Reviewed Preprint
v1 • April 1, 2026
Not revised

Reviewed Preprint
v2 • September 3, 2026
Revised by authors

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

Competing interests: No competing interests declared

Funding: See page 27

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], Heberto 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 presents a promising framework for automated spike sorting in high-density extracellular electrophysiology. The core evidence is **solid**, but could be strengthened with extended simulations, broader benchmarking against existing sorters, and validation via expert curation. It will be of interest to the systems neuroscience community.

https://doi.org/10.7554/eLife.110588.2.sa4

Abstract

Spike sorting is an algorithmic process that extracts 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 (i.e. peak detection, feature extraction and clustering, and template matching). We then leverage these results to create a modular, component-based spike sorter that can outperform Kilosort 4 on dense and large simulated recordings and produce similar quantitative results on real data. 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 component-based spike sorting framework presented here has the potential to foster community engagement in the field by lowering the barrier to contributions and providing 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 [1, 2]. With the development and widespread adoption of high-density micro-electrode arrays both for *in-vivo* [3, 4, 5] and *in-vitro* applications [6, 7, 8], 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 [9, 10, 11, 12, 13, 14, 15, 16, 17, 18].

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 corrected for probe motion), then a selection of putative spikes are detected. The features of 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 redetect spikes using a deconvolution method. Different spike sorters use different methods for each sorting component (see Figure 1 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. Although there has been continuous algorithmic development in spike sorting with new ideas and concepts used to tackle motion correction [19, 20, 10], peak detection [21, 22, 23], feature extraction and clustering [24, 25, 26, 27, 28], and template matching [29, 30, 31], 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 difficult to benchmark new algorithmic ideas. Although notable efforts towards end-to-end spike sorting comparisons have been proposed [32], 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., by using 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 [33] and move on to other projects. In an era where hardware and 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 2021, will reach end-of-life in October 2026), making software developed in older Python versions virtually unmaintainable. 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, software maintenance problems 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[While many of these sorters are integrated in the SpikeInterface framework, their original GitHub repositories have had no commits over the last two years]: Klusta [9], JRClust/IronClust [13], SpyKING-CIRCUS [15], HDSort [14], WaveClus [16], YASS [17].

In this paper, we propose an alternative framework to provide a modular and community-based paradigm for spike sorting development. Building on the SpikeInterface framework [34], 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.

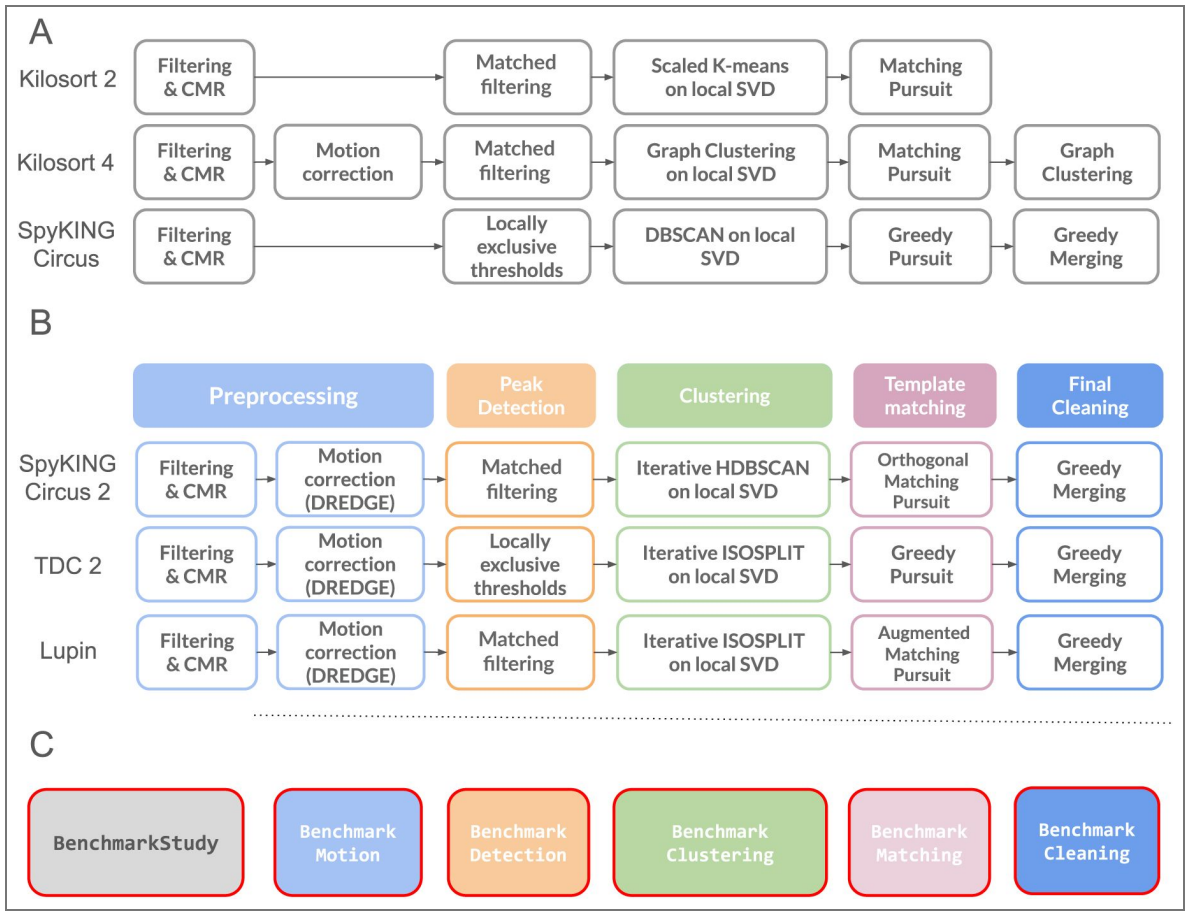

Figure 1.

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 [18, 10], alongside SpyKING-CIRCUS [15]. **B** For each of these key algorithmic steps, we have 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 drift simulation [19]. Next, we showcase the framework with a detailed benchmark of the currently available methods for peak detection, feature extraction plus clustering, and template matching. We then go into detail on a 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 [18], 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. The first two originally motivated the development of such a modular architecture: SpyKING-CIRCUS 2, an enhanced version of SpyKING-CIRCUS [15], and TriDesClous 2. The third, Lupin, is a new component-based sorter created as the optimal combination of the best-performing components, based on our generated data and benchmarks. In our simulated benchmarks, and on real-world datasets, we show that Lupin is competitive with Kilosort4 both in performance and speed, highlighting the 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 regarding how a typical algorithmic pipeline should look. 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] and whitening), peak detection, feature extraction, then clustering to obtain a dictionary of templates, followed by 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 beyond the scope of this current paper (see Methods of the aforementioned spike sorters), all modern spike sorters, including `Kilosort` [10, 18], `SpyKING-CIRCUS` [15], `MountainSort` [12], `YASS` [17] and `TriDesClous` follow such a pipeline. However, these pipelines are always evaluated as "black boxes" [32] so 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. We call each step in a spike sorter a component, and one can now implement a full spike sorter as a series of components, adjusting each one for fine granularity over their pipeline. As suggested in Figure 1B, we port three new spike sorters to this conceptual architecture: SpyKING-CIRCUS 2 and TriDesClous 2 are direct evolutions of previous algorithms, while Lupin is a new spike sorting algorithm (see Methods), created to demonstrate the power of such modularity. Although the three pipelines share similar processing pipelines, sometimes even using similar algorithms, they differ by the particular parameters of the algorithms used, and thus each have different pros and cons, as will be shown in this paper.

Crucially, each key algorithmic step is paired with a higher-order object termed a BenchmarkStudy, see Figure 1C, enabling performance benchmarking at the component level for each algorithm. The exact nature of each benchmark depends on the component, e.g., how many peaks are detected for a detection method or how many clusters can be found for a clustering method. The benchmarks allow us to identify strengths and weaknesses of each step in spike sorting pipelines.

The sortingcomponents module uses a built-in engine to parallelize the computational load, which splits the recordings into temporal chunks. 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. This new object, integrating the built-in comparison features of SpikeInterface, can reduce 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 modular 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 pipelines, one needs to be able to generate artificial ground truth data in a fast and efficient manner. We have designed and implemented a new way to generate artificial ground truth recordings within SpikeInterface. As shown in Figure 2A, ground truth recordings with any number of neurons can be generated on-the-fly once a probe layout is provided. Each neuron can have a unique spatio-temporal 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.

To challenge modern spike sorters and make our ground truth recordings more realistic, we allowed for the inclusion of possible motion (drift) of the probe. As noted in recent papers [1], this is currently one of the major bottlenecks in spike sorting, but the algorithmic methods designed to solve the problem are not entirely satisfactory [19]. In our generator, neurons can drift with precomputed and imposed motion vectors, allowing for non-uniform motion that 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 positions of the somas (see Figure 2B). If templates are taken from external libraries, the templates are interpolated as functions of the drift. In our generator, we model noise as a multivariate normal distribution, but other options could be implemented (1/f and/or multi-unit activity such as in [18], many background units [35], or accurate noise spectrum [36]).

In the rest of the paper, we will use 30 minute long recordings with a Neuropixels 1.0-like layout (384 channels), sampled at 30 kHz, with a trimodal distribution profile for the positions of the cells (see Figure 2C). In Figure 2D, we show some macroscopic statistics from an example 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 spike sorting, we combined the ground truth generator with the BenchmarkStudy objects to assess where mistakes in spike sorting algorithms originate.

Peak detection

The first step we assess with the components framework is peak detection. In peak detection we aim to identify all putative spiking events in a recording. Here, we compare the two most commonly used peak detection methods from recent literature: “locally exclusive” and “matched filtering” (see Methods). 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 neurons are likely to be seen on many channels simultaneously). The second method, introduced in [4, 18], relies on the concept of matched filtering [37]. This method can boost the signal-to-noise ratio of the extracellular signal, assuming we approximately know the spatio-temporal shapes of the motifs we are looking for.

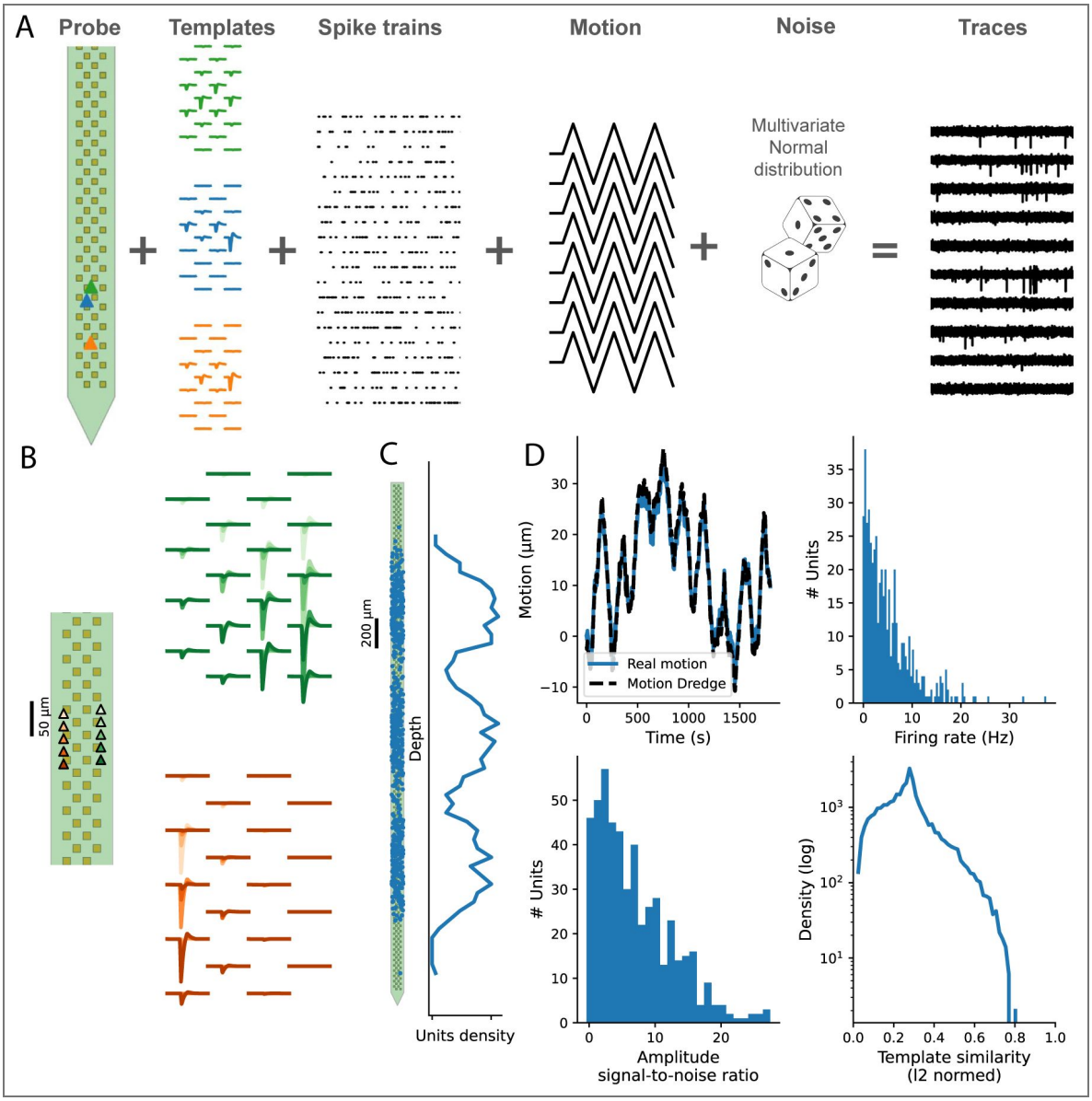

Figure 2. Ground truth recording generation.

A Given a probe layout, templates, activity patterns and inhomogeneous motion vectors, we can generate, on-demand and lazily, extracellular traces to extensively benchmark spike sorting. **B** Left: Positions of two neurons (triangles) during drift, color-coded as a function of position. Right: During drift extra-cellular templates will be dynamically generated as functions of positions of the sources. Each line display the extracellular voltage on a particular spatial channel, colored to match the positions on the left plot. **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.

To examine the difference between these two peak detection algorithms, we generated one static ground truth recording (see Methods) and assessed how many ground truth peaks each method detected. Figure 3 A displays the distribution of ground truth peaks as a function of peak amplitude, and the distribution of peaks that each method detected. We see that the locally exclusive method[used with a detection threshold of 5 median absolute deviations, which is "standard" in spike sorting [38]], finds very few peaks below the detection threshold; whereas the matched filtering method can recover some peaks below threshold. The locally exclusive method seems to be better at finding very large (< −200μV) peaks. This could be partially explained by 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 these events since it simply looks at threshold crossings.

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 functions of the signal-to-noise ratio (SNR) of the units with a rolling average overlaid. The plots show that for low SNR, recall and precision are higher for the matched filtering method. Even with matched filtering, the recall starts to decrease for SNRs lower than 20. This means that even for quite large peaks relative to the noise, some are missed because of collisions and/or noise. This observation is a strong argument for additional steps, such as template matching, to recover these putative spikes. Note that matched filtering should be most beneficial when the electrode density is high, as it uses the spatial redundancy of spike detectability. In Figure 3 D we plot the run times of the two methods. We see that, although matched filtered has performed better overall, especially with small peaks, this performance comes with some computational cost.

In modern spike sorters, the peaks computed in this step are not used directly as the final spike times of neurons. Instead, they are used to build the clusters and templates of each unit. The final spike times are computed during the later template matching step, which has been proven to outperform other methods [39]. Hence, it is not clear if missing spikes during peak detection has a large impact on overall spike sorting.

Feature extraction and clustering

The next component we evaluated is clustering. In this step, each algorithm receives all spiking events as input and clusters them into groups representing putative units. It is a crucial, if not the most crucial, step in spike sorting to disambiguate individual neurons. Before clustering, we must extract features from each event. Because most spike sorters currently rely on Singular Value Decomposition (see [2] for a review) for feature extraction, e.g. [40, 15, 17, 41], we decided to only focus on benchmarking the clustering step. Therefore, in the following, we will consider the feature extraction and the clustering as a whole rather than individually.

In order to quantify the effects of motion correction on each clustering algorithm, we applied our analysis to two conceptually similar datasets: one termed "static" and one termed "motion-corrected". Both datasets have the same initial cell input. For the motion-corrected recording, we imposed rigid motion on this input, then used DREDge to estimate the motion and finally interpolated the recording using kriging (see Methods and [19, 20]). Hence, if the motion was perfectly computed and compensated for, the static and motion-corrected recordings would be equivalent.

In Figure 4A (top), we compare four clustering algorithms that we have extracted and re-implemented from various spike sorting tools. All these algorithms are given the same inputs (i.e. the exact peaks times from the ground-truth recordings) and are 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, the ground truth spike times, we expect to get an upper bound of their performances.

Figure 4A compares the performance of several algorithms (see Methods for a description of their implementations). All methods perform well on static data. Iter-HDBSCAN (used in SpyKING-CIRCUS 2 [15]) and Iter-ISOSPLIT (used in TriDesClous 2 and Lupin) are both based on iterative

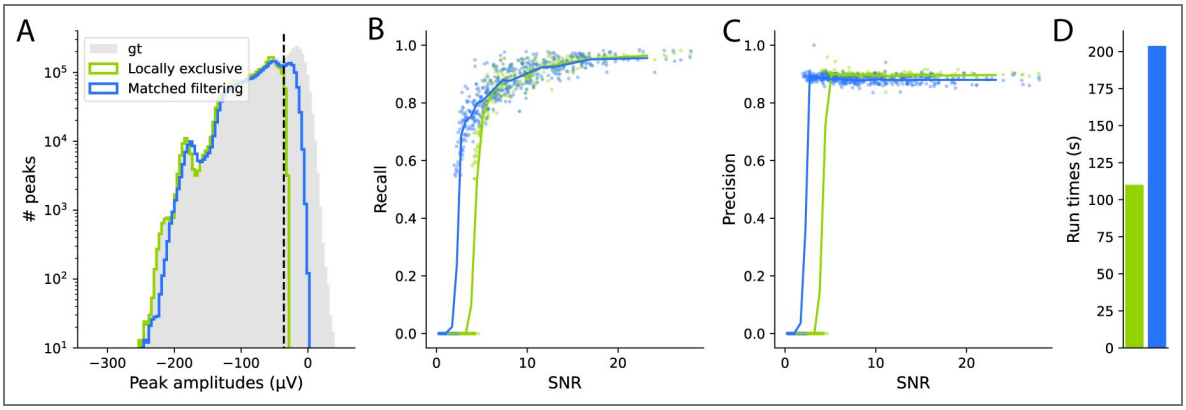

Figure 3. **Peak detection benchmark.**

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

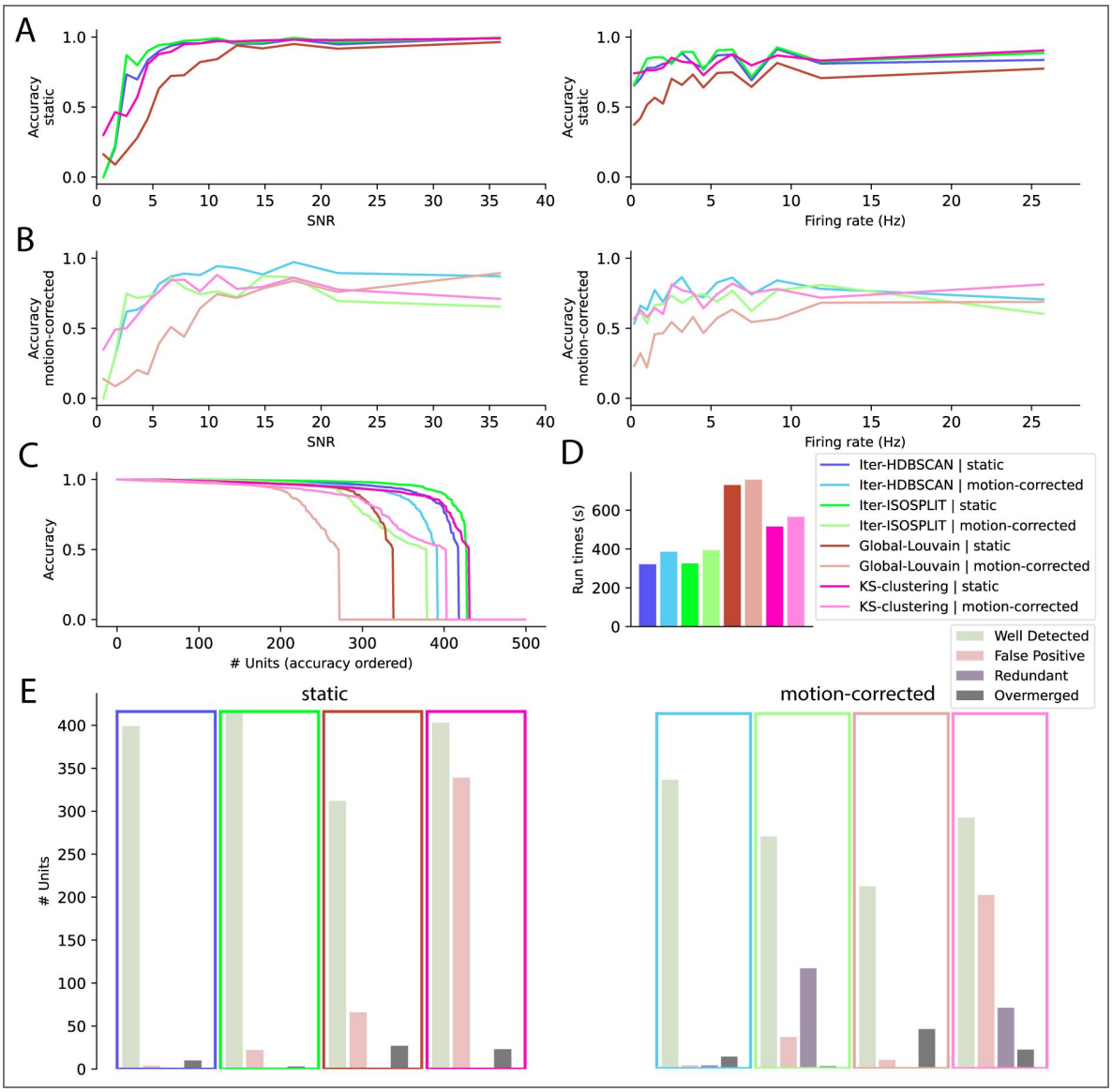

Figure 4. Feature extraction and clustering benchmark.

A Accuracy recall for all the clustering methods when applied on true peak times from a static recording. Left: as a function of the signal-to-noise ratios of the neurons. Right: as a 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).

splits of local density-based clustering performed by grouping the spikes per electrode, and give comparable results. KS-clustering is a re-implementation of the local graph clustering found in [18]. Global-Louvain is an attempt to design a single unified graph-based clustering on sparse connectivity matrices.

Interestingly we found, in Figure 4C, the number of units detected by the sorters is already far from perfect on static recordings (at best, 420 out of 500 units). We observe that clustering fails mainly 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 algorithms drastically degrades for motion-corrected data. As seen in Figure 4B and C the clustering algorithms struggle to properly find the units, in contrast to the static case, suggesting that the motion-correction algorithms do not completely compensate for the probe motion. We can see a dramatic drop in performance, specifically in the number of units outside of the well-detected units category (see Figure 4E), for methods that rely on local clustering (such as KS-clustering). Here again, the influence of the 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 as they rely heavily on parallelization over CPU-cores.

Template matching

We next assessed how effective the various template matching strategies used by different spike sorters are. In template matching we try to describe the incoming trace as a linear combination of spike waveforms, generated from a catalog of templates, plus noise. The temporal position of each waveform reveals the spike times while the template used from the catalog tells us the cluster index. In this way template matching simultaneously detects and clusters spikes. We compared the main algorithms available in the field, providing the ground truth template "catalog" as input. Although sharing similar principles, the exact mathematical nature of these algorithms is different. In the following, we compared the classical matching pursuit algorithm implemented in `Kilosort` (KS-matching [10]), a refined Orthogonal Template matching (Circus-OMP [42]), an augmented matching pursuit algorithm with temporal super-resolution (Wobble [17]), and a simpler greedy pursuit (TDC-peeler) (see Methods for more details of all these algorithms).

As 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). Methods based on full convolution of the traces with the catalog of templates (KS-matching, Circus-OMP and Wobble) have a better accuracy for small units compared to the greedy pursuit algorithm (TDC-peeler). However, once again, the performance for all algorithms is severely impacted for motion-corrected recordings (panel B; see Methods), since the interpolation of the traces considerably blurs the signals. Although there are some differences with respect to run times (Figure 5D), the user's choice of template matching engine depends on some other factors (e.g. correlation levels, firing rates, etc.) that could be tested to make a well-informed choice depending on the input data. For example, Figure 5E shows the recall over a distribution of lags between spikes. Here, the Wobble algorithm has the best performance for spike collisions, i.e, spikes that overlap in space and time (the results are displayed for static recordings, but similar results are obtained for motion-corrected ones). Assuming that fine correlations are important to understand how information is finely encoded by a neuronal population, one might favor Augmented Matching Pursuit algorithms such as Wobble, even if they are slightly slower than the others.

Motion interpolation reduces spike sorting performance

As shown in Figure 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 by the DREDge algorithm [20]. This spike sorting degradation occurs even when the ground truth motion is given. [19]. Instead, the performance drop is mainly due to

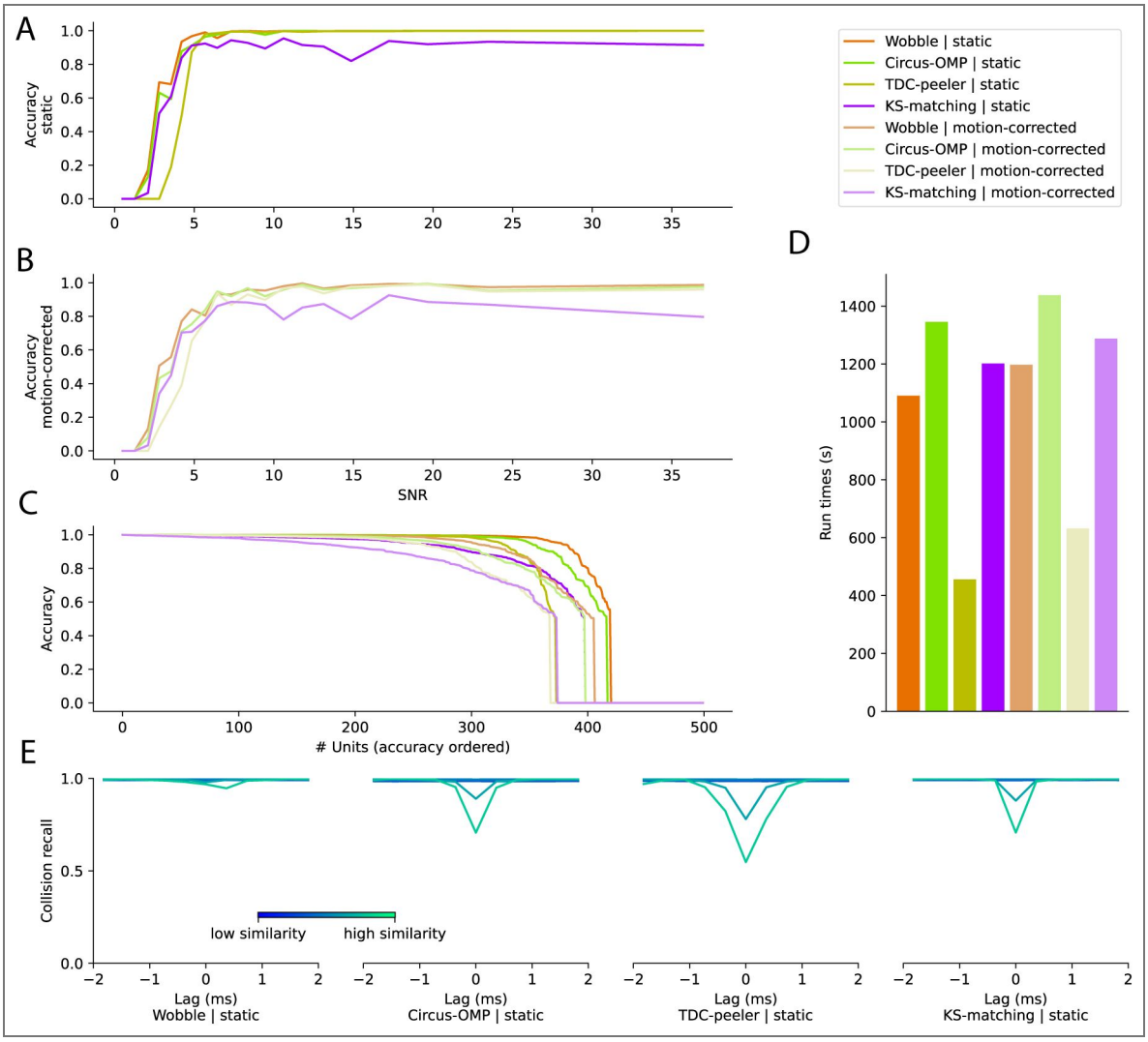

Figure 5. **Template matching benchmark.**

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 [39] 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.

the interpolation of the traces using the kriging kernel method [18]. This 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, which should compensate for the motion itself. Our previous work [19] 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 affects clustering (Figure 4C-E) and template-matching (Figure 5C).

Knowing that motion interpolation causes a significant performance decrease in spike sorting, as shown in Figures 4C-5C, we implemented a new idea for template matching. Instead of interpolating the traces, we directly interpolated the templates [43]. This has three advantages: 1) the interpolation of the templates is less noisy because the templates are smoother than traces; 2) the templates can 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 can be made via better methods (e.g. cubic interpolation). Because these interpolation methods are more computationally demanding they cannot be performed on-the-fly on raw traces, but can be performed 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 can be done with bicubic splines on a grid for a given motion vector. As can be seen, there are still some important non-zero residuals that give rise to errors.

To check the quantitative impact of these errors in the interpolation, we extended the Greedy Matching Pursuit algorithm (TDC-peeler) to have two modes: the standard one (where templates are kept fixed) and a new one with a “drifting templates" 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 algorithm, instead of interpolating the traces, we interpolated templates at any given time and motion prior to the template-matching step. We compared these algorithms for template matching in five different 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 static 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 drifting performance)

The results are shown in Figure 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 degrade the matching procedure.

The final case “True drifting templates", although biologically impossible, provides an upper bound on the performance we could expect for matching in a drifting recording and allows us to estimate the cost of spatial interpolation to compensate for the drift. As shown in Figure 6B-C, the result is as we hypothesized: 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

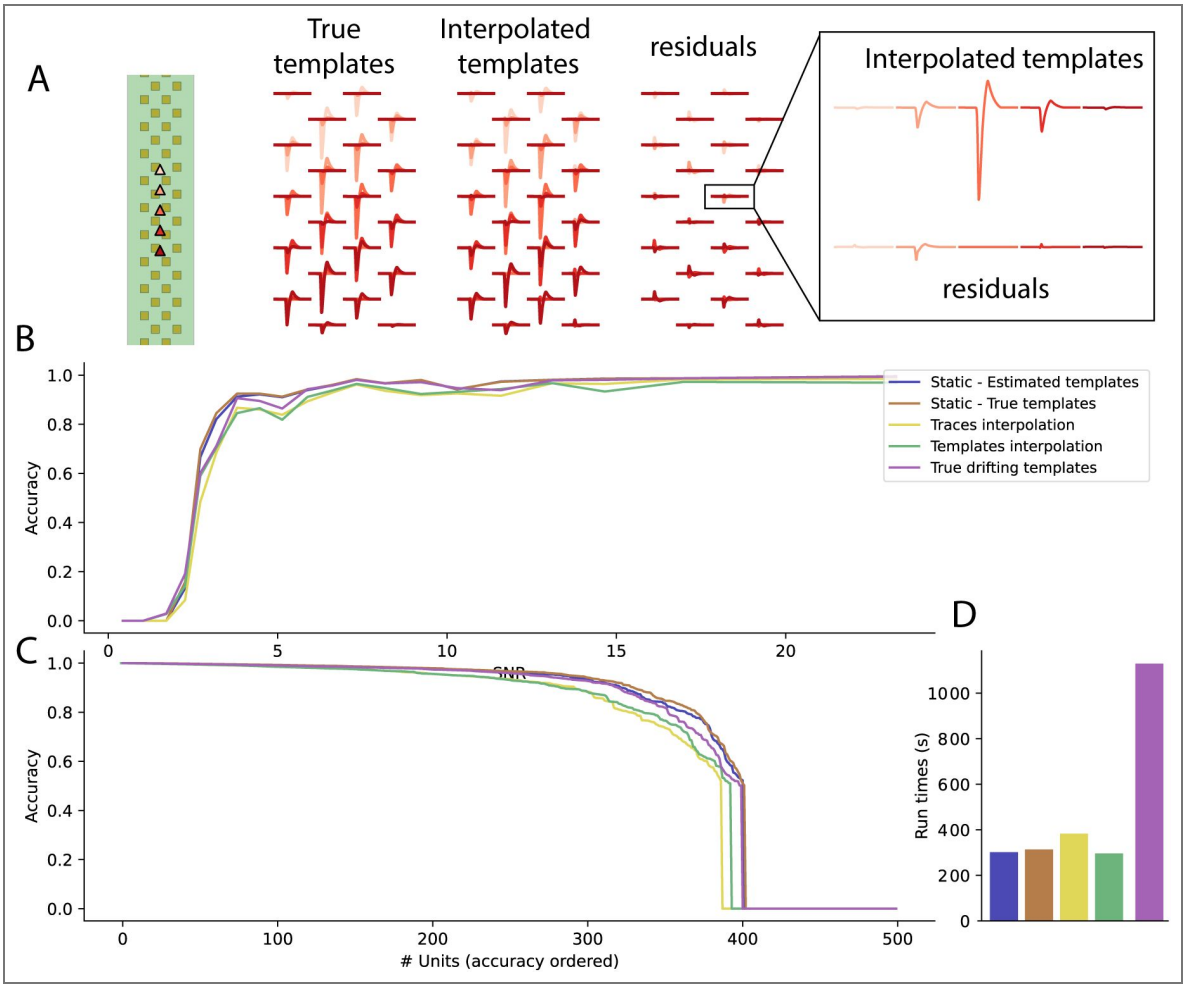

Figure 6. **Motion correction strategies benchmark.**

A Example of template interpolation for a particular neuron drifting along the depth of the electrode. On the Left, the True templates are created from the generative mathematical model, taking into account the real position of the neuron. In the middle, templates are interpolated *via* a 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. The inset displays the interpolated templates and residuals on a single chosen channel, in a non-overlapping manner and as function of depth **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.

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 through interpolation on traces or templates still greatly degrades the performance 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` [18]) to several new sorters entirely built on the components described in this article. We first developed `SpyKING-CIRCUS 2` and `TriDesClous 2`, as updated ports of `SpyKING-CIRCUS` and `TriDesClous`, to show that it was possible to create end-to-end components based sorters. We then constructed `Lupin` as the “best" sorter, based on the benchmarks in this paper. Based on Figures 3, 4 and 5 we choose matched filtering for peak detection, iterative ISOSPLIT for clustering and wobble (Augmented Matching Pursuit) for template matching. These three algorithms were originally developed as part of `Kilosort` [18], `Mountainsort` [12] and `YASS` [17] respectively, showing that `Lupin` benefits from ideas from an array of other spike sorters.

As can be seen in Figure 7A, the performances for all spike sorters on static recordings are comparatively strong, although all sorters sometimes miss obvious units with a large signal-to-noise ratio. This could be 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. Lupin is able to outperform all other spike sorters in both the static and motion-corrected cases.

When recordings have drift and are motion-corrected, there is a 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 for motion. Motion-correction degrades the overall number of cells that can be recovered, slows all pipelines (Figure 7D), and increases the number of False Positives, especially for Kilosort4 (see Figure 7E).

`Kilosort4` performs well on our generated data. It finds many good units and runs quickly, although is the only sorter that runs on a GPU. `Kilosort4` finds many false positives, especially for the motion-corrected recording. Importantly, this demonstrates that our simulated data poses a challenge to state-of-the-art spike sorters. The two spike sorters that were rebuilt as chains of modular components (`SpyKING-CIRCUS 2` and `TriDesClous 2`) have several pros and cons. `SpyKING-CIRCUS 2` has less False Positives but longer run times (Figure 7D, E). On the other hand, `TriDesClous 2` is 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 Positives. This is somewhat expected since such a pipeline is built with the “best” algorithms, evaluated on our generated data, carefully chosen at each step of the spike sorting pipeline. Its performance can be seen as a direct illustration of the gain obtained by sharing ideas within the community.

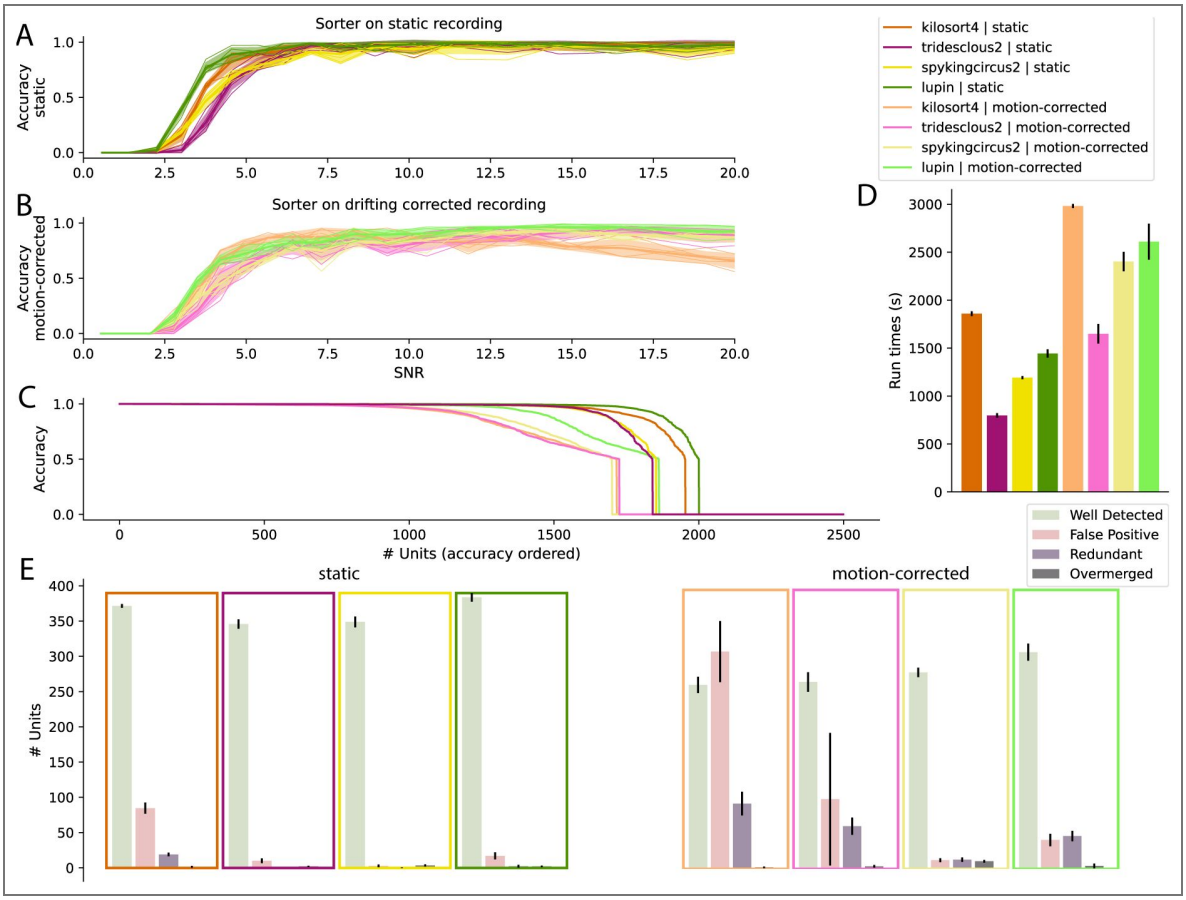

Figure 7. **End-to-end spike sorter benchmark with simulated recordings.**

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).

When designing Lupin we chose the component algorithms which performed best on our Neuropixel 1.0-like generated data. To demonstrate that the results shown in this paper generalize to new probe geometries, we performed additional simulations on ground truth datasets with Neuropixel 2.0, tetrode, SiNAPS [44] and Cambridge Neurotech layouts (see Supplementary Figure S1). In all cases, we found the same general trend observed in Figure 7, showing that Lupin's success generalizes to other probe geometries. In all these cases, our modular pipeline Lupin was on par with state of the art sorters such as Kilosort4 [18], with less False Positives and Redundant Units, especially in the presence of motion.

Finally, we tested the four spike sorters on real world datasets. We chose three datasets with diverse properties (chronic and acute, and from Cambridge Neurotech probes as well as Neuropixels 1.0 and 2.0 probes) and launched end-to-end spike sorters on the data. It is very difficult to assess spike sorters since there is no ground truth for any high-density ephys data. Hence we use results from three automatic curation algorithms as proxies to assess the sorting quality. Both Bombcell [45] and UnitRefine [46] label putative neurons as "good", "mua" (multi-unit activity) and "noise", providing approximate measures of "well detected", "overmerged" and "redundant" units. The SLAy [47] algorithm suggests units that should be merged, a proxy measure for "oversplit". Figure 8A shows the results of the automatic curation algorithms, with grouped columns corresponding to our categories. Note that these metrics are imperfect and only give a suggestion of the quality of the results. Similar to what has been observed with synthetic data, we can see that Lupin is competitive with Kilosort4, finding on average a similar number of good units and less noisy units. TriDesClous2 and SpyKINGCIRCUS 2 find fewer units, but have different advantages that might be interesting given the context. TriDesClous2 is fast, while SpyKING-CIRCUS 2 is mostly tailored for *in-vitro* data, and thus can scale for more thousands of channels while some other algorithms might not [18]. Overall the results show that our method, of optimizing and constructing a spike sorter based on carefully benchmarked results on simulated data, can generate a spike sorter on par with cutting edge end-to-end algorithms when applied to real data. As one can see in Figure 8B, which shows the activity over time for all recordings, the results do not seem to be related to the level of motion artifacts.

Discussion

In this work, we have presented a new and modular framework to dissect and benchmark all 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 step, and allowed us to design new component-based spike sorters SpyKING-CIRCUS 2, TriDesClous 2 and Lupin. The latter sorter, Lupin, which is built using the bestperforming method for each step, is already competitive with the most widely used sorter (Kilosort4) on our simulations and on real data. Unlike Kilosort4 our new sorters have been designed for high performance on CPUs, which is especially advantageous in the context of High Performance Computing and considering the cost and difficulty of getting access to GPUs.

We hope that our initial 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 now have access to a diverse range of adaptable 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, or template matching). They can implement new ideas without needing to write a completely new end-to-end sorter that includes the entire machinery (data reader, preprocessing, and user interface) since 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.

One could argue that such a modular based-approach might still have some limitations. First, it might appear as a suboptimal framework for spike sorters that run different steps iteratively rather than serially. Iterative methods have been used to refine some processing steps, e.g., having a re-clustering step after a first template-matching pass (e.g., as in Kilosort 1 and 2 [10], although

A

Sorter	Tot units	BC good	UR sua	BC mua	UR mua	SLAy merges	BC noise	UR noise
Lebedeva et. al., Chronic, NP2.0, 38 mins								
KS4	808	246	288	489	312	5	73	208
Lupin	683	216	259	460	338	2	7	86
TCD2	617	119	246	491	369	13	7	2
SC2	270	102	133	166	137	1	2	0
IBL, Acute, NP1.0, 67 mins								
KS4	1050	210	459	673	354	24	167	237
Lupin	864	209	379	601	278	6	54	207
TDC2	954	124	417	778	504	33	52	33
SC2	458	97	170	333	271	0	28	17
Duszkiewicz et. al., Chronic, CN 156H5, 211 mins								
KS4	174	41	71	98	68	2	35	35
Lupin	162	56	96	103	63	4	3	3
TDC2	191	11	60	180	128	9	0	3
SC2	58	4	9	53	47	0	1	2

B

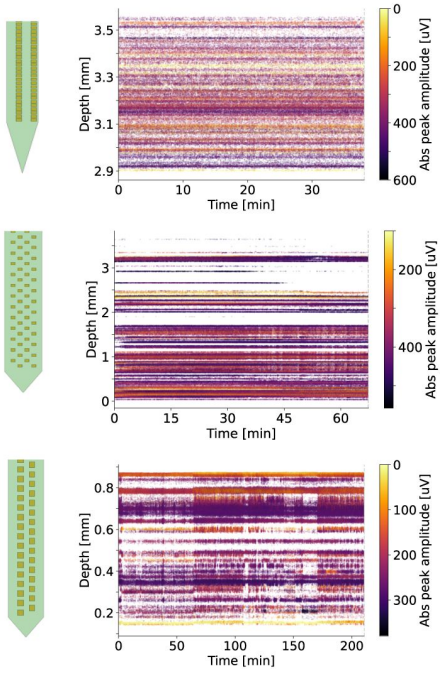

Figure 8. End-to-end spike sorter benchmarks on real data

Results from running four spike sorters (Kilosort4, Lupin, TriDesClous 2 and SpyKING-CIRCUS 2) on three datasets: a chronic Neuropixels 2.0 recording, an acute Neuropixels 1.0 recording [48, 49] and a long chronic recording using a 64-channel Cambridge Neurotech probe [50, 51] **A**: Tabulated proxy measurements of well-detected, overmerged or oversplit, and redundant units. For well-detected, the proxies were units being labeled "good" by `Bombcell` [45] or being labeled "sua" by the Jain lightweight model using `UnitRefine` [46]. The overmerged proxies were being labeled "mua" by `Bombcell` or the Jain model, and the oversplit proxy is the number of suggested merges from the SLAy algorithm [47]. The redundant proxies were being labeled "noise" by `Bombcell` or the Jain model. **B**: a representation of the probe used in each recording, and a raster map of spikes over time, giving an indication of the stability of each recording.

note that Kilosort4 is a serial algorithm). However, nothing prevents creating iterative schemes and repeating individual steps in our modular framework, since all steps are implemented independently of the others. Second, our step-wise benchmarking effort could be extended and made more granular. For example, we did not benchmark the "feature extraction" step individually, but only in conjunction with clustering since most spike sorting algorithms extract spike features in a similar manner. But one could break the two steps up in order to properly benchmark each of them individually. This might provide an understand of how different feature extractions boost performance of particular clustering methods [26]. Third, this modular architecture might not reflect the complexity of "monolithic" spike sorting algorithms that have been optimized to work in specific manners. For example, failures at one particular step might be compensated for in subsequent steps in built-in pipelines, and these nonlinear optimizations might not be found in such a component-based framework. However, even if this is the case, identifying the sources of failures is beneficial to understand, and enhance the performances and the robustness of, the algorithms. The fact that `Lupin`, created here from scratch as the "optimal" combinations of components is *on par* with state of the art spike sorters both on artificial and real data (see Figures 7, 8, S1) is a direct demonstration that our optimization strategy works well.

We have demonstrated the validity of our component-based framework to benchmark individual methods and to build full end-to-end spike sorters, by using a fast and efficient way of generating ground truth data. Although the generative model that we use to simulate ground truth recordings does not capture 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 drift. It has the potential to speed up the development and comparison of spike sorters, not only for benchmarking purposes but also maybe, in the future, for the generation of training/labeled dataset that could be beneficial for end-to-end deep network solutions [52, 53].

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 (in 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 comes from biophysical simulations, which use advanced multi-compartment models to simulate extracellular templates [54] or even the full network activity that gives rise to extracellular recordings [55]. Although the latter types of simulation appear more biophysically plausible, it should be noted that virtually all available cell models are built from *in-vitro* experiments, which affect neuronal physiology and results in waveform shapes and distributions that are different from what is observed *in-vivo* [18]. 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 idea that could be viable for some of the steps that do not require exhaustive ground truth (e.g. template matching or end-to-end comparisons) is hybrid recordings [18, 56]. A potential problem for hybrid recordings 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 precurated spike sorting outputs). Since drift is a major phenomenon for shank-like probes, injected spikes should follow the inherent drift of the recordings. Spikes are therefore moved and interpolated given the estimated motion. When benchmarking the performance of spike sorters on such datasets, a spike sorter that uses the same motion correction method as is 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 benchmarking.

While it has already been shown that drift in a recording causes a major degradation in spike sorting performance [19], in this work we further highlighted that the problem occurs most strongly in the clustering and template matching steps. It is mainly due to the way templates and/or traces are interpolated, even using state-of-the-art motion-correction algorithms [20, 57]. Template matching is very sensitive to residuals, and slight mismatches have a large impact on

performance. The resolution of such issues remains an open challenge in spike sorting, and our work directly addresses this 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. In our opinion, if the estimated motion vectors are below 10 μm, it is better to turn off motion correction and work directly on the raw data. Such motion estimations can easily be computed and visualized with custom methods in `SpikeInterface`, to gain confidence in final results. One potential solution to improve motion correction is to make use of data from the new generation of ultra-dense probes, such as Neuropixels Ultra [58]. These novel probes provide an unprecedented spatial resolution, which could help to gain new insights on biophysical features of extracellular templates and be used to design better interpolation methods for the templates. It is also worth noting that switching from Neuropixels 1.0 to Neuropixels 2.0 seems to reduce the loss of well-detected units in motion corrected recordings compared to static recordings, regardless of the spike sorter (see Supplementary Figure S1 and Figure 7). This suggests that the higher density of electrodes (along the direction of the probe) found in Neuropixel 2 probes reduces the loss of accuracy due to motion.

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□ notation and convolution using □ notation. We consider that the extracellular signals s□(t) are defined on N channels. We use 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 `generation` module in `SpikeInterface` to ease the generation of ground truth artificial recordings, with or without motion drift to mimic *in-vivo* experiments [59]. This new strategy allows us to bypass the use of the `MEArec` simulator [54], at the cost of a slight loss in pseudo-realism. To create a simulated recording, a user must first provide a probe layout supported by `probeinterface` [60]. An in memory recording is created, which can produce traces on-the-fly, avoiding the need to save the generated recording to disk. This lazy mode is crucial, since ground truth recordings can be quite large for high-density probes with long durations. Given the templates, spike times, and the motion vector ^{M}□(t) that affects the units, a seeded reproducible recording is created. 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 w□(t) can either be extracted from third-party libraries or generated *via* a simple generative mathematical model.

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*. This generation procedure has several parameters such as the peak negative/positive amplitudes, the time constant of depolarization and repolarization, and the recovery time. Parameters are 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 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 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 the model is able to reproduce the core features needed to properly challenge modern spike sorters, even while not capturing all the diversity observed *in-vivo* with respect to cell types, morphologies, etc. The precise implementation details can be found by inspecting the generation module of SpikeInterface.

A key feature of this new module 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 $p_i = (x, y, z)$ that can be varied as a function of motion $M_i(t)$ during the course of the experiment. The positions (x, y) are randomly drawn within the area covered by the probe layout, extended by a 20 μm margin. Depths z are considered within the range [5, 40] μm. Second, if the unit templates are taken from an external library or provided by the user, the module will handle drift by interpolating these templates *via* cubic spline interpolation while shifting them with respect to the motion vector $M_i(t)$.

The noise structure of these artificial recordings is simpler than observed in real data, but there are some key properties that can be specified by the user to challenge 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. Similar to [19], since the user has full access to the ground truth, the 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.

In this paper, we focus mainly on 30 minute long simulated recordings with a Neuropixels-1.0 probe layout, 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. In Supplementary Figure S1, we generated four other simulated recordings each with a different probe layout but other conditions kept constant. Here, we use: a Cambridge NeuroTech ASSY-158-H5 64-channel layout; a Neuropixels 2.0-like 128 channel layout; a high density 128-channel SiNAPS layout; and a single 4-channel Tetrode. For all simulations, we fix the average cell density to 2.5 cells per channel.

Real datasets

We tested the new components-based sorters on real data. To do so, we chose three datasets representing a diverse range of experimental conditions: a chronic implantation of a Neuropixels 2.0 probe from the Lebedeva et. al.[Using recording AL032_2020-01-07] [4, 61]; an acute implantation using a Neuropixels 1.0 probe from IBL[Using recording sub-UCLA034_ses-3537d970-f515-4786-853f-23de525e110f_desc-raw_ecephys.nwb] [48, 49] and a long chronic recording using a Cambridge Neurotech probe from Duszkiewicz et. al.[Using recording sub-A3702_ses-191126_behavior+ecephys.nwb] [50, 51]. We ran Lupin, SpyKING-CIRCUS 2 and TriDesClous 2, as well as Kilosort4 (version 4.1.2), on each dataset once without any additional preprocessing. The only default setting we adjusted was to apply motion correction or not depending on whether the recording was acute or chronic.

After sorting, we computed features of the spike sortings using `SpikeInterface`'s sorting analyzer tools (including computing average waveforms, spike amplitude distributions, and template and quality metrics) allowing us to apply several automatic "quality control" tools each output. We applied `BombCell` [45], using default settings, which returns a list of units considered "sua" (singlue-unit activity), "mua" (multi-unit activity) and "noise" determined by a thresholding

decision tree. We also applied the "lightweight" models from Jain using `UnitRefine` [46] These models are classifiers trained on thousands of manually curated units from several datasets. The models output the quality labels "sua", "mua" and "noise", and should represent the decisions of the manual curation decisions fed to the models. Finally, we applied `SLAy` an auto-merging tool [47], which returns paired units it determines to be originally oversplit by the sorters.

Since there is no ground truth for real spike sorted data, we use the output of these tools as a proxy for the categories we assessed the simulated data on: well-detected, oversplit and redundant. We use the `Bombcell` "sua" labels and the `UnitRefine` "sua" label as proxies for well-detected units, `Bombcell` "noise" and `UnitRefine` "noise" labels as proxies for redundant units, `Bombcell` "mua" and `UnitRefine` "mua" labels as a proxies for overmerged units, and number of suggested `SLAy` merges a proxy for oversplit units. These are imperfect metrics. For instance, both the default `Bombcell` parameter tuning the Jain model [46] use Neuropixels data sorted using `Kilosort4` as their input. Regardless, we felt it was fairest to use the default settings to judge the sorting outputs.

Peak detection

Locally exclusive

In this method, commonly performed in spike sorters [15, 11, 17], peaks are detected as negative threshold crossings within a spatio-temporal exclusion zone, to avoid crosscontamination 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 the signal must be to be counted as a peak (`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

This peak detection method relies on the the concept of matched filtering [37]. Because spikes have 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 [18], the idea behind this peak detection method is to create an exhaustive catalog F of artificial templates $f_{n\in1,\dots k}(t)$ at known positions $p_n = (x_n, y_n)$, and to convolve the extracellular signal $s(t)$ with all these spatio-temporal templates to find matches in the data.

To create these artificial templates, the typical waveform $H(t)$ of some detected peaks on a single channel is estimated as a median of, say, 5, `000` normalized waveforms. These initial waveforms are taken from peaks that are detected using the locally exclusive method described above. The computed single channel waveform is duplicated on all nearby channels, such that on every channel c at a position $p_c = (x_c, y_c)$ we have $f_n(c, t) = w_n(c)H(t)$ with a weight $w_n(c)$ that will reflect the spatial decay of the templates. In the following, we model the spatial decay as:

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

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 F are convolved with the extracellular signal $s(t)$, giving rise to a new signal $r(t) \in \mathbb{R}^K$, 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 $r(t)$. Note that because the stereotypical waveform $H(t)$ is the same as the one used to generate all the spatio-temporal templates in the catalog F, the convolutions between F and $s(t)$ can be highly optimized by only performing the convolution between $s(t)$ and $H(t)$ per channel, and then summing these individual convolutions with the weights $w_n(c)$.

Motion estimation

The motion estimation and correction steps of the spike sorting pipeline were modularized in a previous paper [19]. Although SpikeInterface offers multiple methods to handle motion correction, we believe that there is sufficient evidence that DREDge provides the best motion estimation [62, 20]. Hence throughout the paper we always use DREDge to infer the motion of our recordings.

Since motion correction is now a canonical step in most preprocessing pipelines for *in-vivo* recordings with high-density probes, it is important to assess how well various algorithms are with respect to this key preprocessing step. Hence, we often compare the results obtained on static recordings with "motioncorrected" recordings. To be precise, for every ground-truth recording that has been generated using the `generation` module, we create both a static and a drifting version of the recording. A "motioncorrected" recording is a drifting recording in which the motion has been estimated by `DREDge` and then compensated for *via* a kriging interpolation [18]. To our knowledge, this is the closest, albeit not perfect, way to estimate the static recording from the drifting one. If this step was perfect, the "motion-corrected" recording would be equal to the "static" one. Even if such a motion estimation step is pretty good [19], it might still distorts the signal in two ways. First, since it depends on activity levels and cell positions, we need to have many active cells in a localized region to properly estimate the motion [19]. So when there is little activity, the motion estimation is imperfect. Second, because the kriging interpolation used to compensate for the motion smooths the signal and noise, some information from the data is lost during this step.

Clustering

Clustering is one of the most important steps in spike sorting, and numerous clustering methods have been developed. A complete review of all these methods is outside the scope of this manuscript. In the SpikeInterface components module, we have implemented some of the key modern approaches used for high-density probes. Most of these methods start from the observation that clustering in a high-dimensional space is an intractable problem, so one must 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 $w_i(t) \in \mathbb{R}^{1\times M}$, where M is the number of time steps (usually, the temporal duration of a spike *in-vivo* lasts 2 ms), and then perform an SVD to reduce the number of dimensions from M to K = 5, with K < M. Assuming that such an SVD projection is rather typical, it can be extended on a per-channel basis to spatio-temporal waveforms such that snippets $w_i(t) \in \mathbb{R}^{N\times M}$ 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 $w^{SVD} \in \mathbb{R}^{N\times K}$ for the projected representation of $w_i \in \mathbb{R}^{N\times M}$, after SVD. In all the following, we used K = 5, as is commonly used by many sorters.

KS-clustering

The full details of this clustering method can be found in [18]. In a "divide-and-conquer" approach, the waveforms w^{SVD} are split into groups as a function of their estimated depth (the algorithm being 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 [63]). The results of all these individual clusterings are then concatenated while ensuring that cells at the borders of the bin depths, which could give rise to duplicated clusters, are merged before template matching (duplicated templates are removed).

Iterative clusterings (Iter-HDBSCAN and Iter-ISOSPLIT)

These methods are similar to the one used in `Kilosort`, but rely on density-based clustering algorithms. In another "divide-and-conquer" approach, the waveforms w^{SVD} are split into groups as a function of their estimated depth (the algorithm being tailored for Neuropixels-like probes), and a clustering algorithm is applied locally. In particular, all detected peaks are grouped with respect to their peak channel. For all these peaks, all channels in a given neighborhood are

selected, and projected onto a low-dimensional space to obtain the w_{SVD}. These are then clustered, using a density-based clustering algorithm (such as HDBSCAN [64]) or one based on statistical considerations (such as ISOSPLIT [12, 65]). The results of all these individual clusterings are then concatenated while ensuring 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 measurements between found templates, as defined by an l1 norm, are less than a certain threshold (0.8) 2) All small clusters that are likely to be noisy are removed (given a firing rate threshold for found neurons).

Full graph sparse clustering (Global Louvain)

This method attempts to avoid the edge effect of the “divide-and-conquer" approaches encountered with iterative splits and local clusterings. The method also begins with projected waveforms w_{SVD} however, unlike the previous methods, the graph-based clustering is applied to the full connectivity graph at once, in order to avoid border effects. A key to making 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 [63], or even HDBSCAN for the distances). In this manuscript, we used the Louvain algorithm.

Template matching

Once the templates $T_i(t) \in \mathbb{R}^{N\times M}$ have been found and identified as the centroids of the clusters, template matching attempts to describe the signal $s(t)$ 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 [66], and full details can be found in [18, 10]. The signal $s(t)$ is convolved with all the templates $T_i(t) \in \mathbb{R}^{N\times M}$. This convolution leads to the computation of all the scalar products $b_{ij} = T_i(t_j) \cdot s(t_j)$ at any time t_j. 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 $s(t)$, leaving behind the 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 $s(t)$. 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 $T \times T \times (2M-1)$. While 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 based on the Orthogonal Matching Pursuit algorithm [42], which is slightly different to the matching algorithm implemented in `Kilosort`. The main difference is 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 [67]. Again, to speed up the implementation, some internal optimizations can be performed, via Cholesky decomposition [68].

Wobble

This is an augmented Matching Pursuit algorithm, as implemented in YASS [17]. The ideas are similar to the classical Matching Pursuit Algorithm of `Kilosort`, but involve a super-resolution step to enhance the alignment of the templates. In summary, the dictionary of templates $T_i(t) \in$

$\mathbb{R}^{N\times M}$ are "augmented" through multiple slightly time-delayed versions of the templates, to compensate for subsampling jittering. This produces a dictionary of $T_{\square}(t) \in \mathbb{R}^{N' \times M}$, with $N' > N$, 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, originally developed both in SpyKING-CIRCUS [15] and T. In short, peaks are detected as thresholds that cross above k times the median absolute deviation per channel. As in the peak detection step, a spatiotemporal exclusion radius ensures that only the most prominent peaks are kept. At these peak times, we look for the best match to the templates with respect 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 "TDC-peeler" algorithm, which highlights the effect of motion correction while matching templates. It is not part of any of the components based sorters, and is only used in Figure 6. Instead of interpolating the traces, as is done by most modern sorters [18], 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 bi-cubic splines, and once this augmented set of templates has been generated, we apply exactly the same matching algorithm as "TDC-peeler", but now selecting the appropriate template at each time point, based on the inferred motion at this time point.

To generate Figure 5E, we used the same methodology as in [39] 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 [15] based on the modular components implemented in this article. In summary, this spike sorter uses (when motion is present) the DREDge motion correction algorithm [20] 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). Finally, units are merged greedily with built-in function of SpikeInterface suggesting putative merges. 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 [20] 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). Finally, units are merged greedily with built-in function of SpikeInterface suggesting putative merges. 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 [20] 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). Finally, units are merged greedily with built-in function of SpikeInterface suggesting putative merges. The results presented in this paper were created using version 25.12.

Kilosort4 This is the complete standalone algorithm, as implemented in [18]. All the units found by Kilosort were kept for downstream analysis. We used version 4.1.1.

Evaluation

The exact nature of the evaluations that have been performed in every benchmark depends on the nature of the benchmark. In all of our benchmarks, we tried to identify the key goals of every step (for example, capture accurately all peaks for peak detection, find all clusters for clustering, etc.) 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". More details can be found in 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, etc.), we also looked at the number of good, overmerged, and false positive units (see [34] for more details).

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

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

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 the matched sorted unit, respectively, and T P_{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 greater than 20% with more than one GT unit.
- *redundant* : units with an agreement above 20% with a single GT 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).

Supplementary Figures

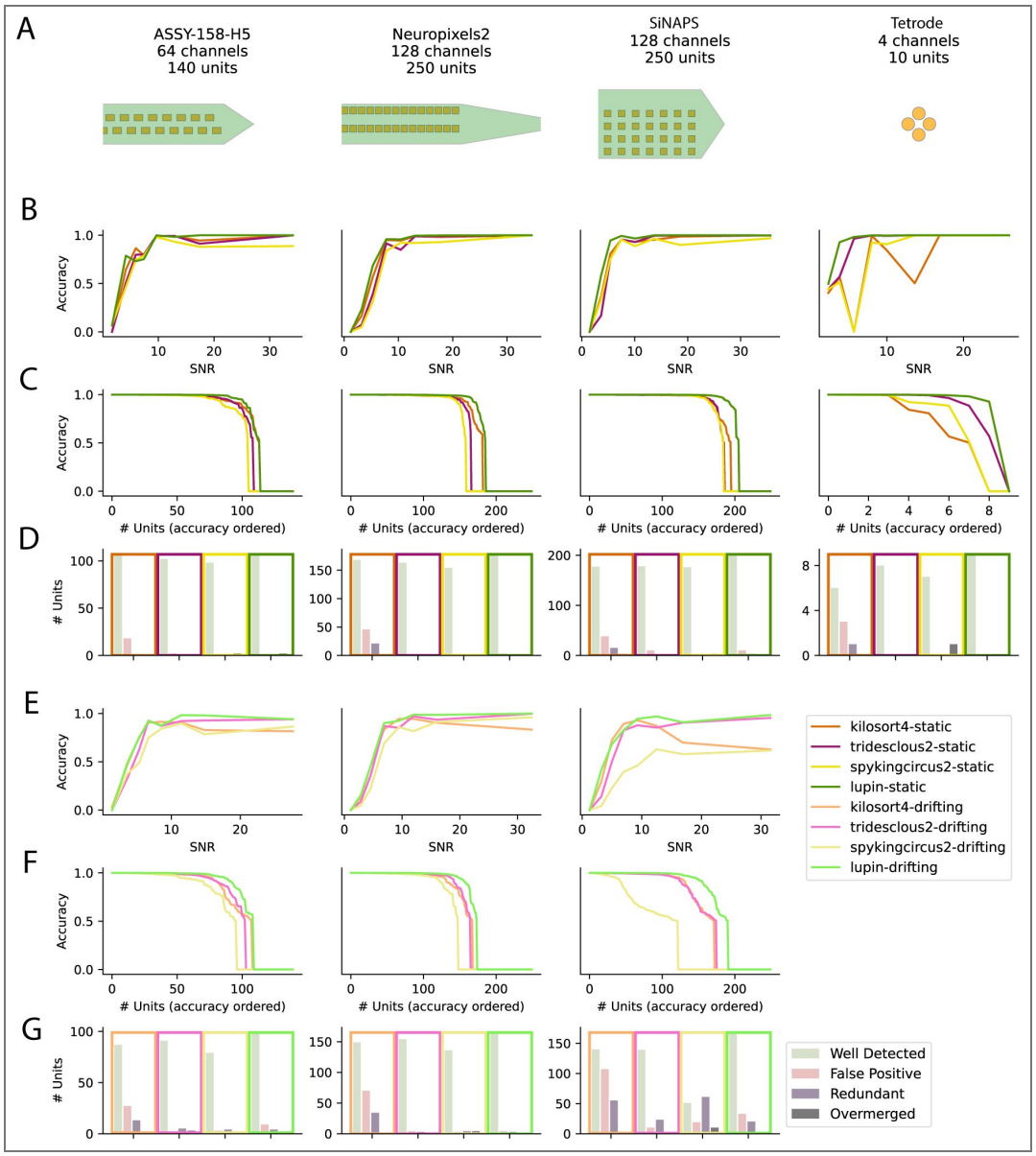

Figure S1. End-to-end spike sorter benchmark with extra simulated recordings. A Different probe geometries considered to generate the synthetic ground truth (see Methods). **B** Averaged accuracy for several spike sorters when applied to several static recordings, as a function of the signal-to-noise ratios of the neurons **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** For all spike sorting methods and recording types, the number of Well Detected, False Positive, Redundant and Overmerged units (see Methods) **E, F, G** Same as in **B, C, D**, but when applied to motion-corrected recordings, with the exact same activity/firing as the static ones (see Methods).

Data availability

All the figures available in this article can be regenerated from Jupyter notebooks that are available at *https://zenodo.org/records/20407862* or *https://github.com/SpikeInterface/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 thank Joe Ziminski for his helpful feedback and comments on the manuscript. PY is supported by the ANR-24-RRII-000, the ANR GNEURO ANR-25-CE42-6535-03 and the *Cross-Disciplinary Project LOOP "Closed-Loop Neurotechnologies: from sensors to applications"* (Initiative d'excellence Université de Lille R-CDP-25-003-LOOP. 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 supported 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)	ANR-24-RRII-000	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	Paul Adkisson-Floro Benjamin K Dichter Heberto Ramon Mayorquin

Author ORCID iDs

Samuel Garcia: https://orcid.org/0000-0001-6389-9779
Alessio P Buccino: https://orcid.org/0000-0003-3661-527X
Pierre Yger: https://orcid.org/0000-0003-1376-5240

References

[1] **Buccino AP,** Garcia S, Yger P (2022) Spike sorting: new trends and challenges of the era of high-density probes. *Progress in Biomedical Engineering* **4**:022005 https://doi.org/10.1088/2516-1091/ac6b96

[2] **Lefebvre B,** Yger P, Marre O (2016) Recent progress in multi-electrode spike sorting methods. *Journal of Physiology-Paris* **110**:327-335 https://doi.org/10.1016/j.jphysparis.2017.02.005 | PubMed

[3] **Jun JJ,** Steinmetz NA, Siegle JH, Denman DJ, Bauza M, Barbarits B, Lee AK, Anastassiou CA, Andrei A, Aydın Ç, *et al.* (2017) Fully integrated silicon probes for high-density recording of neural activity. *Nature* **551**:232 https://doi.org/10.1038/nature24636 | PubMed

[4] **Steinmetz NA,** Aydin C, Lebedeva A, Okun M, Pachitariu M, Bauza M, Beau M, Bhagat J, Böhm C, Broux M, *et al.* (2021) Neuropixels 2.0: A miniaturized high-density probe for stable, longterm brain recordings. *Science* **372** https://doi.org/10.1126/science.abf4588 | PubMed

[5] **Angotzi GN,** 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 https://doi.org/10.1016/j.bios.2018.10.032 | PubMed

[6] **Hua S,** Liu Y, Luo J, Li S, Jiang L, Wu P, Sun S, Shang L, Lu C, Zhang K, *et al.* (2025) Microelectrode arrays cultured with in vitro neural networks for motion control tasks: encoding and decoding progress and advances. *Microsystems & Nanoengineering* **11**:233 https://doi.org/10.1038/s41378-025-01046-7 | PubMed

[7] **Wolff R,** Polito A, Buccino AP, 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 https://doi.org/10.1016/j.xpro.2025.104195 | PubMed

[8] **Schröter M,** Cardes F, Bui C.-VH, Dodi LD, Gänswein T, Bartram J, Sadiraj L, Hornauer P, Kumar S, Pascual-Garcia M, *et al.* (2025) Advances in large-scale electrophysiology with highdensity microelectrode arrays. *Lab on a Chip* **25**:4844-4885 https://doi.org/10.1039/d5lc00058k | PubMed

[9] **Rossant C,** Kadir SN, Goodman DF, Schulman J, Hunter ML, Saleem AB, Grosmark A, Belluscio M, Denfield GH, Ecker AS, *et al.* (2016) Spike sorting for large, dense electrode arrays. *Nature neuroscience* **19**:634 https://doi.org/10.1038/nn.4268 | PubMed

[10] **Pachitariu M,** Steinmetz NA, Kadir SN, 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

[11] **Hilgen G,** Sorbaro M, Pirmoradian S, Muthmann J.-O, Kepiro IE, Ullo S, Ramirez CJ, Encinas AP, Maccione A, Berdondini L, *et al.* (2017) Unsupervised spike sorting for large-scale, high-density multielectrode arrays. *Cell reports* **18**:2521-2532 https://doi.org/10.1016/j.celrep.2017.02.038 | PubMed

[12] **Chung JE,** Magland JF, Barnett AH, et al. (2017) A fully automated approach to spike sorting. *Neuron* **95**:1381-1394 https://doi.org/10.1016/j.neuron.2017.08.030 | PubMed

[13] **Jun JJ,** Mitelut C, Lai C, Gratiy SL, Anastassiou CA, Harris TD (2017) Real-time spike sorting platform for high-density extracellular probes with ground-truth validation and drift correction. *BioRxiv* https://doi.org/10.1101/101030

[14] **Diggelmann R,** Fiscella M, Hierlemann A, Franke F (2018) Automatic spike sorting for highdensity microelectrode arrays. *Journal of neurophysiology* **120**:3155-3171 https://doi.org/10.1152/jn.00803.2017 | PubMed

[15] **Yger P,** Spampinato GL, Esposito E, Lefebvre B, Deny S, Gardella C, Stimberg M, Jetter F, Zeck G, 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 | PubMed

[16] **Chaure FJ,** Rey HG, Quian Quiroga R (2018) A novel and fully automatic spike-sorting implementation with variable number of features. *Journal of neurophysiology* **120**:1859-1871 https://doi.org/10.1152/jn.00339.2018 | PubMed

[17] **Lee J,** Mitelut C, Shokri H, Kinsella I, Dethe N, Wu S, Li K, Reyes EB, Turcu D, Batty E, *et al.* (2020) Yass: Yet another spike sorter applied to large-scale multi-electrode array recordings in primate retina. *bioRxiv* https://doi.org/10.1101/2020.03.18.997924

[18] **Pachitariu M,** Sridhar S, Pennington J, Stringer C (2024) Spike sorting with Kilosort4. *Nature Methods* **21**:914-921 https://doi.org/10.1038/s41592-024-02232-7 | PubMed

[19] **Garcia S,** Windolf C, Boussard J, Dichter B, Buccino AP, Yger P (2024) A Modular Implementation to Handle and Benchmark Drift Correction for High-Density Extracellular Recordings. *eNeuro* **11**:ENEURO.0229–23.2023 https://doi.org/10.1523/eneuro.0229-23.2023 | PubMed

[20] **Windolf C,** Yu H, Paulk AC, 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 https://doi.org/10.1038/s41592-025-02614-5 | PubMed

[21] **Laboy-Juárez KJ,** Ahn S, Feldman DE (2019) A normalized template matching method for improving spike detection in extracellular voltage recordings. *Scientific reports* **9**:12087 https://doi.org/10.1038/s41598-019-48456-y | PubMed

[22] **Saggese G,** Tambaro M, Vallicelli EA, Strollo AG, Vassanelli S, Baschirotto A, Matteis MD (2021) Comparison of sneo-based neural spike detection algorithms for implantable multitransistor array biosensors. *Electronics* **10**:410 https://doi.org/10.3390/electronics10040410

[23] **Toosi R,** Akhaee MA, Dehaqani M.-RA (2021) An automatic spike sorting algorithm based on adaptive spike detection and a mixture of skew-t distributions. *Scientific reports* **11**:13925 https://doi.org/10.1038/s41598-021-93088-w | PubMed

[24] **Souza BC,** Lopes-dos Santos V, Bacelo J, Tort AB (2019) Spike sorting with gaussian mixture models. *Scientific reports* **9**:3627 https://doi.org/10.1038/s41598-019-39986-6 | PubMed

[25] **Eom J,** Park IY, 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 https://doi.org/10.1016/j.neunet.2020.11.009 | PubMed

[26] **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 https://doi.org/10.1088/1741-2552/ac0f4a | PubMed

[27] **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 https://doi.org/10.1038/s41598-022-19771-8 | PubMed

[28] **Keshtkaran MR,** Yang Z (2017) Noise-robust unsupervised spike sorting based on discriminative subspace learning with outlier handling. *Journal of neural engineering* **14**:036003 https://doi.org/10.1088/1741-2552/aa6089 | PubMed

[29] **Shabestari PS,** Buccino AP, Kumar SS, 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). pp. 1-4 https://doi.org/10.1109/biocas49922.2021.9644995 | PubMed

[30] **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 https://doi.org/10.1088/1741-2552/aace8a | PubMed

[31] **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

[32] **Magland J,** Jun JJ, Lovero E, Morley AJ, Hurwitz CL, Buccino AP, Garcia S, Barnett AH (2020) Spikeforest, reproducible web-facing ground-truth validation of automated neural spike sorters. *eLife* **9**:e55167 https://doi.org/10.7554/eLife.55167 | PubMed

[33] **Merow C,** Boyle B, Enquist BJ, Feng X, Kass JM, Maitner BS, McGill B, Owens H, Park DS, Paz A, *et al.* (2023) Better incentives are needed to reward academic software development. *Nature Ecology & Evolution* **7**:626-627 https://doi.org/10.1038/s41559-023-02008-w | PubMed

[34] **Buccino AP,** Hurwitz CL, Garcia S, Magland J, Siegle JH, Hurwitz R, Hennig MH (2020) Spikeinterface, a unified framework for spike sorting. *eLife* **9**:e61834 https://doi.org/10.7554/eLife.61834 | PubMed

[35] **Wyngaard AJG,** Llobet V, Barbour B (2022) 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

[36] **Steinmetz PN** (2024) Simulation of background neuronal activity and noise in human intracranial microwire recordings. *Journal of Neuroscience Methods* **402**:110017 https://doi.org/10.1016/j.jneumeth.2023.110017 | PubMed

[37] **Turin G** (1960) An introduction to matched filters. *IRE Transactions on Information Theory* **6**:311-329 https://doi.org/10.1109/tit.1960.1057571

[38] **Quiroga RQ,** Nadasdy Z, Ben-Shaul Y (2004) Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural computation* **16**:1661-1687 https://doi.org/10.1162/089976604774201631 | PubMed

[39] **Garcia S,** Buccino AP, Yger P (2022) How do spike collisions affect spike sorting performance?. *Eneuro* **9** https://doi.org/10.1523/eneuro.0105-22.2022 | PubMed

[40] **Pachitariu M,** Steinmetz NA, Colonell J (2019) Kilosort2. https://github.com/MouseLand/Kilosort2

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

[42] **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. **1** pp. 40-44 https://doi.org/10.1109/acssc.1993.342465

[43] **Boussard J,** Windolf C, Hurwitz C, Lee HD, 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

[44] **Angotzi GN,** Vöröslakos M, Perentos N, Ribeiro JF, Vincenzi M, Boi F, Lecomte A, Orban G, Genewsky A, Schwesig G, *et al.* (2025) Multi-shank 1024 channels active SiNAPS probe for large multi-regional topographical electrophysiological mapping of neural dynamics. *Advanced Science* **12**:2416239 https://doi.org/10.1002/advs.202416239 | PubMed

[45] **Fabre JMJ,** Beest EH. v, Peters AJ, Carandini M, Harris KD (2023) Bombcell: automated curation and cell classification of spike-sorted electrophysiology data. Zenodo. https://doi.org/10.5281/zenodo.8172821

[46] **Jain A,** Greene R, Halcrow C, Swann JA, Kleinjohann A, Spurio F, Graff S, Pan-Vazquez A, Kampa B, Gall J, *et al.* (2025) UnitRefine: A community toolbox for automated spike sorting curation. *bioRxiv* https://doi.org/10.1101/2025.03.30.645770

[47] **Koukuntla S,** DeWeese T, Cheng A, Mildren R, Lawrence A, Graves AR, Cullen KE, Colonell J, Harris TD, Charles AS (2025) SLAy-ing oversplitting errors in high-density electrophysiology spike sorting. *bioRxiv* https://doi.org/10.1101/2025.06.20.660590 | PubMed

[48] **Angelaki D,** Benson B, Benson J, Birman D, Bonacchi N, Bougrova K, Bruijns SA, Carandini M, Catarino JA, et al. (2025) A brain-wide map of neural activity during complex behaviour. *Nature* **645**:177-191 https://doi.org/10.1038/s41586-025-09235-0 | PubMed

[49] **International Brain Laboratory,** Benson B, Benson J, Birman D, Bonacchi N, Carandini M, Catarino J, Chapuis G, Dayan P, DeWitt E, *et al.* (2026) Ibl - brain wide map. DANDI Archive. https://doi.org/10.48324/dandi.000409/0.260309.1324

[50] **Duszkiewicz AJ,** Orhan P, Skromne Carrasco S, Brown EH, Owczarek E, Vite GR, Wood ER, Peyrache A (2024) Local origin of excitatory–inhibitory tuning equivalence in a cortical network. *Nature neuroscience* **27**:782-792 https://doi.org/10.1038/s41593-024-01588-5 | PubMed

[51] **Duszkiewicz A,** Skromne Carrasco S, Peyrache A (2026) Large-scale recordings of head direction cells in mouse postsubiculum. DANDI Archive. https://doi.org/10.48324/dandi.000939/0.260512.1701

[52] **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

[53] **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

[54] **Buccino AP,** Einevoll GT (2020) Mearec: a fast and customizable testbench simulator for ground-truth extracellular spiking activity. *Neuroinformatics* 1-20 https://doi.org/10.1007/s12021-020-09467-7 | PubMed

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

[56] **Buccino AP,** Sridhar A, Feng D, Svoboda K, Siegle JH (2026) Efficient and reproducible pipelines for spike sorting large-scale electrophysiology data. *eLife* https://doi.org/10.7554/elife.110170 | PubMed

[57] **Watters N,** Buccino A, Jazayeri M (2025) MEDiCINe: Motion correction for neural electrophysiology recordings. *eNeuro* https://doi.org/10.1523/eneuro.0529-24.2025 | PubMed

[58] **Ye Z,** Shelton AM, Shaker JR, Boussard J, Colonell J, Birman D, Manavi S, Chen S, Windolf C, Hurwitz C, *et al.* (2025) Ultra-high-density Neuropixels probes improve detection and identification in neuronal recordings. *Neuron* **113**:3966-3982.e12 https://doi.org/10.1016/j.neuron.2025.08.030 | PubMed

[59] **Steinmetz N** (2021) "Imposed motion datasets" from Steinmetz et al. Science 2021. figshare. https://doi.org/10.6084/m9.figshare.14024495.v1

[60] **Garcia S,** Sprenger J, Holtzman T, Buccino AP (2022) ProbeInterface: A Unified Framework for Probe Handling in Extracellular Electrophysiology. *Frontiers in Neuroinformatics* **16**:823056 https://doi.org/10.3389/fninf.2022.823056 | PubMed

[61] **Lebedeva A,** Okun M, Krumins MK, Carandini M (2023) Chronic recordings from Neuropixels 2.0 probes in mice. figshare. https://doi.org/10.5522/04/24411841.v1

[62] **Varol E,** Boussard J, Dethe N, Winter O, Urai A, Laboratory TIB, 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

[63] **Blondel VD,** Guillaume J.-L, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* **2008**:P10008 https://doi.org/10.1088/1742-5468/2008/10/p10008

[64] **McInnes L,** Healy J, Astels S (2017) hdbscan: Hierarchical density based clustering. *Journal of Open Source Software* **2**:205 https://doi.org/10.21105/joss.00205

[65] **Magland JF,** Barnett AH (2016) Unimodal clustering using isotonic regression: ISO-SPLIT. *arXiv* https://doi.org/10.48550/arXiv.1508.04841

[66] **Mallat S,** Zhang Z (1993) Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing* **41**:3397-3415 https://doi.org/10.1109/78.258082

[67] **Bian S,** Zhang L (2021) Overview of match pursuit algorithms and application comparison in image reconstruction. In: 2021 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC). pp. 216-221 https://doi.org/10.1109/ipec51340.2021.9421295

[68] **Lubonja A,** Præsius SK, Tran TD (2024) Efficient Batched CPU/GPU Implementation of Orthogonal Matching Pursuit for Python. *arXiv* https://doi.org/10.48550/arXiv.2407.06434

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 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 be on par or maybe even outperform existing spike-sorting packages on their benchmark. While the modules forming Lupin were chosen based on a single benchmark, end-to-end evaluation of different sorters now use multiple benchmarks, including different available datasets.

Overall, this is a strong tool/resource contribution with clear potential to accelerate spike-sorting development and enable more rigorous comparisons. While not the main point of the paper and therefore less important, remaining claims regarding Lupin outperforming other sorters are not well supported. Lupin does not necessarily outperform all other sorters in real data if you consider 'finding most good units' more important than decreasing false positives - something that quality metrics post-sorting can take care off.

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 facility uptake and community contributions.

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

Weaknesses:

(1) I appreciate the use of automatic curation tools to define the quality of units obtained from different spike sorters. However, a discussion on what is considered a good trade-off is missing. For example, one could argue that as long as you apply these curation tools post-sorting, finding more good-quality units is more important than keeping the number of false positives low, as these can be filtered out at a later stage by these curation methods. The manuscript currently seems to argue the balance between high number of good and low number of false positives is more important. This needs to be discussed and rationalized more explicitly, rather than using terms like 'clearly outperforms' and 'good trade-off'.

(2) Although I agree that overfitting to specific data is a general problem with spike sorting (as you can also change parameters of individual sorters), and perhaps this is even required for the best results, I still miss an explicit discussion of this.

(3) While the end-to-end evaluation is a very good addition, defining the best strategy for each module (which in turn led to choices what to use for Lupin) is still based on one benchmark only. This remains a weakness.

Cmments on revised version:

Previously identified weaknesses in limited support for Lupin being superior to other spike sorters, especially using only a single benchmark, have been largely addressed by 1) making superiority of Lupin less of a point in the manuscript and 2) adding more simulations and real datasets. Also, clarification on serial versus iterative spike sorting has been added to the discussion.

https://doi.org/10.7554/eLife.110588.2.sa3

Reviewer #2 (Public review):

Summary

Spike sorting, that is, assigning events detected in extracellular electrophysiology data to 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 relative motion of the tissue and recording sites, and large volumes of data. Experimental ground truth data - where the correct assignment of spikes in 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 these 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 voltages traces yields similar accuracy to interpolation of the spike templates.

Strengths

The paper shows that useful insight can be gained through analyzing process step by step. While this analysis has also been done in papers presenting spike sorters (for example, Pachitariu (2024)) the tools presented here allow users and developers to do similar studies for their own work. This toolset will be 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/Limitations:

The model ground truth data used in testing spike sorting and its components 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. Therefore, the insights gained concerning optimization of the individual steps should be interpreted as "correct for that model data. The comparison of the paper's new sorter to standard sorters on experimental recordings suggests that the benchmarking data is reasonable. Nevertheless, users of these tools will need to assess how well the simulated data matches their recordings.

https://doi.org/10.7554/eLife.110588.2.sa2

Reviewer #3 (Public review):

Summary:

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).

The validity of the work is poor. In particular, the simulator is unrealistic and the ground truth dataset is too short. Several spike sorters are used as straw men, and most sorters compared have never been described in peer-reviewed literature or in sufficient detail. The purely feedforward architecture of the modules is very limiting and irrelevant for modern sorters. Finally, the reporting of results is sporadic and does not follow scientific reporting standards.

General comments:

(1) Abstract, lines 14-16: "We then leverage these results to create a modular component-based spike sorter that can outperform Kilosort 4 on dense and large simulated recordings and produce similar quantitative results on real data." However, the artificial data are not "large" - they are very short. And "similar quantitative results" cannot be assessed on real data because those data do not have any ground truth. Because this is a revision and the authors have already received similar feedback from this Reviewer, I am not sure what to recommend.

(2) In a previous comment, I indicated that the simulator itself is overly simplistic, and indicated that as far as I am concerned, the authors must improve it in one of two ways: (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. The authors explained in their answer - but not in the MS - some of the shortcomings of biophysical simulators, but chose to do neither. My comment therefore remains unaddressed in the MS.

(3) In a previous comment, I indicated that the duration of 10 minutes is too short. The authors chose to extend the duration to 30 minutes, which is insufficient for units that have low firing rates. Units that fire e.g., 0.1 spikes/s would have fewer than 200 spikes. If this MS is to be taken seriously, the simulation should be done on durations that (1) are similar to the potential applications - which may be many hours or even days; (2) allow identification of low-firing neurons. Therefore, about 3 hours is the bare minimum. Therefore, at present, my comment remains unaddressed.

(4) In a previous comment, I indicated that some sorters have never been described in peer reviewed papers and therefore, they must either be removed from the present comparisons or be described in full. The authors chose to persist in relying on un-reviewed online documentation. Thus, my comment remains unaddressed, and the comparisons that involve those sorters (e.g., TDC, TDC2, SpyKING Circus 2) are simply invalid.

(5) In a previous comment, I indicated that some sorters (SpyKING Circus 2 and TDC2) are effectively straw men and suggested to reorganize the MS to demonstrate its main goal. Specifically, I suggested to reorganize 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. The authors agreed that these are straw men and gave a historical account of why these were included. However, they did not explain these

considerations in the MS itself, and chose not to reorganize the MS. Therefore, my comment remains unaddressed.

(6) In a previous comment, I commented about the presentation, description, and interpretation of the results, and indicated that the choice to report point estimates makes any conclusions based on those results invalid. The authors did not make any changes to the reporting. Therefore, none of the results reported in the MS can be taken as an outcome of scientific inquiry. In other words, the MS does not deliver any solid findings - neither scientific nor methodological.

https://doi.org/10.7554/eLife.110588.2.sa1

Author response:

The following is the authors' response to the original reviews.

As requested by all three reviewers we have added a new figure which applies our new end-to-end sorters on real openly available data. This demonstrates that our modular algorithms, optimized to work on simulated data, also performs well in real world cases. In addition, to demonstrate that our results are not due overfitting on simulated data on a specific probe geometry (Neuropixels 1.0), we added a Supplementary Figure demonstrating that our positive results for the end-to-end spike sorters are observed for numerous probe geometries (Neuropixels 2.0, SiNAPS, tetrode and Cambridge NeuroTech). We hope that the extended applications will convince the reviewers and readers of the robustness of our results.

> ***Reviewer #1 (Public review):***
>
> *Weaknesses:*

The reviewer identifies several 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.*

We would like to thank the reviewer for the comments and the valuable feedback. We revised our paper to answer the major concerns that were raised by the reviewer. Regarding the claims about Lupin (1,2), we modified the manuscript in two directions: (i) we attempted to stress that the goal of the paper is not the introduction of the new Lupin sorter *per se*, but rather presenting and highlighting the modularity of the sorting components framework. In doing so, we also toned down our claims of superiority; (ii) we added simulations on a diverse range of probes and included three real experimental datasets, obtained with different probes, in the results. Regarding the iterative aspect of some sorters – point (3) – we added a paragraph to the discussion highlighting that Kilosort4, unlike previous versions of KiloSort, [9] does not iterate over modules anymore, and thus it can be regarded as a serial

algorithm. In addition, although all sorters we present are serial, our proposed framework does not prevent iterative schemes: for example, one could add a re-clustering step after a first template-matching pass. In fact, the modular framework makes this task even easier than before.

Finally, related to point (4), we added some comments on the problem of overfitting with modular benchmarks in the Discussion, but we think that the risks have been mitigated with the addition of more end-to-end examples with various probe types and experimental data in the revised manuscript.

The reviewer also points toward some possible 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.*

In response to these comments, the manuscript has been extended both with more simulated ground truth recordings and experimental data. For additional ground truth, we generated recordings with various probe geometries, to check that the results observed for our modular pipeline could generalize (see Supplementary Figure). Regarding real experiments, we chose three open dataset of various types (chronic and acute implantations, IMEC and Cambridge Neurotech devices) and compared the results of several sorters at a macroscopic level relying on high-level automatic curation tools [7, 3, 6] to quantify how many "good", "oversplit" and "noise" units are found. This is now a new Figure 8 in our manuscript. We believe that the results from all these datasets demonstrate that Lupin is, as is claimed in the paper, *on par* with the most popular spike sorting algorithm, Kilosort4 [9].

Regarding overfitting, we do not believe this is a direct consequence of the modular approach introduced in this article, but rather a general potential "risk" in the spike sorting field. Each spike sorter exposes a large array of parameters that users can tweak to attempt to optimize outcomes on specific datasets, but this end-to-end fine-tuning is hard to control and quantify. We believe that the modular benchmarks introduced in this paper may enable a finer, more controlled, and quantifiable parameter exploration. As an example, very high firing rates such as those observed in the cerebellum might require different parameters for peak detection than for neurons in the cortex. To test this, one could use our generation framework to mimic key macroscopic features of the system you are studying, and benchmark the peak detection step to find optimal parameters. Overall, we do not see this optimization strategy as a problem. Extracellular electrophysiology is so diverse based on brain regions, species, conditions, tasks, etc. that generalizable "best practices" might not exist.

> ***Reviewer #1 (Recommendations for the authors):***
>
> *(1) Tone down or further support the Lupin and specific modules' superiority claims.*

This has been modified in the manuscript, and we added a final figure to discuss how Lupin is *on par* with Kilosort on real data, but with no claim of superiority.

> *(2) Add benchmark diversity, this would test generalization and mitigate benchmark overfitting. Specifically: add more probe-geometries, and allow for different depth-profiles of clusters of neurons.*

We thank the reviewer for the suggestion, and indeed, we added some more benchmarks to convince the readers that the results observed can be generalized properly. More specifically, we extended the duration of the recordings to 30 min, and included additional benchmark datasets with four different probe geometries (Neuropixels 2.0, a Cambridge Neurotech probe, a SiNAPS probe and a tetrode).

> *(3) Clarify how (x,y) positions of neurons are distributed around the shanks.*

This has been clarified in the Methods section. The (*x,y*) positions are generated uniformly within a rectangle covering the probe boundaries, plus a 20µm margin. Regarding the depth, *z* positions (distance from the probe) are drawn uniformly from the range [5,40] µm.

> *(4) Add at least one real-data benchmark. Some suggestions for evaluation are: stability of firing rates over time, agreement across sorters, quality metrics, functional validation, expert curation.*

As suggested by almost all reviewers, we added some real-data benchmarks (see last figure). Since experimental data don't have ground truth, we used automatic curation tools as a proxy for "goodness" of the results. We used automatic labels from Bombcell [3] and UnitRefine, which label units as good, multi-unit activity (MUA) and noise, and SLAy [7] for automatic merging, which correlates with the amount of putative oversplits. We felt it was fairer to use external curation tools instead of creating our own methods to assess quality.

> *(5) Clarify recommendations for users facing drift. While your statement of 'not having drift is ideal' is true, the reality is that many recordings have drift. Some practical solutions would be useful. For example, when to trust results and how to report drift sensitivity.*

The reviewer is right, and we added a sentence to clarify when motion correction methods should be used, in our opinions.

> *(6) It would help to see what types of signals are most often missed, for example: low-SNR units, drifting units, bursty units, and show which modules affect which of these issues.*

This is already shown in Figures 4 of the manuscript, at the clustering level. These Figures show that cells with low firing rates and/or low SNR are most likely to be missed by all sorters. Our ground truth simulator does not include a bursting mechanism yet. We think that this would be an interesting aspect to simulate and we plan to include bursting units, with bursty spike trains and waveform modulation, in future releases. We thank the reviewer for the suggestion.

> *(7) To overcome the issue of overfitting to specific datasets rather than generalization, it would help if a 'default' or 'recommended starting point' for users were described in more detail.*

We overcome the issue of overfitting by adding other artificial ground truth recordings (see Supplementary Figure S1), and also real world dataset (see Figure 8). In all these simulations, Lupin is used with default parameters, and this is, we believe, a good starting point. Of course, for very special needs (animal species, brain structures, ...), one might need to adapt parameters, but so as for any other sorters, and such an exploration of the parameter space is out of the scope of the current manuscript. This has been added in the Discussion.

> *(8) A lot of the plots have ticks / labels too small to read in 100%, or show quite low-resolution. For example, Figure 3 and Figure 5. Please homogenize across all figures.*

The figures have been regenerated and homogenized as suggested.

> *(9) Consider archiving the GitHub version used to generate the figures on Zenodo (DOI) for posterity.*

This has been done for the current state of the manuscript at *https://zenodo.org/records/20407862* and the code is available at *https://github.com/SpikeInterface/sorting_components_benchmark_paper*

> *(10) To make the manuscript more reader-friendly, I recommend adding graphics representing the different methods. For example, in Figure 3 one could add schematics of the two peak-detection methods.*

We thank the reviewer for this suggestion. However, this project and our manuscript is not introducing these methods, only re-implementing them in a modular framework. Hence we feel it is out of the scope of this manuscript to produce schematics of the many methods discussed in the paper readers should view the original sources to find out more information.

> *(11) Discuss what is meant by KS-like clustering, as I was under the impression that KS4 is also iterative. This may be on the template-matching side, but it's difficult to know where you draw the border between iterative clustering and (iterative) template-matching. Potentially, we would want to see these processes as one module together, as many sorters work in an iterative fashion across these steps.*

By KS-like clustering, we meant that the code is a direct port, in Python, of the clustering algorithm implemented in KiloSort4. However, we cannot guarantee that this is the exact same clustering method, because of the way the clustering of KiloSort is interleaved with some others steps part of the algorithm. To be more specific, Kilosort has its own special way of performing matched filtering, with a custom grid of templates generated internally with a higher resolution compared to the recording channels positions. These templates are then used to estimate a putative position of each spikes, and these positions are the ones that are used to start the clustering. Thus, the term KS-like comes from the fact that we do not reproduce this exact same mechanism. In SpikeInterface matched filtering is performed using an equivalent approach, but positions are not estimated in the exact same manner. Since the clustering code is equivalent, the results might differ. Regardless of these details, the clustering is not iterative. In former versions of KiloSort, the core algorithm used ideas from k-SVD algorithms, often used in the Machine Learning community. In such an algorithm, the goal is to learn a sparse dictionary of templates to reconstruct the signals, and indeed, there was some iterations between optimizing the templates and the spike times. However, this is no longer the case in KiloSort4. The algorithm works in a serial manner, following the global strategy mentioned in our paper.

> *(12) Rather than showing results for one specific simulation, it would be more convincing if we saw the average result of multiple simulations. We don't need to see individual neurons necessarily (e.g., Figure 3B and C, but this applies throughout the manuscript).*

While we tend to agree with the reviewer than averaging over multiple datasets might be more informative (as we did in the final figure, for end-to-end sorter comparison), we want to say that given the fact that we are using ground-truth recordings that are randomly generated, as long as we do not change the macroscopic properties of the recordings, results on various instances of the noise will be very similar, and averages might not be as informative as one could hope for. One option would be to vary the parameter space of these ground truth recordings, but then there are so many parameters (noise levels, firing rates,

distributions of the cells, ...) than averaging everything, and/or even choosing what should be primarily studied is an open question on its own. However, to ease the redibility of the figures, individual neurons were removed from the plots.

> *(13) Legends are often incomplete. For example, describe what individual dots are and what lines are in the different figures, even if it seems obvious.*

Legends have been updated.

> *(14) Figure 7 would benefit from average thick curves for each model (optionally with individual thin lines for each instance, or error bars). Individual neurons can be left out. Figure 7E (and likewise Figure 5E) would benefit from having x labels to indicate precise labels, so the color attribution is reserved for specific models. It's quite an intense 'search' game to understand these figures.*

All the figures have been regenerated, and for the sake of readability, we removed the scatter plots for the individual neurons, focusing only on the averaged lines.

> *(15) While I appreciate the effort is gigantic, it may be better for the reader to conclude that.*

This has been rephrased.

> ***Reviewer #2 (Public review):***
>
> *Major comments*
>
> *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.*

We agree with the reviewer that extrapolating our results to real data is difficult, and the same point was raised by the other referees. We have now extended the results to include three experimental datasets from different probes. Due to the lack of ground truth, we used automatic curation tools (Bombcell [3] and UnitRefine for labeling, SLAy for merging [7]) as a proxy for performance and showed that Lupin is on par with Kilosort4 on all datasets. We hope this gives users some idea of how well the new sorters will work on their data.

> ***Reviewer #2 (Recommendations for the authors):***
>
> *(1) Any comparison to experimental data would be welcome. Is it possible to add firing rate, amplitude, and template similarity distributions from measured recordings to the panels in Figure 2D?*

Experimental data is very diverse, depending on brain region, species, recording technology, etc. Instead of extending the comparison between simulated and experimental data, we rely on new Figure 8 to showcase the applicability and performance of the presented methods on real recordings.

> *(2) Do yields of units passing basic quality metrics look different with the new sorter vs. established sorters? Another option that would help establish the range of applicability of the results would be a different model ground truth system, e.g., the hybrid ground truth data used by the authors in reference 7.*

As it can be seen on synthetic recordings (e.g Figure 7A-B), all sorters behave similarly with respect to firing rates, signal to noise ratios. To help establish the range of applicability, we added an extra Figure 8 in the manuscript, as described above.

> *(3) Interpolation errors are likely larger for NP1.0 probes - which have 40 um vertical pitch - than NP2.0 probes, with 15 um pitch. If it's possible to include even a small-scale comparison of the results from the 2.0 geometry, that would be very valuable for readers trying to decide what probe type to use.*

In the revised manuscript, we added new ground truth benchmarks with a NP2, and other, layouts to show that the key results do not depend on the geometry of the probe (see Supplementary Figure 1). But to address more specifically the point raised by the reviewer, we note that in the case of applying Lupin to NP2 layouts, there is only a loss of ≈ 7.5% of well-detected units between static and motion-corrected cases (see Supplementary Figure 1). Where when we apply Lupin to NP1 layouts (see Figure 7) there is a decrease of 20% of well-detected units. So clearly, a smaller pitch and the columnar arrangement of NP2 seems to help motion-correction methods, and thus reduce the failures due to motion. This has been added in the Discussion.

> *About the text:*
>
> *(1) In panel E of Figures 5 and 7: Adding the name of the sorter algorithm under the bar charts would be helpful. It is encoded by the color of the outline of each box, but I found that cue rather subtle.*

The figures have been regenerated, with increased ticks and label fonts. We found that adding the names made the Figures too dense, and acronyms would not simplify the figures. In the end we decided not to add them.

> *(2) Is the number of features (K) used for clustering already mentioned in the main text or methods? I couldn't find it.*

We thanks the reviewer for pointing out this problem, and the answer ($K = 5$) has been added in the manuscript, in the methods section.

> *(3) Equation 2 in Methods, defining accuracy, appears to be incorrect. I believe the correct equation is: accuracy = TPij/(Ni + Nj - TPij).*

The reviewer is right, and this has been corrected

> *(4) In the abstract, line 18: Component based spike sorters => component based spike sorter.*

This has been corrected

> *(5) Line 169 "we can artificially boost the signal-to-noise..." I don't really see anything artificial about leveraging the extra information in neighboring sites. Maybe just remove that adjective?*

We removed "artificial" from the sentence.

> *(6) Line 196-197, describing the result in Figure 3: Especially since 3A is a log plot, it would be helpful to add a percentage to the spikes missed on the high end. These are probably pretty unusual cases, under 0.1%?*

This has been commented in the text, but both because this number is only an approximation (the problem of pairing peaks between detected and ground truth is slightly ill-defined), and because it depends on the particular seeds and parameters of the artificial ground-truth, we avoided numerical values.

> *(7) Line 567: "number of dimensions from M to K 5" => "number of dimensions from M to K[5]" That is, is the 5 meant to be a reference? Or is 5 the number of dimensions retained*

> *to describe the temporal waveform?*

5 is the value of K, and this has been corrected in the manuscript.

> *(8) Line 585-590: Does this description of how to handle borders between groups of sites also apply to the KS-clustering method?*

For the KS-clustering methods, we used the original method implemented in KiloSort. In fact, KiloSort aggregates all the clusters found during the clustering steps, launched per bins, but duplicated templates (based on their shapes only) are removed before matching (and those are the cells at the borders found numerous times). After the template-matching step, once cells have been "populated" with theirs spikes, templates are once again removed and/or merged. However, as explained in the methods, currently this step has not been implemented in our framework. To be more explicit, during template matching, KiloSort stores the features of all discovered spikes, in a space where the effects of nearby spikes have been subtracted, to get a clearer picture of these features. In this "denoised" space, the clustering algorithms is launched again on all spikes (decimated), to assign labels.

> *(9) Line 687: "In fact, a major main with Kilosort lies after the template matching step..." => "In fact, a major difference with Kilosort lies after the template matching step...".*

This has been corrected.

> ***Reviewer #3 (Public review):***
>
> *Major 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.*

The reviewer is right when pointing out that the current ground truth generator is not "biophysical", and this is why in the manuscript we used the terms "biophysically plausible". However, we decided to changed this phrase to "phenomenological" in order to avoid confusion. We believe that our generation tool has the key ingredients to challenge (and also demonstrate limits of) modern spike sorters, which is the goal of the proposed simulator. Spike sorters performing well on such phenomenological simulated data should be a necessary, but not sufficient, condition to convince experimentalists that they will work well on real data. In previous papers [5, 4], we used MEArec [1], which relies on biophysical modeling of the reconstructed neurons to simulate extracellular potentials to generate ground-truth recordings. However, such biophysical simulations have two shortcomings: i) simulations are very slow and resource-hungry; ii) it is not guaranteed that the superior simulation environment (multicompartment modeling) translates to simulated data that are more similar to experimental data. In fact, the Kilosort4 paper [9] shows that the action potentials generated by MEArec [1] have an almost doubled duration compared to realistic data, which requires further ad-hoc parametrization. We believe this discrepancy can be due to the fact that virtually all multi-compartment models are built from *in vitro* slices, not *in vivo* recordings. Given these limitations, we decided to rely on a simpler but better controllable model for generating templates. Despite not being "biophysical", the generation model can replicate, to some extent, the variability in waveforms (using different parameters for waveforms widths and spatial decays) and allows us to have a ground-truth model for drifting as well, that we can use to assess interpolation errors. Nevertheless, we agree with

the reviewer that some aspects of the simulator can be improved, such as the structure of the noise in the data. We are currently working on the Spikeinterface side to improve the generation module so that it supports temporally correlated noise (spatial correlation is already supported and used in this manuscript).

> *(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.*

We extended the simulated recording to an half an hour duration. However, as it can be seen in the Figures, this does not affect the main results of the paper.

> *(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.*

We believe the that Neuropixels 1.0 geometry is a good choice: the NP1.0 paper it is the most highly cited paper about a high-density electrophysiology probe, suggested that it is currently the most widely used probe in the world. However, the reviewer is correct that there is a risk of overfitting. To demonstrate generalizability of our algorithm, we have added results obtained with NP2.0 layout, a Cambridge NeuroTech layout, a SINAPs probe layout and tetrodes. The results demonstrate that the Lupin sorter is not overfitted.

> *(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 peerreviewed 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.*

The reviewer is rising a valid point, but we think it is beyond the scope of this paper to fully describe every component of every spike sorter mentioned in the paper. We made the deliberate choice to describe the sorters as chains of components in order to demonstrate the flexibility of our modular approach. Full details of every components can be found in the online documentation. In order to ensure the manuscript felt more complete and concrete, we revised the descriptions in our Methods section, in order to give some more details.

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

We are not sure of what exactly is suggested by the reviewer. In order to clarify the situation, we pushed the notebooks and all the code needed to reproduce the Figures of the paper in a Zenodo archive *https://zenodo.org/records/19695406*

> *(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.*

Although we agree that these two sorters might be seen as straw men, we decided to keep them in the paper for various reasons. This has been clarified in the manuscript, but the primary reason is an historical one: the developers of these sorters decided to unite their efforts while designing new tools and algorithms, which led to the initial work on the modular framework described here. Because SpyKING CIRCUS and TriDesClous had to evolve for maintenance, it was decided to try to write a common "grammar" that would allow these two spike sorters to be described in the same framework. The reviewer is right in the fact that once the foundations were stabilized, most of the development efforts were put to Lupin, that was built as the best combination of all the expertise gained on the two aforementioned sorters. A second reason to keep them is that, once again, we decided that Lupin should not be the main focus of the paper. Of course, this is a nice illustration of what the modular framework can do, but we do not want to push it *per se*. What matters most is the methodology, especially since we can not claim, here, that Lupin would be the best spike sorter regardless of data types, probe geometries, ... We extended the paper with other probe geometries (see added Supplementary Figure 1) and real world data (see Figure 8), and we observed that Lupin was on par, and/or slightly better than KiloSort 4 with respect to number of False Positives for examples. But ultimately, the paper is really about the development of a common ecosystem such that all tools can be improved upon, at the community level.

> *(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.*

As suggest by the reviewer, we tried to extend the methods of the clustering and the template matching steps, bearing in mind that some of them have already been published in detail elsewhere. We really want to underline that the central point of the paper is the modularity of the architecture, not so much the low-level details. To populate our framework and demonstrate its generalization, we implemented some key algorithms. But describing with schematics and in depth every individual methods is something that is not even done in papers focused on the spike sorters themselves. Later, the reviewer complains that the paper sounds like a technical report. Delving into more details would only amplify this problem.

> *(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.*

To try and simply the plots and their interpretation, we decided to remove the scatter plots of the individual neurons in all Figures, to really focus on the core trends. We believe that the results, such as the dependence on accuracy as a function of SNR, are difficult to capture with summary statistics, and that the results are best understood by looking at the plots we have made. If we wanted to make strong claims about one algorithm being more suitable for a specific task, then these summary statistics would be suitable. Instead, we are trying to demonstrate the general utility of the components framework, and we optimize Lupin simply to maximize the accuracy of each step.

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

The manuscript has been intensively proofread by native english speakers.

> ***Reviewer #3 (Recommendations for the authors):***
>
> *(1) Lines 14-15: "...a... sorters...": either "...sorter..." or "...a... sorter...".*

This has been corrected

> *(2) Line 30: "peak detection" or "event detection"?*

We prefer the term “peak detection", since this is exactly what the algorithm are looking for: spatio-temporal extrema in the signals

> *(3) Line 30: the purely sequential structure of the modules is very limiting and generally incorrect. Template matching may replace peak detection, as is the case in many real-time hardware implementations. The feedforward process is limiting, and many sorters use feedback or multiple loops. It is unclear whether and how the proposed framework supports such structures.*

The reviewer raises a good point that we did not explain well in the manuscript. Since our framework is modular, with each component independent of the others, a developer has freedom to create “iterative" sorters. E.g. they could loop over a pair of steps until a criteria is met. In fact, this is one of the advantages of creating modular components. The examples we show in the manuscript are sequential feedforward structures, leading to this confusion. We have clarified the point in the text. We show sequential sorters because to our knowledge there are currently no iterative sorters in wide use. Older versions of Kilosort were iterative, but KiloSort4 is not. Modularity also allows us to isolate one component for a specific task. Hence, for a real-time implementation, we can pre-compute templates using the peak-detection and clustering components. Then these steps would be skipped for “online” sorting, which would only use the template matching component. The reviewer states that “template matching may replace peak detection”. Indeed, in Lupin, SC2 and TDC2, template matching does replace peak detection – the initial peak detection is only used to construct templates for downstream matching. We have clarified this point in the text.

> *(4) Lines 62-63: Are all of these sorters supported by the spike Interface framework? Please include a table of which are and which are not. The same for every module.*

All the sorters listed are indeed supported by Spike Interface, and this has been added in the text.

> *(5) Lines 84, 98, and elsewhere: TriDesClous 2 is mentioned. What about TriDesClous - is there such a sorter, and if yes, what is the scientific reference?*

TriDesClous (*https://github.com/tridesclous/tridesclous*) is a spike sorting pipeline that has not been properly published with a DOI, but that has been developed by the first author of the manuscript and has been used by many papers [8].

> *(6) Line 84: Lupin - suggest reorganizing the manuscript around this sorter and evaluating it on multiple datasets.*

The paper is not about Lupin, and we tried to rewrite the manuscript in order to make this point more explicit. The paper is intended to be a proof of concept of the benefits that can be obtained thanks to a modular approach. This is why we do not want to reorganize the paper on Lupin itself.

> *(7) Line 104, Figure 1, and elsewhere: What is the advantage of evaluating three sorters, if two are predicted to be worse than the third/state of the art? Suggest to reorganize the MS: (a) describe a proper simulator; (b) describe every individual modules: mention the existing algorithms and elaborate + demonstrate the new algorithms; (c) evaluate every*

> *individual module - on properly realistic dataset; (d) evaluate existing complete spike sorters + the proposed best combination - on the same ground truth dataset; (e) compare the best two sorters on multiple datasets. In addition, may identify the WEAKEST link in each sorter and demonstrate the improvement by replacing ("upgrading") that link alone.*

As clarified in the introduction and in the “End-to-end evaluation..." section, we decided to keep three sorters for various reasons. The first is historical: SpyKING CIRCUS 2 and TridesClous 2 were the first two sorters that motivated the creation of the sortingcomponents framework. Because both authors realized that they had so much code in common, they decided to unite their efforts while rewriting them and share some common building blocks. The second reason is that the paper is not about Lupin, *per se*, but more about the general philosophy of the modular architecture presented here. We want to push forward the idea, in the community, that we should share tools, ideas and algorithms in order to enhance the analysis pipelines. Keeping several sorters, even if sub-optimal, is a way to showcase the flexibility of the framework, and this is why we did not re-organize the manuscript as suggested by the reviewer.

> *(8) Line 121: "can drastically cut the time.." - provide quantitative support. In general, avoid superlatives and unsupported statements.*

Spikeinterface has been primarily design to ease the comparison between spike sorting pipelines [2]. Thus all comparison metrics such as agreement matrices, false positives, false negatives, ... are available out of the box when using these Benchmark objects. It would be hard to provide a quantitative support for such a speedup since it will depend on the algorithm, but it allows developer to simply focus on the core implementation while benefiting for free of the whole ecosystem that will launch benchmarks and compare it, with appropriate metrics validated by a large community. Since this validation process, on its own, can be quite complex depending on the processing step, we truly believe the development gain is important, despite the fact that it might be hard to quantify. However, we rewrote the sentence in order to avoid superlatives.

> *(9) Line 129: "powerful and fast way to generate artificial" - again, avoid statements with superlatives and lacking quantitative support.*

We rewrote the sentence to avoid superlatives.

> *(10) Line 129: "powerful and fast way to generate artificial" - the assumptions made in constructing the artificial data critically and strongly affect the conclusions of the benchmark processes. For instance, it is well known that about 10% of the spikes in the cortex have a positive extrema, but the generator is limited to negative spikes. Also, the generator produces spatially-displaced and scaled versions of the waveform generated by the putative soma, but extracellular waveforms almost never behave that way. Even if the simulator is improved, it will always remain a simulator, and therefore the caveat should always be kept in mind.*

The reviewer is right: our simulator has some limitations compared to biological data. But the fact is that, even on synthetic data, there is still plenty of room for improvements of current spike sorting pipelines before even getting to real data, where ground truth are unknown. We believe that such ground truth simulator is a necessary, but not sufficient, condition to validate sorting algorithms. Other options such as hybrid recordings would also suffer from the same flaws, and might be even more questionable. In the revised version of the manuscript, we also added tests on real data to compare more qualitatively Lupin and KiloSort. The fact that results are in line with what is observed on synthetic data gives us confidence in our observations.

> *(11) The authors should add a limitations section to the MS.*

Some limits have been more extensively discussed in the Discussion of the paper, with respect to the feedforward architecture, the validity of the ground truth data.

> *(12) Line 129: Can the benchmark object be used with other data (e.g., existing)? The methods indicate that this is the case - please demonstrate.*

We think that a proper demonstration would be out of the scope of this paper, but indeed, as long as the user can provide a recording alongside with a sorting (exhaustive or not), then the Benchmark objects can be used. This allows the use of hybrid recordings, and/or manually curated datasets where users would be able to provide a ground truth. Of course, depending if the ground truth is exhaustive or not (if one knows the activity of all the neurons in the recordings), the metrics might not be the same. But everything is built-in in the object.

> *(13) Line 144: 10 minutes is too short and nearly irrelevant. In particular, many units spike at rates much lower than 0.1 spikes/s, and in 10 minutes would emit less than a score of spikes (e.g., 0.01 spikes/s would accumulate, on average, 6 spikes..).*

We regenerated all the figures in the paper with 30 min long recordings, and the results remain similar to the 10 minute recordings.

> *(14) Line 151: firing rates are not independent of the waveform as implicitly assumed; this should be accounted for, at least in the options in the simulator. Same for library waveforms - the firing rates should be a parameter that is optionally provided along with the waveform library.*

We are not sure what is meant by the reviewer. We believe that the point is that some cell types, with particular waveforms, have particular firing rates, such as fast-spiking interneurons. This could be dealt with in the current simulator, since users can provide, on a per cell basis, some particular values to generate the waveforms of the neurons. One could clearly imagine having some particular cell types with dedicated waveforms and firing rate parameters. However, for the sake of simplicity in the paper, we chose not to use such granularity. What the simulator can not do, at the moment, is to perform amplitude modulation of the templates as function of bursts for example. But this could easily be implemented, and should be part of future works to consolidate the generator.

> *(15) Figure 2A, D: It is unclear why the specific zig-zag motion was simulated. Please rationalize, or use a motion from a real dataset. In particular, simulate (a) breathing-induced micro-motions, (b) gradual drift, and/or (c) a step jump.*

We used a zig-zag plus a Brownian motion to cover a rather broad range of continuous motion. Of course, as pointed out by the reviewer, the heterogeneity of real drifts is large, and again, we believe it is out of the scope of the manuscript to cover them all. In previous works, we explored how motion correction methods were working as function of the drifts [5], and the conclusion was that this simple continuous drift was already challenging enough to make the spike sorters fail. Adding discontinuities, as often encountered in experiment, would only make things worse. Finally, we added real world data (Figure 8) from three randomly picked dataset with heterogeneous probe geometries. They all come with some drift, that might be representative of what is typically dealt with.

> *(16) Line 181: "more computationally demanding" - quantification?*

We forgot to add the reference to panel 3D, and this has been corrected in the manuscript.

> *(17) Line 192: "(Figure 3A" - add ")"*

This has been corrected

> *(18) Lines 195-6: "... not that missing a few spikes at peak detection will not have a large impact..." - this statement should be quantified.*

We reformulated the sentence, but the point here is simply that template-matching based algorithms use the template-matching step exactly for this purpose: to label spikes that would have been ignored/missed by the peak detection and clustering steps. The fact that template-matching based pipelines such as KiloSort, SpyKING-CIRCUS, ... outperforms clustering-based solution when detecting spikes [4] is a clear support for our sentence.

> *(19) Lines 195-6: "... not that missing a few spikes at peak detection will not have a large impact..." - this statement, if correct, exposes a key weakness of the purely modular approach. For instance, assume that module 2.1 performance can be 0.5 and module 2.2 performance is 0.9, and that will have zero impact on the overall performance of a sorter that has module 4.1 as its fourth module, yielding an overall performance of 0.7. But when module 4.2 is used, module 2.1 performance of 0.5 results in an overall performance of 0.5, whereas module 2.2 performance of 0.9 translates to an overall performance of 0.9. The point should be clear now: evaluating every module in isolation cannot fully predict the behavior of the full system - even if a purely feedforward, single iteration (no loop) architecture is assumed. This requires algorithmic support in the proposed framework, and at the very least explicit discussion.*

The reviewer is right, benchmarking every module in isolation cannot fully predict the behavior of the full system. However, the fact that Lupin, built as an optimal combination of the components and can be *on par*, if not better in some situations, than KiloSort 4 on various artificial and real dataset makes a compelling argument in favor of assembling the best algorithmic pieces one after the other. In fully assembled spike sorting pipelines, the failures at one stage might be compensated by some algorithmic optimizations latter on. But still, being able to identify these failures, and eventually correct them at the appropriate level, i.e. as soon as they appear might be beneficial for the development of the tool, and to ease their readability/maintenance. This has been added in the discussion.

> *(20) Line 203: "we hope that future efforts": This statement is strange for two reasons: (a) if the authors hope for it, why not simply do it? (b) if the authors cannot do it / it is out of scope, the proper place to mention their hopes is in the Discussion section.*

This has been moved to the Discussion section

> *(21) Line 232: "motion is only partially compensated for" - so what is the utility of the compensation? This becomes clear later, but the statement is obscure at this point - please clarify.*

This has been clarified.

> *(22) Figure 4A, right: The fact that the lines do not reach unity even for high FRs suggests that the FRs are NOT the key limiting factor. Please provide a scalar measure and a 2D analysis of the accuracy as a function of FRs and SNRs, separately for static and motion-corrected.*

We are not sure that we understand what is being requested by the reviewer. A scalar measure with a 2D analysis of the accuracy as a function of FRs and SNRs would mean, per case (static and motion corrected) at least 4 panels, thus a total of 8 panels. This seems like an overly dense figures. Further, a scalar measure is unlikely to provide more insight into the analysis.

> *(23) Line 275: "kriging method" - describe.*

For a full description we refer readers to the original paper describing the method [9] and other work on motion correction [5]. We have added a brief description of the method in the text in the "Motion interpolation reduces spike sorting performance" section.

> *(24) Lines 309-310: Where are the templates from in this case - true or estimated? If the latter, say it.*

This has been clarified.

> *(25) Line 317: This is the point where I almost gave up - the MS up to this stage seemed very much like a technical report and is not organized in a clear, results-oriented manner. Even for a Methods-oriented paper, one expects to learn the key results, but those are lost in this MS.*

While we agree that this part of the manuscript is rather technical, this is something that we believe is at the core of some scientific questions that have not been yet properly addressed by most of the spike sorting pipelines. The idea of performing template-matching, i.e. to seek for spatio-temporal pattern in the signals while at the same time compensating only partially the motion has never been properly benchmarks. While template-matching is clearly a game-changer in static recordings, i.e. without motion, the extra-addition of motion might limit its performances. We firmly believe that our modular approach can be used to isolate such core questions from the whole integrated pipelines, and guide both users and developers.

> *(26) Figure 6A, second and third panels: The fourth (rightmost) panel shows that there are differences between the "true" and "interpolated" templates, but this cannot be seen in the two middle panels - probably because too much information is overloaded on every channel. My suggestion is to show a reduced number of channels, but for each of those, show all five waveforms (corresponding to different spatial positions of the centers of mass) in a NON-overlapping display.*

The figure has been redrawn, as requested by the reviewer.

> *(27) Figure 6 and the entire issue of the failure in motion correction + lines 331-332: It is not 100% convincing that the failure is not at the DREDGE level - please demonstrate that the motion is estimated perfectly.*

We checked that the DREDGE algorithm is correctly estimating the motion. To convince the reader this is the case, this has now been shown in Figure 2, panel D, where the simulated motion is displayed on top of the motion estimated by DREDGE. As shown, both are very similar.

> *(28) Figure 6 and the entire issue of the failure in motion correction: If motion is estimated perfectly and interpolation is done properly, then is the failure an outcome of discrete spatial sampling? In other words, if the movement in space over time (in the simulator) were limited to discrete steps that correspond exactly to the positions of the electrodes, would the motion correction allow perfect performance? Stated differently: if the electrodes were not 15 or 20 micro-meters apart but rather 1 micro-meter apart, would the difference (e.g, between Figure 4A/4B, or between Figure 5A and Fig. 5B) be reduced? Please check and report.*

The reviewer is raising an interesting point, and indeed, this is something that we are planning to investigate. We felt it would have been too technical for the scope of this paper, which is aimed at showcasing the modular framework and its possibilities. But we are planning to explore such failures with ultra-dense probes as has been done in the DREDGE

paper [10]. We also have the intuition that errors are originating from failures of discrete spatial sampling: hence the larger the sampling, the more pronounced the errors.

> *(29) Figure 6B: It is impossible to discern which line is which - this underscores the general point of summarizing every CDF by a scalar with measures of dispersion (i.e., descriptive statistics) + statistical testing (i.e., quantitative statistics).*

While we agree with the reviewer that the plots are dense, they convey the global message of the paper and are the same that were used in [9]. To ease the comparison, we decided to stick to this representation.

> *(30) Figure 6D: add error bars.*

There are no error bars, since there was only a single run in this figure, i.e. on a single dataset.

> *(31) Line 318, line 328: So what is the result here - that the interpolation idea is not useful? If yes, what is the source of the failure? Is it discrete/poor spatial sampling as suggested above? Something else?*

The result is that interpolation, rather than motion estimation, is the bottleneck in recoding with motion. A careful analysis using data from dense probes, as already stated above, would be the focus of further work. But we suspect that errors results mostly from bad interpolation due to discrete spatial sampling.

> *(32) Line 334: up to this point, there are no quantified results - and actually, no clear results at all.*

We rewrote the section accordingly, to make the key observations more striking. The goal of this section is not to propose quantified results and say which interpolation methods is the best (a full paper should be devoted to this question), but rather to show the possibilities offered by the modular framework, looking at questions that have been not yet properly addressed in spike sorting algorithms.

> *(33) Line 345: "This is likely due" - provide support? Example?*

We rephrased the sentence accordingly.

> *(34) Line 349: "this is mostly because both clustering..." - but feature detection was evaluated together with clustering, so a conclusion specifically about clustering is unwarranted. This is a general point - can the proposed software/modular framework evaluate feature extraction separately from clustering? If yes, please demonstrate.*

The reviewer is right about the fact that feature extraction was evaluated together with clustering, however, we want to stress that it is exactly the same method used by all the clustering algorithms developed in the paper. Thus, even if the feature detection method might not be optimal, it is not introducing any biases. Currently, almost all spike sorting algorithms these days are using PCA (or truncated SVD) as a feature detection method, on nearby channels where peaks are detected, and this is why we decided to only focus on this method.

> *(35) Lines 358-374: If SpyKING-CIRCUS 2 and TriDesClous do not provide any advantage, why include them in the MS? Many other sorters could be included as well. In other words, what do we LEARN from the failure of these sorters to compete with KS4 and Lupin? If nothing, please remove them. If something, please state the conclusions clearly.*

Both these software have advantages, but we decided to keep them as they offer direct illustrations than implementation of fully integrated pipelines, other than Lupin, are possible

within the proposed framework. TriDesClous2 is fast, and SpyKingCircus2 (in line with Spyking Circus [11]) is mostly tailored for *in-vitro* data, and thus can scale for more than thousands of channels while some other algorithms might not [9]. Of course, listing all the pros and cons of each software would be impossible within the scope of the paper, but we decided to keep them for illustrative purpose.

> *(36) Line 369: "the main advantages of these sorters lie in their modularity" - modularity is an advantage only if it is useful for something - if the sorters are modular but perform poorly, the point is nullified.*

We agree with the reviewer, but we would like to point out that here, none of the modular sorters shown in the paper are performing poorly, and all have pros and cons as said in the previous point.

We believe that this is important to show that modularity can offer some options both for users and developers with respect to probe geometries, animal species, data types, ...

> *(37) Line 371: "on our dataset" - this is a very important limitation. See general comments, and discuss in the Limitations section of the Discussion.*

We extended the paper by adding more datasets, for different probe layouts, and also real datasets with meta comparison of state of the art spike sorters (see Supplementary Figure S1).

> *(38) Figure 7C: the difference between the cyan (KS-like) and the orange (KS4) lines makes the KS-like irrelevant. Please improve or remove.*

The reviewer is right, and we removed the KS-like pipeline, since it at the stage of the manuscript it is not yet exactly like KiloSort 4.

> *(39) Line 379: "each individual algorithmic steps" - should be "step".*

This has been corrected

> *(40) Line 380: Is Lupin available for download and usage as a separate, standalone package - i.e., not as part of the spike interface framework? If not, please make it available. And if yes, indicate this clearly.*

Lupin is part of the SpikeInterface project, and thus can not be installed in a standalone mode, outside of the SpikeInterface framework

> *(41) Line 385: "our... gigantic...effort" - avoid superlatives, especially to self.*

This has been removed

References

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

(2) A. P. Buccino, C. L. Hurwitz, S. Garcia, J. Magland, J. H. Siegle, R. Hurwitz, and M. H. Hennig. Spikeinterface, a unified framework for spike sorting. Elife, 9:e61834, 2020.

(3) J. M. J. Fabre, E. H. v. Beest, A. J. Peters, M. Carandini, and K. D. Harris. Bombcell: automated curation and cell classification of spike-sorted electrophysiology data.

(4) S. Garcia, A. P. Buccino, and P. Yger. How do spike collisions affect spike sorting performance? Eneuro, 9(5), 2022.

(5) S. Garcia, C. Windolf, J. Boussard, B. Dichter, A. P. Buccino, and P. Yger. A Modular Implementation to Handle and Benchmark Drift Correction for High-Density Extracellular

Recordings. eNeuro, 11(2):ENEURO.0229–23.2023, Feb. 2024.

(6) A. Jain, R. Greene, C. Halcrow, J. A. Swann, A. Kleinjohann, F. Spurio, S. Graff, A. Pan-Vazquez, B. Kampa, J. Gall, S. Grün, O. Winter, A. Buccino, M. H. Hennig, and S. Musall. UnitRefine: A community toolbox for automated spike sorting curation.

(7) S. Koukuntla, T. DeWeese, A. Cheng, R. Mildren, A. Lawrence, A. R. Graves, K. E. Cullen, J. Colonell, T. D. Harris, and A. S. Charles. SLAy-ing oversplitting errors in high-density electrophysiology spike sorting. bioRxiv: The Preprint Server for Biology, page 2025.06.20.660590, 2025.

(8) J. Magland, J. J. Jun, E. Lovero, A. J. Morley, C. L. Hurwitz, A. P. Buccino, S. Garcia, and A. H. Barnett. Spikeforest, reproducible web-facing ground-truth validation of automated neural spike sorters. Elife, 9:e55167, 2020.

(9) M. Pachitariu, S. Sridhar, J. Pennington, and C. Stringer. Spike sorting with Kilosort4. Nature Methods, 21(5):914–921, May 2024. Publisher: Nature Publishing Group.

(10) C. Windolf, H. Yu, A. C. Paulk, D. Meszéna, W. Muñoz, J. Boussard, R. Hardstone, I. Caprara, M. Jamali, Y. Kfir, D. Xu, J. E. Chung, K. K. Sellers, Z. Ye, J. Shaker, A. Lebedeva, R. T. Raghavan, E. Trautmann, M. Melin, J. Couto, S. Garcia, B. Coughlin, M. Elmaleh, D. Christianson, J. D. W. Greenlee, C. Horváth, R. Fiáth, I. Ulbert, M. A. Long, J. A. Movshon, M. N. Shadlen, M. M. Churchland, A. K. Churchland, N. A. Steinmetz, E. F. Chang, J. S. Schweitzer, Z. M. Williams, S. S. Cash, L. Paninski, and E. Varol. DREDge: robust motion correction for high-density extracellular recordings across species. Nature Methods, 22(4):788–800, Apr. 2025.

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

https://doi.org/10.7554/eLife.110588.2.sa0