. We plan to include curation tools for all implemented quality metrics and for more complicated curation steps (merging, splitting, etc.) in future versions.

3.3 `SpikeSorters`

The `spikesorters`[6] 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 the underlying program language, 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.

To allow for a simple, overarching API despite inherent differences between the sorting algorithms, each sorting wrapper is implemented as a subclass of a `BaseSorter` class. To run a spike sorting algorithm in SpikeInterface, the user passes a `RecordingExtractor` object to the wrapper and sets parameters for the underlying algorithm. Internally, each spike sorter wrapper creates and modifies the configuration based on these user-defined parameters and then runs the sorter on the dataset

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

encapsulated by the `RecordingExtractor`. Once the spike sorting algorithm is finished, the sorting output is saved and a corresponding `SortingExtractor` is returned for the user. Spike sorters can be invoked either by using the wrapper directly or by using a simple function call.

In the following code snippet, Mountainsort4 and Kilosort2 are used to sort an extracellular recording. Running each algorithm (and setting its associated parameters) can be done using a single function:

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

Along with running each sorting algorithm normally, our spike sorting wrappers 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` also provides a launcher function which handles any internal complications associated with running multiple sorters and returns a nested dictionary of `SortingExtractor`s corresponding to each sorting output.

Currently, SpikeInterface supports 9 semi-automated spike sorters which are listed in Table 1. We encourage developers to contribute to this expanding list in future versions. We provide comprehensive documentation on how to do so[7].

3.4 SpikeComparison

The `spikecomparison` package[8] 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 sorting outputs.
2. `compare_multiple_sorters` - Compares multiple sorting outputs.
3. `compare_sorter_with_ground_truth` - Compares a sorting output to ground truth.

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

[7] `https://spikeinterface.readthedocs.io/en/latest/contribute.html`
[8] https://github.com/SpikeInterface/spikecomparison

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 [38] 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.

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

- true positive (tp): spike found both in the ground-truth spike train and tested spike train.
- false negative (fn): spike found in the ground-truth spike train, but not in the tested spike train.
- false positive (fp): spike found in the tested spike train, but not in the ground-truth spike train.

Using these labels, the following performance measures are computed:

- accuracy: $\frac{\#tp}{(\#tp+\#fn+\#fp)}$
- recall: $\frac{\#tp}{(\#tp+\#fn)}$
- precision: $\frac{\#tp}{(\#tp+\#fn)}$
- miss rate: $\frac{\#fn}{(\#tp+\#fn)}$
- false discovery rate: $\frac{\#fp}{(\#tp+\#fp)}$

Based on the matching results and the scores, the units of the sorting output are classified as *well-detected*, *false positive*, *redundant*, and *over-merged*. 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). Over-merged units have an agreement score above 0.2 with two or more ground-truth units. All threshold scores are adjustable by the user.

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 [48]. An example ground-truth study is shown in Section 5.2.

3.5 SpikeWidgets

The `spikewidgets` package[9] 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 one only needs a `RecordingExtractor` for its visualization. This category includes widgets for time series, electrode geometry, signal spectra, and spectrograms. The second category only needs 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, and final, category needs 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])
```

The widget class is easily extendable, and will likely grow rapidly as new visualization tools are added. We also plan to introduce interactive widgets that allow the user to quickly explore the underlying elements of a spike sorting pipeline.

4 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 a

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

robust spike sorting pipeline. The spike sorting pipeline that we construct using SpikeInterface is depicted in Figure 2A and consists of the following analysis steps:

1. Loading an Open Ephys recording [66].
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.

4.1 Using the Python API

Using SpikeInterface's Python API to build the pipeline shown in Figure 2A is straightforward. Each of the seven steps is implemented with a single line of code (as shown in Figure 2B). Additionally, data visualizations can be added for each step of the pipeline using the appropriate widgets (as described in Section 3.5). Unlike handmade scripts, SpikeInterface has a wide range of unit tests 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.

4.2 Using the `spikely` GUI

Along with our Python API, we also developed `spikely`[10], 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 2C shows a screenshot from `spikely` where the pipeline in Figure 2A 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.

[10] https://github.com/SpikeInterface/spikely

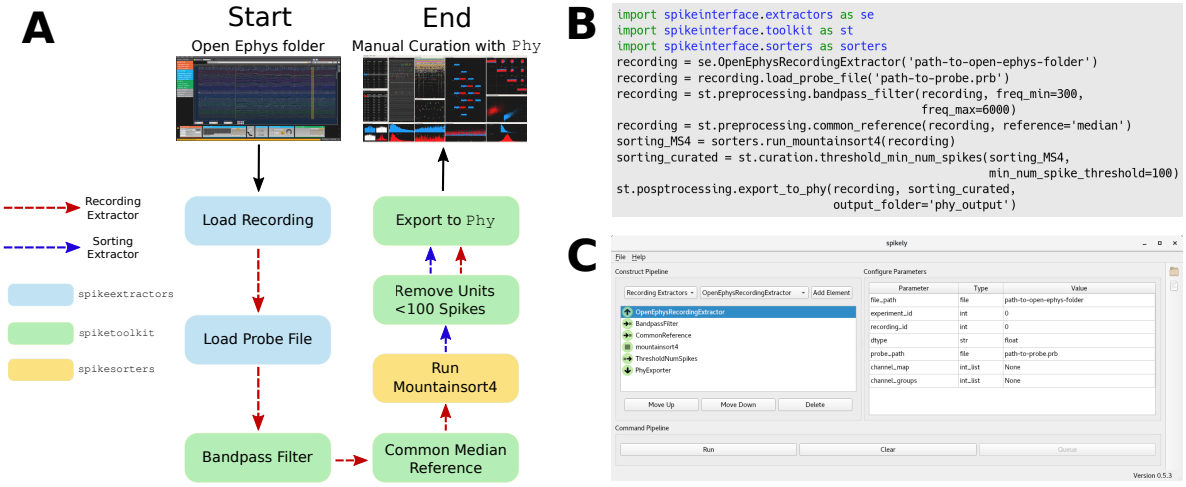

Figure 2: Sample spike sorting pipeline using SpikeInterface. (A) A diagram of a sample spike sorting pipeline. Each processing step is colored to represent the SpikeInterface package in which it is implemented and the dashed, colored arrows demonstrate how the `Extractors` are used in each processing 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).

5 Applications

We present two applications of the SpikeInterface framework in this section. In Application 1, we sort a Neuropixels dataset with six popular spike sorters. After sorting, we quantify and visualize the agreement among the spike sorters. In Application 2, we sort a simulated, ground-truth dataset with the same six spike sorters. Afterwards, we systematically evaluate and visualize the performance of each sorter (based on their default parameters). These applications demonstrate the advantages of using SpikeInterface for spike sorting analysis and highlight unsolved issues in the field. All analysis is done with PyPI version 0.9.0 of `spikeinterface`.

5.1 Application 1: Comparing Spike Sorters on Neuropixels Data

In this application, we utilize SpikeInterface to sort a dense *in vivo* recording with many different spike sorters. After sorting and without ground-truth information, we use SpikeInterface to assess the level of agreement between spike sorters.

The dataset we use in this application is a recording from a rat cortex using the Neuropixels probe ([47, 46] – recording `c1`). It has a duration of 270 seconds, 384 channels, and a sampling frequency of 30 kHz. The raw data are first pre-processed with a bandpass filter (highpass cutoff 300 Hz - lowpass cutoff 6000 Hz) and are subsequently pre-processed with a common median reference filter.

For this analysis, we choose to run six different spike sorters: HerdingSpikes2 [33], Kilosort2 [54], IronClust [35], SpykingCircus [74], Tridesclous [28], and Mountainsort4 [19][11]. As each of these algorithms are semi-automatic, we fix their parameters to default values to allow for straightforward comparison. We do not include Klusta [61], WaveClus [18], and Kilosort [55] in this analysis as Klusta can only

[11]The versions for each spike sorter are as follows: SpykingCircus==0.8.2, Tridesclous==1.2.2, Mountainsort4==0.3.2, HerdingSpikes2==0.3.2, IronClust==4.8.8, Kilosort2==GitHub commit 2a39926.

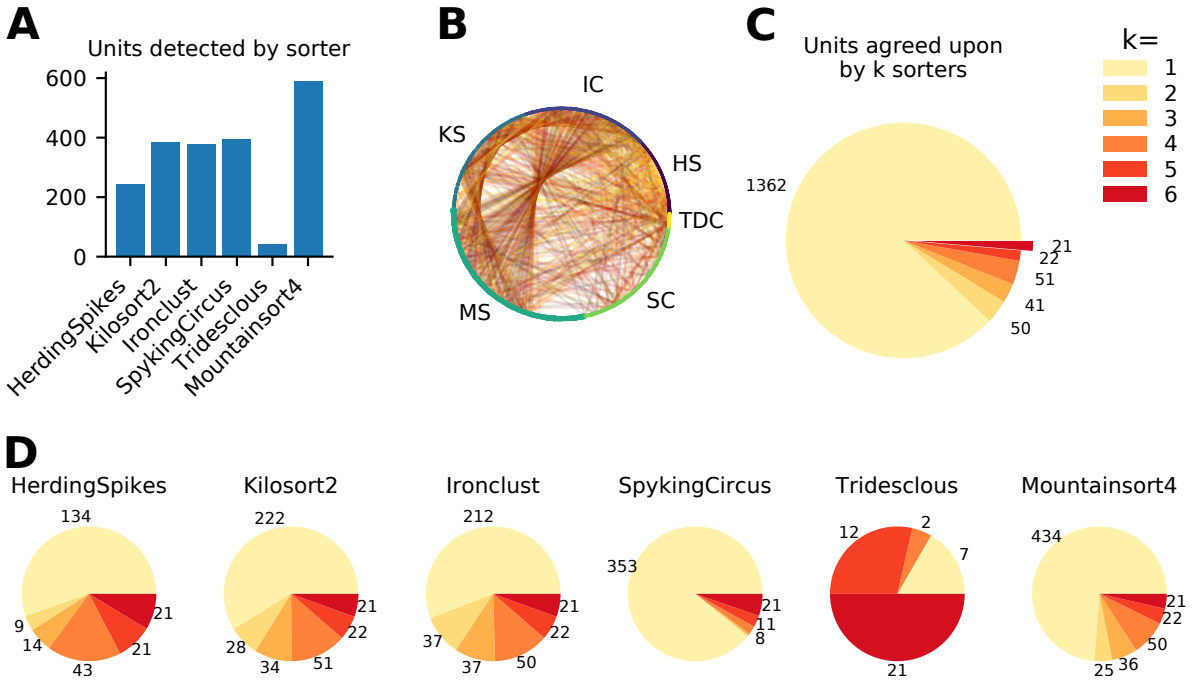

Figure 3: Analysis of Neuropixels recording with six spike sorters. (A) Number of units found by each spike sorter. (B) Network representation of the comparison between multiple sorters: each node is a unit and edges connect agreed upon units (edge color indicates agreement score). (C) Total number of units where k out of six sorters agree at a level of at least 0.5. (D) Number of units found and their agreement levels for each spike sorter.

handle up to 64 channels, WaveClus is designed for probes with a low channel count, and Kilosort is superseded by Kilosort2.

By quickly comparing the outputs of all six sorters, large discrepancies are immediately apparent. Figure 3A shows the number of units found by each sorter. While four of the sorters find between 200 and 400 putative units, Tridesclous only identifies 42, and Mountainsort4 almost 600.

Next, we use the `compare_multiple_sorters` function of the `comparison` module to explore these differences in more depth. As explained in Section 3.4, this function builds a weighted graph in which each node is a unit detected by a sorter and in which each edge is the best-match between a pair of units from different sorters. The edges are weighted by the respective agreement scores (Eq. 1; Figure 3B), and only edges with a score of at least 0.5 are kept. Once constructed, this graph can be interrogated to extract the units agreed upon by different sorters.

Figure 3C shows the overall agreement statistics. Surprisingly, out of a total of 1547 units, 1362 (∼88%) are not matched at all, i.e. they are only found by a single sorter. The number of units found by all six sorters is just 21 which is only the 1.36% of the total number of units. Panel 3D breaks this result down for each spike sorter. For HerdingSpikes, Kilosort2, and IronClust, about half of the units are not matched by any other sorters. For SpykingCircus and Mountainsort4, an overwhelming majority of their units are not matched to another sorter (∼90% for SpykingCircus, ∼74% for Mountainsort4). Tridesclous is more conservative, as it finds very few units, but ∼83% of them are matched by at least three sorters.

As units with little agreement are potentially noisy or very low-SNR units, we suggest a consensus-based

strategy for removing them. Using the multiple sorting comparison function, the units in agreement can automatically be extracted from the output of a sorter. This leads, potentially, to a subset of the putative units that are well-isolated and suited for downstream analysis. In future work, we plan to better understand low agreement units and to explore this consensus-based curation method.

5.2 Application 2: Benchmarking Spike Sorters on Simulated Data

In this application, we utilize SpikeInterface to evaluate and benchmark multiple spike sorters on a simulated, ground-truth dataset. We then illustrate that a popular, unsupervised quality metric for evaluating sorting outputs, SNR, can be correlated with a spike sorter's accuracy on the underlying ground-truth units. To be clear, the main goal of this application is to illustrate the capabilities of SpikeInterface to perform such comparisons and not to thoroughly analyze and benchmark the performance of different sorters which may be improved using different parameter sets or dedicated curation tools.

We use a simulated dataset[12] created with the `MEArec` Python package [16]. The probe is a square MEA with 100 channels, organized in a 10x10 configuration with an inter-electrode distance of 15 µm. The recording contains spiking activity from 50 neurons (from the Neocortical Micro Circuit Portal [56, 45]) that exhibit independent Poisson firing patterns. The recording also has an additive Gaussian noise with $10\mu V$ standard deviation. For preprocessing, the recording is bandpass filtered (highpass cutoff 300 Hz - lowpass cutoff 6000 Hz).

For this analysis, we choose to benchmark the same six sorters as used in Application 1. We use the `GroundTruthStudy` class of the `comparison` module to run and benchmark all the algorithms in a systematic manner (as described in Section 3.4). Again, we use the default parameters of each sorter to allow for straightforward comparison.

As the full ground-truth information is available, we are able to thoroughly quantify the performance of each sorter with a variety of metrics. Figure 4A shows swarm plots of the accuracy, precision, and recall (these terms are defined in Section 3.4) for each sorter on all 50 ground-truth units. This type of analysis provides a good first insight into the strengths of each sorter, but does not tell the whole story. In this analysis, Kilosort2 appears to be the best performing sorter with a mean accuracy of 0.88 and the least variability across each of the metrics.

While assessing the accuracy of each sorter on the ground-truth units is important, it is also critical to analyze **all** the units found by the sorters, not just the well-detected ones. Figure 4B shows the number of well detected, redundant, false positive, and over-merged units for each sorter (these terms are defined in Section 3.4). From this analysis we can see that although Kilosort2 finds many well-detected units (43), it also returns a large number of false positive (58), redundant (6), and over-merged (3) units. Other sorters, in contrast, display a more conservative behavior. IronClust, HerdingSpikes, and Tridesclous, for example, find fewer well-detected units (30, 26, and 26, respectively), but also significantly fewer false positives, redundant, and over-merged units. This suggests that there may be a trade-off between unit isolation and reliability, a factor that has to be taken into account in subsequent analysis of sorted spike trains.

Additionally, SpikeInterface records the runtime of each sorter (Figure 4C). The spike sorters specifically designed to deal with high-density probes (HerdingSpikes, Kilosort2, and IronClust), as expected, have a lower computation time than more general-purpose software (Tridesclous, SpykingCircus, and Mountainsort4). All spike sorters were run on an Ubuntu 18.04 machine, an Intel(R) Core(TM) i7-8700 CPU 3.20GHz processor, and 64 GB of RAM. Additionally, IronClust and Kilosort2 were run using a

[12] https://doi.org/10.5281/zenodo.3260283

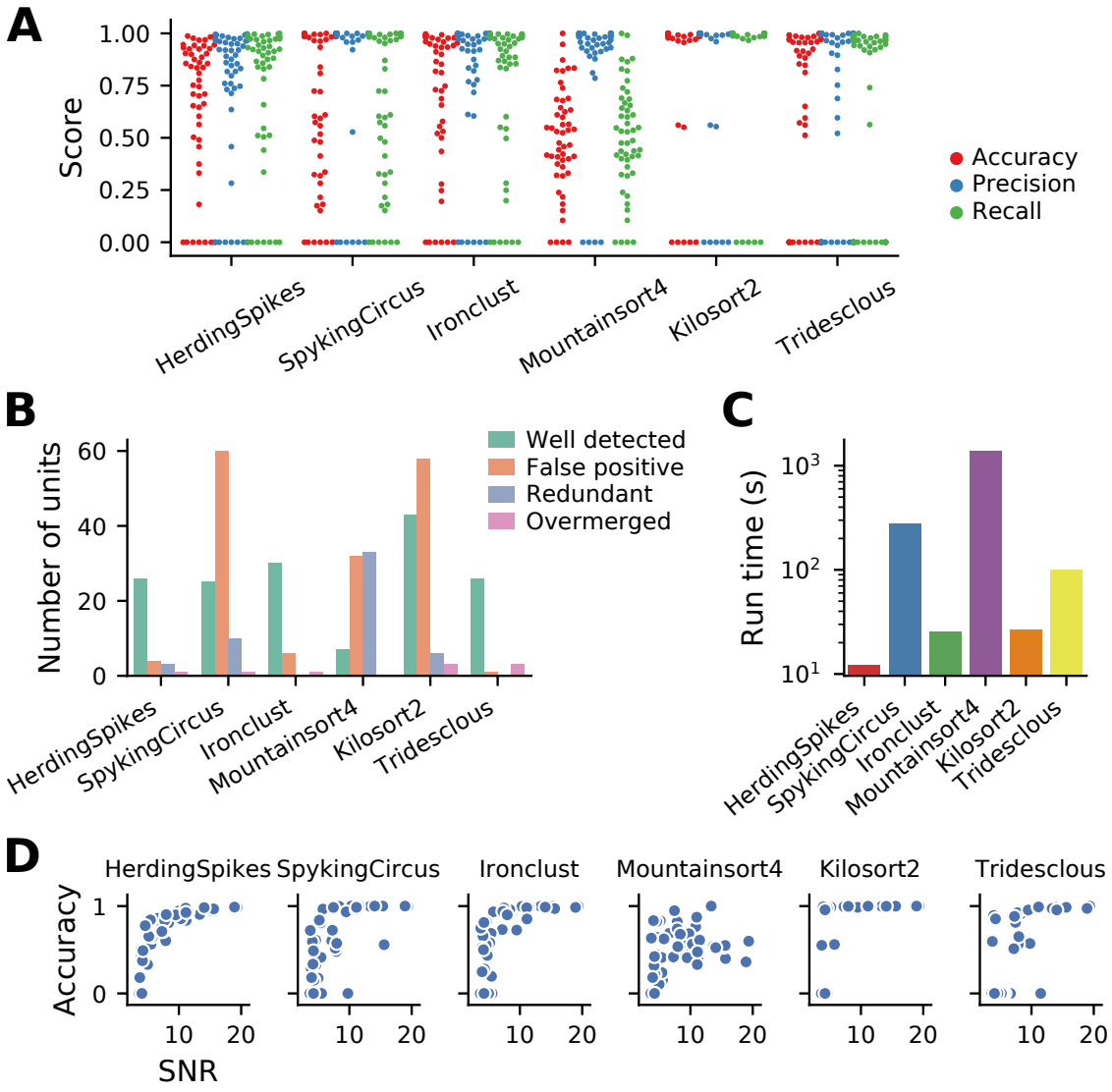

Figure 4: Analysis of a simulated ground-truth dataset. (A) Run times for each spike sorter. (B) Number of well detected, false positive, redundant, and over-merged units for each spike sorter. (C) Accuracy, precision, and recall for all ground-truth units for each spike sorter. (D) Accuracy on ground-truth units with respect to the SNR for each spike sorter.

GeForce RTX 2080 Ti GPU.

Finally, performance metrics can be also related to unsupervised quality metrics. In Figure 4D, for example, we plot the accuracy of each unit with respect to its SNR for each sorter. This plot illustrates that spike sorters generally are capable of isolating units with strong signals, but may differ in their ability to separate units with signals closer to the background noise level.

6 Discussion

We have introduced SpikeInterface, a Python framework designed to consolidate a complex ecosystem of software tools and file formats and to enhance the accessibility, reliability, and reproducibility of spike sorting. To highlight the modularity and careful design of SpikeInterface, we provided an overview of, and code examples for, each of the five main packages (Figure 1). To demonstrate how SpikeInterface can be used to construct flexible spike sorting workflows, we implemented an example pipeline (Figure 2A) using both the Python API (Figure 2B) and the `spikely` GUI (Figure 2C). Finally, to demonstrate potential applications of SpikeInterface, we evaluated the results of six spike sorters on both a Neuropixels and a simulated recording.

6.1 Supported File Formats and Spike Sorters

The file formats and spike sorters currently supported by SpikeInterface are summarized in Table 1. We expect this list to grow in future versions as both spike sorting developers and the general neuroscience community contribute to the growth of SpikeInterface. In order to facilitate contributions to SpikeInterface, we provide documentation on how to add a `RecordingExtractor`, a `SortingExtractor`, or a spike sorter[13] to our framework. At present, several `Extractors` have already been developed by or in collaboration with external contributors (SpikeGLX, Neurodata Without Borders, MCS H5, MaxOne, NIX, and Neuroscope).

Raw File Formats	Sorted File Formats	Sorters
Klusta	Klusta	Klusta [61]
Mountainsort (MDA)	Mountainsort (MDA)	Mountainsort4 [36]
Phy/Kilosort/Kilosort2 [59, 55, 60]	Phy/Kilosort/Kilosort2	Kilosort [55]
SpyKING Circus	Spyking Circus	Kilosort2 [54]
Exdir [22]	Exdir	SpyKING Circus [74]
MEArec [16]	MEArec	HerdingSpikes2 [33]
SpikeGLX [37]	HerdingSpikes2	Tridesclous [28]
Open Ephys [66]	Trideclous	IronClust [35]
Intan [2]	NPZ (numpy zip)	Wave clus [18]
Neurodata Without Borders (NWB) [70]	Neurodata Without Borders (NWB)	
NIX [5]	NeuroScope [6]	
MaxOne [3]		
MCS H5 [4]		
Neuroscope [32]		
Biocam HDF5 [1]		
Binary		

Table 1: In this table, we show SpikeInterface's currently supported file formats and spike sorting algorithms. With the help of the neuroscience community, we plan to expand these lists in future versions.

6.2 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

[13] https://spikeinterface.readthedocs.io/en/latest/contribute.html

knowledge) and compare them to SpikeInterface.

`Nev2lkit` [14] 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 [61]. 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[14], but it lacks version-control and automated testing with continuous integration platforms.

`SigMate` [44] 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 large scope, or because of it, the spike sorting step in `SigMate` is limited to `Wave clus` [18], 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[15], but again, it lacks version-control and automated testing with continuous integration platforms.

Regalia et al. [57] 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` [63]). 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 [4]). The software is distributed upon request.

Most recently, Nasiotis et al. [52] 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` [18], `UltraMegaSort2000` [34], and `Kilosort` [55]). 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[16] and its functionality is documented[17]. `IN-Brainstorm` does not include the latest spike sorting software [61, 74, 19, 35, 54, 33], however, and it does not cover any post-sorting analysis such as validation, curation, and sorting output comparison.

SpikeInterface overcomes all limitations of the aforementioned analysis frameworks by following rigorous design principles. As the scope of SpikeInterface is **focused** on spike sorting only, we were able to provide a **comprehensive** framework that encompasses all the functionality required for spike sorting. This includes interfacing with a wide range of commonly used file formats for extracellular recordings and sorting outputs, handling probe file information, pre-processing, spike sorting, post-processing, validation, curation (automatic or manual with `Phy`), comparison, and visualization. The **modularized** and object-oriented design of SpikeInterface enables users to build custom analysis pipelines using the Python API or the `spikely` GUI and for the codebase to expand gracefully with community contributions of new `Extractors` and spike sorters. Since SpikeInterface is **efficient** and already implements 9 modern spike sorters, it can be used to analyze large-scale recordings from next-generation multi-electrode arrays as shown in Section 5. Finally, SpikeInterface allows users to implement **reproducible** analysis pipelines with careful version control, fixed random seeds, and a standardized API. All source code is open-source, version-controlled, and tested with a continuous integration platform[18].

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

6.3 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 studying similar problems. 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 and software deployment. For developers who work on projects that use spike sorting, SpikeInterface can be used out-of-the-box, providing more reliability and functionality then handmade 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. In one application, SpikeInterface is being used as the engine of a related project called SpikeForest [43]. SpikeForest is an interactive website for benchmarking and tracking the accuracy of publicly available spike sorting algorithms. At present, it includes ten sorting algorithms and more than 300 extracellular recordings with ground-truth firing information. These recordings include both simulations and paired recordings where ground-truth is obtained from juxtacellular signals.

Overall, we hope that SpikeInterface can become a standard tool in neuroscience and can help foster a stronger relationship between spike sorting users and developers. To this end, we are maintaining an open forum[19] that can be a common space for the community to discuss any and all spike-sorting-related topics. We look forward to sharing and growing SpikeInterface over the years to come.

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 a doctoral fellow in the Simula-UCSD-University of Oslo Research and PhD training (SUURPh) program, an international collaboration in computational biology and medicine 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.

[19] www.spikeforum.org

References

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

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

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

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

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

[6] Neuroscope. `http://neurosuite.sourceforge.net/`.

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

[8] 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.

[9] M. Ballini, J. Muller, P. Livi, Y. Chen, U. Frey, A. Stettler, A. Shadmani, V. Viswam, I. L. Jones, D. Jackel, M. Radivojevic, M. K. Lewandowska, W. Gong, M. Fiscella, D. J. Bakkum, F. Heer, and A. Hierlemann. 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.

[10] 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.

[11] L. Berdondini, P. D. 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 & Bioelectronics*, 21(1):167–74, jul 2005.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] A. P. Buccino and G. T. Einevoll. Mearec: a fast and customizable testbench simulator for ground-truth extracellular spiking activity. *bioRxiv*, page 691642, 2019.

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

[18] 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.

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

[20] M. Denker, G. T. Einevoll, F. Franke, S. Grün, E. Hagen, J. N. D. Kerr, M. P. Nawrot, T. V. Ness, R. Ritz, L. S. Smith, T. Wachtler, and D. K. Wójcik. 1st incf workshop on validation of analysis methods. 2018.

[21] 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.

[22] 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.

[23] 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.

[24] B. Eversmann, M. Jenkner, F. Hofmann, C. Paulus, R. Brederlow, B. Holzapfl, P. Fromherz, M. Merz, M. Brenner, M. Schreiter, R. Gabl, K. Plehnert, M. Steinhauser, G. Eckstein, D. Schmitt-landsiedel, and R. Thewes. A 128 128 CMOS Biosensor Array for Extracellular Recording of Neural Activity. *IEEE Journal of Solid-State Circuits*, 38(12):2306–2317, 2003.

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

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

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

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

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] J. Magland, J. Jun, E. Lovero, L. Greengard, A. Barnett, et al. SpikeForest, 2019. `https://spikeforest.flatironinstitute.org`.

[44] 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.

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

[46] 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`.

[47] 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.

[48] 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.

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

[50] 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.

[51] J. Müller, M. Ballini, P. Livi, Y. Chen, M. Radivojevic, A. Shadmani, V. Viswam, I. L. Jones, M. Fiscella, R. Diggelmann, A. Stettler, U. Frey, D. J. Bakkum, A. Hierlemann, J. Muller, M. Ballini, P. Livi, Y. Chen, M. Radivojevic, A. Shadmani, V. Viswam, I. L. Jones, M. Fiscella, R. Diggelmann, A. Stettler, U. Frey, D. J. Bakkum, and A. Hierlemann. High-resolution CMOS MEA platform to study neurons at subcellular, cellular, and network levels. *Lab on a Chip*, 15(13):2767–2780, 2015.

[52] 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.

[53] 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.

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

[55] 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.

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

[57] 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.

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

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

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

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] L. Scrucca, M. Fop, T. B. Murphy, and A. E. Raftery. mclust 5: clustering, classification and density estimation using gaussian finite mixture models. *The R journal*, 8(1):289, 2016.

[66] 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.

[67] A. Sobolev, A. Stoewer, A. Leonhardt, P. L. Rautenberg, C. J. Kellner, C. Garbers, and T. Wachtler. Integrated platform and api for electrophysiological data. *Frontiers in neuroinformatics*, 8:32, 2014.

[68] A. Stoewer, C. J. Kellner, J. Benda, T. Wachtler, and J. Grewe. File format and library for neuroscience data and metadata.

[69] J. Teeters, J. Benda, A. Davison, S. Eglen, R. C. Gerkin, J. Grethe, J. Grewe, K. Harris, C. Kellner, Y. L. Franc, et al. Requirements for storing electrophysiology data. *arXiv preprint arXiv:1605.07673*, 2016.

[70] 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.

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

[72] M. Weeks, M. Jessop, M. Fletcher, V. Hodge, T. Jackson, and J. Austin. The carmen software as a service infrastructure. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1983):20120080, 2013.

[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.