Spyglass: a data analysis framework for reproducible and shareable neuroscience research

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

[1]Department of Physiology, University of California, San Francisco
[2]Howard Hughes Medical Institute
[3]Scientific Data Division, Lawrence Berkeley National Laboratory
[4]Center for Computational Mathematics, Flatiron Institute
[5]DataJoint
[6]Department of Electrical and Computer Engineering, Rice University
[7]Centre de Recherche en Neuroscience de Lyon, CNRS
[8]Allen Institute for Brain Science

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

Abstract

Sharing data and reproducing scientific results are essential for progress in neuroscience, but the community lacks the tools to do this easily for large datasets and results obtained from intricate, multi-step analysis procedures. To address this issue, we created Spyglass, an open-source software framework designed to promote the shareability and reproducibility of data analysis in neuroscience. Spyglass integrates standardized formats with reliable open-source tools, offering a comprehensive solution for managing neurophysiological and behavioral data. It provides well-defined and reproducible pipelines for analyzing electrophysiology data, including core functions like spike sorting. In addition, Spyglass simplifies collaboration by enabling the sharing of final and intermediate results across custom, complex, multi-step pipelines as well as web-based visualizations. Here we demonstrate these features and showcase the potential of Spyglass to enable findable, accessible, interoperable, and reusable (FAIR) data management and analysis by applying advanced state space decoding algorithms to publicly available data from multiple laboratories.

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 based on the analysis of complex, multi-modal datasets. While the community has significantly improved tools and algorithms for data collection and analysis (Buccino et al., 2020; Lopes et al., 2015; Nath et al., 2019; Pachitariu et al., 2016; Siegle et al., 2017; Yatsenko et al., 2018), deriving consistent and reproducible insights from data remains a complex and time-consuming task.

The difficulty stems from how the scientific community performs analysis and synthesis of data. Often, researchers take years to collect and organize data, which is then transformed through intricate analysis. This begins with preprocessing steps that extract specific signals from the data, followed by a series of custom-written scripts to further examine 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. For instance, experimental protocols might 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 missing. Essential tasks, like 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 (Abe et al., 2022). In collaborations between multiple scientists, these problems are 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. For example, a new trainee might struggle to examine existing data due to issues with understanding critical details. Similarly, shared code might not be standardized or documented, causing multiple teams to duplicate efforts implementing the same tools.

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 data (Wilkinson et al., 2016) and analysis pipelines (Goble et al., 2020). 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 (De Vries et al., 2023; Hider et al., 2022; The International Brain Laboratory et al., 2023). However, these efforts have not fully resolved the issues related to sharing data and reproducing analysis. For instance, these organizations often do not share raw data, and when they do, it may not be in a standardized format accessible to the broader community. In addition, they do not disclose every step of the processing pipeline, such as spike sorting, which can significantly affect the results (Magland et al., 2020). Some of the technologies they use, such as cloud computing services (e.g., CodeOcean) and sophisticated databases and APIs (Hider et al., 2022; Matelsky et al., 2021; Sanchez et al., 2022), 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 advancements, 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 (Rübel et al., 2022; Teeters et al., 2015). It uses DataJoint (Yatsenko et al., 2018, 2015) to manage reproducible analysis pipelines with a relational database and novel software tools such as Kachery and Figurl for sharing data and web-based visualizations to enable collaboration within and across labs. Being Python-based and using standard data types, it is compatible with a wide array of analysis packages including SpikeInterface (Buccino et al., 2020), GhostiPy (Chu and Kemere, 2021), and Pynapple (Viejo et al., 2023). Spyglass also offers ready-to-use pipelines for analyzing behavior and electrophysiology data, as well as extensive documentation and tutorials for training new users. 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 reliable and reproducible analysis of neuroscience data and sharing of the results with collaborators and the broader community. The process begins with raw data and experimental metadata stored in NWB, a format that meets the data management standards of BRAIN Initiative (BRAIN Initiative, 2019; Rübel et al., 2022). Spyglass then ingests these NWB files into a relational database using DataJoint, ensuring reproducible analysis 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 has recently become a standard for neurophysiology and behavior data (BRAIN Initiative, 2019; Rübel et al., 2022). 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 (Ghosh et al., 2023).
- 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.

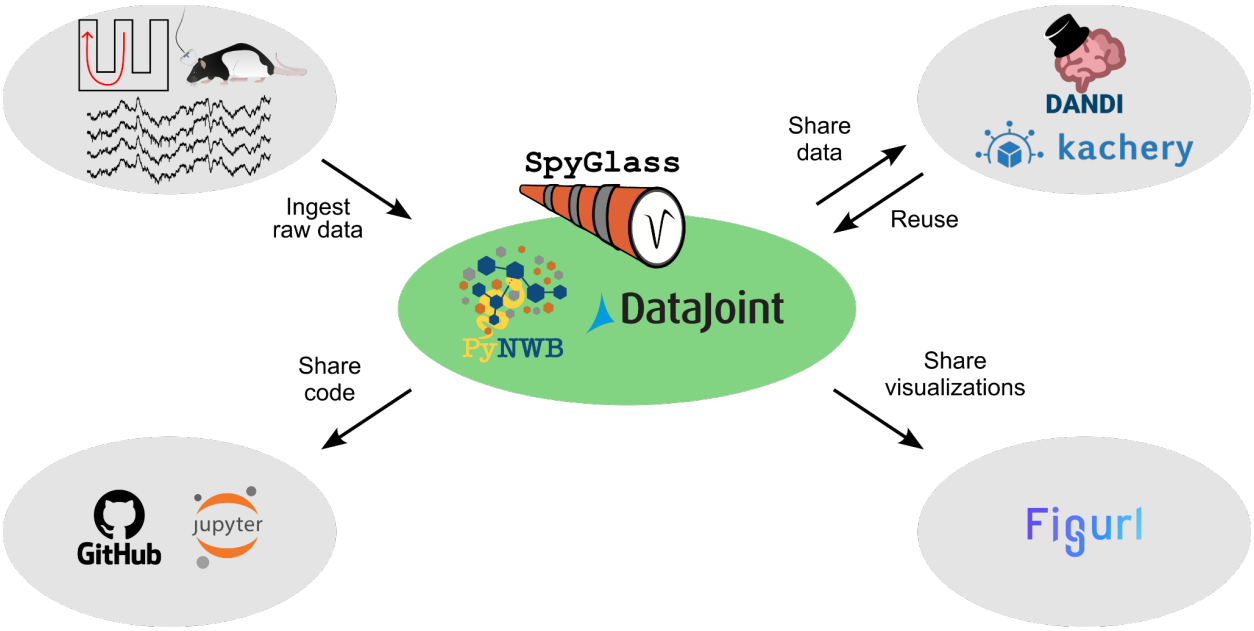

Figure 1: Overview of Spyglass. The raw data consisting of information about the animal, the behavioral task, the neurophysiology data etc. is converted to the NWB format and ingested into Spyglass pipelines via DataJoint. The raw and processed data are shared with the community by depositing them to 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 code 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). We note that while 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 to accommodate a broad range of experiments and lab-specific requirements. 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 session of an experiment 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), we include a link to them in the NWB file. In addition, virtually all intermediate results from downstream analysis pipelines are also stored in NWB (see Data Ingestion and Analysis Pipelines section). This approach ensures that all data associated with the analysis, including intermediate results, can be easily shared.

Data Ingestion and Analysis Pipelines

One significant challenge with data analysis lies in managing its complexity. Virtually every analysis involves a complex series of steps, including "preprocessing" (spike sorting for electrophysiological data, region-of-interest identification for optical physiology 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 (Yatsenko et al., 2018, 2015). DataJoint makes it possible to develop a standardized and searchable structure to organize and store each step of analysis and is concurrently accessible by multiple users. As a result, the entire provenance of a particular analysis can be easily retrieved and understood.

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, such as maintaining the integrity of data entries based on the dependency structure of the pipelines (e.g., deleting a table entry causes cascading deletion of dependent entries in downstream tables).

To begin analysis, a user first ingests the NWB file into the database, during which 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 which are stored in the NWB format and includes only the critical metadata from the original NWB file.

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

Spyglass offers ready-to-use pipelines for standard analysis tasks in systems neuroscience, such as analysis of local field potentials (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 (Chu and Kemere, 2021) and SpikeInterface (Buccino et al., 2020). These pipelines simplify these tasks and enable the application of multiple algorithms to a single data set, enhancing the robustness of results. Finally, as the analyzed results are stored in the NWB format, they can be easily fed into downstream pipelines that use other analysis software packages within the NWB ecosystem. More generally, a user can create custom pipelines that use the results of these upstream analyses (See Methods). As a result, the entire set of analyses and results that constitute a scientific finding can be implemented within the same framework.

To introduce the structure of analysis pipelines, we first describe two examples that are relatively straightforward: (i) filtering the 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.

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`), 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`). 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, making it easy to retrieve the data object and track its provenance.

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

To detect sharp-wave ripples (SWRs), a prominent LFP event in hippocampal areas, we use a pipeline illustrated in Figure 2C. The SWR detection pipeline begins with a row in the `LFP` table and involves two further steps: an additional band-pass filtering to isolate the 150-250 Hz band, and the detection of SWR events. Each of these steps uses the same basic scheme shown in Figure 2A. First, the user populates the `Parameter` tables with the time interval and the specific band-pass filter to use. These are associated with a row in the `LFP` table in the `LFPBandSelection` table, allowing the `LFPBand` table to be populated and generate an NWB file containing the ripple-band data. The user then selects `RippleParameters` to detect the ripple events and associates it with filtered data in `LFPBand` in the `RippleLFPSelection` table. Finally, the `RippleTimes` table is populated. Again, the output (the start and stop times of each ripple event) is saved in NWB.

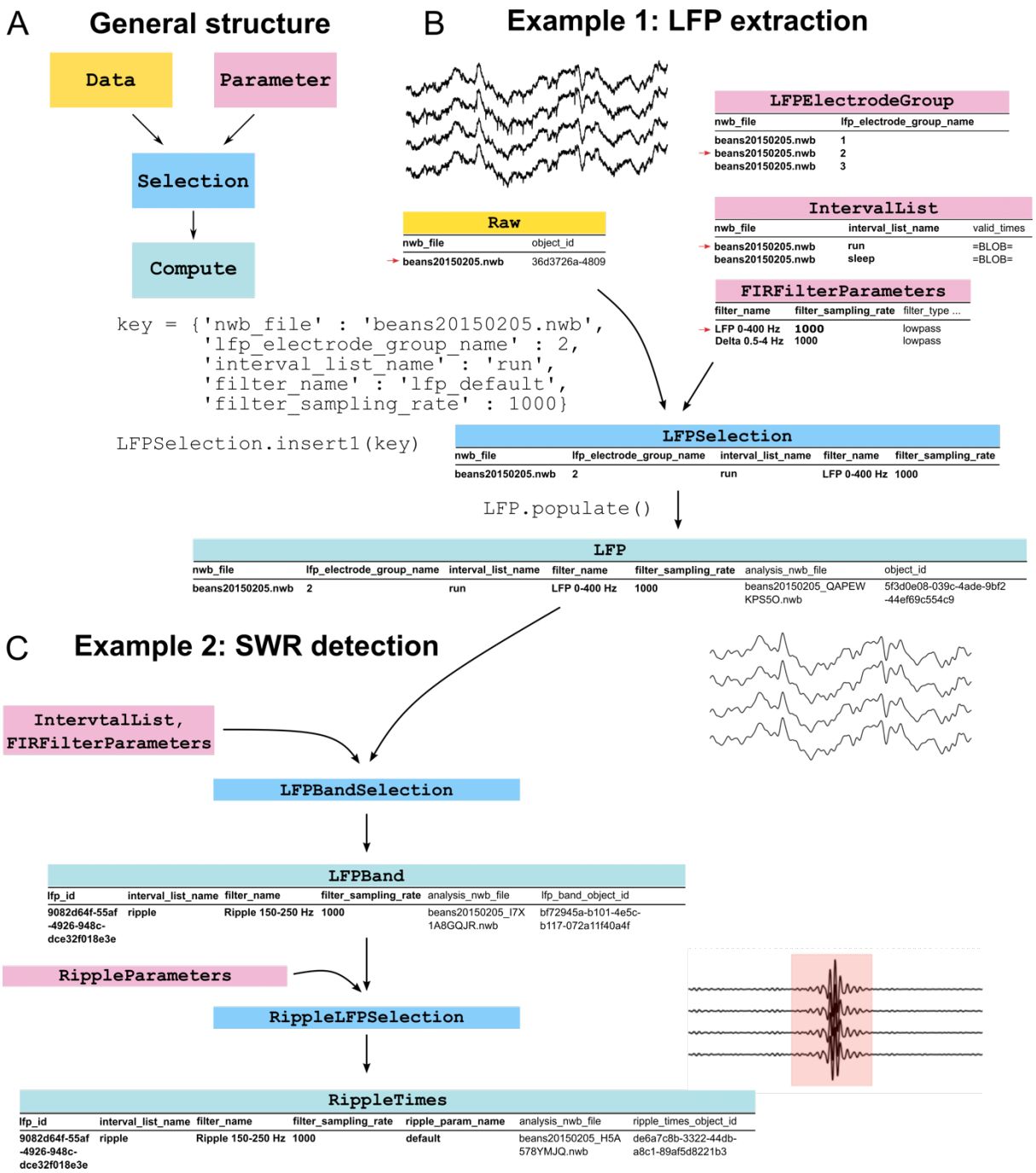

Figure 2: Analysis pipelines in Spyglass. (A) A general structure for a Spyglass pipeline. (B) Example 1: LFP extraction. Note the correspondence to the pipeline structure in panel A as shown by the color scheme. (C) Example 2: Sharp-wave ripple (SWR) detection. This pipeline is downstream of the LFP extraction pipeline and consists of two steps: (i) further extraction of a frequency band for SWR; and (ii) detection of SWR events in that band. Note that the output of LFP extraction serves as the input data for the SWR detection pipeline. In each of the compute tables, the results are saved as NWB files, and the object ID of the analysis result within the NWB file are also stored. Columns in bold are primary key. Arrows depict dependency structure within the pipeline.

Ensuring Reproducibility and Future Compatibility

A central challenge to reproducible science is balancing the rigidity required for reproducibility with the flexibility required to adopt new and better analytical approaches. DataJoint excels for reproducibility but does not include systems to ensure that new analysis pipelines can be developed and applied while older pipelines and results remain accessible. We therefore developed a versioning system and an approach to allow old and new pipelines to co-exist and feed into the same downstream pipelines.

As a result, a new version of any pipeline can be created (typically denoted by a version number at the end of the table name) and the outputs of these pipelines can be "merged" with the outputs of the previous pipeline to allow users to perform the same downstream analyses using both the old and new approaches. This facilitates comparisons of methods across old and new pipelines as well as reanalysis of older data using new pipelines. It also ensures that results that are difficult to regenerate (e.g., because it requires substantial user input) remain available even as methods change. We provide additional documentation describing this design motif and the associated helper functions (Table 1, *03_Merge_Tables*).

Extended example: spikes sorting pipeline

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

Overview

Spike sorting consists of the following steps: preprocess the recording (e.g., filter and whiten to remove noise); apply spike sorting algorithm (e.g. MountainSort4, Kilosort3, etc); curate the results (e.g. either manually or automatically by computing quality metrics); and 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 ("temporal support"). 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` feeds into both `SpikeSortingRecordingSelection` (for preprocessing the recording) and `SpikeSortingSelection` (for running spike sorting algorithms). In addition, the intervals during which artifacts (transient high-amplitude events; e.g. those caused by licking events) occur are computed and fed back into `IntervalList` (dashed arrow in Figure 3).

Iterative divergence ("one-to-many") motif for curation

Certain tasks, such as curating the output of spike sorting, must be done multiple times. For example, one might first compute quality metrics to remove noise clusters (Automatic), apply curation labels to the result with an external viewer (Manual), and compute another set of metrics to identify pairs of units to be merged (Automatic), resulting in the sequence (Automatic, Manual, Automatic). The user may generate many such sequences whose order and length cannot be predicted in advance. This presents a challenge in modeling the pipeline under the relational database framework. To enable this iterative application, we have used the following design motif. 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 would have linking information, such as the previous curation from which it descended. This allows the user to keep track of every round of curation while applying as many steps as they desire. It is also easily extensible; if new automatic curation algorithms are developed in the future, it can simply be added downstream to the `CurationV1` table and the user will be able to use it along with the existing pipelines.

Merge ("many-to-one") 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 be able 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 methods but produces output of the same datatype. 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 that new columns be added every time a new version or new data stream is added, which is cumbersome and risky. To solve this problem, we have used the merge motif, in which `Parts` tables (a data tier within DataJoint) 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*).

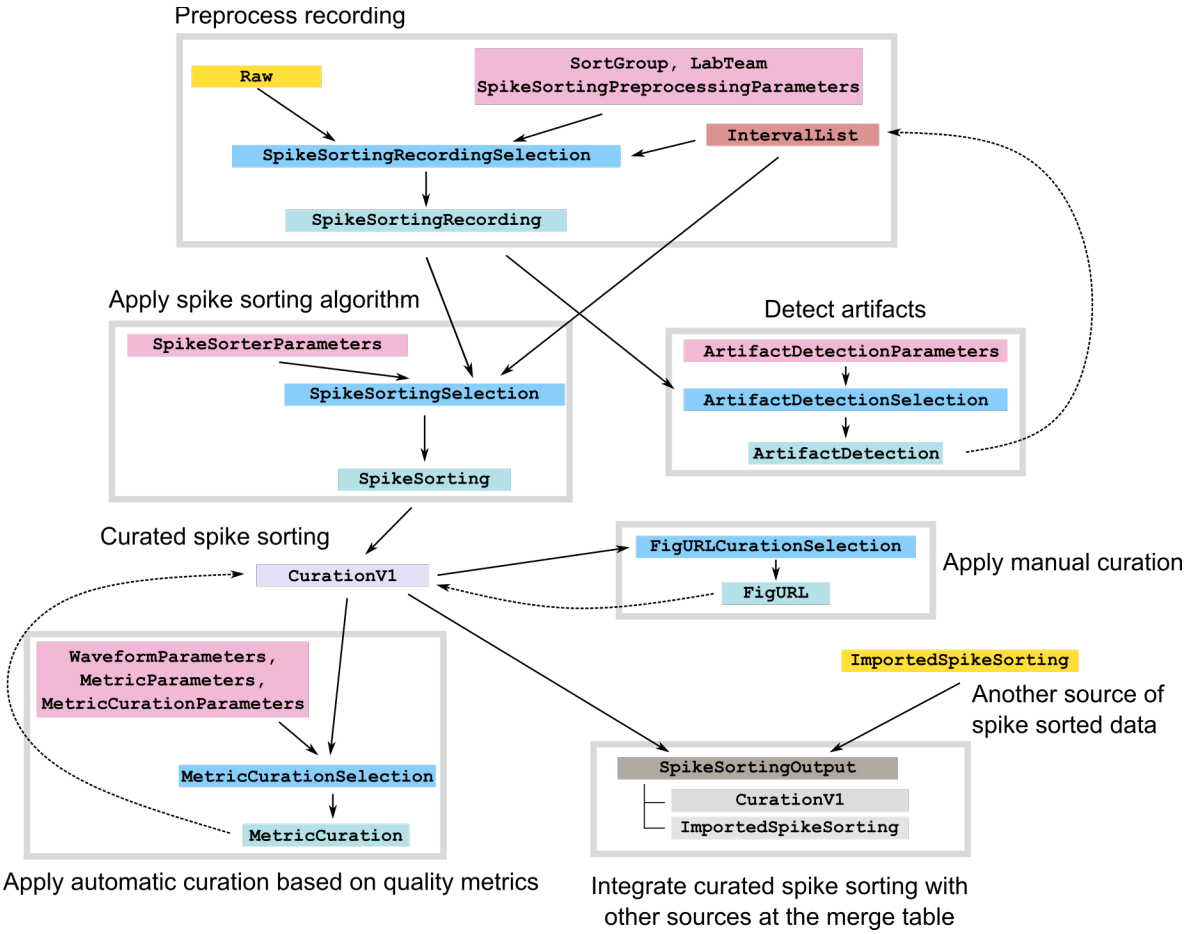

Figure 3: Spike sorting pipeline. The spike sorting pipeline consists of six components (gray boxes): preprocess recording, identify artifacts to omit from sorting, apply spike sorting algorithm, curate with quality metrics, curate manually, and merge with other sources of spike sorting for downstream processing. 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): iterative divergence for curation and merge for consolidating data streams.

Sharing Data and Analysis

Spyglass streamlines sharing of data and analysis both within and across laboratories. Within a laboratory, data and code sharing are simplified as 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 offers a secure way to share data and analyses during ongoing projects. For example, the collaborators can be given access to the database hosted by the lab. Doing so does not necessarily grant access to files containing results, as the database only stores links to NWB files and not the files themselves. The database owner can then selectively make NWB files available for download by specific collaborators using Kachery, a content-addressed sharing tool for scientific data (Figure 4A). This is done by registering collaborators' credentials on the Kachery web page and managing membership and permission settings for each project. Collaborators can then access

results, add their pipelines, and share their work. This functionality is detailed in a Jupyter Notebook tutorial (Table 1, *02_Data_Sync*).

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 linked views. Moreover, the visualizations can be shared simply by sharing a web link without the need for any local software installation or specialized hardware. Thus, collaborators anywhere in the world can access and explore the data.

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 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.

Compared to file hosting services, like Dropbox and Google Drive, or alternative architectures like the IBL data architecture, Kachery offers distinct advantages. It does not require a central location to track available files, and files can be easily retrieved via a user-friendly Python API. This decentralized approach enhances flexibility and accessibility, aligning with Spyglass's commitment to fostering collaboration and transparency in neuroscience research.

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_Curation.ipynb	Curate the results of spike sorting manually
Position processing	20_Position_Trodes.ipynb	Process information about animal's position from video recording of the behavior
	21_DLC	Detects keypoint markers with DeepLabCut
	22_DLC_Loop	Detects keypoint markers with DeepLabCut over multiple epochs
	23_Linearization	Converts 2D track geometry to 1D
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	41_Extracting_Clusterless_Waveform_Features	Extract waveform features for clusterless decoding
	42_Decoding_Clusterless	Apply the decoding algorithm from threshold crossing events.
	43_Decoding_SortedSpikes	Apply the decoding algorithm from spikes of sorted and curated units.

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

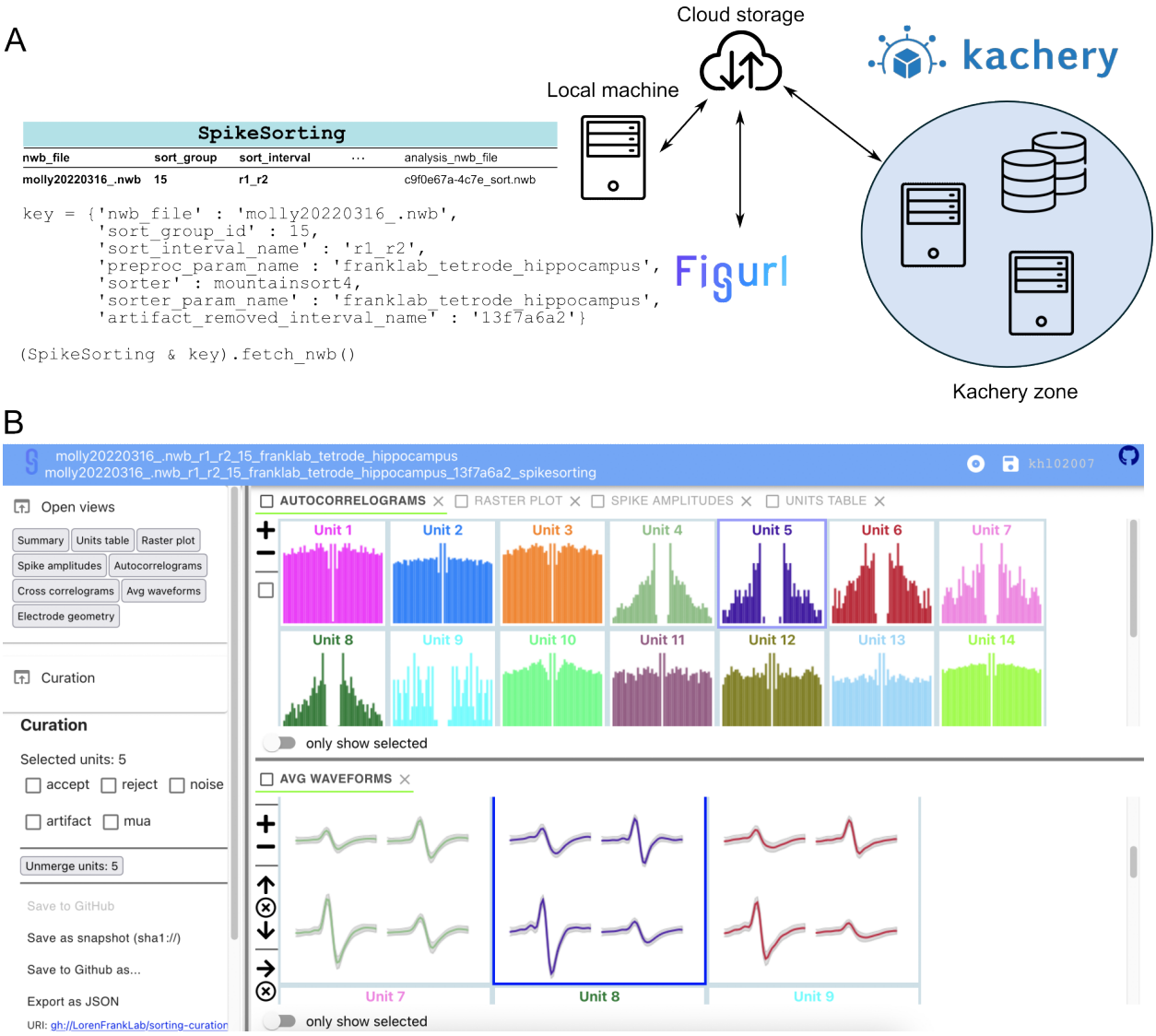

Figure 4: Sharing data and visualizations. (A) Kachery provides a convenient Python API to share data over a content-addressable cloud storage network. To retrieve data from a collaborator's Spyglass database, one can make a simple function call (`fetch_nwb`) that would pull it from a node in the Kachery 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

Spyglass can be used to apply complex analysis pipelines to NWB files from many different sources, such as those from collaborators or public archives. Despite the significant heterogeneity in NWB files, Spyglass can work with them by allowing the user to provide missing information in the form of configuration files (Table 1, *04_PopulateConfigFile*). To demonstrate the generalizability of our system, we decode the latent neural position of hippocampal spiking activity from two NWB datasets – one from Frank lab at UCSF and one from Buzsaki lab at NYU (Petersen and Buzsáki, 2020) – using a switching state space model (Denovellis et al., 2021, 2019). This is a complex, non-trivial analysis which involves integrating multiple data sources such as position and neural spiking activity. Pipelines in Spyglass enable the user to prepare the data (e.g. linearize the 2D position of the animal, perform spike sorting

or import units that have already been sorted), fit the model to the data, and then quickly visualize the results on the browser using Figurl. Because the data and parameters are tightly integrated in Spyglass, this allows the user to quickly iterate on parameters and understand the complex output of the data.

Figure 5A shows the results of running the decoding pipeline on the UCSF dataset (which can also be interactively visualized via Figurl). The decoded position (Figure 5A, second panel from the top) sweeps ahead of the actual position and then back during movement bouts (Figure 5A, third panel). This pattern recurs at 8-10 Hz, a phenomenon known as theta sequences (Foster and Wilson, 2007; Skaggs et al., 1996). When the animal is immobile (Figure 5A, fourth panel from top), the hippocampal activity becomes burst-like (Figure 5A, bottom panel) and there are systematic deviations of the decoded position from the actual position known as hippocampal replay (Buzsaki, 2015). This general pattern is replicated when the same model is applied to the NYU dataset (Figure 5B, Figurl). This highlights the ability to use Spyglass to perform sophisticated analyses with complex, multi-modal data collected in multiple labs, which could enable meta-analyses and promote data reuse.

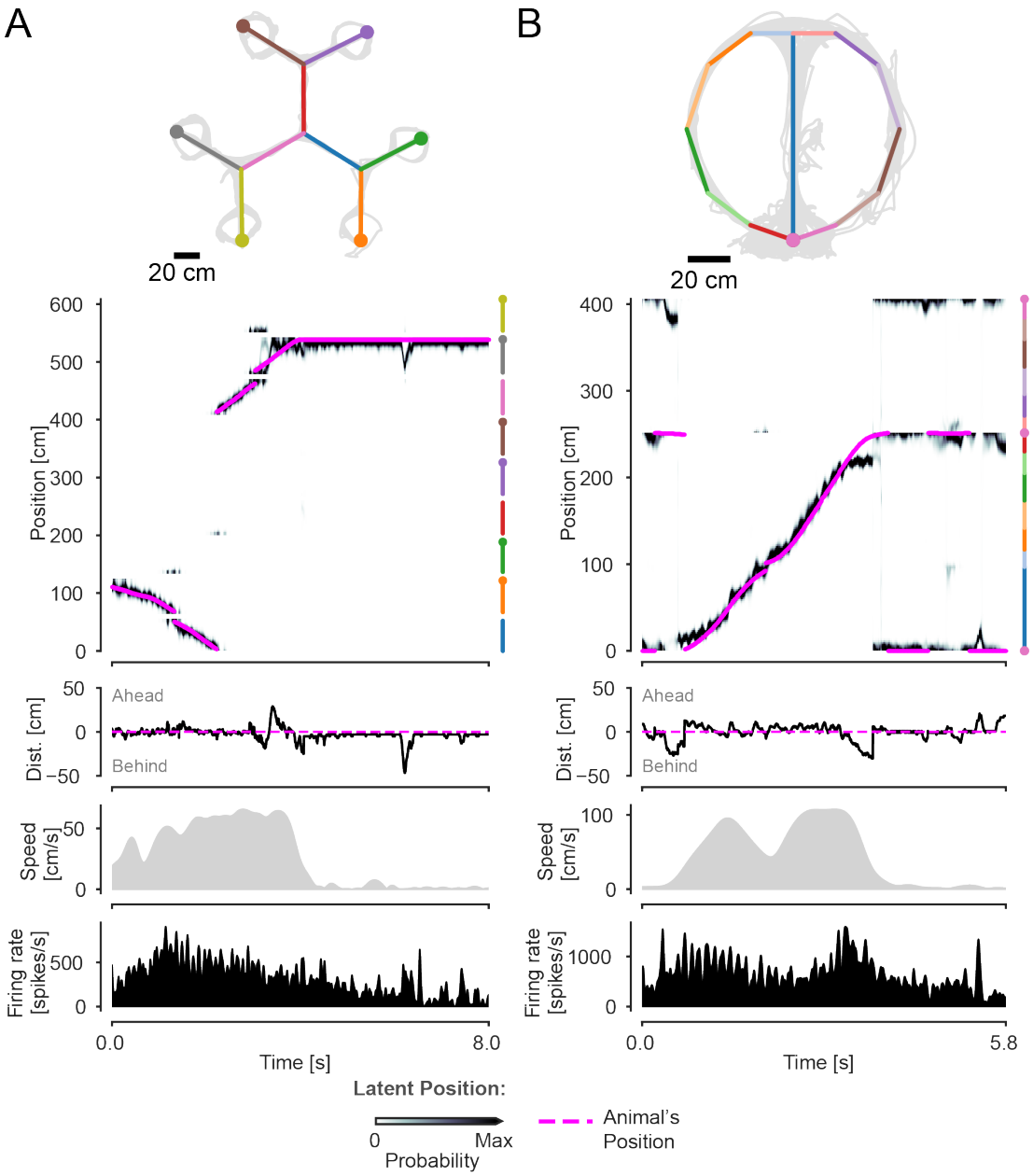

Figure 5: Applying decoding pipelines to multiple data sets from different labs (A) Decoding neural position from rat hippocampal CA1 using a clusterless state space model (UCSF dataset). In the top panel, grey lines represent positions the rat has occupied in the spatial environment. Overlayed lines in color are the track segments used to linearize position for decoding. Dots 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. Note the 8-10 Hz systematic deviations from the animal's position known as theta sequences when the animal is moving through the track, as well as "replay" - the deviations that occur when the animal is stationary. 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. Circle in the linearization represents the reward zone rather than the reward well.

Discussion

Summary of results

Scientific data analysis must be reliable, reproducible, and shareable. Spyglass is a data analysis framework designed to meet these goals. Based on a robust foundation of community standards and community-supported open-source tools, it provides data management and formalized analysis pipelines that facilitate both reliability and reproducibility. Its integration of data and visualization sharing 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 (Reimer et al., 2021), as well as publications highlighting datasets available in NWB (Chandravadia et al., 2020). More broadly, DataJoint is used by many labs with lab-specific pipelines (Yatsenko et al., 2021), but to our knowledge none of these efforts integrate cross-laboratory data and visualization tools. Parallel efforts from large collaborative groups like The International Brain Laboratory include a data architecture with similar elements, such as a system to organize neurophysiology data for sharing with collaborators and a module to automatically run analyses (The International Brain Laboratory et al., 2023). But the conversion to a standardized format (outside the collaboration or group) and public data sharing are only done following substantial analysis, hindering easy replication of the full analysis.

In 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. Furthermore, as better tools and algorithms become available, Spyglass offers a straightforward way to re-sort the data to determine how results depend on the quality of spike sorting. 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 constraints 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 requires 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 in limited cases. 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 (Ensuring Reproducibility and Future Compatibility section). 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.

The costs and benefits of doing reproducible research

The benefits of doing reproducible research are clear. For instance, 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 (Soergel, 2015). There are also more 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 – 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 (Pierré et al., 2024). 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. We believe these challenges will likely be addressed over time. There are significant efforts going into tools to simplify the raw data conversion into NWB, such as 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. We are also excited about collaborating with other groups to enhance Spyglass and include pipelines for other data modalities like optical physiology. Our hope is that 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. This work was supported by HHMI funds and NIH grants RF1MH130623 and RF1MH133778 to L.M.F. and a Helen Hay Whitney Post-doctoral Fellowship to K.H.L.

Author contributions

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

Contribution	JHB	MC	XS	AT	OR	TN	DY	JC	CK	SG	AB	LMF
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 versions up to 3.10. 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 extension:

- Time:
  - Spyglass inherits from the source NWB file either the explicit or implicit timestamps. Many NWB files from Frank lab have explicit timestamps for each sample in Unix time (seconds since 12:00 am January 1st, 1970). Others may use implicit timestamping consisting of the start time and sampling rate.
- `ElectrodeTable` and `ElectrodeGroup`:
  - `ElectrodeGroup`s 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 accidentally deletes.

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 easily link analysis NWB files to be shared to 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 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 to 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 will be uploaded to DANDI. The Buzsaki lab data was obtained from DANDI (Dandiset 59, `sub-MS22_ses-Peter-MS22-180629-110319-concat_desc-processed_behavior+ecephys.nwb`). 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*). The additional code to generate the figure from output of the decoding pipeline will be made public in the near future.

Reference

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

BRAIN Initiative, 2019. Notice of Data Sharing Policy for the BRAIN Initiative.

Buccino, A.P., Hurwitz, C.L., Garcia, S., Magland, J., Siegle, J.H., Hurwitz, R., Hennig, M.H., 2020. SpikeInterface, a unified framework for spike sorting. eLife 9, e61834. https://doi.org/10.7554/eLife.61834

Buzsaki, G., 2015. Hippocampal sharp wave-ripple: A cognitive biomarker for episodic memory and planning. Hippocampus 25, 1073–188.

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., Valiante, T.A., Mamelak, A.N., Rutishauser, U., 2020. A NWB-based dataset and processing pipeline of human single-neuron activity during a declarative memory task. Sci. Data 7, 78. https://doi.org/10.1038/s41597-020-0415-9

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

De Vries, S.E., Siegle, J.H., Koch, C., 2023. Sharing neurophysiology data from the Allen Brain Observatory. eLife 12, e85550. https://doi.org/10.7554/eLife.85550

Denovellis, E.L., Frank, L.M., Eden, U.T., 2019. Characterizing hippocampal replay using hybrid point process state space models. Presented at the 2019 53rd Asilomar Conference on Signals, Systems, and Computers, IEEE, pp. 245–249.

Denovellis, E.L., Gillespie, A.K., Coulter, M.E., Sosa, M., Chung, J.E., Eden, U.T., Frank, L.M., 2021. Hippocampal replay of experience at real-world speeds. Elife 10. https://doi.org/10.7554/eLife.64505

Foster, D.J., Wilson, M.A., 2007. Hippocampal theta sequences. Hippocampus 17, 1093–9.

Ghosh, S., Baker, C., Choudhury, R., Dichter, B., Jarecka, D., Dehghani, N., Ioanas, H., Lamanna, D., Nesbitt, J., Vandenbrugh, M., Wodder, J., Halchenko, Y., 2023. Dandi: an archive and collaboration space for neurophysiology projects. Presented at the Neuroscience.

Goble, C., Cohen-Boulakia, S., Soiland-Reyes, S., Garijo, D., Gil, Y., Crusoe, M.R., Peters, K., Schober, D., 2020. FAIR Computational Workflows. Data Intell. 2, 108–121. https://doi.org/10.1162/dint_a_00033

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

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., Medina, R.E., Calcaterra, L., Dreosti, E., Paton, J.J., Kampff, A.R., 2015. Bonsai: an event-based framework for processing and controlling data streams. Front. Neuroinformatics 9. https://doi.org/10.3389/fninf.2015.00007

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

Matelsky, J.K., Rodriguez, L.M., Xenes, D., Gion, T., Hider, R., Wester, B.A., 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). Presented at the 2021 43rd Annual International Conference of the IEEE

Engineering in Medicine & Biology Society (EMBC), IEEE, Mexico, pp. 2413–2418. https://doi.org/10.1109/EMBC46164.2021.9630199
Nath, T., Mathis, A., Chen, A.C., Patel, A., Bethge, M., Mathis, M.W., 2019. Using DeepLabCut for 3D markerless pose estimation across species and behaviors. Nat Protoc 14, 2152–2176. https://doi.org/10.1038/s41596-019-0176-0
Pachitariu, M., Steinmetz, N., Kadir, S., Carandini, M., Harris, K., 2016. Kilosort: realtime spike-sorting for extracellular electrophysiology with hundreds of channels. bioRxiv 061481. https://doi.org/10.1101/061481
Petersen, P.C., Buzsáki, G., 2020. Cooling of Medial Septum Reveals Theta Phase Lag Coordination of Hippocampal Cell Assemblies. Neuron 107, 731-744.e3. https://doi.org/10.1016/j.neuron.2020.05.023
Pierré, A., Pham, T., Pearl, J., Datta, S.R., Ritt, J.T., Fleischmann, A., 2024. A perspective on neuroscience data standardization with Neurodata Without Borders.
Reimer, M.L., Bangalore, L., Waxman, S.G., Tan, A.M., 2021. Core principles for the implementation of the neurodata without borders data standard. J. Neurosci. Methods 348, 108972. https://doi.org/10.1016/j.jneumeth.2020.108972
Rübel, O., Tritt, A., Ly, R., Dichter, B.K., Ghosh, S., Niu, L., Baker, P., Soltesz, I., Ng, L., Svoboda, K., Frank, L., Bouchard, K.E., 2022. The Neurodata Without Borders ecosystem for neurophysiological data science. eLife 11, e78362. https://doi.org/10.7554/eLife.78362
Sanchez, M., Moore, D., Johnson, E.C., Wester, B., Lichtman, J.W., Gray-Roncal, W., 2022. Connectomics Annotation Metadata Standardization for Increased Accessibility and Queryability. Front. Neuroinformatics 16, 828458. https://doi.org/10.3389/fninf.2022.828458
Siegle, J.H., López, A.C., Patel, Y.A., Abramov, K., Ohayon, S., Voigts, J., 2017. Open Ephys: an open-source, plugin-based platform for multichannel electrophysiology. J. Neural Eng. 14, 045003. https://doi.org/10.1088/1741-2552/aa5eea
Skaggs, W.E., McNaughton, B.L., Wilson, M.A., Barnes, C.A., 1996. Theta phase precession in hippocampal neuronal populations and the compression of temporal sequences. Hippocampus 6, 149–72.
Soergel, D.A.W., 2015. Rampant software errors may undermine scientific results. F1000Research 3, 303. https://doi.org/10.12688/f1000research.5930.2
Teeters, J.L., Godfrey, K., Young, R., Dang, C., Friedsam, C., Wark, B., Asari, H., Peron, S., Li, N., Peyrache, A., Denisov, G., Siegle, J.H., Olsen, S.R., Martin, C., Chun, M., Tripathy, S., Blanche, T.J., Harris, K., Buzsáki, G., Koch, C., Meister, M., Svoboda, K., Sommer, F.T., 2015. Neurodata Without Borders: Creating a Common Data Format for Neurophysiology. Neuron 88, 629–634. https://doi.org/10.1016/j.neuron.2015.10.025
The International Brain Laboratory, Acerbi, L., Aguillon-Rodriguez, V., Ahmadi, M., Amjad, J., Angelaki, D., Arlandis, J., Ashwood, Z.C., Banga, K., Barrell, H., Bayer, H.M., Benson, J., Benson, B., Bhagat, J., Birman, D., Bougrova, K., Boussard, J., Bruijns, S.A., Carandini, M., Catarino, J., Cazettes, F., Dan, Y., Davatolagh, F., Dayan, P., Denève, S., Dong, L.L., Engel, T., Fabbri, M., Fiete, I., Findling, C., Freitas-Silva, L., Ganguli, S., Gercek, B., Ghani, N., Gordeliy, I., Haetzel, L.M., Hausser, M., Hiratani, N., Hofer, S., Hu, F., Huber, F., Hurwitz, C., Khanal, A., Krasniak, C.S., Krishnagopal, S., Krumin, M., Langdon, C., Latham, P., Lau, P., Lee, H., Liu, A., Mainen, Z.F., Vergara, H.M., Mcgrory, C., McMannon, B., Meijer, G.T., Melin, M., Meshulam, L., Miska, N.J., Mitelut, C., Mohammadi, Z., Mrsic-Flogel, T., Murakami, M., Noel, J.-P., Nylund, K., Vazquez, A.P., Paninski, L., Pezzotta, A., Picard, S., Pillow, J.W., Pouget, A., Roth, N., Roy, N.A., Saniee, K., Schaeffer, R., Shi, Y., Socha, K.Z., Soitu, C., Svoboda, K., Taheri, M., Tessereau, C., Urai, A.E., Varol, Erdem., Wells, M.J., Whiteway, M.R., Windolf, C., Witten, I., Wool, L.E., Zador, A.M., Bonacchi, N., Chapuis, G.A., Churchland, A.K.,

DeWitt, E.E.J., Faulkner, M., Harris, K.D., Huntenburg, J.M., Hunter, M., Laranjeira, I.C., Rossant, C., Sasaki, M., Schartner, M.M., Shen, S., Steinmetz, N.A., Walker, E.Y., West, S.J., Winter, O., Wells, M.J., 2023. A modular architecture for organizing, processing and sharing neurophysiology data. Nat. Methods 20, 403–407. https://doi.org/10.1038/s41592-022-01742-6

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

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., Bouwman, J., Brookes, A.J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C.T., Finkers, R., Gonzalez-Beltran, A., Gray, A.J.G., Groth, P., Goble, C., Grethe, J.S., Heringa, J., 't Hoen, P.A.C., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S.J., Martone, M.E., Mons, A., Packer, A.L., Persson, B., Rocca-Serra, P., Roos, M., van Schaik, R., Sansone, S.-A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M.A., Thompson, M., van der Lei, J., van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., Mons, B., 2016. The FAIR Guiding Principles for scientific data management and stewardship. Sci. Data 3, 160018. https://doi.org/10.1038/sdata.2016.18

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

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

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