Spyglass: a framework for reproducible and shareable neuroscience research

Kyu Hyun Lee[1,2,3,♦], Eric L. Denovellis[1,2,3,♦], Ryan Ly[4], Jeremy Magland[5], Jeff Soules[5], Alison E. Comrie[1,3], Daniel P. Gramling[6], Jennifer A. Guidera[1,3,7,8], Rhino Nevers[1,3], Philip Adenekan[1,3], Chris Brozdowski[1,3], Samuel R. Bray[1,3], Emily Monroe[1], Ji Hyun Bak[1], Michael E. Coulter[1,3], Xulu Sun[1,2,3], Emrey Broyles[1,3], Donghoon Shin[1,3,7], Sharon Chiang[9], Cristofer Holobetz[10], Andrew Tritt[4], Oliver Rübel[4], Thinh Nguyen[11], Dimitri Yatsenko[11], Joshua Chu[12], Caleb Kemere[12], Samuel Garcia[13], Alessio Buccino[14], Loren M. Frank[1,2,3,*]

[1]Department of Physiology, University of California, San Francisco
[2]Howard Hughes Medical Institute, University of California, San Francisco
[3]Kavli Institute for Fundamental Neuroscience, University of California, San Francisco
[4]Scientific Data Division, Lawrence Berkeley National Laboratory
[5]Center for Computational Mathematics, Flatiron Institute
[6]Graudate Program in Neural and Behavioral Sciences, University of Tübingen
[7]UCSF-UC Berkeley Graduate Program in Bioengineering, University of California, San Francisco
[8]Medical Scientist Training Program, University of California, San Francisco
[9]Department of Neurology, University of California, San Francisco
[10]Sainsbury Wellcome Centre, University College London
[11]DataJoint
[12]Department of Electrical and Computer Engineering, Rice University
[13]Centre de Recherche en Neuroscience de Lyon, CNRS
[14]Allen Institute for Brain Science

♦Equal contribution
*Corresponding author: loren.frank@ucsf.edu

Abstract

Scientific progress depends on reliable and reproducible results. Progress can also be accelerated when data are shared and re-analyzed to address new questions. Current approaches to storing and analyzing neural data typically involve bespoke formats and software that make replication, as well as the subsequent reuse of data, difficult if not impossible. To address these challenges, we created Spyglass, an open-source software framework that enables reproducible analyses and sharing of data and both intermediate and final results within and across labs. Spyglass uses the Neurodata Without Borders (NWB) standard and includes pipelines for several core analyses in neuroscience, including spectral filtering, spike sorting, pose tracking, and neural decoding. It can be easily extended to apply both existing and newly developed pipelines to datasets from multiple sources. We demonstrate these features in the context of a cross-laboratory replication by applying advanced state space decoding algorithms to publicly available data.

New users can try out Spyglass on a Jupyter Hub hosted by HHMI and 2i2c: https://spyglass.hhmi.2i2c.cloud/.

Introduction

A central goal of neuroscience is to understand how the structure and dynamics of neural activity relate to the internal states of the organism and the external world. This understanding is derived from the analysis of complex, multi-modal datasets. While the community has significantly improved tools and algorithms for data collection and analysis[1–6], extracting consistent and reproducible insights from data remains a complex and time-consuming task.

The difficulty stems from how the scientific community analyzes and synthesizes data. Often, researchers take years to collect and organize data, which is then transformed through complicated analysis. Analyses often begin with preprocessing steps that extract specific signals from the data, followed by a series of custom-written scripts to further examine and quantify them. The results from multiple experiments are then synthesized into coherent findings, and when these are shown to be consistent upon limited replication, they are reported in scientific literature—with the data and analysis scripts documented to varying degrees.

Despite this traditional approach's potential to yield reliable and reproducible results, there are significant implementation issues. Experimental protocols can be well defined, but analyses usually are not. Raw data are seldom shared, critical metadata are often unavailable, and the full set of analysis steps (including relevant parameters) is typically missing. Essential tasks, including manual curation of clustered spikes and artifact rejection, are often hidden or irretrievable from the written record. Efforts to reproduce findings are hampered by idiosyncratic data and code organization, poor documentation, and obscured vital details, including hardware requirements[7]. In collaborations among multiple scientists, these problems can be further exacerbated due to the variability in how each participant carries out analysis. Consequently, the full validation of a result usually requires repeating the experiment and reconstructing the analysis.

These problems also hinder the reusability of data and code. A new trainee might struggle to analyze existing data due to issues with understanding critical details. A scientist who downloads the data from a previous study may find that the analyses they wanted to carry out are impossible because the raw data are not available. Alternatively, the raw data may be available, but the scientist may need processed data (e.g. sorted spikes) that are not included. Similarly, shared code might not be standardized or documented, causing multiple teams to duplicate efforts and implement the same tools. In addition, visualizations that facilitate exploring the data may be difficult to generate and share with others.

A system that could address these challenges should therefore enable:

- recording of raw data with sufficient metadata required for analysis and reuse;
- sharing of data and all intermediate analysis results in an accessible form;
- reproducible analysis via well-documented, organized, and searchable pipelines;
- generation of shareable visualizations to facilitate communication and collaboration;
- easy use by scientists with minimal formal training in data management.

Achieving these goals would represent a major step towards meeting the FAIR guiding principles for findable, accessible, interoperable, and reusable[8] data and analysis pipelines[9]. For example, it would become possible to easily find publicly available data, analyze it with a standardized pipeline that keeps track of all the parameters, and generate a visualization to share the results over the web—a stark contrast to how science is practiced today.

In pursuit of this vision, many organizations, such as the Allen Institute for Brain Science (AIBS), Johns Hopkins Applied Physics Lab (APL), and the International Brain Laboratory (IBL), have made strides by standardizing and sharing data and analysis[10–12]. However, these efforts have not fully resolved the issues related to data sharing and reproducible analysis. For instance, the raw data is not always shared, and when they are, they may not be in a standardized format accessible to the broader community. In addition, they do not disclose every step of the processing pipeline (e.g. the criteria used for manual or automatic curation of spike sorting), which can significantly affect the results[13]. Some of the technologies used, such as cloud computing services (e.g., CodeOcean) and sophisticated databases and APIs[11,14,15], can be cost prohibitive or require specialized software engineering expertise that is beyond the reach of most labs. Furthermore, many of these existing efforts are focused on the needs of specific projects, data types, and behavioral paradigms, limiting their scope. Thus, while these efforts mark important advances, there remains a need for user-friendly, integrated solutions that can be more widely adopted across individual labs and groups in the neuroscience community.

To address this need, we developed Spyglass, an open-source neuroscience data management and analysis framework written in Python. Spyglass leverages widely available community-developed tools and embraces the Neurodata Without Borders (NWB) standardized format[16,17]. It uses DataJoint[6,18] to manage reproducible analysis pipelines with a relational database and incorporates novel software tools (Kachery and Figurl) for sharing data and web-based visualizations to enable collaboration within and across labs. It is Python-based and uses standard data types, and can thus include pipelines that use a wide array of analysis packages including SpikeInterface[1], GhostiPy[19], DeepLabCut[3], and Pynapple[20]. Spyglass also offers ready-to-use pipelines for analyzing behavior and electrophysiological data, including spectral analysis of local field potential (LFP), spike sorting, video processing to extract position, and decoding neural data. In addition to the extensive documentation and tutorials, new users can try out a demo version of Spyglass hosted on the web by HHMI and 2i2c as a Jupyter Hub instance. Here we describe the structure of Spyglass and demonstrate its potential by applying the same analysis pipelines to NWB files from different labs and comparing the results.

Results

Overview of Spyglass

Spyglass is an open-source software framework for reproducible analysis of neuroscience data and sharing of the results with collaborators and the broader community. Analyzing data with Spyglass begins with the raw data and experimental metadata stored in NWB, a format that meets the data management standards of the BRAIN Initiative[16,21]. Spyglass then ingests these NWB files into a relational database using DataJoint, ensuring reproducible analyses and parameter caching. Finally, Spyglass integrates with tools to share results and interactive data visualizations over the web such as Kachery and Figurl (Figure 1). In the following sections, we provide detailed descriptions of these components. We also present a use case demonstrating how Spyglass can apply complex analysis to a publicly available NWB file to enable comparison of results across multiple laboratories.

Data Format

A typical neuroscience experiment consists of multiple data streams stored in different formats. Managing such heterogeneous data in a shareable and accessible manner is challenging. A practical solution is to save the data in a community-supported format like NWB, which is emerging as a standard for neurophysiology and behavior data[16,21]. We have chosen NWB as the data specification in Spyglass for the following reasons:

- The versatility of NWB accommodates various data types and allows metadata to be saved with the data in a single self-annotated file.
- Converting raw data to NWB makes it immediately shareable in an accessible format.
- NWB is supported by public data archives like DANDI[22].
- All subsequent data processing steps are applied to the NWB file, ensuring complete reproducibility of subsequent analyses.
- Tools developed for NWB files are immediately accessible to users.

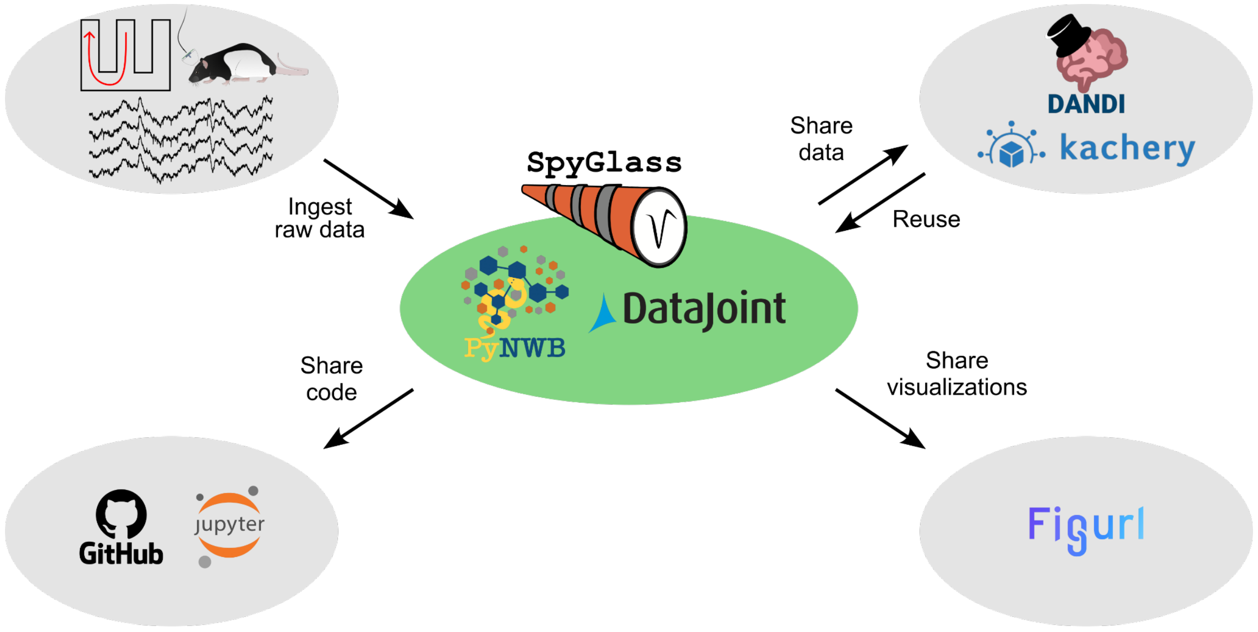


Figure 1: Overview of Spyglass. The raw data consisting of information about the animal, the behavioral task, the neurophysiological data, etc. is converted to the NWB format and ingested into Spyglass pipelines via DataJoint. The raw and processed data are shared with the community by depositing them to public archives like DANDI or shared with collaborators via Kachery. Code is shared by hosting the codebase for Spyglass and project-specific pipelines on online repositories like GitHub, as well as the Jupyter Notebooks for recreating the state of the database. Finally, visualizations of key analysis steps are shared over the web via Figurl.

Conversion to NWB can be done using software tools developed by the community (see Methods for recommendations).

NWB provides a community standard for neurophysiology data and has a list of best practices, but it also allows some flexibility in the specification of data to accommodate a broad range of experiments and lab-specific requirements. As a result, naming conventions for some types of data can differ across labs. We have therefore developed a system that makes it possible to ingest NWB files into Spyglass even when they do not adhere to our naming conventions or best practices (see Methods and Table 1, *04_PopulateConfigFile*).

Often each NWB file contains a single day of experimental data from one animal. This includes not only the electrophysiological voltage traces from recording devices but also other details about the implantation surgery (e.g. brain region), behavioral tasks (e.g. periods of foraging vs. resting), and the data resulting from the animal's interaction with the environment (e.g. digital inputs and outputs that indicate times of beam breaks, reward delivery, or optogenetic manipulations). For data not efficiently stored in NWB (e.g. video recordings of the animal's behavior), a link to the external file can be included within the NWB file.

NWB permits storage of many types of data, and in Spyglass we store virtually all intermediate results from downstream analysis pipelines in NWB (see Data Ingestion and Analysis Pipelines section). This approach ensures that all data associated with the analysis, including intermediate results, can be read using the same software tools and can be easily shared.

System Design

One significant challenge with data analysis lies in managing its complexity. Virtually every analysis involves an extended series of steps, including “preprocessing” (spike sorting for electrophysiological data, region-of-interest identification for optical physiological data, video processing for behavioral data, etc.) as well as various downstream analyses. Each step depends on a different algorithm and a specific set of parameters for that algorithm. Each step also generates distinct intermediate data. Tracking these numerous components is challenging, and understanding how another investigator has managed them can be even more daunting. This complexity hinders collaboration and data reuse.

These issues motivated our use of a formal software system that associates each step of the analysis with one or more tables[1] and paired processing methods. Starting from NWB files containing the raw data, we apply and track every transformation to the data using DataJoint, an open-source Python package for managing relational databases with a focus on scientific computing pipelines and data integrity[6,18]. DataJoint makes it possible to develop a standardized and searchable structure to organize and store each step of analysis. It is also concurrently accessible by multiple users. As a result, the entire provenance of a particular analysis can be easily retrieved and understood. DataJoint also makes it easy to apply the same pipeline to many datasets, greatly enhancing efficiency.

Our DataJoint pipelines consist of hierarchically organized tables in a relational database that contains information about the data, metadata, and analysis parameters (Figure 2). Users initiate the pipelines by entering new entries into these tables, and the results of analysis are saved as entries in downstream tables. This style of data analysis offers several advantages:

- Within Spyglass, the code for running the computation is intrinsically associated with the table that will store the result. This allows the users to specify the data and parameters for computation ("what") rather than the execution details ("how"), simplifying the process.
- It naturally organizes the analysis parameters, data, and outputs into different tables.
- It enables easy access to multiple datasets via queries.

DataJoint also provides additional features that are useful for reproducible data analysis including maintenance of data integrity based on the dependency structure of the pipelines (e.g. deleting a table entry causes cascading deletion of dependent entries in downstream tables).

Data Ingestion and Analysis Pipelines

To begin analysis, a user first ingests one or more NWB files into the database. Here some flexibility is required, as the NWB format allows files to contain different types of information and for users to use different naming conventions for the same types of data. We therefore allow users to provide mappings from NWB files to Spyglass naming conventions and any missing data (e.g. the layout of recording probes used in a study) in the form of configuration files (Table 1, *04_PopulateConfigFile*).

[1] A structure composed of rows and columns that organize and store data.

During ingestion, information from the NWB file is automatically extracted and stored in tables of the `Common` module. Each `Common` table corresponds to a data object in the NWB file and serves as an interface to retrieve it with simple function calls (`fetch_nwb`). Because these tables only store pointers to the data objects, they allow for "lazy loading" (i.e., loading a specific part of the data only when used, instead of the entire NWB file at the beginning of analysis).

Analysis pipelines then build upon these tables, and each step in the analysis consists of populating one of four table types (Figure 2A):

- `Data` tables contain pointers to data objects in either the original NWB file or ones generated by another upstream analysis.
- `Parameter` tables contain a list of the parameters needed to fully specify the desired analysis.
- `Selection` tables associate parameter entries with data object entries, making it easy to create different data/parameter pairs and analyze the same data using multiple parameters sets.
- `Compute` tables execute computations and store results, often creating new data that are stored in the NWB format. This new NWB file includes only the critical metadata from the original NWB file.

Spyglass includes pipelines for standard analysis tasks in systems neuroscience, such as analysis of LFP, spike sorting, video and position processing, and fitting state-space models for decoding neural data. Tutorials for all pipelines are available on the Spyglass documentation website (Table 1). Many pipelines are powered by community-developed, open-source packages, like GhostiPy[19], SpikeInterface[1] and DeepLabCut[3]. These pipelines store a complete record of the analysis and simplify the application of these tools. Furthermore, multiple versions of the pipelines can co-exist to apply different algorithms to a single data set, enhancing the robustness of results (see *Merge motif* below).

Critically, users can create custom pipelines that start from the results of these upstream analyses (see Methods). These custom pipelines can also take advantage of the fact that analyzed results are stored in the NWB format, enabling the use of other analysis software packages within the NWB ecosystem. Whether or not a user decides to use those packages, this structure makes it possible to implement the entire set of analyses and results that constitute a scientific finding within the same framework.

Example Pipelines

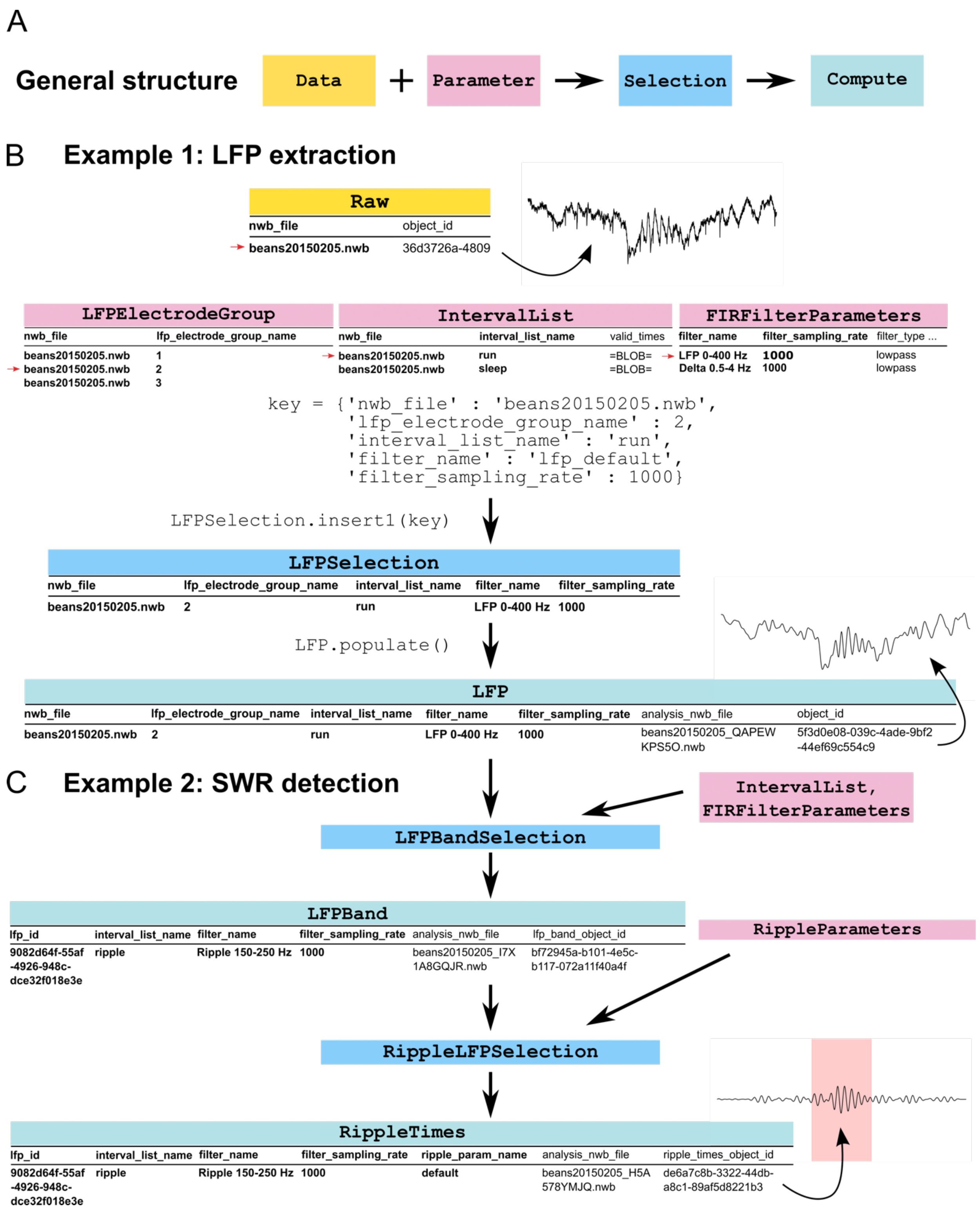
To illustrate how analysis pipelines in Spyglass are organized, we first describe two simple examples: (i) filtering broadband extracellular voltage traces to extract the lower-frequency LFP bands; and (ii) the detection of discrete events (e.g. sharp-wave ripples, a hippocampal event marking the time of bursts of population activity) in the LFP signals. We then present a more complex example: spike sorting and curation.

Example 1: LFP extraction (Figure 2B)

To extract the LFP signal (below 400 Hz), we use the pipeline illustrated in Figure 2B. First, we select the `Data`, in this case a `Raw` object that points to an `ElectricalSeries` in the NWB file. We then specify the `Parameters`: the list of channels for which LFP should be extracted (`LFPElectrodeGroup`), the time interval for the LFP extraction (`IntervalList`; see below for addition details), and the coefficients for the filter that will be used on the data (`FIRFilterParameters`). These parameters are linked to the `Data` by defining a Python dictionary object and inserting it into a `Selection` table (`LFPSelection`) (Figure 2B). Finally, we apply the filter to the selected data over the selected interval in the `Compute` table (`LFP`) by calling the `LFP.populate` method. The resulting filtered data is saved to disk in the NWB format, and the object ID associated with the LFP object within the NWB file is also stored in the table for easy retrieval. The corresponding entry in the `LFP` table contains all the details about the data and the parameters, allowing a user to fully track the provenance of the data. In addition, a single function call to the `fetch_nwb` method enables retrieval of the data object and access to the data.

Example 2: Sharp-wave ripple detection (Figure 2C)

Once the LFP extraction is completed, we can build on the results by applying another filter to isolate a specific frequency band and identifying sharp-wave ripples (SWRs), a prominent LFP event within hippocampal data. This pipeline is illustrated in Figure 2C. It applies two additional steps to a row in the `LFP` table: another band-pass filter to isolate the 150-250 Hz band and a subsequent detection of SWR events. Each steps uses the same basic scheme shown in Figure 2A. These include defining a specific band-pass filter in the `Parameter` tables; selecting a time interval for the bandpass filtering; and adding an entry to `LFPBandSelection` table that associates both the filter parameters and the time interval with a row in the `LFP` table. A call to `LFPBand.populate` generates an NWB file containing the ripple-band data and an entry in the `LFPBand` table with information about which data and parameters were used. Next, the user selects an entry in `RippleParameters` to define the parameters for detecting the ripple events (e.g. threshold) and associates it with filtered data in `LFPBand` in the `RippleLFPSelection` table. Finally, the `RippleTimes` table is populated (`RippleTimes.populate)`, which identifies the start and end times of each ripple event and saves these to a new NWB file.



Raw

nwb_file	object_id
beans20150205.nwb	36d3726a-4809

LFPElectrodeGroup

nwb_file	lfp_electrode_group_name
beans20150205.nwb	1
beans20150205.nwb	2
beans20150205.nwb	3

IntervalList

nwb_file	interval_list_name	valid_times
beans20150205.nwb	run	=BLOB=
beans20150205.nwb	sleep	=BLOB=

FIRFilterParameters

filter_name	filter_sampling_rate	filter_type ...
LFP 0-400 Hz	1000	lowpass
Delta 0.5-4 Hz	1000	lowpass

```
key = {'nwb_file' : 'beans20150205.nwb',
 'lfp_electrode_group_name' : 2,
 'interval_list_name' : 'run',
 'filter_name' : 'lfp_default',
 'filter_sampling_rate' : 1000}
```

```
LFPSelection.insert1(key)
```

LFPSelection

nwb_file	lfp_electrode_group_name	interval_list_name	filter_name	filter_sampling_rate
beans20150205.nwb	2	run	LFP 0-400 Hz	1000

```
LFP.populate()
```

LFP

nwb_file	lfp_electrode_group_name	interval_list_name	filter_name	filter_sampling_rate	analysis_nwb_file	object_id
beans20150205.nwb	2	run	LFP 0-400 Hz	1000	beans20150205_QAPEW KPS5O.nwb	5f3d0e08-039c-4ade-9bf2 -44ef69c554c9

LFPBand

lfp_id	interval_list_name	filter_name	filter_sampling_rate	analysis_nwb_file	lfp_band_object_id
9082d64f-55af -4926-948c- dce32f018e3e	ripple	Ripple 150-250 Hz	1000	beans20150205_I7X 1A8GQJR.nwb	bf72945a-b101-4e5c- b117-072a11f40a4f

RippleTimes

lfp_id	interval_list_name	filter_name	filter_sampling_rate	ripple_param_name	analysis_nwb_file	ripple_times_object_id
9082d64f-55af -4926-948c- dce32f018e3e	ripple	Ripple 150-250 Hz	1000	default	beans20150205_H5A 578YMJQ.nwb	de6a7c8b-3322-44db- a8c1-89af5d8221b3

Figure 2: Analysis pipelines in Spyglass. (A) A general structure for a Spyglass pipeline. (B) Example 1: LFP extraction. Note the correspondence to the pipeline structure in panel A as shown by the color scheme. The trace next to `Raw` table is raw data sampled at 30 kHz and is represented by a row in `Raw` table. This, along with parameters from `LFPElectrodeGroup`, `IntervalList`, and `FIRFilterParameters` tables (red arrow), are defined in a Python dictionary and inserted into `LFPSelection` table (code snippet). When the

`populate` method is called on `LFP` table, the filtering is initiated. The results (e.g. the trace above `LFP` table) are stored in NWB format and its object ID within the file is also stored as a row in `LFP` table, enabling easy retrieval. (C) Example 2: Sharp-wave ripple (SWR) detection. This pipeline is downstream of the LFP extraction pipeline and consists of two steps: (i) further extraction of a frequency band for SWR (`LFPBand`); and (ii) detection of SWR events in that band (`RippleTimes`). Note that the output of LFP extraction serves as the input data for the SWR detection pipeline and can thus be thought of as both `Compute` and `Data` type. As in (B), for each step, the results are saved in NWB files and the object ID of the analysis result within the NWB file are stored as rows in the corresponding `Compute` tables. The trace above `RippleTimes` table is the SWR-filtered LFP around the time of a single SWR event (pink shade). In each table, columns in bold are primary key. Arrows depict dependency structure within the pipeline.

Example 3: Spike sorting and curation (Figure 3)

Combining the principles of analysis pipeline design and merging we outlined in the previous sections, we now describe the spike sorting pipeline (Figure 3) in detail and discuss additional design decisions it embodies.

Spike sorting consists of the following steps: (1) preprocess the recording (e.g. filter and whiten to remove noise); (2) apply spike sorting algorithm (e.g. MountainSort4, Kilosort3, etc.); (3) curate the results (e.g. either manually or automatically by computing quality metrics); and (4) consolidate the output with other sources of sorted units (e.g. those already present in the NWB file) for downstream analysis. Each of these steps follow the general design shown in Figure 2A.

Global `Parameter` tables (e.g. `IntervalList`)

An important object in any analysis is the time interval during which the data were collected or to which analysis procedures should be applied. This is stored in the `IntervalList` table in Spyglass. To avoid having a separate table for time intervals in each pipeline, `IntervalList` serves as the source of time interval information for all pipelines. For example, in the spike sorting pipeline (Figure 3), `IntervalList` is provides time interval the preprocessing of the (`SpikeSortingRecordingSelection`) and running a spike sorting algorithm (`SpikeSortingSelection`). In addition, the intervals during which artifacts (high-amplitude voltage transients from behavioral events such as licking) occur can be identified and fed back into `IntervalList` (dashed arrow in Figure 3).

"Cyclic iteration" motif for curation

Certain tasks, such as curating the output of spike sorting, are often done iteratively. For example, one might first compute quality metrics to identify noise clusters and potential candidates for merging over-clustered units (Automatic); then inspect, merge, and apply curation labels to the result with an external viewer (Manual); and finally, compute a final set of metrics to describe the quality of each unit (Automatic). This results in the following sequence of steps: (Automatic, Manual, Automatic). Depending on the data, the user may choose a different sequence, and the order and length of these sequences might change as new algorithms and metrics are developed. This presents a challenge in modeling the pipeline under the relational database framework.

We therefore developed a specific design motif to enable this iterative curation with a finite number of tables (Figure 3). First, a given row of the `CurationV1` table (the output of the spike sorting step) is taken through automatic or manual curation steps downstream. Upon completion, the spike sorting object may enter this curation pipeline again as a new row in the `CurationV1` table. Importantly, the new row has information about previous curation from which it descended. This allows the user to keep track of every round of curation while applying as many steps as desired. It can also be easily extended; if new automatic curation algorithms are developed in the future, it can simply be added downstream to the `CurationV1` table, enabling application of the latest methods to previously collected data.

"Merge" motif for consolidating data streams and versioning pipelines

A different challenge arises when the user wants to feed multiple streams of data of the same type into a single downstream pipeline. For example, once curation is completed, the spike sorting is saved in `CurationV1`. But some NWB files may already contain curated spike sorting (as `ImportedSpikeSorting`), and one may want to apply the same downstream pipeline to both data sources to compare the results. In yet another case, the other data stream could be a different version of the spike sorting pipeline (e.g. `CurationV2`) that uses different algorithms but produces output of the same type. Adding the same downstream pipeline to all these individually would result in code redundancy and database bloat. Simply having these converge onto a single downstream table is not desirable either, as it will require new columns to be added every time a new version or new data stream is added. This involves modifying an existing table, which is cumbersome and risky.

To solve this problem, we have designed a "merge" motif (Figure 3). Here `Parts` tables (a table type within DataJoint tightly associated with a parent table) are used to implement the merging of multiple data streams onto a single table. The downstream pipeline then gets data from this table without any duplication. More details for the implementation and helper functions to maintain data integrity can be found in the tutorial notebook (Table 1, *03_Merge_Tables*).

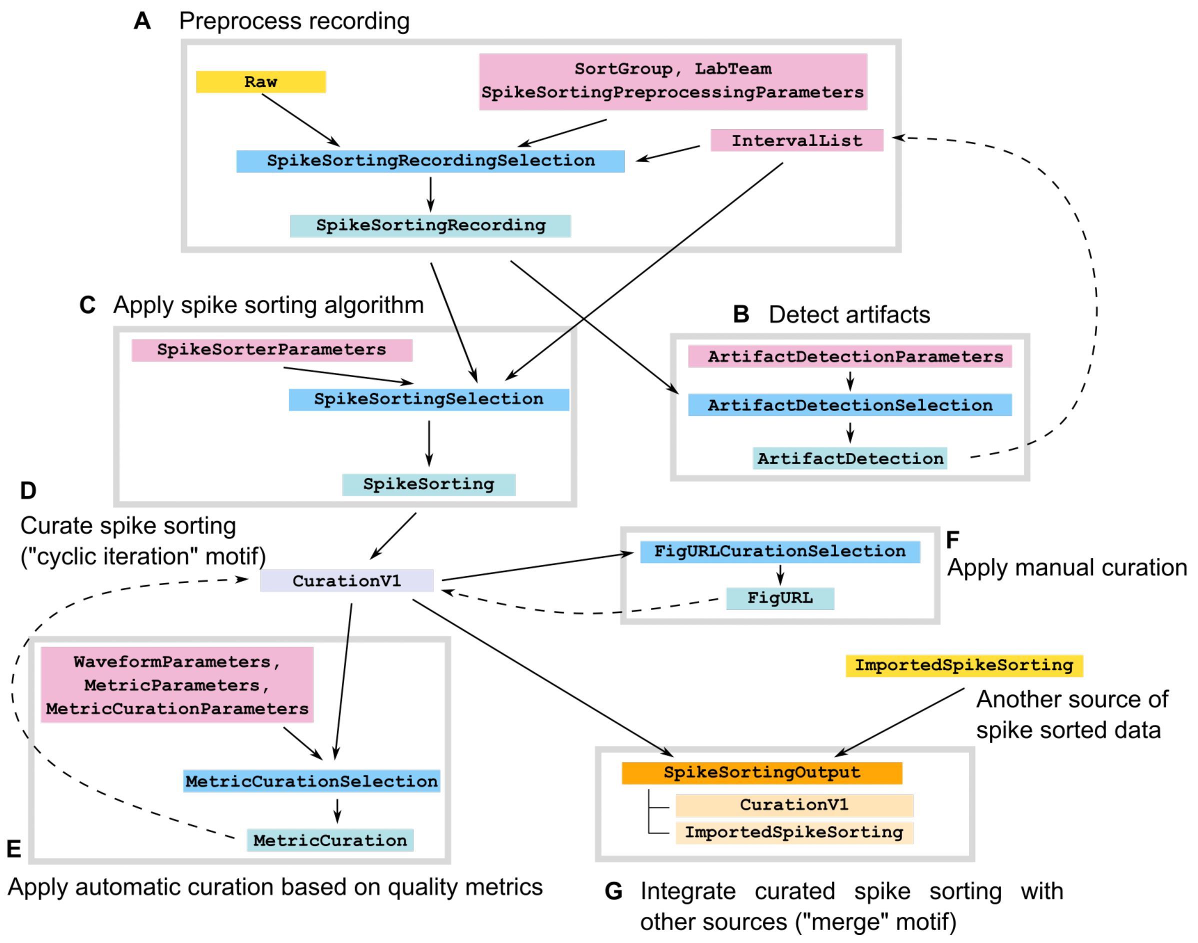


Figure 3: Spike sorting pipeline. The Spyglass spike sorting pipeline consists of seven components (large gray boxes): preprocess recording (A); detect artifacts to omit from sorting (B); apply spike sorting algorithm (C); curate spike sorting (D), either with quality metrics (E) or manually (F); and merge with other sources of spike sorting for downstream processing (G). Solid arrows describe dependency relation and dashed arrows indicate that the data is re-inserted upstream for iterative processing. Note the two design motifs (see text): "cyclic iteration" for curation and "merge" for consolidating data streams. Color scheme is the same as Figure 2, except for light purple (cyclic iteration table), orange (merge table), and peach (`Parts` table of the merge table).

Sharing Data, Analysis, and Visualization

Spyglass also includes tools that simplify sharing within and across laboratories.

Sharing data and analyses within and across labs

Within a lab, all data and analysis pipelines share the same organization and codebase. Once an NWB file is ingested into the database, multiple lab members can access the data, collaborate on analysis, and apply pipelines across various projects sharing the same input data types.

For multi-lab collaborations that are increasingly common in neuroscience, Spyglass also provides a secure way to share data and analyses across labs while projects are ongoing. Here, two steps are required. First, the collaborators are given access to the database hosted by the lab. Importantly, this does not grant access to the NWB files containing results, as the database only stores links to NWB files and not the files themselves. The owner of the files can then selectively make NWB files available to specific collaborators using Kachery, a content-addressed sharing tool for scientific data (Figure 4A). Specifically, collaborators' credentials are registered on the Kachery web page, which enables management of membership and permission settings for each project. Once these credentials are in place, a call to a method that fetches the data first looks for the data on the user's system. If the data are not present but listed in the sharing table, the corresponding files are automatically uploaded from their location to a cloud storage server and then downloaded to the user's computer. Collaborators can then access the data, develop their own pipelines, and share the code and the new results with the other members of the team. This feature is detailed in a tutorial (Table 1, *02_Data_Sync*). Kachery offers advantages over file hosting services (e.g. Dropbox and Google Drive) or alternative architectures (e.g. IBL data architecture) by not requiring a central location to track available files and providing a user-friendly Python API. This decentralized approach enhances flexibility and accessibility.

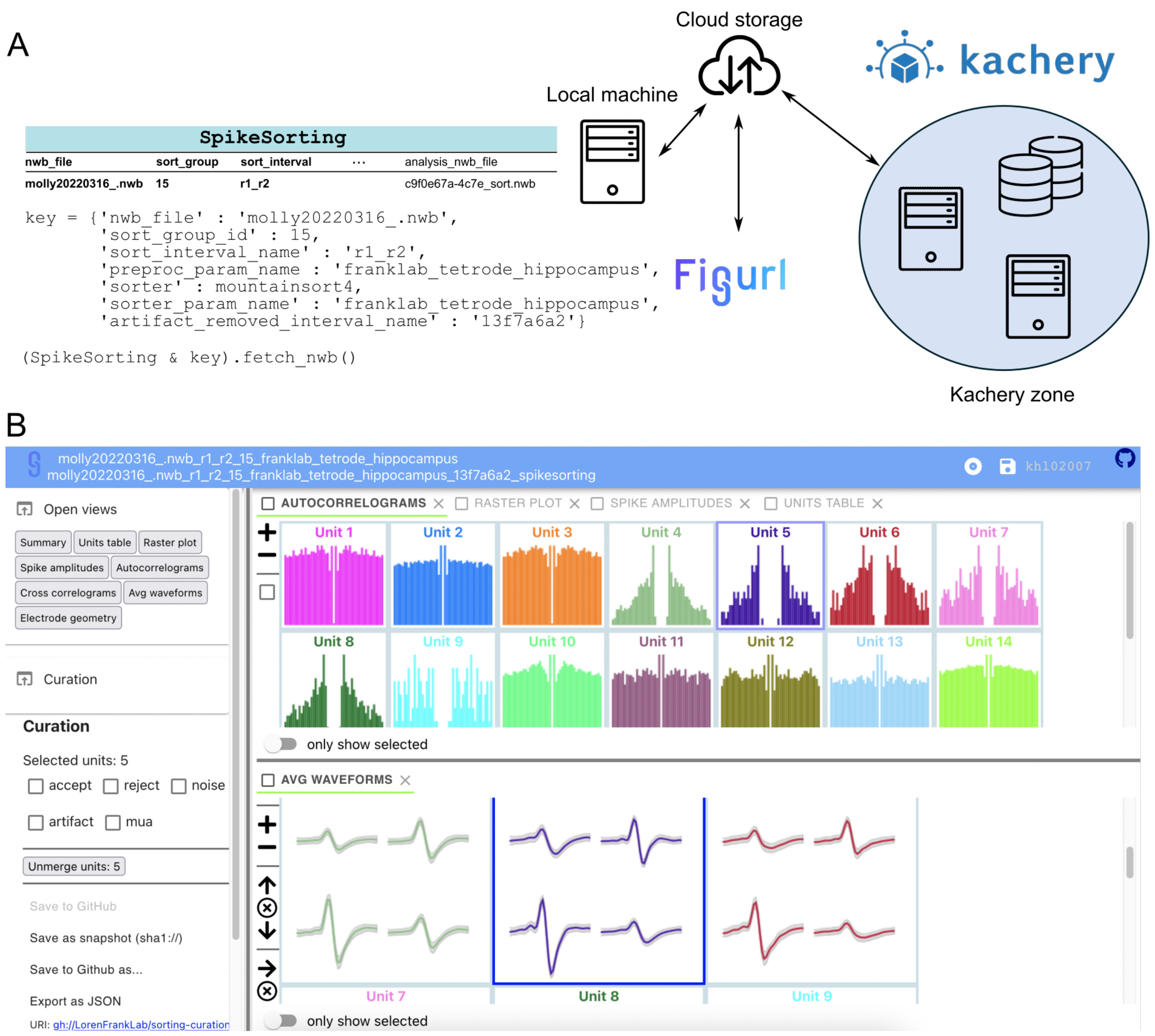
In addition, Spyglass simplifies the process of sharing when results are ready to be published. Adopting the NWB format makes sharing of raw data and intermediate results straightforward: at the end of the project lifetime (e.g. publication), we can deposit the associated NWB files in DANDI, a NIH-backed public archive for neuroscience data. Sharing the analysis code is also relatively easy: simply share both the codebase for the analysis pipelines (e.g. Spyglass, any project-specific pipelines defined on top, and the versions of the various python libraries used) as well as the scripts used to populate the database. Others can then download the raw data from DANDI, set up the database with Spyglass, and recreate all results locally. Alternatively, a snapshot of the database can be shared in a container (e.g. using Docker) or hosted on the cloud, providing community access without requiring database setup or re-running time-consuming analysis steps. This ensures complete transparency and reproducibility of the analysis.

Sharing visualization within and across labs

Spyglass also enables users to create and share interactive visualizations of analysis results through the Figurl package. Figurl is integrated within Spyglass and provides simple interfaces to generate visualizations from analysis outputs, such as spike sorting (Figure 4B) and neural decoding (Figure 5). These visualizations include the ability to explore complex, multi-dimensional time series across multiple views whose time axes can be linked. Moreover, the visualizations can be shared as web links without the need for any local software installation or specialized hardware. Thus, collaborators anywhere in the world can readily access and explore the data.

Pipeline	Tutorial notebook	Description
Data ingestion	00_Setup	Introduction to Spyglass and its structure
	01_Insert_Data	How to insert data into Spyglass
	02_Data_Sync	How to share data with collaborators who have access to the database
	03_Merge_Tables	A new table tier unique to Spyglass that allows the user to use different versions of pipelines on the same data
	04_PopulateConfigFile	Ways to ingest NWB files into the Spyglass database using yaml-based configuration file
Spike sorting	10_Spike_SortingV0	Detect spikes from electrophysiological recording and separate them to individual neurons (example of multiple versions of the same pipeline)
	10_Spike_SortingV1	
	11_CurationV0.ipynb	Curate the results of spike sorting manually for V0
Position processing	20_Position_Trodes.ipynb	Process information about animal's position from video recording of the behavior using Trodes
	21_DLC	Detect keypoint markers with DeepLabCut
	22_DLC_Loop	Detect keypoint markers with DeepLabCut over multiple epochs
	23_Linearization	Convert 2D position to 1D position using track geometry
LFP analysis	30_LFP	Filter broadband electrophysiology data to isolate low-frequency LFP bands
	31_Theta	Filter LFP to isolate the theta band
	32_Ripple_Detection	Detect sharp-wave ripples from filtered LFP
Decoding	40_Extracting_Clusterless_Waveform_Features	Extract waveform features for clusterless decoding
	41_Decoding_Clusterless	Apply the decoding algorithm using clusterless waveform features
	42_Decoding_SortedSpikes	Apply the decoding algorithm from spikes of sorted and curated units
MUA	50_MUA_Detection	Detect times of high multiunit firing

Table 1: Tutorials included in Spyglass and their descriptions. All available from https://github.com/LorenFrankLab/spyglass.



SpikeSorting

nwb_file	sort_group	sort_interval	⋯	analysis_nwb_file
molly20220316_.nwb	15	r1_r2		c9f0e67a-4c7e_sort.nwb

```
key = {'nwb_file' : 'molly20220316_.nwb',
       'sort_group_id' : 15,
       'sort_interval_name' : 'r1_r2',
       'preproc_param_name : 'franklab_tetrode_hippocampus',
       'sorter' : mountainsort4,
       'sorter_param_name' : 'franklab_tetrode_hippocampus',
       'artifact_removed_interval_name' : '13f7a6a2'}

(SpikeSorting & key).fetch_nwb()
```

Figure 4: Sharing data and visualizations. (A) Kachery provides a convenient Python API to share data over a content-addressable cloud storage network. To retrieve data from a collaborator's Spyglass database, one can make a simple function call (`fetch_nwb`) that pulls the data from a node in the Kachery Zone to the local machine. (B) Example of a Figurl interactive figure for visualizing and applying curation labels to spike sorting over the web.

Demonstration of generalizability: neural decoding of position in multiple data sets

A major goal of Spyglass is to facilitate analyses of data across multiple datasets. These datasets might arise from a single laboratory or might be compiled from multiple laboratories. To illustrate this second case, we ingested and analyzed two NWB files, one from our laboratory and another from the Buzsáki laboratory at NYU[23]. Specifically, we applied a switching state space model [24,25] to the data. This is a complex analysis that involves integrating multiple data sources, including position and neural spiking activity, and applying an advanced statistical model with many user-settable parameters. Pipelines in Spyglass enable the user to carry out every step of this analysis, including "preprocessing" the data (e.g. linearize the 2D position of the animal, perform spike sorting, or import units that have already been sorted) and fitting the model to the data. We can then visualize the results in the browser using Figurl. Because the data and parameters are available and systematized within Spyglass, the user can also quickly iterate to explore different parameter sets and their influence on the results.

The UCSF dataset contains large scale hippocampal recordings in an animal performing a foraging task in a maze with six reward sites and dynamic reward probabilities (Figure 5A, top panel). Applying the decoding pipeline to these data yields a probability distribution over space in 2 ms bins that describes our estimate of the "mental" position of the animal. This mental position tracks the animal as it traverses the maze (Figure 5A, 2nd panel from top; see interactive visualization via Figurl) but also shows interesting deviations from actual position. Computing the distance between the peak of the probability distribution and the actual location also reveals characteristic pattens of deviation from the actual position (Figure 5A, 3rd panel from top) wherein the decoded position sweeps ahead of the actual position and then back during movement bouts. This pattern recurs at ~8 Hz, reflecting the well-known "theta sequences" seen in the hippocampus[26,27].

We then applied this same pipeline to the NYU dataset, where animals performed a spatial alternation task on a maze with a figure-8 topology (Figure 5B, top panel). As expected, we could identify theta sequences in these data as well, highlighting the robustness of these phenomena (Figure 5B, 2nd and 3rd panels from top, see interactive visualization via Figurl). Moreover, the NYU dataset includes a specific manipulation in which the medial septum, a brain region critical for pacing the theta rhythm, was cooled, reducing the theta frequency from 8-10 Hz to 5-8 Hz. The authors carried out several detailed analyses to demonstrate that cooling reduced theta frequency and impaired behavior without changing the overall spatial tuning of single neurons or their tendency to fire sequentially within theta cycles. They did not apply state-space decoding methods, however, and did not characterize the effects of cooling on the decoded representation of space in relation to the animal's actual position. We therefore applied our pipeline to the cooling trials ("cooling on") as well as the control trials preceding it ("pre-cooling"), just after it ("cooling off"), and the recovery trials 10-12 minutes after cooling ("post-cooling").

The results of these analyses were consistent with the published findings and provided new characterizations that could serve as the foundation for additional discoveries. We first estimated the multiunit firing rate as a proxy for the theta LFP and characterized its power spectrum before and after cooling. As expected, cooling decreased the power above ~8 Hz and increased the power below ~8 Hz, consistent with the slowing of theta LFP shown in the original manuscript (Figure 5C, top panel). We then applied the same analysis described above to the distance between the decoded and the actual position during movement ("decode distance"), expecting cooling to have a similar effect on its power spectrum. Interestingly, here cooling led to a decrease in power at essentially all frequencies (Figure 5C, bottom panel). Consistent with this result, the decode distance decreased from the pre-cooling to cooling period, with a partial recovery during the post-cooling period (Figure 5D, top panel). Similarly, the average speed at which the decoded position moved ahead and behind the animal was also reduced during cooling and showed a partial recovery after the cooling period (Figure 5D, bottom panel). These results indicate that cooling reduces both the extent and the rate at which the decoded position deviates from the actual position. This was unexpected given that cooling had no effect on the average spatial tuning of these cells[23]. It also raises an interesting hypothesis: hippocampal representations of distant locations may be exquisitely tuned to the specific frequency of the rhythmic input from medial septum, such that slowing the rhythm down by just 2-3 Hz significantly limits their expression.

More broadly, these findings illustrate the power of a framework that enables both replication of results across datasets and the re-analysis of previously collected data.

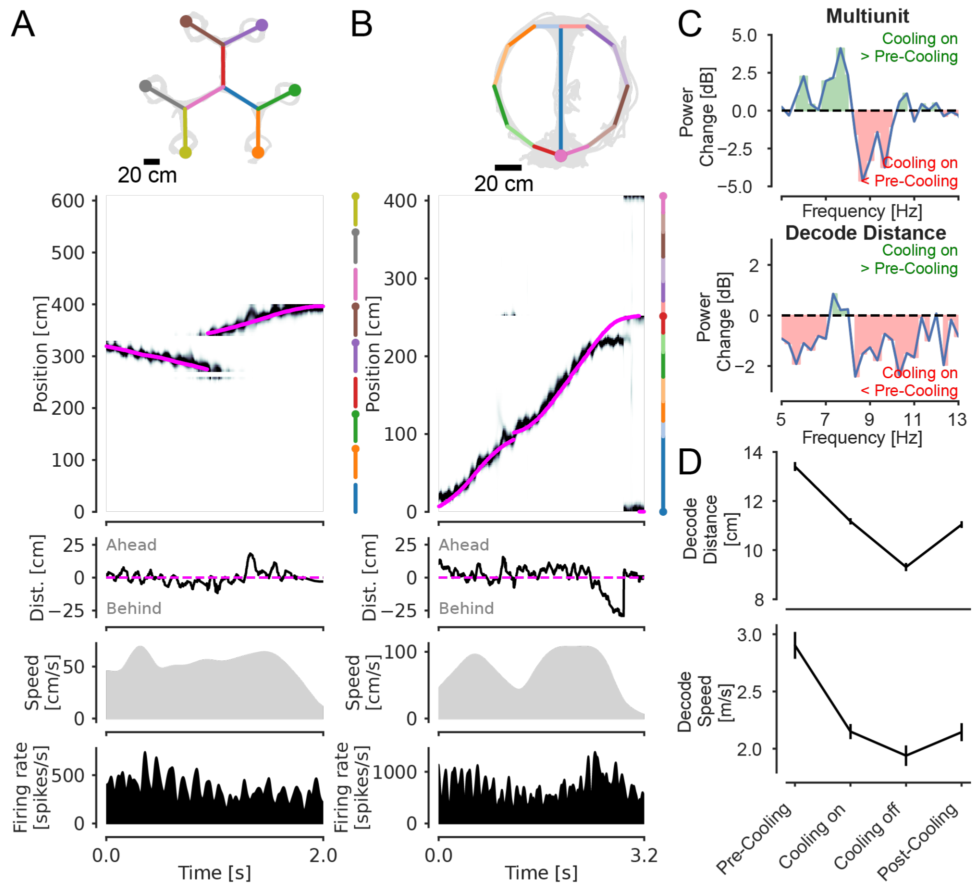


Figure 5: Applying decoding pipelines to multiple data sets from different labs (A) Decoding neural position from rat hippocampal CA1 using a clusterless state space model (UCSF dataset). In the top panel, grey lines represent positions the rat has occupied in the spatial environment. Overlayed lines in color are the track segments used to linearize position for decoding. Filled circles represent reward wells. The second panel from the top shows the posterior probability of the latent neural position over time. The magenta line represents the animal's actual position. The vertical lines on the right represent the linearized track segments with the colors corresponding to the top panel. The third panel from the top shows the distance of the most likely decoded position from the animal's actual position and sign indicates the direction relative to the animal's head position. The fourth panel from the top is the speed of the animal. The final panel is the multiunit firing rate. (B) Decoding from rat hippocampal CA1 using existing spike sorted units (NYU dataset). Conventions are the same as in A. Filled circle in the linearization represents the reward zone rather than the reward well. (C) Decoding analysis of the NYU dataset. The top panel shows the power difference of the multiunit firing rate between the medial septal cooling period and the pre-cooling period in the 5-13 Hz range. The power at 8-10 Hz is attenuated during cooling while the power at 5-8 Hz is enhanced, showing a slowing of the theta rhythm during cooling. The bottom panel shows that the power of the distance between decoded and actual position (decode distance) is mostly reduced throughout the 5-13 Hz range. (D) Cooling decreases the decode distance and speed and this effect may only recover partially after cooling. Bars represent 95% confidence intervals.

Discussion

Summary of results

Science is a social enterprise, in which the accumulation and dissemination of knowledge rely heavily on collaboration and transparency among researchers. Reproducible and sharable data analysis plays a critical role in this context, as it ensures that scientific findings can be independently verified and built upon by others. Spyglass is a framework designed to promote these goals of reproducible and sharable data analysis. Based on a robust foundation of community-supported standards and open-source tools, it provides an effective data management solution and reliable and reproducible analysis pipelines. Its integration of data and web-based sharing of visualizations simplifies collaboration within and across labs, making it well-suited as a community framework for neurophysiological and behavioral data analysis.

Comparison to prior work

Our work builds on and integrates many previous approaches that have been proposed for scientific data management and reproducible analysis pipelines. This includes work from individual laboratories that have illustrated how a few elements of an NWB file could be read into a DataJoint database[28], as well as publications highlighting datasets available in NWB[29]. More broadly, DataJoint is used by many labs with lab-specific pipelines[30], but to our knowledge none of these efforts integrate cross-laboratory data and visualization tools or use NWB as foundation to facilitate sharing. Our system also contains elements similar to those developed by large collaborative groups like The International Brain Laboratory, a system designed to organize neurophysiology data for sharing with collaborators and a module to automatically run analyses[12]. But the conversion to a standardized format (outside the collaboration or group) and public data sharing are only done following substantial analysis, complicating replication of the full analysis. Another recent project called DataLad uses version control tools such as git and git-annex to manage both code and data as files[31]. This interesting project shares similar goals to Spyglass by enabling the creation of a reproducible data analysis environment and decentralized data sharing, though it does not provide formal structures such as relational databases to organize the analysis pipelines.

By contrast, Spyglass begins with a shared data format that includes the raw data and offers both transparent data management and reproducible analysis pipelines for real-world, large-scale data analysis. One distinguishing feature of Spyglass is the emphasis placed on combinatorial matching of data and method in a reproducible way. For example, Spyglass makes it possible to apply multiple spike sorting algorithms to a given dataset and to compare the results. Similarly, as we illustrated, Spyglass makes it straightforward to apply complex analyses like decoding to datasets from multiple labs, facilitating replication and data re-use. Furthermore, as better tools and algorithms become available, Spyglass offers a straightforward way to re-analyze the data to determine how results depend on the choice of algorithm. We feel that it is critical to provide this kind of future-compatibility to maximize the impact of the years of experimental work that go into each dataset.

Balancing reproducibility and flexibility

There is an inherent tension between reproducibility and flexibility in data analysis. The former requires that every analysis run the same way in as many contexts as possible, while the latter emphasizes the ability to try out different algorithms, including those that may become available in the future. Maximal flexibility is achieved by an individual scientist implementing their own analysis pipelines, but this comes at the cost of a lack of reproducibility. Other data analysis environments (e.g. CodeOcean) also provide substantial flexibility to the user in developing and executing analyses, but a lack of constraint on how metadata and analyses are organized can impede reproducibility.

Spyglass provides a system that emphasizes reproducibility but also includes tools to ensure flexibility. Reproducibility is enhanced by both the standardized structure of each analysis, with tables for `Data`, `Parameter`, `Selection`, and `Compute`, as well as by the strict data integrity requirements of DataJoint. While this ensures that the provenance of every entry in the table can be reconstructed, substantial changes to the structure of the tables require either regenerating the results with the new structure or creating a new version of the pipeline.

Although we believe such rigidness in our system is a "feature, not a bug," we also recognize that some flexibility is required and have made efforts to implement it. For example, we have made it easy to supplement or override information in the NWB file with a configuration file (see Methods and Table 1, *04_PopulateConfigFile*). We also have developed a versioning system and database design motifs to allow specification of novel pipelines that can be swapped in place of an old one without disrupting downstream pipelines. These features provide flexibility and future-compatibility, as there can be multiple versions of a pipeline for any given analysis task.

Furthermore, because upstream analyses feed into downstream analyses, users can build their own pipelines on top of the pipelines already provided (see Methods). These pipelines can branch from any point in a previously created pipeline: for example, they could start with the raw data, the filtered LFP, the results of spike sorting, etc. The pipeline can then define its own set of `Data`, `Parameter`, `Selection`, and `Compute` tables to carry out the desired tasks. And once this new pipeline is validated and published, other users could build on it to achieve their own goals, minimizing the re-implementation of analyses that is endemic in our field.

The costs and benefits of doing reproducible research

The benefits of doing reproducible research are clear. Adhering to this high standard allows one to be more confident of one's own results, makes it easier for others in the community to verify or build on them, and increases robustness against problems arising from errors in the analysis scripts [32]. There are also technical benefits; for example, organizing one's analysis around a system like Spyglass makes it easy to scale the compute when processing a large dataset, as one just needs to recruit more compute nodes.

Despite these benefits, many scientists are hesitant to fully embrace reproducible research because it often requires much time and knowledge to implement properly. For example, it may be unclear how one should adopt tools like NWB and DataJoint for one's needs, as they can be used in many ways. One goal of Spyglass is to provide an example of how these tools can be integrated seamlessly to increase the transparency of analyses, facilitate collaborations, and improve reproducibility. We hope that this "existence proof" will reduce the mental barrier to dedicating more efforts to reproducible research.

At the same time, we acknowledge the substantial costs in time and effort for a laboratory to convert their raw data to NWB and adopt a system like Spyglass[33]. These costs include the necessity of learning Python as well as the requirements to set up and maintain a relational database. In addition, Spyglass was designed for Unix-based systems (Linux and MacOS) and is not yet compatible with the Windows operating system. Fortunately, there are ongoing efforts to address these challenges, including tools to simplify the raw data conversion into NWB, such as NeuroConv, a package to convert neurophysiology data in common formats to NWB automatically, and NWB GUIDE, a desktop app that guides users through the process of converting data to NWB without writing any code. These efforts, along with data sharing mandates by funding agencies, are expected to boost demand for tools like Spyglass. Over time, our hope is that Spyglass will include pipelines for other data modalities like optical physiology. As these tools become more user friendly and the data and code sharing requirements become more stringent, the adoption of Spyglass or a similar system will become increasingly appealing, particularly for young scientists just starting out.

Future applications

Spyglass and similar tools have the potential to transform scientific data analysis. Beyond facilitating examination or extension of published results, they enable meta-analysis across studies and easy testing of novel methods across multiple datasets. The machine-readable form of data and analysis pipelines also opens doors for machine-driven analysis and hypothesis testing. As these tools continue to develop and become more accessible, we believe that frameworks like Spyglass will likely become essential for neuroscience researchers.

Acknowledgements

We thank members of the Frank and Gillespie laboratory for bug reports and testing. We also thank Peter Petersen for consultations about analyzing his publicly available data, Daniel Liu for initial discussions about standardization of pipelines, and Abhilasha Joshi for consulting on the DeepLabCut pipeline. Finally, we thank Vanessa Bender for comments on the manuscript. This work was supported by HHMI funds and NIH grants RF1MH130623 and RF1MH133778 to L.M.F. and a Helen Hay Whitney Postdoctoral Fellowship to K.H.L.

Author contributions

Contribution	KL	ED	RL	JM	JS	AC	DG	JG	RN	PA	CB	SB	EM	JB
Conception	■	■	■											
Pipeline design	■	■	■			■	■	■			■	■		
Pipeline implementation	■	■	■			■	■	■	■		■	■		■
Tool development	■	■	■	■	■					■				
Documentation and tutorials	■	■	■				■			■	■	■		
Testing and bug fixes	■	■	■	■	■	■	■	■	■	■	■	■	■	■
Data collection						■		■					■	
Data analysis	■	■												
Figure generation	■	■												
Drafting manuscript	■	■	■	■	■	■		■		■	■	■		

Contribution	MC	XS	EB	DS	SC	CH	AT	OR	TN	DY	JC	CK	SG	AB	LF
Conception															■
Pipeline design															■
Pipeline implementation	■	■	■	■	■	■									■
Tool development							■	■	■	■	■	■	■	■	■
Documentation and tutorials															■
Testing and bug fixes	■	■	■	■	■	■									■
Data collection	■														
Data analysis															
Figure generation															
Drafting manuscript	■	■	■	■	■	■		■	■	■	■	■	■	■	■

Declaration of interests

The authors declare no competing interests.

Methods and materials

Coding environment

Spyglass was developed in Python 3.9 and is compatible with version 3.10 as well. See our dependency list for a full list of Python packages used.

NWB conversion

To facilitate conversion of raw data to NWB format, we offer `trodes-to-NWB`, a sister package to Spyglass for converting data acquired with the SpikeGadgets hardware to NWB. This comes with a web-based GUI for conveniently generating a YAML file containing the metadata used by `trodes-to-NWB`. For converting data not acquired with SpikeGadgets, users can use NWB conversion tools developed by the NWB team, such as NeuroConv and NWB GUIDE.

NWB file conventions

We adopted a specific set of conventions for our NWB files. Some of these conventions rely on a specific set of Frank lab-specific NWB extensions:

- Time:
    - Spyglass inherits from the source NWB file either the explicit or implicit timestamps. NWB files from Frank lab have explicit timestamps for each sample in Unix time (seconds since 12:00 am January 1st, 1970). This enables users to know exactly when data were collected. Spyglass is also compatible with other approaches, however, including implicit timestamping consisting of the start time and sampling rate.
- `ElectrodeTable` and `ElectrodeGroup`:
    - `ElectrodeGroups` are stored in a custom NWB extension that also includes the name of the targeted brain region for each group.
    - The NWB file contains information about the relative locations of each of the electrodes within each physical device used for data collection. This ensures that the relative locations of the electrodes are available for spike sorting and registration to histology.
- Video files
    - The relative path to the video files collected along with the recordings are stored in the NWB file.
- Additional files
    - Other files important to recreate the conditions of the experiments can be saved, depending on the format. For example, the code used for implementing the behavioral paradigm or reward contingency can be stored as text objects in the NWB file.

NWB file ingestion

Although the NWB format serves as a community standard for neurophysiology data and has a list of best practices, it allows some flexibility in the specification of data within NWB files to accommodate user preferences. For example, the `ElectricalSeries` object that stores the electrophysiology data may have different names depending on the convention chosen by the investigator, which may complicate programmatic access to the data. To make Spyglass interoperable with NWB files of varying degrees of NWB-compliance, we have created an option to supply or override information that is missing in the NWB file but is nevertheless required by Spyglass via a configuration file that can accompany the NWB file. We provide an example of this approach in a Jupyter Notebook tutorial (Table 1, *04_PopulateConfigFile*).

Permission-handling and cautious delete

Spyglass is based on a relational database that is accessible to multiple users. In some cases, the type of operations that can be applied to individual data entries (i.e., rows of a table) may need to be restricted to a specified set of users. This is particularly true for operations that are irreversible or time consuming, such as deleting a row from a table storing analysis results. However, there is no inherent mechanism within MySQL or DataJoint that allows permission handling at the level of individual rows of a table. To solve this problem, we have implemented a `cautious_delete` function, in which the user's permission to carry out a delete operation is checked before it is applied. The permission is granted based on team membership within the lab, reflected in the `LabTeam` table. Though this is not a formal permission-management system, it can prevent accidental deletions.

Sharing files via Kachery

One way to share the results of Spyglass analysis pipelines is to make the database publicly available. This gives anyone the permission to access the rows of the tables that make up the pipelines and inspect the metadata and the parameters associated with each step of the analysis. But because Spyglass only saves a path to the NWB files containing analysis results within the tables, external viewers cannot download the data and examine it by default.

To enable controlled external access to the data, we have created a system to share selected analysis NWB files with a specified group of users via Kachery. We define a set of tables (`KacheryZone` and `AnalysisNWBfileKachery`) where users can associate analysis NWB files to be shared with a Kachery Zone, making it available to all remote clients who are members of the zone through cloud storage services like Cloudflare R2 bucket or self-hosted servers. Once linked, Spyglass automatically requests, downloads, and manages analysis data for remote users attempting to access shared data through Spyglass tables. This provides a convenient way to provide access to the Spyglass pipelines and associated data files to collaborators.

Customizing pipelines

To alleviate the challenges associated with database design, we have identified design principles that have been tested extensively by multiple users in the Frank lab. These are described in the text and illustrated with examples in Figures 2 and 3. We recommend users adopt these design elements for building their custom pipelines. We also describe the naming conventions for the tables defined as Python classes and important methods associated with them (e.g. for multiple versions of a pipeline) in our Developer Notes available online. Once the pipeline is sufficiently mature and potentially useful to other scientists, we encourage users to submit their pipelines as a pull request to our GitHub repository.

Decoding of position from NWB files from multiple laboratories

The Frank lab data is available on the DANDI archive (DANDI:000937). The Buzsaki lab data was also obtained from DANDI (DANDI:000059/0.230907.2101). For decoding the Frank lab data, we applied the clusterless decoding pipeline by detecting the amplitude of threshold-crossing events in the tetrode recordings. For decoding the Buzsaki lab data, we applied a sorted-spikes decoding pipeline. The code for these decoding pipelines, as well as detailed tutorials describing them, are available online (Table 1, *41_Extracting_Clusterless_Waveform_Features, 42_Decoding_Clusterless, 43_Decoding_SortedSpikes*). Code to generate Figure 5 can be found at: https://github.com/LorenFrankLab/spyglass-paper. Briefly, decoding the latent neural position and extracting the distance between the most likely decoded position and the animal's position used methods described in Denovellis et al. (2021). We used a timestep of 4 ms and a position bin size of 2 cm with a continuous (6 cm variance Gaussian random walk) and fragmented (uniform distribution) discrete state. Place intensity receptive fields were estimated using a Gaussian kernel density estimate with a standard deviation of 6 cm for position and 24 mV for amplitude space (amplitude space was used for the clusterless analysis only). We calculated the power of the multiunit firing rate and the decoded distance from the animal by using a multitaper spectrogram during the pre-cooling and cooling periods. The time resolution was 3 seconds and the frequency resolution of 2/3 Hz with a single taper. We excluded immobility periods by using a threshold of 10 cm/s. Power difference was calculated by converting to the Decibel scale and taking the difference of average power under the cooling and pre-cooling condition. The decoded speed of theta sequences was calculated by taking the absolute value of the second-order difference of the decoded distance from the animal (function numpy.gradient) multiplied by the sampling frequency (250 Hz).

Reference

1. Buccino, A.P., Hurwitz, C.L., Garcia, S., Magland, J., Siegle, J.H., Hurwitz, R., and Hennig, M.H. (2020). SpikeInterface, a unified framework for spike sorting. eLife *9*, e61834. 10.7554/eLife.61834.

2. Lopes, G., Bonacchi, N., FrazÃ£o, J., Neto, J.P., Atallah, B.V., Soares, S., Moreira, L., Matias, S., Itskov, P.M., Correia, P.A., et al. (2015). Bonsai: an event-based framework for processing and controlling data streams. Front. Neuroinformatics *9*. 10.3389/fninf.2015.00007.

3. Nath, T., Mathis, A., Chen, A.C., Patel, A., Bethge, M., and Mathis, M.W. (2019). Using DeepLabCut for 3D markerless pose estimation across species and behaviors. Nat Protoc *14*, 2152–2176. 10.1038/s41596-019-0176-0.

4. Pachitariu, M., Steinmetz, N., Kadir, S., Carandini, M., and Harris, K. (2016). Kilosort: realtime spike-sorting for extracellular electrophysiology with hundreds of channels. bioRxiv, 061481. 10.1101/061481.

5. Siegle, J.H., López, A.C., Patel, Y.A., Abramov, K., Ohayon, S., and Voigts, J. (2017). Open Ephys: an open-source, plugin-based platform for multichannel electrophysiology. J. Neural Eng. *14*, 045003. 10.1088/1741-2552/aa5eea.

6. Yatsenko, D., Walker, E.Y., and Tolias, A.S. (2018). DataJoint: A Simpler Relational Data Model. ArXiv180711104 Cs.

7. Abe, T., Kinsella, I., Saxena, S., Buchanan, E.K., Couto, J., Briggs, J., Kitt, S.L., Glassman, R., Zhou, J., Paninski, L., et al. (2022). Neuroscience Cloud Analysis As a Service: An open-source platform for scalable, reproducible data analysis. Neuron *110*, 2771-2789.e7. 10.1016/j.neuron.2022.06.018.

8. Wilkinson, M.D., Dumontier, M., Aalbersberg, Ij.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L.B., Bourne, P.E., et al. (2016). The FAIR

Guiding Principles for scientific data management and stewardship. Sci. Data *3*, 160018. 10.1038/sdata.2016.18.

9. Goble, C., Cohen-Boulakia, S., Soiland-Reyes, S., Garijo, D., Gil, Y., Crusoe, M.R., Peters, K., and Schober, D. (2020). FAIR Computational Workflows. Data Intell. *2*, 108–121. 10.1162/dint_a_00033.

10. De Vries, S.E., Siegle, J.H., and Koch, C. (2023). Sharing neurophysiology data from the Allen Brain Observatory. eLife *12*, e85550. 10.7554/eLife.85550.

11. Hider, R., Kleissas, D., Gion, T., Xenes, D., Matelsky, J., Pryor, D., Rodriguez, L., Johnson, E.C., Gray-Roncal, W., and Wester, B. (2022). The Brain Observatory Storage Service and Database (BossDB): A Cloud-Native Approach for Petascale Neuroscience Discovery. Front. Neuroinformatics *16*, 828787. 10.3389/fninf.2022.828787.

12. The International Brain Laboratory, Acerbi, L., Aguillon-Rodriguez, V., Ahmadi, M., Amjad, J., Angelaki, D., Arlandis, J., Ashwood, Z.C., Banga, K., Barrell, H., et al. (2023). A modular architecture for organizing, processing and sharing neurophysiology data. Nat. Methods *20*, 403–407. 10.1038/s41592-022-01742-6.

13. Magland, J., Jun, J.J., Lovero, E., Morley, A.J., Hurwitz, C.L., Buccino, A.P., Garcia, S., and Barnett, A.H. (2020). SpikeForest, reproducible web-facing ground-truth validation of automated neural spike sorters. eLife *9*, e55167. 10.7554/eLife.55167.

14. Matelsky, J.K., Rodriguez, L.M., Xenes, D., Gion, T., Hider, R., Wester, B.A., and Gray-Roncal, W. (2021). An Integrated Toolkit for Extensible and Reproducible Neuroscience. In 2021 43rd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC) (IEEE), pp. 2413–2418. 10.1109/EMBC46164.2021.9630199.

15. Sanchez, M., Moore, D., Johnson, E.C., Wester, B., Lichtman, J.W., and Gray-Roncal, W. (2022). Connectomics Annotation Metadata Standardization for Increased Accessibility and Queryability. Front. Neuroinformatics *16*, 828458. 10.3389/fninf.2022.828458.

16. Rübel, O., Tritt, A., Ly, R., Dichter, B.K., Ghosh, S., Niu, L., Baker, P., Soltesz, I., Ng, L., Svoboda, K., et al. (2022). The Neurodata Without Borders ecosystem for neurophysiological data science. eLife *11*, e78362. 10.7554/eLife.78362.

17. Teeters, J.L., Godfrey, K., Young, R., Dang, C., Friedsam, C., Wark, B., Asari, H., Peron, S., Li, N., Peyrache, A., et al. (2015). Neurodata Without Borders: Creating a Common Data Format for Neurophysiology. Neuron *88*, 629–634. 10.1016/j.neuron.2015.10.025.

18. Yatsenko, D., Reimer, J., Ecker, A.S., Walker, E.Y., Sinz, F., Berens, P., Hoenselaar, A., Cotton, R.J., Siapas, A.S., and Tolias, A.S. (2015). DataJoint: managing big scientific data using MATLAB or Python. BioRxiv, 031658.

19. Chu, J.P., and Kemere, C.T. (2021). GhostiPy: An Efficient Signal Processing and Spectral Analysis Toolbox for Large Data. eneuro *8*, ENEURO.0202-21.2021. 10.1523/ENEURO.0202-21.2021.

20. Viejo, G., Levenstein, D., Carrasco, S.S., Mehrotra, D., Mahallati, S., Vite, G.R., Denny, H., Sjulson, L., Battaglia, F.P., and Peyrache, A. (2023). Pynapple: a toolbox for data analysis in neuroscience (elife) 10.7554/eLife.85786.1.

21. BRAIN Initiative (2019). Notice of Data Sharing Policy for the BRAIN Initiative.

22. Ghosh, S., Baker, C., Choudhury, R., Dichter, B., Jarecka, D., Dehghani, N., Ioanas, H., Lamanna, D., Nesbitt, J., Vandenbrugh, M., et al. (2023). Dandi: an archive and collaboration space for neurophysiology projects. In.

23. Petersen, P.C., and Buzsáki, G. (2020). Cooling of Medial Septum Reveals Theta Phase Lag Coordination of Hippocampal Cell Assemblies. Neuron *107*, 731-744.e3. 10.1016/j.neuron.2020.05.023.

24. Denovellis, E.L., Gillespie, A.K., Coulter, M.E., Sosa, M., Chung, J.E., Eden, U.T., and Frank, L.M. (2021). Hippocampal replay of experience at real-world speeds. Elife *10*. 10.7554/eLife.64505.

25. Denovellis, E.L., Frank, L.M., and Eden, U.T. (2019). Characterizing hippocampal replay using hybrid point process state space models. In (IEEE), pp. 245–249.

26. Skaggs, W.E., and McNaughton, B.L. (1996). Replay of neuronal firing sequences in rat hippocampus during sleep following spatial experience. Science *271*, 1870–1873.

27. Foster, D.J., and Wilson, M.A. (2007). Hippocampal theta sequences. Hippocampus *17*, 1093–1099.

28. Reimer, M.L., Bangalore, L., Waxman, S.G., and Tan, A.M. (2021). Core principles for the implementation of the neurodata without borders data standard. J. Neurosci. Methods *348*, 108972. 10.1016/j.jneumeth.2020.108972.

29. Chandravadia, N., Liang, D., Schjetnan, A.G.P., Carlson, A., Faraut, M., Chung, J.M., Reed, C.M., Dichter, B., Maoz, U., Kalia, S.K., et al. (2020). A NWB-based dataset and processing pipeline of human single-neuron activity during a declarative memory task. Sci. Data *7*, 78. 10.1038/s41597-020-0415-9.

30. Yatsenko, D., Nguyen, T., Shen, S., Gunalan, K., Turner, C.A., Guzman, R., Sasaki, M., Sitonic, D., Reimer, J., Walker, E.Y., et al. (2021). DataJoint Elements: Data Workflows for Neurophysiology (Neuroscience) 10.1101/2021.03.30.437358.

31. Halchenko, Y., Meyer, K., Poldrack, B., Solanky, D., Wagner, A., Gors, J., MacFarlane, D., Pustina, D., Sochat, V., Ghosh, S., et al. (2021). DataLad: distributed system for joint management of code, data, and their relationship. J. Open Source Softw. *6*, 3262. 10.21105/joss.03262.

32. Soergel, D.A.W. (2015). Rampant software errors may undermine scientific results. F1000Research *3*, 303. 10.12688/f1000research.5930.2.

33. Pierré, A., Pham, T., Pearl, J., Datta, S.R., Ritt, J.T., and Fleischmann, A. (2024). A perspective on neuroscience data standardization with Neurodata Without Borders. Preprint at arXiv.