eLife

Reviewed Preprint
v1 • April 1, 2026
Not revised

For correspondence:
samuel.garcia@cnrs.fr
pierre.yger@inserm.fr

Competing interests: No competing interests declared

Funding: See page 24

Reviewing editor: Adrien Peyrache, McGill University, Canada

Opening the black box: a modular approach to spike sorting

Samuel Garcia[1], Chris Halcrow[2], Charlie Windolf[3], Zachary M McKenzie[4,5], Paul Adkisson-Floro[6], Herberto Ramon Mayorquin[6], Benjamin Dichter[6], Alessio P Buccino[6,7], Pierre Yger[8]

[1]Centre de Recherche en Neuroscience de Lyon, CNRS, Lyon, France • [2]University of Edinburgh, Edinburgh, United Kingdom • [3]Columbia University, New York, United States • [4]Harvard Medical School, Boston, United States • [5]Massachusetts General Hospital, Boston, United States • [6]CatalystNeuro, Casper, United States • [7]Allen Institute for Neural Dynamics, Seattle, United States • [8]Lille Neurosciences & Cognition (LilNCog) – U1172 (INSERM, Lille), Univ Lille, CHU Lille, Lille, France

eLife Assessment

This **valuable** study introduces a new framework for improving the automated sorting of extracellular action potentials. However, the evidence is **incomplete**; the biophysical model used for simulation is based on one simulation that does not necessarily reflect real experimental data, the test datasets are insufficiently diverse, and essential algorithmic details are currently missing. This work will be of interest to neuroscientists using high-density multichannel electrophysiology.

https://doi.org/10.7554/eLife.110588.1.sa3

Abstract

Spike sorting is an algorithmic process to extract the activity of individual neurons from extracellular electrophysiology recordings. With the ballooning use of high density probes, such as Neuropixels, this essential processing step is increasingly becoming time consuming and computationally expensive. Although many software tools have been proposed to address spike sorting, they are usually constructed and benchmarked as monolithic “black boxes”, making it difficult to factor out the effects of individual algorithmic steps on the final outcome, especially when varying datasets and parameters. To address this issue, we developed a modular and common framework to develop, benchmark, and assemble the key computational steps that are used in state-of-the-art spike sorting algorithms. Relying on fast and efficient ground truth generation of biophysically plausible recordings, we show that we are able to individually benchmark and precisely quantify the performance of different steps in a spike sorting pipeline (peak detection, feature extraction and clustering, and template matching). We then leverage these results to create a modular component-based spike sorters that can outperform Kilosort4 on large and dense simulated recordings. In addition, we find that the major bottleneck of all modern spike sorting pipelines is in the physical motion of probes, regardless of the drift-correction strategy. The presented component-based spike sorting framework has the potential to foster community engagement in the field by lowering the barrier to contributions, and provides a flexible yet powerful framework to construct end-to-end spike sorting solutions.

Introduction

Spike sorting is a processing step to extract individual spike times of individual neurons from extracellular recordings [5, 25]. With the development and widespread adoption of high-density micro-electrode arrays both for *in-vivo* [1, 20, 43] and *in-vitro* applications [18, 39, 50], automated

solutions to the spike sorting problem are essential. In recent years, the neuroscience community has therefore produced a wide range of open-source tools to optimize spike sorting [8, 9, 10, 17, 19, 24, 32, 35, 37, 56].

Almost all spike sorting algorithms follow a similar data processing pipeline which we break down into five components: preprocessing, peak detection, feature extraction, clustering and template matching. In more detail: raw signals are preprocessed (sometimes with correction for probe motion), then a selection of putative spikes are detected. Features from these spikes are computed and used to form clusters. Templates, representing the spatial-temporal footprint of the "average" spike in the cluster, are then computed to detect spikes using a deconvolution method. Different spike sorters use different methods for each sorter component (see Figure 1 for an example comparing `Kilosort2`, `Kilosort4`, and `SpyKING-CIRCUS`). Despite this shared structure, each spike sorting tool is designed and used as an end-to-end "monolithic" pipeline, which leads to several problems.

First, it is difficult to test, implement, and contribute new methods for individual spike sorting steps. While there has been continuous algorithmic development in spike sorting with new ideas and concepts used to tackle motion correction [14, 35, 49], peak detection [22, 38, 45], feature extraction and clustering [11, 21, 41, 53, 57], and template matching [40, 51, 52], these potentially powerful methods for spike sorting remain largely unused, mainly because they are not embedded in a ready-to-use spike sorting tool. This challenge creates a very high-barrier for developers to enter the spike sorting field.

Second, it is also difficult to benchmark new algorithmic ideas. Although notable efforts towards end-to-end spike sorting comparisons have been proposed [27], direct benchmarks of individual spike sorting steps are still limited and scattered in the field. Every new contribution, such as a new peak detection or clustering method, requires a large overhead effort to compare it to other methods, i.e., ad-hoc and non-validated re-implementations, or the development of sub-optimal integrated pipelines to compare with other existing tools. It is also currently very difficult to ask questions of the type "how does a new peak detection method affect the downstream results of a given spike sorter?".

Finally, it is a challenge for developers to maintain their spike sorting packages. The maintenance burden of the tools falls almost exclusively on the original developers, who, in most cases, follow academic incentives [31] and move on to other projects. In an era where hardware and core software are in constant development, software maintenance is an essential part of software development. For example, `Python` versions have an official support of 5 years from their release (`Python` 3.10, released in late 2020, will reach end-of-life in October 2026), which makes software developed in older Python versions virtually uninstallable. Similarly, `MATLAB` versions have rolling support for GPU hardware, which makes GPU-accelerated software developed in older `MATLAB` versions (e.g., `Kilosort` 1/2/2.5/3) difficult to sync with modern hardware. In general, the software maintenance problem can render otherwise state-of-the-art tools unusable within a few years of their development. This is a genuine problem in academic software development and most of the spike sorting tools mentioned above are currently unmaintained (the GitHub repositories have no commits over the last two years): `Klusta` [37], `JRClust`/`IronClust` [19], `SpyKING-CIRCUS` [56], `HDSort` [10], `WaveClus` [8], `YASS` [24].

In this paper, we propose an alternative framework to provide a modular and community-based paradigm for spike sorting development. Building on top of the `SpikeInterface` framework [6], a mature and well-maintained software ecosystem for extracellular electrophysiology data analysis, we introduce a new `sortingcomponents` module. This module provides a flexible modular framework based on the five sorting components discussed earlier. The `sortingcomponents` module includes several methods for each component, with an extensible framework to support new contributions. Each component is paired with a benchmarking tool, allowing for easy comparisons between existing methods and new ones. We believe that the approach implemented here can solve many of the issues detailed above.

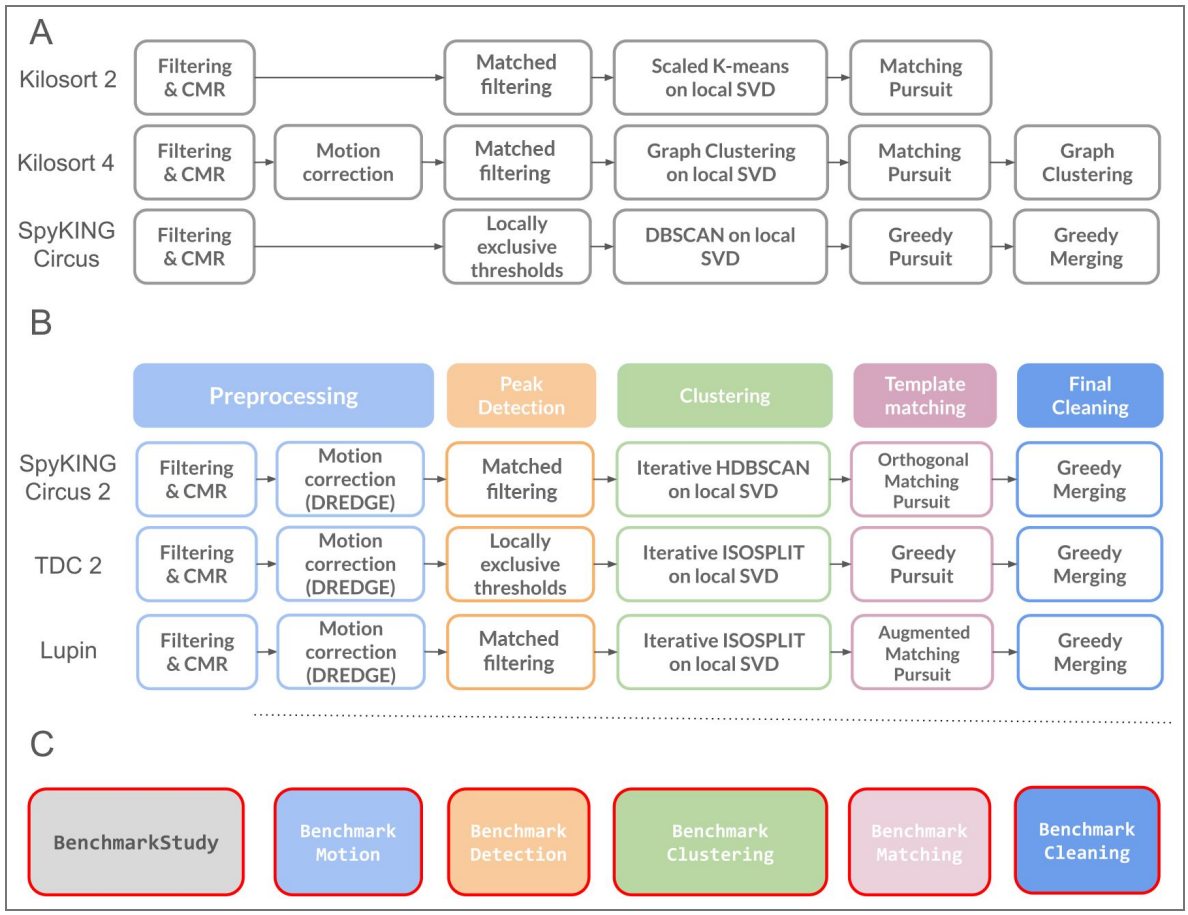

Figure 1.

A Most of modern spike sorters tend to have the same sequence of algorithmic steps, with particular details and/or implementations choices. Particular examples are shown for `Kilosort2` and 4 [32, 35], alongside `SpyKING-CIRCUS` [56]. **B** For each of these key algorithmic steps, we factorized and re-implemented various algorithms to properly benchmark each of them on a per step basis, bypassing the need to compute performance on the whole sequence. **C** Relying on the modular architecture, `BenchMarkStudy` objects on a per-step basis can be created, to test/optimize individual components.

To kickstart the proposed community effort, we have re-implemented or ported several methods available in the literature and from existing spike sorting tools. In the following sections, we first introduce the `sortingcomponents` modular framework and present a fast and efficient ground-truth simulation module, which includes a drift simulation [14]. Next, we showcase the framework with a thorough benchmark of the currently available methods for peak detection, feature extraction and clustering, and template matching. We then go into detail on the template matching failure mode, leveraging our ground-truth simulations to pinpoint issues with interpolation as a source of spike sorting failures for recordings with drift. Finally, we compare end-to-end spike sorting solutions implemented *via* this `sortingcomponents`-based solution with `Kilosort4` [32], which is arguably the most widely used state-of-the-art spike sorter in the field. To demonstrate the net gain of using this component-based approach, we present three new sorting pipelines built with the modular approach of the components: `SpyKING-CIRCUS 2,` an enhanced version of `SpyKING-CIRCUS` [56], `TriDesClous 2`,and `Lupin`, a new component-based sorter created as the optimal combination of all the best-performing components. In our simulated benchmarks, we show that such a sorter outperforms `Kilosort4` both in performance and speed, highlighting the clear benefit of step-wise benchmarks and modularity.

Results

A modular approach to the spike sorting problem

In recent years, a broad consensus has emerged among the latest spike sorting algorithms on what a typical algorithmic pipeline should be. On a macroscopic scale and taking into account some internal subtleties in preprocessing, most modern algorithms are now structured like those shown in Figure 1A. This involves preprocessing (typically filtering and denoising *via* Common Median Reference – CMR, whitening), peak detection, feature extraction, then clustering to obtain a dictionary of templates, before a template matching step and a final gathering and/or cleaning of the results. Although the exact nature of all these steps may vary and is out of the scope of this current paper (see Methods of the aformentionned spike sorters), all modern spike sorters including `Kilosort` [32, 35], `SpyKING-CIRCUS` [56], `MountainSort` [9], `YASS` [24] and `TriDesClous` follow such a pipeline. However, these pipelines are always evaluated as “black boxes” [27] such that it is hard to properly assess the pros and cons of each of the individual steps.

We have implemented a new `sortingcomponents` module in the popular `SpikeInterface` package. Each step in a spike sorter is called a component, and one can now implement a full spike sorter as a series of components, adjusting each one for fine granularity over the pipeline. As can be seen in Figure 1B, we port three new spike sorters to this conceptual architecture: while `SpyKING-CIRCUS 2` and `TriDesClous 2` are direct evolutions of previous algorithms, `Lupin` is a new spike sorting algorithm (see Methods), created to demonstrate the power of such modularity. While the three pipelines share similar processing pipelines, sometimes even with similar algorithms, they differ by the particular parameters of the algorithms they are using, and thus have different pros and cons, as it will be shown in the paper.

More importantly, as seen in Figure 1C, for every key algorithmic step, we have also implemented a higher-order object termed a `BenchmarkStudy`, so that one can easily benchmark and assess the performance, at the component level, of the particular algorithm implemented. While the exact nature of the benchmarks depend on the steps (how many peaks are detected for a detection method, how many clusters can be found for a clustering method, etc.), it is indeed crucial to be able to open the boxes of the spike sorters in order to combine their strengths and identify their weaknesses more easily along the course of the algorithmic pipelines.

The `sortingcomponents` module comes with a built-in engine to parallelize the computational load, splitting the recordings into temporal chunks and relying on multi-core computing. This ensures that the flexibility and modularity offered do not adversely affect the processing times of the algorithms.

The `BenchmarkStudy` object can be used to compare different methods in a given step or to compare different parameters of the same method. The new object can drastically cut the time needed for developers to implement a new algorithmic idea and compare it to other available methods, without benchmarking the whole pipeline, which can itself bias the results. This modularity framework thus turns spike sorting into a community-based effort, where the pros and cons of individual steps can be efficiently understood, and the best ones reused in many different pipelines.

A fast and efficient ground truth generation module

In order to quickly benchmark the numerous components of the spike sorting methods, one needs to be able to generate artificial ground truth data in a fast and efficient manner. This is why, within `SpikeInterface`, we designed and implemented a powerful and fast way to generate artificial ground truth recordings. As shown in Figure 2A, ground truth recordings with any number of neurons can be generated on-the-fly as soon as a probe layout is provided. Each neuron can have a unique spatiotemporal waveform (also known as a template), which can be generated from a mathematical model or loaded from external libraries, as well as a position and firing rate. Each probe channel can have its own noise profile, with or without correlations between channels.

In addition, to challenge modern spike sorters and make our ground truth recordings more realistic we allowed for the inclusion of possible motion (or drift) of the tissue. As noted in recent papers [5], this is currently one of the major bottlenecks in spike sorting, and the algorithmic methods designed to solve the problem are not entirely satisfactory [14]. In our generator, neurons can drift with precomputed and imposed motion vectors, including allowing for non-uniform motion which changes as a function of the neuronal depth along the recording probes. If neuronal waveforms are generated from a mathematical model (see Methods), as drift occurs the waveforms are modified accordingly as functions of the updated positions of the somas (see Figure 2B). On the other hand, if templates are taken from external libraries, then they are interpolated as functions of the drift. In the rest of the paper, we will consider 10 minute long recordings with a Neuropixels-like layout (384 channels), sampled at 30 kHz, with a trimodal distribution profile for the positions of the cells (see Figure 2C and Methods). Note that in our generator, we made the choice to add noise as a multivariate normal distribution, but other options could be easily implemented (1/f and/or multi-unit activity such as in [32], many background units [54], or accurate noise spectrum [44]).

In Figure 2D, we show some macroscopic statistics from an example of such an artificially generated recording (see Methods for more details). The rigid motion was generated as a random walk, while firing rates were drawn from a gamma distribution (see Methods). The analytical model used to generate the templates as a function of the position provides a large diversity of template shapes and amplitudes. This is visible in the distribution of signal-to-noise ratios for the templates, and in the distribution of Euclidean distances between pairs of templates.

Benchmarking individual spike sorting steps improves overall spike sorting quality

To demonstrate the power of our modular approach to benchmarking spike sorting, we combined our ground truth generator with `BenchmarkStudy` objects to assess where mistakes in spike sorting algorithms originate.

Peak detection

The first step we assessed with the proposed framework was peak detection. This first step dramatically influences how the various spike sorters are able to identify units and eventually reconstruct the signals. Here, we compare the two most commonly used methods in the recent literature, one termed “locally exclusive” and the other termed “matched filtering” (see Methods). In brief, the first method detects peaks as local extrema within a spatio-temporal exclusion radius (to take into account the fact that when the channel density is high, action potentials emitted by

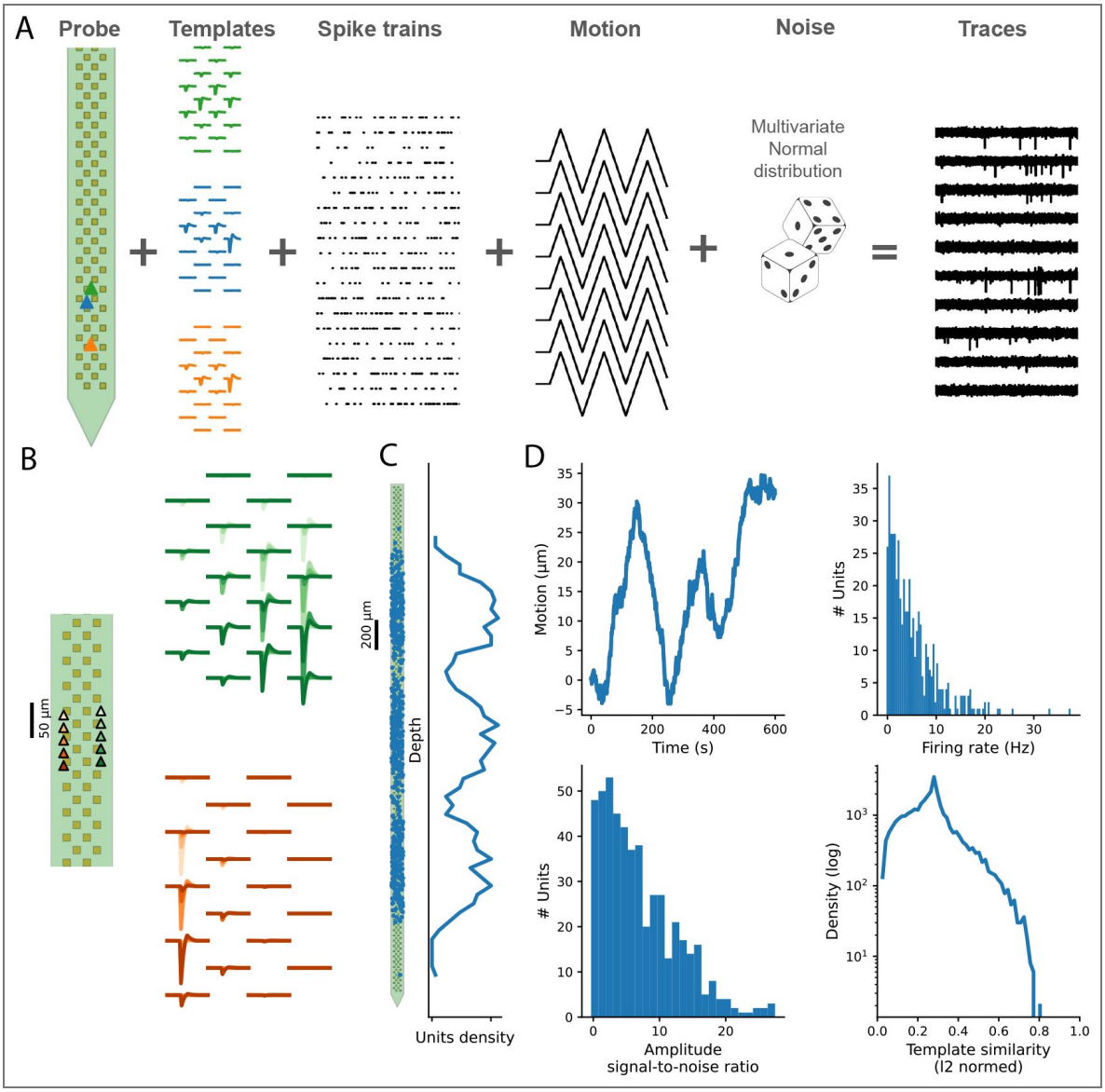

Figure 2. Ground truth recording generation.

A Given a probe layout, templates, activity patterns and inhomogeneous motion vectors, we can generate, on-demand and lazily, extracellular traces to extensively benchmark online spike sorting. **B** When neurons are drifting, templates will be dynamically generated as functions of updated positions of the sources. **C** The positions of the cells are drawn randomly from a trimodal Gaussian along the depth of a Neuropixels-like probe (see Methods) **D** Top left: the rigid motion vector impacting all the cells during the course of the recording. Bottom Left: the distribution of the signal-to-noise ratio for all the templates generated given the cells positions. Top Right: the distribution of firing rates for all the cells in the recording. Bottom Right: the distribution of Euclidean distances between the templates, for all pairs of cells in the recording, on a log scale.

neurons are likely to be seen on many channels simultaneously). The second method, introduced in [32, 43], relies on the concept of matched filters [46] and the fact that, while detecting peaks, because we know roughly the spatio-temporal shapes of the motifs we are looking for in the extracellular signal, we can artificially boost the signal-to-noise ratio. However, the difference between these two methods, when plugged in and integrated in spike sorters, has not yet been thoroughly examined.

To benchmark the quality of peak detection algorithms, we generated one static ground truth recording (see Methods) and assessed how many of the ground truth peaks each method detected. Figure 3A-C shows that, as expected with the locally exclusive method (used with a detection threshold of 5 MADs - median absolute deviations, which is "standard" in spike sorting), very few peaks below the detection thresholds are detected. However the matched filtering method can recover some peaks below thresholds, even if the impact is not drastic. In Figure 3 B and C we plot the recall and precision for each unit from a single generated recording, for both methods. These are plotted as a function of the signal-to-noise ratio (SNR) of the units, with a rolling average overlayed. The plots show that for low snr, recall and precision are higher for the matched filtering method. However, the run times of the two methods show that matched filtering is more computationally demanding. In the future our implementation of this method could be optimized with graphical processing units (GPU) since it mostly relies on computing convolutions.

Overall, one can see in Figure 3B and C that while matched filtering is slightly better at detecting small templates, this positive effect comes with some computational costs, as shown in Figure 3D. In addition, note that matched filtering should mostly be beneficial when the electrode density is high enough, as it uses spatial redundancy of spike detectability. It is also interesting to note that, as observed in Figure 3B, even with matched filtering the recall starts to decrease for SNR lower than 20. This means that even for quite large peaks, some are missed because of collisions and/or noise, and this observation is a strong argument for additional steps, such as template matching, in order to recover these putative spikes. The locally exclusive method seems to be better at finding very large peaks (Figure 3A. This could be partially explained in light of spike collisions: when two large spikes overlap in space and time, the resulting waveform could be highly distorted. Since matched filtering looks for "stereotypical" shapes, it might miss these events. The locally exclusive method could still detect them since it simply looks at threshold crossings. Finally, it is also important to note that missing a few spikes at peak detection will not have a large impact on the overall spike sorting, since the goal of the final template matching step is to recover all spikes in the traces.

Feature extraction and clustering

The next step that can be evaluated as a standalone module is the clustering step. Because most spike sorters currently rely on the same feature extraction, i.e. Singular Value Decomposition [24, 33, 34, 56] (see [25] for a review), we decided not to benchmark this particular extraction step and focus only on the clustering. Therefore, in the following, we will consider the extraction of features and the clustering as a whole, but we hope that future efforts will additionally assess feature extraction methods and individually benchmark each of the steps. Clustering is a crucial, if not the most crucial, step in spike sorting to disambiguate the individual neurons. Here again, spike sorting algorithms have the tendency to combine all steps such that benchmarking the effect of clustering alone is almost impossible. In Figure 4A (top), we compare the clustering algorithms of various spike sorting tools, re-implemented as standalone modules. All these algorithms are given the same inputs (i.e. the exact peaks times from the ground-truth recordings) and evaluated in a similar manner, suppressing any biases that might be due to differences in peak detection methods. Given the fact that we are providing the theoretically ideal input data to these algorithms, i.e., the ground truth spike times, we expect to get an upper bound of their performances. In order to quantify the effects of motion correction on these algorithms, we applied the analysis to two conceptually similar datasets: one termed "static" and one termed "motion-corrected" (see Methods for more details). The latter corresponds to a recording whose cells and activities are exactly the same as the static one, except that they are moving according to a rigid motion. Motion is then compensated for and corrected using the current state-of-the-art

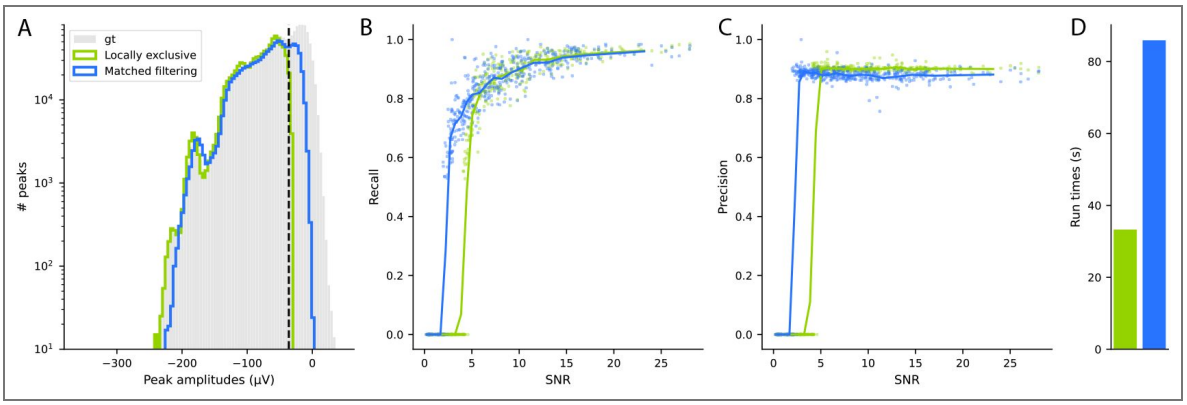

Figure 3. **Peak detection benchmark.**

A Given the true distribution of the peak amplitudes in the ground truth recordings (grey area), we compute the recovered distribution for the peaks detected either via the "locally exclusive" method (blue), or the "matched filtering" (green). **B** The recall for all the neurons in the ground truth recording as function of their signal-to-noise ratios for the two aforementioned methods. **C** Same as in **B**, but for Precision. **D** The run times (in seconds) for the two methods.

motion correction algorithm (see Methods and [14, 49]) such that, in theory, the two recordings (static and motion-corrected) should be equivalent, up to the interpolation of the traces when motion is compensated.

Figure 4A compares the performances of several algorithms (see Methods for a full description of their implementations). One can immediately see that they all perform well on static data. Iter-HDBSCAN (used in `SpyKING-CIRCUS 2` [56]) and Iter-ISOSPLIT (used in `TriDesClous 2` and `Lupin`) are very similar, both algorithms are based on iterative splits of local density-based clustering performed by grouping the spikes per electrode. KS-clustering is a re-implementation of the local graph clustering found in [32]. Global-Louvain is an attempt to design a single unified graph-based clustering on sparse connectivity matrices (see Methods). Note that, in Figure 4C, the number of units found is already far from perfect on static recordings (at best, 420 out of 500 units). We observe that clustering mostly fails for small SNR units, as opposed to small firing rate units (see Figure 4A left vs right). Hence the failure to observe all units is driven by units with small SNRs being indistinguishable from noise.

The performance of all clustering degrades drastically for motion-corrected data. As seen in Figure 4B and in Figure 4C, because the motion is only partially compensated for, the clustering algorithms struggle to properly find the units, in contrast to the static case. We can see a dramatic drop in performance, specifically in the number of not well-detected units (see Figure 4E) for methods that rely on local clustering (such as KS-clustering). This is, to some extent, less true for global clustering, such as the Global-Louvain one, or for clustering solutions that explicitly remove very small clusters under the hypothesis that they are likely to be too noisy (Iterative-HDBSCAN and Iterative-Isosplit, see Methods). Here again, the influence of signal to noise ratio seems to prevail over firing rates (see Figure 4B, right). Regarding the run times (Figure 4D), it is important to stress that Iterative methods of the `sortingcomponents` framework (Iter-HDBSCAN and Iter-ISOSPLIT) are fast, since they rely heavily on parallelization over cpu-cores.

Template matching

We also assessed how good the various template matching strategies used by different spike sorters are, bypassing the pitfalls and limitations of the peak detection and clustering steps. We compared the main algorithms available in the field and provided the ground truth template "catalog" as input. Currently, most of the algorithms are using a template matching approach, looking in the extracellular signal for all the possible times where there are matches between the templates in the catalog and the spike waveforms. Although sharing similar principles, however, the exact mathematical natures of these template matching algorithms are different. This is why we precisely compared the classical matching pursuit algorithm implemented in `Kilosort` (KS-matching [35]), a refined Orthogonal Template matching (Circus-OMP [36]), an augmented matching pursuit algorithm with temporal super-resolution (Wobble [24]), and a simpler greedy pursuit (TDC-peeler) (see Methods for more details on all these algorithms).

As can be seen in Figure 5A-C, all algorithms perform similarly at reconstructing the signal *through* template matching based approaches, with an accuracy that rises sharply as a function of the SNR of neurons, at least for static recordings (panel A). Template-matching based solutions (KS-matching, Circus-OMP and Wobble), based on full convolution of the traces with the catalog of templates, have a better accuracy for small units compared to greedy pursuit (TDC-peeler). However, once again, as soon as we are dealing with motion-corrected recordings (see Methods), the interpolation of the traces considerably blurs the signals such that performances are severely impacted for all algorithms (panel B). Although there are some differences with respect to run times (Figure 5D), the choice of the template matching engine might depend on some other factors (correlation levels, firing rates, etc.) that could be tested to make a well-informed choice depending on the input data. For example, as shown in Figure 5E, which shows the recall over a distribution of lags between spikes, the Wobble algorithm has the best performance when it comes to spike collisions, i.e, spikes that overlap in space and time (the results are displayed for static recordings, but similar results are obtained for motioncorrected ones). Assuming that fine

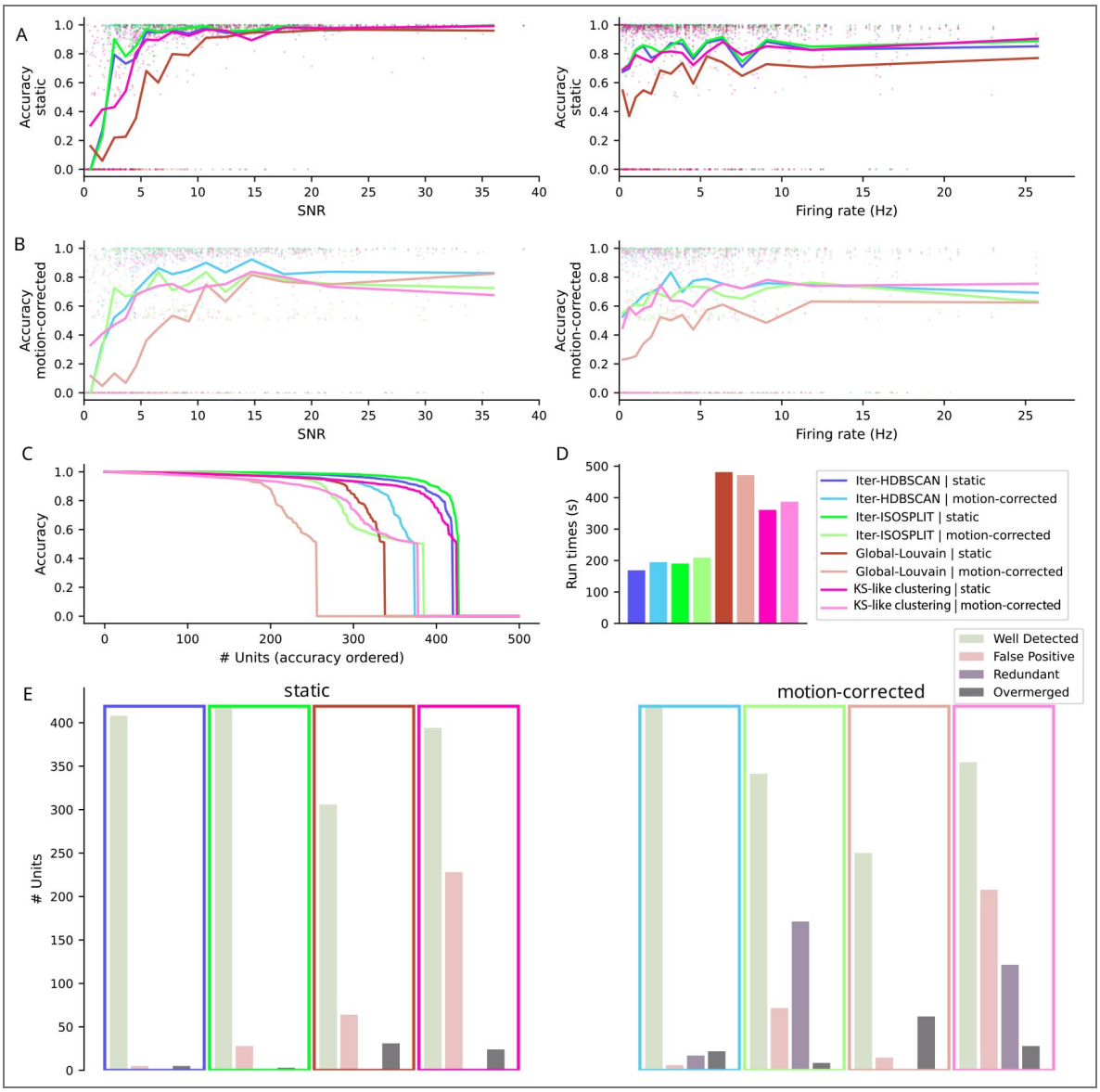

Figure 4. Feature extraction and clustering benchmark.

A Accuracy recall for all the clustering methods when applied on true peak times from a static recording. Left: as function of the signal-to-noise ratios of the neurons. Right: as function of the firing-rate of the neurons **B** Same as in **A**, but applied to motion-corrected recordings, with the exact same activity/firing as the static one (see Methods) **C** Sorted accuracy levels for all clustering methods and recording types, as function of the neurons present in the artificial recordings **D** The run times for all the methods and datasets **E** For all clustering methods and recording types, the number of Well Detected, False Positive, Redundant and Overmerged units (see Methods)

correlations are important to understand how information is finely encoded by a neuronal population, one might want to favor Augmented Matching Pursuit algorithms such as Wobble, even if they are slightly slower than the others.

Motion interpolation reduces spike sorting performance

As is shown in Figures 5, all template matching algorithms suffer from a lack of performance when dealing with motion-corrected recordings. This is not due to the estimation step, since motion can be estimated very accurately the DREDGE algorithm [49] and even providing ground truth motion strongly degrades spike sorting performance [14]. Instead, it is mainly due to the interpolation of the traces using the kriging kernel method [32]. The method constructs a kernel for every time bin of the estimated motion (1 s in our case) based on the motion vector, which is applied using a scalar product to the traces to compensate for motion. In short, every sample will be interpolated in space by the weighted average of the neighboring channels, and this should compensate for the motion itself. Our previous work [14] has already shown a strong degradation in the performance of spike sorters when drift is present, which is mainly due to this interpolation step. Thanks to the modular approach introduced in this paper, we can now highlight how this performance loss occurs due to clustering (Figure 4C-E), and from template-matching (Figure 5C).

Knowing that motion interpolation causes a major performance decrease in spike sorting, as shown in Figures 4C-5C, we implemented a new idea for template matching. Instead of interpolating the traces, one could directly interpolate the templates instead [3]. This has three advantages: 1) the interpolation of the templates should be less noisy because the templates are smoother than traces; 2) the templates could be pre-computed for some predefined motion steps (i.e. spatial bins) to cover the entire motion vector, resulting in faster computation; 3) the interpolation could potentially be made via better methods (like spline, i.e. cubic interpolation). Such interpolation procedures, because they are more computationally demanding, cannot be performed on-the-fly at the raw trace level but could be performed once on the templates. In Figure 6, we tested this idea of interpolating the templates instead of the traces during the template matching step. The core nature of the matching problem can be observed in Figure 6A. Here, we compare the “True” drifting templates (as defined by the generative mathematical model as a function of the cell position, see Methods) and the best interpolation that one can do with bi-cubic splines on a grid for a given motion vector. As can be seen, there are still some important non-zero residuals which will give rise to errors.

To check the quantitative impact of such errors in the interpolation, we extended the Greedy Matching Pursuit algorithm (TDC-peeler) with two modes: the standard one (where templates are kept fixed) and a new one with a “so-called” drifting template based algorithm, where we pre-compute interpolated versions of each template in a spatial range ([-100 μm, 100 μm]) for a given spatial step (1 μm)). In this new mode, instead of interpolating the traces, we will interpolate templates at any given time and motion, prior to the template-matching step. Given the modular nature of the `SpikeInterface` framework, we can compare these algorithms for template matching. In total, we compare five situations:

1. “Static - Estimated Templates”: no drift is present in the recording, templates are estimated from raw data
2. “Static - True templates”: no drift is present, true templates are taken from the biophysical model (upper bound on performance)
3. “Traces interpolation”: drift is present in the recording, traces are motion-corrected as a preprocessing step
4. “Templates interpolation”: drift is present in the recording, templates are estimated, and then motion-corrected via bicubic splines before template matching
5. “True drifting templates”: drift is present in the recording, perfect templates from the biophysical model are used (upper bound on performance)

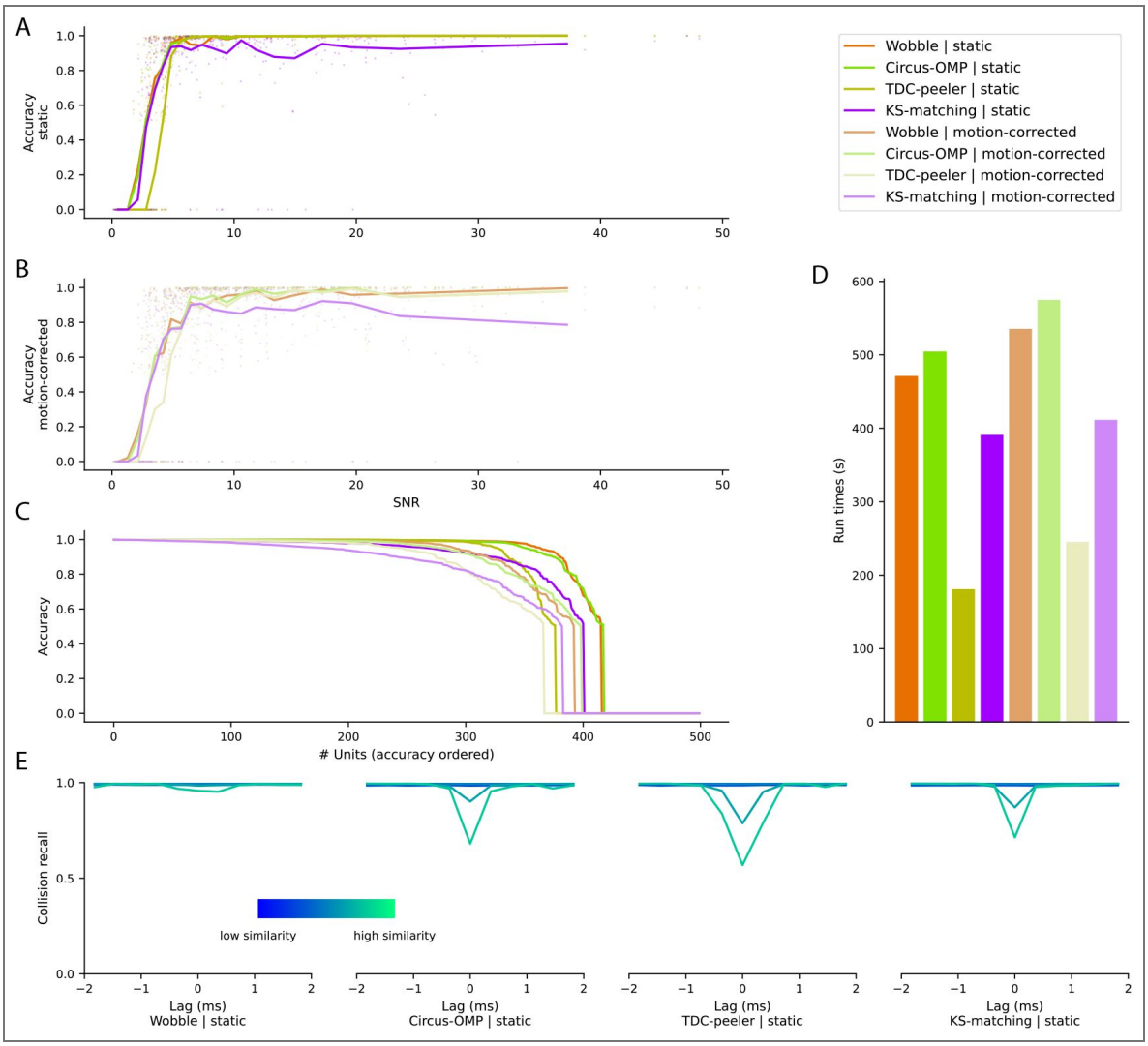

Figure 5. Template matching benchmark.

A Accuracy recall for all the matching methods when launched with a perfect catalog of templates, and as a function of the signal-to-noise ratios of the neurons **B** Same as in **A**, but when applied to a motion-corrected recording, with the exact same activity/firing as the static one (see Methods) **C** Sorted accuracy levels for all matching methods and recording types, as a function of the neurons present in the artificial recordings **D** The run times for all the methods and datasets **E** For all matching methods, collisions levels [12] computed on the static recording, as a function of the temporal lag between all pairs of spikes, and the cosine similarity between pairs of templates.

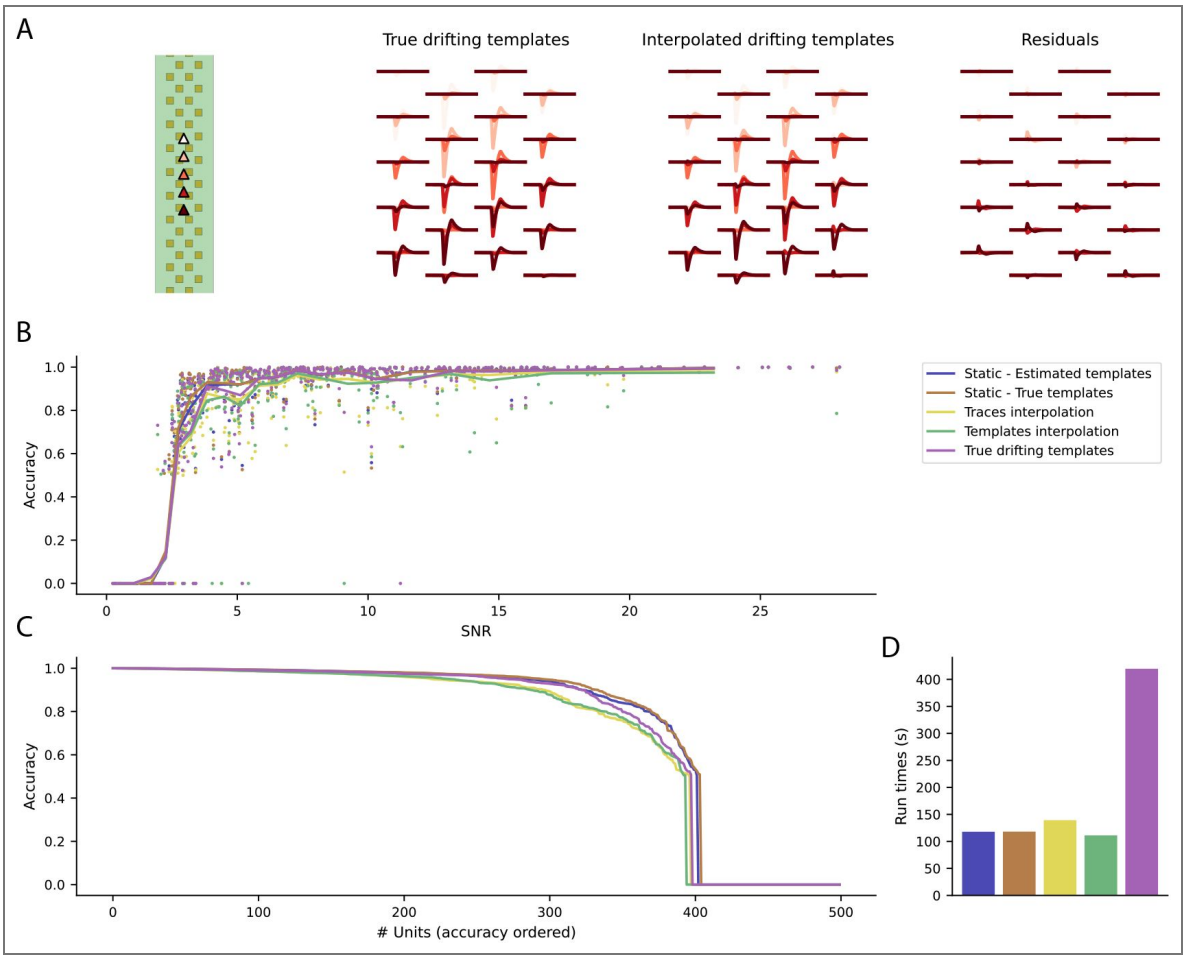

Figure 6. **Motion correction strategies benchmark.**

A Example of template interpolation for a particular neuron drifting along the depth of the electrode. On the Left, the real templates are generated from the generative mathematical model, taking into account the real position of the neuron. In the middle, templates are interpolated *via* cubic spline interpolation for a particular displacement. On the right, the residuals (i.e. the differences) between estimated and real templates, as functions of the positions **B** Accuracy for several matching methods when run on a static recording, when Templates or Traces are interpolated given the estimated motion, or when we use the True static or drifting templates (perfect dictionaries of templates) and as function of the signal-to-noise ratios of the neurons **C** Sorted accuracy levels for all matching methods and types of dictionary, as function of the neurons present in the artificial recordings **D** The run times for all the methods and datasets

The result of the benchmark is shown in 6. As expected, the best performance occurs for static recordings. For recordings with drift, the performances of both traces-based and templates-based interpolations are similar and the accuracy is almost the same (Figure 6B,C). The results also show that estimating the templates from raw data or using real ground truth templates from the biophysical model has no clear impact on the performances for static recordings (Figure 6B,C). This is a good sign that, even for low-firing neurons, estimating the templates from a small subset of spikes does not dramatically harm the matching procedure.

To investigate further, we ran a final case, referred to as "True drifting templates". In this case, the templates used by the template-matching engine for any given spike are the "true" templates generated by the ground truth simulator (knowing the position of the underlying neuron at each time point). Although we are aware that this case is impossible with biological recordings, as we do not have access to such information, this gives an upper bound on how good we could expect the matching to be. It also allows us to estimate the cost of spatial interpolation to compensate for the drift. As shown in Figure 6B-C, the result is quite clear: the matching performance of "True drifting templates" is not as good as for the static recordings, but is better than the other methods applied to drifting recordings (with an increased computational cost, Figure 6D, since true drifting templates are denser than interpolated ones). This leads us to the conclusion that, in a recording with drift, estimating the drift can be done accurately, but correcting for it by an interpolation on traces or templates still greatly degrades the ability of spike sorting. This conclusion paves the way for future improvements, i.e. finding better ways of interpolating either templates or traces when motion is known or estimated.

End-to-end evaluation of component-based sorters

Finally, the modular benchmark proposed in this article allows one to perform exhaustive comparisons of fully integrated spike sorters. In order to get a robust estimate of the performances, we computed the performance of five end-to-end spike sorters on multiple instances of artificially generated recordings, all with the same properties (see Methods). To demonstrate the full potential of the modular framework described in this paper, we compared a state-of-the-art spike sorting algorithm (`Kilosort4` [32]) to several new sorters entirely built on the components described in this article, `SpyKING CIRCUS 2`, `TriDesClous 2` and `Lupin` (see Methods), and also to a partial re-implementation of `Kilosort4` built on the components, termed `Kilosort-like` (see Methods).

As can be seen in Figure 7A, the performances for static recordings are rather good, although all sorters sometimes missed obvious units with a large signal-to-noise ratio. This is likely due to similar looking units, i.e. units whose physical positions are close in space, thus leading to similar extracellular waveforms that are hard to disambiguate for the clustering algorithms. However, when recordings have drift and are motion-corrected, there is a similarly large decrease in performance for all spike sorters, as displayed in Figure 7B. As we saw in previous figures, this is mostly because both clustering and template-matching steps are severely impacted by the interpolation required when correcting motion.

Several observations can be made from Figure 7. First, as shown in Figure 7C, `Lupin` is able to outperform all other spike sorters both in the static and in the motion-corrected case. This is because such a pipeline is built with the "best" algorithms carefully chosen at each step of the spike sorting pipeline as described in Figure 1. We can also see how motion-correction harms all the results in several ways, regardless of the spike sorting method: by degrading the overall number of cells that can be recovered, by slowing down all pipelines (Figure 7D) and by increasing the number of False Positives, especially for `Kilosort4` and the `Kilosort-like` implementation (see Figure 7E).

It is important to note that our `Kilosort-like` sorter, implemented via our modular implementation (see Methods) is not as good as `Kilosort4`, and this is especially pronounced in the motioncorrected case. This is because `Kilosort`, after template matching, applies a final re-clustering of all detected spikes that we did not yet re-implement. Therefore, the code has a tendency to produce more oversplit units (see Methods), which are labeled as False Positive.

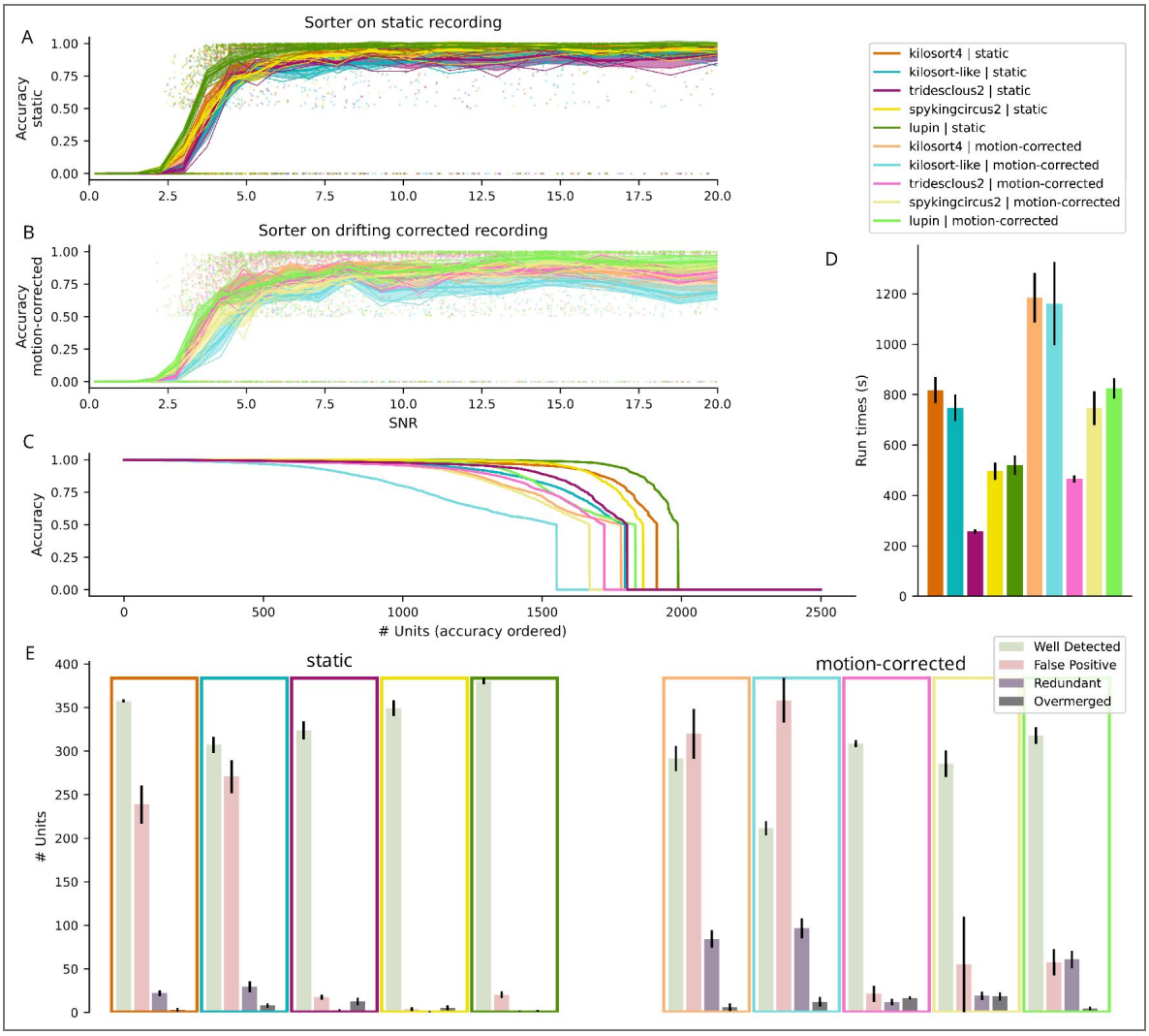

Figure 7. **End-to-end spike sorter benchmark.**

A Averaged accuracy for several spike sorters when applied to several static recordings (with various seeds), as function of the signal-to-noise ratios of the neurons **B** Same as in **A**, but when applied to motion-corrected recordings, with the exact same activity/firing as the static ones (see Methods) **C** Sorted accuracy levels for all spike sorters and recording types, as a function of the total number of neurons present in all artificial recordings **D** The run times for all the methods and datasets **E** For all spike sorting methods and recording types, the number of Well Detected, False Positive, Redundant and Overmerged units (see Methods)

However, it does mimic `Kilosort` and, although a bit slower, we note that it runs on CPUs rather than a GPU. Two of the other spike sorters, built as chains of modular components (`SpyKING-CIRCUS 2` and `TriDesClous 2`), have several pros and cons. `SpyKING-CIRCUS 2` finds more units, at the cost of more False Positives and longer run times (Figure 7D, E). On the other hand, `TriDesClous 2` is rather fast, mostly due to its matching engine (working only on peak times). Such an implementation choice degrades its performances for static recordings, but in the case of motion-corrected ones, the effect is less pronounced. However, the main advantages of these sorters lie in their modularity, so that it is straightforward to test ideas, customize pipelines and adapt parameters depending on the data. Finally, `Lupin` is designed to be the best combination of all currently available components, on our dataset. It clearly outperforms all other sorters in all conditions (static or motion-corrected, including `Kilosort4`), with a good trade-off between the high number of found units, the speed, and the small number of False Positive. Its performance can be seen as a direct illustration of the gain obtained by sharing ideas within the community.

Discussion

In this work, we have presented a new and modular framework to dissect and benchmark all the steps of modern spike sorters, alongside a fast and efficient ground truth generator to quickly create artificial data to challenge the algorithms. This has allowed us to get a better understanding of the pros and cons of each individual algorithmic steps, and, more importantly, allowed us to design new component-based spike sorters (`SpykingCircus 2`, `TriDesClous 2` and `Lupin`). The latter sorter, `Lupin`, which is built using the best-performing method for each step, is already faster and more accurate than the *de-facto* standard (`Kilosort4`) on our Neuropixels simulations, with the worthwhile advantage that users do not need a GPU, especially in the context of High Performance Computing (HPC) and the cost and difficulty of getting access to GPUs.

We hope that our initial (and gigantic) effort of dissecting and re-implementing several methods for each step of any spike sorter into a single unified framework will be beneficial for the spike sorting community at several levels. First, users have access to a better spike sorters to obtain more accurate spike trains from their recordings. Second, developers of new methods will now be able to focus on a specific detail of a specific step (e.g., motion estimation, feature extraction, clustering, template match-ing, or cleaning). They can implement new ideas without needing to write a full new package including the entire machinery (data reader, preprocessing, and user interface) because `SpikeInterface` already handles all these details. Finally, advanced users will be able to construct from scratch or tweak their own spike sorting solution that best fits their needs: a balance between accuracy, computational costs, and speed.

We have demonstrated the validity of our component-based framework to benchmark individual methods and to build full end-to-end spike sorters, given a fast and efficient way of generating ground truth data. Although the generative model that we use to simulate ground truth recordings does not grasp all the complexities observed in experimental settings and could be enhanced, we believe that it already has the key ingredients to challenge spike sorting algorithms, such as cell inhomogeneities, randomness, and drifts. In order to benchmark some of the individual steps, such as peak detection and clustering, datasets with partial ground truth (such as paired recordings or “hybrid” recordings) cannot be used (for peak detection, for example, the ground truth spikes will only be a very minor portion of all spikes in the recording). An alternative source of full ground truth datasets can come from biophysical simulations, which use advanced multi-compartment models to simulate extracellular templates [4] or even the full network activity which give rise to extracellular recordings [23]. While the latter types of simulations might sound more biophysically plausible, it is worth noting that virtually all available cell models are built from *in-vitro* experiments, which could affect neuronal physiology and result in waveform shapes and distribution that are different from what observed *in-vivo* [32]. A second drawback of biophysically plausible simulations is their computational complexity. Our proposed simulation framework trades off some biophysical “correctness” for efficiency and speed: our simulated data are generated almost instantaneously, on-the-fly, and in memory upon request, requiring no disk space. Another source of ground truth that could be viable for some of the steps that do not

require exhaustive ground truth (like template matching) or for end-to-end comparisons is hybrid recordings [7, 32]. A potential problem for hybrid recordings, that could be underestimated, is one of circularity when it comes to motion correction. Hybrid data are built from experimental recordings by injecting ground truth spikes (from collections of templates or pre-curated spike sorting outputs). Since drift is a major phenomenon for shank-like probes, injected spikes should follow the inherent drift of recordings. Spikes are therefore moved and interpolated given the estimated motion. When benchmarking performance of spike sorters on such datasets, a spike sorter that uses the same motion correction method used to inject spikes will be favored. It is therefore important to make sure that motion correction is consistent across methods and potentially run prior to the benchmark.

While it was already shown that drifts in the recording causes a major degradation in spike sorting performance [14], in this work we further highlighted that the problem is mainly in the clustering and template matching steps, and that it is especially due to the way templates and/or traces are interpolated, even using state-of-the-art motion-correction algorithms [48, 49]. Template matching is indeed very sensitive to residuals, and slight mismatches can have a large impact on performance. The resolution of such issues remains an open challenge in spike sorting, and our work does not improve the situation in the field: recordings with high drift amplitudes can have a very poor spike sorting quality. Therefore, it is important to stress that experimenters should try to minimize the drift during acquisition, since motion correction only works to a certain extent. One potential solution to at least ameliorate motion correction could be to make use of data from the new generation of ultra-dense probes, such as Neuropixels Ultra [55]. These novel probes provide an unprecedented spatial resolution, which could help to gain new insights on biophysical features of extracellular templates and use them to design better interpolation methods for the templates.

Embedding this modular spike sorting framework in the `SpikeInterface` framework facilitates continued and distributed maintenance of the new modules. `SpikeInterface` is a mature ecosystem with several core developers across multiple institutes, over one hundred external contributors, an extensive testing suite and continuous integration across multiple operating systems and software infrastructure (e.g., Python versions). Finally, all the methods and options evaluated in this work are readily available to the electrophysiology community within `SpikeInterface` and can be immediately deployed with a few lines of code.

Methods

Notation

Throughout the article, vector variables are represented using $\rightarrow$ notation and convolution with using notation. We consider that the extracellular signals $\vec{s}(t)$ are defined on N channels. We use $\vec{w}_i(t) \in \mathbb{R}^{N \times M}$ to represent the spatio-temporal waveforms emitted by the neuron i at time t, where N is the total number of channels and M is the number of time samples. We further use the term Ground Truth (GT) to refer to the fully controlled variables in our synthetic recordings (either the motion signal or the spike times of the units).

Simulated datasets

We have implemented a new module in `SpikeInterface` to ease the generation of ground truth artificial recordings, with or without motion drifts to mimic what has been observed during *in-vivo* experiments [42]. This new strategy allows us to bypass the use of the `MEArec` simulator [4], at the cost of a slight loss in pseudo-realism.

To be more precise, the `generation` module of `SpikeInterface` can now generate, on-the-fly and in memory, any recording given a probe layout supported by `probeinterface` [13]. This lazy mode is crucial, since ground truth recordings can be quite large for high-density probes with long durations. The implementation avoids saving the data to disk and data chunks can be generated upon request in memory only. Moreover this is done in a seeded and reproducible manner, given

the templates, the spike times, and potentially the motion vector $\vec{M(t)}$ that affects the units. Similar to what was done in the `MEArec` simulator, the user can control the number of units, their spike times and their physical positions in space. The spatio-temporal templates of the units $\vec{w}(t)$ can either be extracted from third-party libraries or generated *via* a simple generative mathematical model.

Although the full details of this mathematical model are available in the `generation` module of `SpikeInterface`, we will now explain its core principles. To generate a template given the position of a neuron, we first generate a single prototypal waveform as a sum of decaying exponentials with various time constants to create the typical bi-phasic shape often observed *in-vivo*. Note that this generation procedure has several parameters, such as the peak negative/positive amplitudes, the time constant of the depolarization and of the repolarization and the recovery time. Parameters are always randomly drawn from uniform distributions to make them different for every cell. Once a single waveform has been generated, it is scaled on every nearby channel by a spatial decay factor, formulated as a power law on the distances between the cell and the channel positions. In order to ensure spatial anisotropies while computing these distances, we modeled the cell as an elongated ellipsoid whose axes and rotation are also randomized. Finally, the model also takes into account a propagation speed such that waveforms are also temporally shifted as a function of the distances. Combined together we firmly believe that, while not capturing all the diversity observed *in-vivo* w.r.t cell types, morphologies, etc., the model is able to reproduce most of the core features needed to properly challenge modern spike sorters.

In this paper, we focus mainly on 10 minute long recordings with Neuropixels-1.0 probe layouts, generated with a sampling rate of 30 kHz. The number of units is fixed to 500, with a trimodal distribution of their positions along with the depth of the electrodes. Cell firing rates are drawn from a gamma distribution (shape 1 and scale 5) leading to rates from 0.1 to 30Hz. A key feature of this new module which generates ground-truth recordings is that it can handle motion drifts in two different ways. First, if cells are generated via the biophysical generative model, then they all have a source position $\vec{p} = (x, y, z)$ that can be varied as a function of motion $\vec{M(t)}$ during the course of the experiment. Second, if the unit templates are taken from an external library or provided by the user, the module will handle drifts by interpolating these templates *via* cubic spline interpolation while shifting them with respect to the motion vector $\vec{M(t)}$.

Although the noise structure of these artificial recordings are simpler than the ones observed in real data, it still has some key properties that can be specified by the user to challenge the spike sorters. For example, the user can specify the noise levels per channel and even impose a given covariance matrix for the noise structure within the channels of the probes. Note, however, that the module does not consider any model of Local Field Potentials, thus making it only suitable for spike sorting benchmarks. However, similarly to what has been done in [14], since the user has full access to the ground truth, such a generation module can be used to impose a particular spatial distribution for the neurons (to mimic the layered organization of recorded structures), a particular distribution of firing rates and/or activity profiles (to mimic particular subtypes), and motion drifts that can be non-homogeneous as a function of the electrode depths. Combined all together, we believe that such a lazy generation module can tremendously speed up the development and comparison of spike sorters, not only for benchmark but even maybe, in the future, for the generation of training/labeled dataset that could be beneficial for end-to-end deep network solutions [15, 16].

Peak detection

Locally exclusive

In this method, which is commonly performed in spike sorters [17, 24, 56], peaks are detected as negative threshold crossings within a spatio-temporal exclusion zone, to avoid cross-contamination between channels (this is implemented as the “locally exclusive” method of `SpikeInterface`). The parameters of such a detection method are the detection threshold,

defined as how many times above the median absolute deviation that peaks must be (*detection_threshold*=5), the spatial radius in which such a peak must be a local extrema (*local_radius_um* = 50 µm) and the temporal exclusion zone during which such an extrema must be unique (*exclude_sweep_ms* = 2ms).

Matched filtering

The idea behind this peak detection method is matched filtering [46]. Because spikes have somewhat stereotypical temporal waveforms, we can increase the signal-to-noise ratio of the peaks during the detection process by specifically looking for such shapes in the signal. Introduced in [32], the idea behind this peak detection method is to create an exhaustive catalog $\mathscr{F}$ of artificial templates $\vec{f}_{n\in 1,\dots k}(t)$ at known positions $\vec{p}_n = (x_n, y_n)$, and to convolve the extracellular signal $\vec{s}(t)$ with all these spatio-temporal templates.

To create these artificial templates, the typical waveform $\vec{H}(t)$ of some detected peaks on a single channel is estimated as a median of, say, 10000 normalized waveforms, and then this waveform is duplicated on all nearby channels, such that on every channel *c* at a position $\vec{p}_c = (x_c, y_c)$ we have $\vec{f}_n(c,t) = w_n(c)\vec{H}(t)$ with a weight $w_n(c)$ that will reflect the spatial decay of the templates. In the following, we model the spatial decay as:

$$\vec{f}_n(c,t) = e^{-(\vec{p}_c-\vec{p}_n)^2/(2\sigma^2)}\vec{H}(t) \qquad (1)$$

In this formula σ controls the spatial decay of the templates and to further extend the catalog, multiple values of σ can be used (in the range of 10 to 50 µm). Once the catalog is obtained, all these templates $\mathscr{F}$ are convolved with the extracellular signal $\vec{s}(t)$, giving rise to a new signal $\vec{r}(t) \in \mathbb{R}^K$, where *K* is the number of templates. Typically, *K* is much larger than *N*, the number of channels.

The final peaks are then detected via the aforementioned locally exclusive method, but on this new signal $\vec{r}(t)$. Note that because the stereotypical waveform $\vec{H}(t)$ is the same as the one used to generate all the spatio-temporal templates in the catalog $\mathscr{F}$, the convolutions between $\mathscr{F}$ and $\vec{s}(t)$ can be highly optimized by only performing the convolution between $\vec{s}(t)$ and $\vec{H}(t)$ per channel basis, and then summing these individual convolutions with the weights $w_n(c)$.

Motion estimation

In this paper, and similar to what has already been performed in [14], we use state-of-the-art methods to infer the motion from a Neuropixels-like recording. Note that the modularity described in the present article has already been applied to these motion correction methods [14]. Although `SpikeInterface` offers multiple methods to handle motion correction [14], throughout the paper, we decided to use the one that gives the best results after an exhaustive exploration, i.e., the DREDGE method [47, 49] [14].

Since motion correction is now a canonical step in most preprocessing pipelines for *in-vivo* recordings with high-density probes, it was important to assess how good the various algorithms are with respect to such a key preprocessing step. This is why, in this paper, we often compare the results obtained on static recordings with what we will refer to as “motion-corrected” recordings. To be more precise, for every ground-truth recording that has been generated using the generation module (see Methods), we always create both a static and a drifting version of the recording. What we termed a “motion-corrected” recording is a recording in which the motion has been estimated by DREDGE and then optimally compensated for *via* a kriging interpolation [32]. To our knowledge, this is the closest, albeit not perfect, way to estimate the static recording from the drifting one. However, this motion correction step distorts the signal in two ways. Firstly, because motion estimation is not perfect and depends at least on activity levels and cell positions: we need to have many active cells in a localized region to properly estimate the motion [14]. Secondly, because the kriging interpolation used to compensate for the motion smooths the signal and noise, some information from the data is destroyed during this step.

Clustering

Clustering is one of the most important steps in spike sorting, and numerous methods have been tried. A complete review of all these methods is outside the scope of the manuscript, but in the `SpikeInterface` components module, we have focused on some of the key modern approaches used for high-density probes. Most of these methods start from the simple observation that clustering in a high-dimensional space is an intractable problem, so there is a clear need to reduce the dimensionality before clustering. The most obvious way to do so, while keeping the spatio-temporal features of the waveforms needed for clustering, is to perform a Singular Value Decomposition (SVD). To do so, several methods first gather single-channel waveforms $\vec{w}_i(t) \in \mathbb{R}^{1\times M}$, where M is the number of time steps (usually, the temporal duration of a spike *in-vivo* lasts 2 ms), and then perform a SVD to reduce the number of dimensions from *M* to *K* 5. Assuming that such a SVD projection is rather typical, it can be extended on a per-channel basis to spatiotemporal waveforms such that snippets $\vec{w}_i(t) \in \mathbb{R}^{N\times M}$ observed on N channels can be projected into a space $\mathbb{R}^{N\times K}$. In such lower-dimensional spaces, the clustering is easier. In the following we use $\vec{w}_i^{SVD} \in \mathbb{R}^{N\times K}$ for the projected representation of $\vec{w}_i \in \mathbb{R}^{N\times M}$, after SVD.

KS-clustering

The full details of this clustering method can be found in [32]. In a so-called divide-and-conquer approach, the waveforms $\vec{w}_i^{SVD}$ are split into groups as a function of their estimated depth (the algorithm being mostly tailored for Neuropixels like probes), and a graph-based clustering algorithm is applied per group (as a modified version of the Louvain or Leiden algorithms [2]). The results of all these individual clustering are then concatenated, taking account of the fact that cells at the border of the bin depths could give rise to duplicated clusters, which need to be merged afterwards.

Iterative clusterings (Iter-HDBSCAN and Iter-ISOSPLIT)

This method is similar to the one used in `Kilosort`, but relies on more density-based clustering algorithms. In another so-called divide-and-conquer approach, the waveforms $\vec{w}_i^{SVD}$ are split as a function of their estimated depth (the algorithm is primarily geared toward Neuropixels-like probes), and a clustering algorithm can be applied locally. To give more details, all detected peaks are grouped as function of their peak channels. For all these peaks, channels in a given neighborhood are selected, and projected onto a low-dimensional space to obtain the $\vec{w}_i^{SVD}$. In practice, this can be a density-based clustering algorithm (such as HDBSCAN [30]) or one based on statistical considerations (such as ISOSPLIT [9, 28]). The results of all these individual clusterings are then concatenated, so that cells at the border of the bin depths which could give rise to duplicated clusters, are merged afterwards. To do this, these clustering algorithms contain an additional cleaning step that ensures: 1) all similarities between found templates, as defined by l1 norm, are less than a certain threshold (0.8) 2) All small clusters that are likely to be noisy are removed (given a threshold expressed as a firing rate for found neurons).

Full graph sparse clustering (Global Louvain)

This method is slightly different from the previous ones, to try and avoid the edge effect of the divide-and-conquer approaches encountered with iterative splits and local clusterings. The method also relies on waveforms $\vec{w}_i^{SVD}$, however, as opposed to the previous methods, graph-based clustering methods are applied to the full connectivity graph at once, in order to avoid border effects. The trick to make this work is that distances (and thus edges) of the connectivity graph between all waveforms are only computed for waveforms that are spatially close enough. Once the connectivity matrix is computed for all local distances, we can apply the clustering algorithm of our choice (Louvain [2], or even HDBSCAN for the distances). In the following, we used the Louvain algorithm.

Template matching

Once the templates $\vec{T}_i(t) \in \mathbb{R}^{N\times M}$ have been found and identified, as the centroids of the clusters, template matching attempts to describe the signal $\vec{s}(t)$ as a linear sum of the templates, plus noise. Here again, various methods have been tried, with small differences and subtleties.

KS-matching

This is a simple Matching Pursuit algorithm [29], and full details can be found in [32, 35]. The signal $\vec{s}(t)$ is convolved with all the $\vec{T}_i(t) \in \mathbb{R}^{N\times M}$. This convolution leads to the computation of all the scalar products $b_{ij} = \vec{T}_i(t_j) \cdot \vec{s}(t_j)$ at any time t_j. Note that to speed up these convolutions, one can use an SVD representation of the templates, and/or optimized libraries such as `torch`. Once the b_{ij} are computed, the algorithm iteratively takes the one with the largest value b_{i^*,j^*} (i.e. the best match), and subtracts $b_{i^*,j^*}T_{i^*}(t_{j^*})$ from the signal $\vec{s}(t)$, to be left with what is called a residual. This procedure is repeated until no more matches can be found, leading, in theory and if the dictionary is complete, to residuals that should only represent the noise present in the signal $\vec{s}(t)$. In practice, however, the dictionary of templates is not fully accurate, and a stopping criteria must be imposed. Here again, to speed up the algorithms, operations are performed in the space of the scalar products instead of the raw data. This requires precomputation of all pairwise scalar products of all pairs of templates $T_{i,j}(t)$ for all possible lags. Assuming that we have N_T templates $T_i(t) \in \mathbb{R}^{N\times M}$, this lookup table M has a size $T \times T \times (2M - 1)$. However, while this is not the case in `Kilosort`, such a lookup table can be sparsified in practice because not all templates interact with each other.

Circus-OMP

This is the Orthogonal Matching Pursuit algorithm [36], which is slightly different compared to the one implemented in `Kilosort`. The main difference being that each time a template is selected (and thus $b_{i*,j*}$ is chosen), this selection updates the values of $b_{i*,j*}$ that were selected so far. Intuitively, this means that adding a new template to the reconstruction updates the weights of the ones that were selected before. Such an algorithm has been shown to be more efficient when templates are non-orthogonal. Again, to speed up the implementation, some internal optimizations can be performed, via Cholesky decomposition [26].

Wobble

This is an augmented Matching Pursuit algorithm, as implemented in YASS [24]. The ideas are similar to the classical Matching Pursuit Algorithm of `Kilosort`, but it involves a superresolution step to enhance the alignment of the templates. In summary, the dictionary of templates $\vec{T}_i(t) \in \mathbb{R}^{N\times M}$ is "augmented" through multiple slightly time-delayed versions of the templates, to compensate for subsampling jittering. This produces a dictionary of $\vec{T}_i(t) \in \mathbb{R}^{N'\times M}$, and then uses an algorithm similar to the one of `Kilosort`.

TDC-peeler

This is a greedy Matching Pursuit algorithm that works only at peak times, as in [56]. In short, peaks are detected as thresholds that cross above k times the median absolute deviation (MAD) per channel. As in the peak detection step, a spatio-temporal exclusion radius ensures that only the most prominent peaks are kept. At these peaks times, we look for the best match of the templates w.r.t. to Euclidean distances. Once a match is found, it is subtracted from the raw traces, and peaks are redetected on the residuals traces until there are no further matches.

TDC-peeler drift aware

This is a special case of the algorithm used in Figure 6, in order to highlight the effect of motion correction while matching templates. In short, instead of interpolating the traces as is commonly done by most modern sorters [32], we extend the dictionary of templates by moving them from −100 μm to 100 μm along the y −axis via 1 μm spatial increments. Interpolation is performed with

bicubic splines, and once this augmented set of templates has been generated, we apply exactly the same matching algorithm as the classical Tridesclous, but now selecting the appropriate template at each time points, based on the inferred motion at this time point.

Note that to generate Figure 5, we used the same methodology as explained in [12] to assess how well the matching strategies were able to detect collisions. In particular, we used the extension of the ground-truth comparison class `CollisionGTComparison`, which computes performance metrics by spike lag. In addition to the agreement score computation and the matching, this method first detects and flags all "synchronous spike events" in the ground-truth spike trains. Two spikes from two separate units are considered a "synchronous spike event" if their spike times occur within a time delay of 2 ms. The synchronous events are then divided into 11 bins that span the [−2, 2] ms interval, and *collision recall* is computed for each bin. The similarities between templates are computed as the normalized l2 norm, with a number between 0 and 1 quantifying how similar two templates are. Collision recalls as function of the lags are plotted by grouping pairs of templates as a function of their similarities, with the assumption that the task of solving temporal collision is different if templates are dissimilar or not.

End-to-end spike sorter comparison on synthetic recordings

We include here a brief description of all the end-to-end spike sorters compared in Figure 7:

`SpyKING-CIRCUS 2` This is an updated version of `SpyKING-CIRCUS` [56] based on the modular components implemented in this article. In summary, this spike sorter uses (when motion is present) the DREDGE motion correction algorithm [49] before whitening the data. On this whitened data, the chain of components that are used are: matched filtering for peak detection, iterative splits for clustering (Iter-HDBSCAN), and orthogonal matching pursuit for template reconstruction (Circus-OMP). The results presented in this paper were created using version 25.12.

`TriDesClous 2` This is an updated version of `TriDesClous` based on the modular components implemented in this article. In summary, the code uses (when motion is present) the DREDGE motion correction algorithm [49] before filtering the data. On this filtered data, the chain of components that are used are: locally exclusive for peak detection, iterative splits for clustering (Iter-ISOPLIT), and fast greedy partial deconvolution, only applied at peak times for template reconstruction (TDC-peeler). The results presented in this paper were created using version 25.12.

`Lupin` This is a direct demonstration of the potential unlocked by the modular components implemented in this article. In summary, the code uses (when motion is present) the DREDGE motion correction algorithm [49] before filtering and whitening the data. On this whitened data, the chain of components that are used are: matched filtering for peak detection, iterative splits for clustering (Iter-ISOPLIT), and augmented matching pursuit for the spike deconvolution (Wobble). The results presented in this paper were created using version 25.12.

`Kilosort4` This is the complete standalone algorithm, as implemented in [32]. All the units found by `Kilosort` were kept for downstream analysis. We used version 4.1.1.

`Kilosort-like` This is an attempt to prove the validity of our modular pipelines to re-implement `Kilsort` as a chain of components. To be more precise, this spike sorter uses (when motion is present) the `Kilosort` correction algorithm [14, 32] before whitening the data. On this filtered data, the chains of components that are used are: matched filtering for peak detection, attempted port of the graphbased clustering from `Kilosort` (KS-clustering), and the exact same matching pursuit for template reconstruction from `Kilosort` (KS-matching). Note that while the performances are not exactly the same as the ones of `Kilosort`, because of implementation details, the results/behavior is nevertheless very similar, and offer the possibility to dissect the computational steps of the algorithm. In fact, a major main with `Kilosort` lies after the template matching step. In `Kilosort4`, all found spikes are then re-clustered, with the exact same graph-based clustering algorithm in the "denoised" feature space (where collisions have been resolved *via* template matching). This step has not yet been ported into our framework, and this explains part of the discrepancies. We used version 25.12

Evaluation

The exact nature of the evaluations that have been performed in every benchmark depends on the nature of the benchmark *per se*. In all of our benchmarks, we tried to identify as accurately as possible what are the key goals of every step (for example, capturing accurately all peaks for peak detection, finding all clusters for clustering, …) and design quality metrics accordingly. Quite often, this relies on a comparison between the ground truth labels of the spike trains and the one provided as output by the different algorithms (in clustering, matching, merging). This comparison is performed based on the agreement matrix and the so-called "matches". To get more details, one can read the `SpikeInterface` documentation, but roughly such matches allow us to compute the average accuracy per matched units. For some particular figures (such as clustering, matching, …), we also looked at the number of good, overmerged, and false positive units (see [6] for more details).

Knowing the ground-truth spiking activity, we can compute the *accuracy* of each ground-truth unit i as:

$$\mathrm{acc}_i = \frac{TP_{ij}}{TP_{ij} - N_i - N_j} \tag{2}$$

where j is the sorted unit matched to the Ground Truth (GT) unit i, N_i and N_j are the number of spikes in the GT and matched sorted unit, respectively, and TP_{ij} is the number of *true positive* spikes, i.e., the spikes found both in the GT and sorted spike trains. From this accuracy metric, we further classified spike sorted units as:

- *well detected* : units with an accuracy greater than or equal to 80%.
- *overmerged* : units with an agreement above 20% with more than one GT unit.
- *redundant* : units with an agreement above 20% with a GT unit that are not the best matching unit. These units can be oversplit or duplicated sorted units.
- *false positive*: sorted units with an agreement below 20%.

Hardware Specifications

All simulations and spike sorting jobs have been run on a Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz Machine with a Nvidia Quadro RTX 4000 GPU (used only by `Kilosort`).

Data availability

The current manuscript is a computational study, so no data have been acquired for this manuscript. All the figures available in this article can be regenerated from Jupyter notebooks that are available at *https://github.com/samuelgarcia/sorting_components_benchmark_paper*. All the methods and options evaluated in this work are readily available to the electrophysiology community within the `SpikeInterface` package *https://github.com/SpikeInterface/spikeinterface* and can be immediately deployed with a few lines of code. The implementations of the Kilosort clustering and matching are available at *https://github.com/SpikeInterface/spikeinterface-kilosort-components*.

Acknowledgements

This work has been funded through several grants. We would like to thanks Joe Ziminski for his helpful feedbacks and comments on the manuscript. PY is supported by the INRIA PIQ ID20240902, the ANR GNEURO ANR-25-CE42-6535-03 and the Cross Disciplinary Project LOOP of the Lille University. ZM is supported by NIH grants F31129103, T32GM007753, T32GM144273. CH is supported by the UKRI Biotechnology and Biological Sciences Research Council (BBSRC) grant number BB/X01861X/1. PAF, HRM, and BD received support by NIH grant U19NS123716-02. CF is supporterd by NSF Neuronex Award DBI-1707398 and Simons Foundation grant 344 543023. We further thank the Simons Foundation for supporting the SpikeInterface project.

Additional information

Funding

Funder	Grant reference number	Author
Institut national de recherche en informatique et en automatique (INRIA)	ID20240902	Pierre Yger
Agence Nationale de la Recherche (ANR)	ANR-25-CE42-6535-03	Pierre Yger
HHS | National Institutes of Health (NIH)	F31129103	Zachary M McKenzie
HHS | National Institutes of Health (NIH)	T32GM007753	Zachary M McKenzie
HHS | National Institutes of Health (NIH)	T32GM144273	Zachary M McKenzie
UKRI | Biotechnology and Biological Sciences Research Council (AFRC)	BB/X01861X/1	Chris Halcrow
National Science Foundation (NSF)	DBI-1707398	Charlie Windolf
Simons Foundation (SF)	344 543023	Charlie Windolf
HHS | National Institutes of Health (NIH)	U19NS123716-02	Benjamin K Dichter Paul Adkisson-Floro Heberto Ramon Mayorquin

Author ORCID iDs

Samuel Garcia: https://orcid.org/0000-0001-6389-9779
Chris Halcrow: https://orcid.org/0000-0002-0246-8075
Herberto Ramon Mayorquin: https://orcid.org/0000-0002-5937-7537
Alessio P Buccino: https://orcid.org/0000-0003-3661-527X
Pierre Yger: https://orcid.org/0000-0003-1376-5240

References

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

[2] **Blondel V. D.,** Guillaume J.-L., Lambiotte R., Lefebvre E. (2008) Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* **2008**:P10008

[3] **Boussard J.,** Windolf C., Hurwitz C., Lee H. D., Yu H., Winter O., Paninski L. (2023) DARTsort: A modular drift tracking spike sorter for high-density multi-electrode probes. *bioRxiv* https://doi.org/10.1101/2023.08.11.553023

[4] **Buccino A. P.,** Einevoll G. T. (2020) Mearec: a fast and customizable testbench simulator for ground-truth extracellular spiking activity. *Neuroinformatics* **19**:185-204

[5] **Buccino P.,** Garcia S., Yger P. (2022) Spike sorting: new trends and challenges of the era of high-density probes. *Progress in Biomedical Engineering* **4**:022005

[6] **Buccino P.,** Hurwitz C. L., Garcia S., Magland J., Siegle J. H., Hurwitz R., Hennig M. H. (2020) Spikeinterface, a unified framework for spike sorting. *eLife* **9**:e61834 https://doi.org/10.7554/eLife.61834

[7] **Buccino P.,** Sridhar A., Feng D., Svoboda K., Siegle J. H. (2025) Efficient and reproducible pipelines for spike sorting large-scale electrophysiology data. *bioRxiv* https://doi.org/10.1101/2025.11.12.687966

[8] **Chaure J.,** Rey H. G., Quiroga R. Quian (2018) A novel and fully automatic spike-sorting implementation with variable number of features. *Journal of neurophysiology* **120**:1859-1871

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

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

[11] **Eom I. Y. Park,** Kim S., Jang H., Park S., Huh Y., Hwang D. (2021) Deep-learned spike representations and sorting via an ensemble of auto-encoders. *Neural Networks* **134**:131-142

[12] **Garcia S.,** Buccino A. P., Yger P. (2022) How do spike collisions affect spike sorting performance?. *Eneuro* **9**

[13] **Garcia S.,** Sprenger J., Holtzman T., Buccino A. P. (2022) ProbeInterface: A Unified Framework for Probe Handling in Extracellular Electrophysiology. *Frontiers in Neuroinformatics* **16**:823056

[14] **Garcia S.,** Windolf C., Boussard J., Dichter B., Buccino A. P., Yger P. (2024) A Modular Imple-mentation to Handle and Benchmark Drift Correction for High-Density Extracellular Recordings. *eNeuro* **11**:ENEURO.0229–23.2023

[15] **Han Y.,** Wang S. (2024) E-Sort: Empowering End-to-end Neural Network for Multi-channel Spike Sorting with Transfer Learning and Fast Post-processing. *arXiv* https://doi.org/10.48550/arXiv.2409.13067

[16] **He Y.,** Marin-Llobet A., Sheng H., Liu R., Liu J. (2024) End-to-end multimodal deep learning for real-time decoding of months-long neural activity from the same cells. *bioRxiv* https://doi.org/10.1101/2024.10.14.618046

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

[18] **Hua Y. Liu,** Luo J., Li S., Jiang L., Wu P., Sun S., Shang L., Lu C., Zhang K., Liu J., *et al.* (2025) Microelectrode arrays cultured with in vitro neural networks for motion control tasks: encoding and decoding progress and advances. *Neuron* **11**:233

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

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

[21] **Keshtkaran M. R.,** Yang Z. (2017) Noise-robust unsupervised spike sorting based on discriminative subspace learning with outlier handling. *Journal of neural engineering* **14**:036003

[22] **Laboy-Juárez K.J.,** Ahn S., Feldman D. E. (2019) A normalized template matching method for improving spike detection in extracellular voltage recordings. *Scientific reports* **9**:12087

[23] **Laquitaine S.,** Imbeni M., Tharayil J., Isbister J. B., Reimann M. W. (2024) Spike sorting biases and information loss in a detailed cortical model. *bioRxiv* https://doi.org/10.1101/2024.12.04.626805

[24] **Lee J.,** Mitelut C., Shokri H., Kinsella I., Dethe N., Wu S., Li K., Reyes E. B., Turcu D., Batty E., *et al.* (2020) Yass: Yet another spike sorter applied to large-scale multi-electrode array recordings in primate retina. *bioRxiv*

[25] **Lefebvre P. Yger,** Marre O. (2016) Recent progress in multi-electrode spike sorting methods. *Journal of Physiology-Paris* **110**:327-335

[26] **Lubonja A.,** Præsius S. K., Tran T. D. (2024) Efficient Batched CPU/GPU Implementation of Orthogonal Matching Pursuit for Python. *arXiv* https://doi.org/10.48550/arXiv.2407.06434

[27] **Magland J.,** Jun J. J., Lovero E., Morley A. J., Hurwitz C. L., Buccino A. P., Garcia S., Barnett A. H. (2020) Spikeforest, reproducible web-facing ground-truth validation of automated neural spike sorters. *eLife* **9**:e55167 https://doi.org/10.7554/eLife.55167

[28] **Magland J. F.,** Barnett A. H. (2016) Unimodal clustering using isotonic regression: ISO-SPLIT. *arXiv* https://doi.org/10.48550/arXiv.1508.04841

[29] **Mallat S.,** Zhang Z. (1993) Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing* **41**:3397-3415

[30] **McInnes L.,** Healy J., Astels S. (2017) hdbscan: Hierarchical density based clustering. *Journal of Open Source Software* **2**:205

[31] **Merow B. Boyle,** Enquist B. J., Feng X., Kass J. M., Maitner B. S., McGill B., Owens H., Park S., Paz A., et al. (2023) Better incentives are needed to reward academic software development. *Nature Ecology & Evolution* **7**:626-627

[32] **Pachitariu M.,** Sridhar S., Pennington J., Stringer C. (2024) Spike sorting with Kilosort4. *Nature Methods* **21**:914-921

[33] **Pachitariu M.,** Sridhar S., Stringer C. (2023) Solving the spike sorting problem with kilosort. *bioRxiv* https://doi.org/10.1101/2023.01.07.523036

[34] **Pachitariu M.,** Steinmetz N. A., Colonell J. (2019) Kilosort2. https://github.com/

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

[36] **Pati Y.,** Rezaiifar R., Krishnaprasad P. (1993) Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In: Proceedings of 27th Asilomar Conference on Signals, Systems and Computers. pp. 40-44

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

[38] **Saggese G.,** Tambaro M., Vallicelli E. A., Strollo A. G., Vassanelli S., Baschirotto A., Matteis M. D. (2021) Comparison of sneo-based neural spike detection algorithms for implantable multitransistor array biosensors. *Electronics* **10**:410

[39] **Schröter M.,** Cardes F., Bui C.-V. H., Dodi L. D., Gänswein T., Bartram J., Sadiraj L., Hornauer P., Kumar S., Pascual-Garcia M., *et al.* (2025) Advances in large-scale electrophysiology with high-density microelectrode arrays. *Lab on a Chip* **25**:4844-4885

[40] **Shabestari P. S.,** Buccino A. P., Kumar S. S., Pedrocchi A., Hierlemann A. (2021) A modulated template-matching approach to improve spike sorting of bursting neurons. In: 2021 IEEE Biomedical Circuits and Systems Conference (BioCAS). IEEE. pp. 1-4

[41] **Souza C.,** Santos V. Lopes-dos, Bacelo J., Tort A. B. (2019) Spike sorting with gaussian mixture models. *Scientific reports* **9**:3627

[42] **Steinmetz N.** (2021) "Imposed motion datasets" from Steinmetz et al. Science 2021. science 2021. https://doi.org/10.6084/m9.figshare.14024495

[43] **Steinmetz N. A.,** Aydin C., Lebedeva A., Okun M., Pachitariu M., Bauza M., Beau M., Bhagat J., Broux M., et al. (2021) Neuropixels 2.0: A miniaturized high-density probe for stable, longterm brain recordings. *Science* **372**

[44] **Steinmetz P. N.** (2024) Simulation of background neuronal activity and noise in human intracranial microwire recordings. *Journal of Neuroscience Methods* **402**:110017

[45] **Toosi R.,** Akhaee M. A., Dehaqani M.-R. A. (2021) An automatic spike sorting algorithm based on adaptive spike detection and a mixture of skew-t distributions. *Scientific reports* **11**:13925

[46] **Turin G.** (1960) An introduction to matched filters. *IRE Transactions on Information Theory* **6**:311-329

[47] **Varol J. Boussard,** Dethe N., Winter O., Urai A., Laboratory T. I. B., Churchland A., Steinmetz N., Paninski L. (2021) Decentralized motion inference and registration of neuropixel data. In: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1085-1089

[48] **Watters N.,** Buccino A., Jazayeri M. (2024) MEDiCINe: Motion correction for neural electrophysiology recordings. *bioRxiv* https://doi.org/10.1101/2024.11.06.622160

[49] **Windolf C.,** Yu H., Paulk A. C., Meszéna D., Muñoz W., Boussard J., Hardstone R., Caprara I., Jamali M., Kfir Y., *et al.* (2025) DREDge: robust motion correction for high-density extracellular recordings across species. *Nature Methods* **22**:788-800

[50] **Wolff R.,** Polito A., Buccino A. P., Chiappalone M., Tucci V. (2025) Protocol for the enhanced analysis of electrophysiological data from high-density multi-electrode arrays with nicespike and spikeNburst. *STAR Protocols* **6**:104195

[51] **Wouters J.,** Kloosterman F., Bertrand A. (2018) Towards online spike sorting for high-density neural probes using discriminative template matching with suppression of interfering spikes. *Journal of neural engineering* **15**:056005

[52] **Wouters J.,** Kloosterman F., Bertrand A. (2019) A data-driven regularization approach for template matching in spike sorting with high-density neural probes. In: 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). pp. 4376-4379

[53] **Wouters J.,** Kloosterman F., Bertrand A. (2021) A data-driven spike sorting feature map for resolving spike overlap in the feature space. *Journal of Neural Engineering* **18**:0460a7

[54] **Wyngaard A. J. G.,** Llobet V., Barbour B. (2025) Lussac: a fully-automated consensus method that increases the yield and quality of spike-sorting analyses. *bioRxiv* https://doi.org/10.1101/2022.02.08.479192

[55] **Ye Z.,** Shelton A. M., Shaker J. R., Boussard J., Colonell J., Birman D., Manavi S., Chen S., Hurwitz C., Yu H., *et al.* (2025) Ultra-high-density Neuropixels probes improve detection and identification in neuronal recordings. *Neuron* **113**:3966-3982.e12

[56] **Yger P.,** Spampinato G. L., Esposito E., Lefebvre B., Deny S., Gardella C., Stimberg M., Jetter F., Picaud S., et al. (2018) A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo. *eLife* **7**:e34518 https://doi.org/10.7554/eLife.34518

[57] **Zhang Y.,** Han J., Liu T., Yang Z., Chen W., Zhang S. (2022) A robust spike sorting method based on the joint optimization of linear discrimination analysis and density peaks. *Scientific reports* **12**:15504

Peer reviews

Reviewer #1 (Public review):

Summary:

This work presents a flexible spike-sorting framework that allows users to run, swap, and benchmark individual modules commonly used in spike sorting. The paper argues and demonstrates that "opening the black box" is essential for understanding which components drive performance differences and for making progress toward more accurate and transparent spike sorting.

Using this modular benchmarking pipeline, the work identifies electrode drift as a primary bottleneck for accurate sorting and introduces an end-to-end sorter ("Lupin") that combines the best-performing modules and is reported to outperform existing spike-sorting packages on their benchmark.

Overall, this is a strong tool/resource contribution with clear potential to accelerate spike-sorting development and enable more rigorous comparisons. However, several claims, particularly around Lupin's or individual modules' superiority, are not yet supported robustly enough for the strength of the conclusions stated.

Strengths:

This work has high community value and practical utility. The effort to make benchmarking and spike sorting modules accessible and standardized is substantial and likely to be broadly useful.

Treating spike sorting as a set of interchangeable modules is a useful approach to some extent, and it enables targeted improvements rather than 'new sorters' popping up, which are difficult to fully understand.

Implementing this resource within SpikeInterface, an already widely used tool, will facilitate uptake and community contributions.

Overall, I am positive about this manuscript as a resource paper. The core framework is compelling and timely.

Weaknesses:

(1) The main concern is the limited support for the claim that 'Lupin' and individual modules' outperform existing spike sorters.

(2) Evidence is primarily from a single benchmark based on an intentionally simplified simulation. While the authors discuss the trade-offs between simulated and real data, the current evaluation does not provide enough diversity to justify claims of superiority.

(3) While improving individual modules that run in a serial fashion could aid overall spike sorting performance, acknowledging that some end-to-end sorters work in an iterative fashion across multiple of these modules would be fair. Perhaps the optimal spike sorter is not a serial set of modules.

(4) There is also a risk of benchmark overfitting. A modular approach makes it easy to select components that excel on specific benchmarks (or a specific project's data characteristics) without generalizing.

Concrete ways to strengthen this work:

(1) Evaluate on multiple simulation regimes, consider adding at least one biophysically detailed simulation, benchmark on multiple probe-geometries with neurons also clustered in different depth profiles (as this will affect drift solutions), and provide real-data validation. Even without full ground truth, real-data can be evaluated with expert curation, functional validation (e.g., refractory violations, quality metrics, unit waveform consistency), agreement across sorters, and consistency across time.

(2) Related to real-data applicability, it is also important to acknowledge that modulatory approaches can enable overfitting to the needs of individual projects. Without real-data benchmarking (or benchmark diversity), it is unclear how the framework will guide users towards generalizable 'best practices' rather than optimized configurations that work for their specific conditions.

https://doi.org/10.7554/eLife.110588.1.sa2

Reviewer #2 (Public review):

Summary:

Spike sorting, that is, assigning events detected in extracellular electrophysiology data to the firing of individual neurons, is an inherently difficult computational problem involving multiple steps. The difficulty arises from low signal-to-noise, instability in signal due to the

relative motion of the tissue and recording sites, and large volumes of data. Experimental ground truth data - where the correct assignment of spikes is known - is not available in large enough quantities to test algorithms. This paper describes a tool for creating fully synthetic ground truth data and benchmarking the individual steps of spike sorting to dissect the impact of signal-to-noise, firing rate, and motion correction on each step. This information is used to construct an optimized algorithm for sorting the ground truth data. One result of particular interest is the dominant role of motion correction in degrading accuracy. Another important technical result is that motion correction via interpolation of the voltage traces yields similar accuracy to interpolation of the spike templates.

Strengths:

The paper clearly shows the benefits of analyzing the complex process of spike sorting step by step. While this analysis has also been done in papers presenting spike sorters (for example, reference [32]), the tools presented here allow users and developers to do similar studies for their own work. This toolset will be very useful to many labs, especially those working in less studied brain areas or model systems, cases where the tuning of standard spike sorting tools is not a good match to the data.

Weaknesses:

The model ground truth data used in the paper does not need to be a perfect match to experimental data to provide useful benchmarking. However, as with all measurements of spike sorting accuracy, extrapolation to experimental data can be complicated. Users of these tools will need to assess how well the simulated data matches their recordings.

https://doi.org/10.7554/eLife.110588.1.sa1

Reviewer #3 (Public review):

Overview:

In this manuscript, the authors describe two additions to an existing toolbox (SpikeInterface, Buccino et al., 2020, eLife). The first addition is an empirical simulator for extracellular recordings, in which spikes from predefined templates are added up with Gaussian noise. The second addition involves granting user-level access to intermediate processing steps along spike sorting algorithms. The authors demonstrate the toolbox by evaluating functions (e.g., event detection) or sets of functions (e.g., feature extraction + clustering) on their simulated data, and suggest that a specific combination of function implementations provides performance improvement relative to kilosort4 (Pachitariu et al., 2024, Nature Methods).

If the authors are interested in making this manuscript a suitable scientific contribution, the entire work has to be revised extensively. In particular, the simulator has to be extended and improved; the implementation of existing spike sorters has to be improved; the feedforward architecture of the modules has to be extended; the reporting of results has to follow standard reporting standards; new algorithms have to be explained in sufficient detail; and the manuscript has to undergo extensive proofreading.

Notably, even assuming perfect implementation and descriptions, it is unclear to me whether the scope of the present work warrants a publication in a scientific journal, or is more suitable for an internal technical report or an e.g., a GitHub version release. To go beyond a scientifically-sound technical report, the authors may choose to demonstrate the utility of their new proposed sorter ("Lupin") and compare it to existing tools on multiple datasets.

General comments:

(1) The simulator itself has to be improved and extended. Right now, it simply generates, for every unit, a mother waveform from a sum of exponentials, scales that over channels, and then adds up multiple instantiations of every unit on every channel, along with noise. This is not a biophysical simulator: it is an ad hoc procedure, and the sentence "we firmly believe that.." (lines 482-483) does not make the procedure convincing. To make the simulator credible, the authors should: (1) use a set of biophysical equations, with multi-compartmental modeling of currents and return currents; (2) use noised data from extracellular recordings; or (3) some combination thereof.

(2) The simulated dataset has to be extended in time. Maybe I missed something, but 500 units over 10 minutes, with some units having firing rates as low as 0.1 spikes/s, corresponds to some of the units firing an expected 60 spikes. This is clearly too short, and does not replicate the standard situation in extracellular experiments.

(3) The simulated dataset has to be extended in space. The choice of using NeuroPixels 1.0 geometry is a poor one. Many labs use other monolithic electrode arrays (MEAs, silicon probes, other rigid arrays); tetrodes remain a major tool, and flexible probes (polyimide, mesh) are evolving. Assessing algorithms over a single spatial architecture is likely to lead to local maxima in performance and potentially erroneous conclusions.

(4) The existing spike sorters evaluated are not completely described. Some sorters (e.g., SpyKING Circus and KS4) were described in previous publications, but it is unclear whether the implementation that was used for the present tests is exactly the same as those previously published. More importantly, some of the sorters evaluated (e.g., TDC, TDC2, SpyKING Circus 2) were never described in a peer-reviewed paper. This does not mean that they cannot be evaluated - but if they are, they must be described in full. Relying on the fact that the code is open source cannot replace a complete and accurate scientific description.

(5) Related to the above, all relevant code should be made available online in permanent repositories, not only in author-controlled ones.

(6) It is unclear why SpyKING Circus 2 and TDC2 are evaluated - these could potentially be described as straw men. I recommend reorganizing the manuscript so that after every module is evaluated separately based on a limited ground truth dataset, a single "best" sorter would be constructed, and then tested extensively (and compared to the de facto state of the art). Such reorganization would both demonstrate the utility of a modular approach and clarify the general usefulness of the outcome.

(7) The new algorithms developed, for example, clustering and template matching, have to be described in more detail, and demonstrated graphically on simple datasets. This can be done in supplementary material if the authors prefer not to extend the manuscript too much.

(8) This reviewer finds the description and interpretation of the results to be inadequate. As an example, focusing on Figure 5: The results in Figure 5A have to be supplemented and summarized as a scalar point estimate (e.g., median accuracy), an estimate of dispersion (e.g., using MAD, IQR, or SD), evaluated over multiple runs, and compared using statistical tests between tools and conditions (e.g., using a multi-dimensional analysis of variance, a mixed effect model, etc.). The results in Figure 5D must have an indication of dispersion. Any conclusions based on the numerical experiments must be based on these metrics and statistical evaluations.

(9) The entire MS would benefit from expert proofreading; there are many language errors, mostly in indefinite articles and grammatical numbers.

https://doi.org/10.7554/eLife.110588.1.sa0