scientific data

Check for updates

OPEN

ARTICLE

Improving data sharing and knowledge transfer via the Neuroelectrophysiology Analysis Ontology (NEAO)

Cristiano A. Köhler[1,2]✉, Sonja Grün[1,3,4] & Michael Denker[1]

Describing the analysis of data from electrophysiology experiments investigating the function of neural systems is challenging. On the one hand, data can be analyzed by distinct methods with similar purposes, such as different algorithms to estimate the spectral power content of a measured time series. On the other hand, different software codes can implement the same analysis algorithm, while adopting different names to identify functions and parameters. These ambiguities complicate reporting analysis results, e.g., in a manuscript or on a scientific platform. Here, we illustrate how an ontology to describe the analysis process can assist in improving clarity, rigour and comprehensibility by complementing, simplifying and classifying the details of the implementation. We implemented the Neuroelectrophysiology Analysis Ontology (NEAO) to define a vocabulary and to standardize the descriptions of processes for neuroelectrophysiology data analysis. Real-world examples demonstrate how NEAO can annotate provenance information describing an analysis. Based on such provenance, we detail how it supports querying information (e.g., using knowledge graphs) that enable researchers to find, understand and reuse analysis results.

Background & Summary

Neuroelectrophysiology is a common approach to investigate the function of the nervous system using electrodes to measure electrical properties of neural tissue[1–3]. Several techniques to perform neuroelectrophysiological recordings are available, such as intracellular recordings, recordings of single unit spiking activity, local field potential (LFP) recordings, or electroencephalography (EEG), leading to a large body of available data[4]. This diversity allows researchers to gain an understanding of the nervous system's activity dynamics ranging from individual cells[5] to complex neural networks[2,6,7]. Deriving insights from these recorded signals requires the careful analysis of such data, i.e., transforming the data into meaningful measures or visual representations. The nature of such an analysis of neuroelectrophysiology experiments is often highly specialized and complex, such that a detailed description of the data analysis process is essential for a reliable interpretation of the findings.

Achieving a detailed description of the processes involved in analyzing neuroelectrophysiology data must consider three aspects. First, a given feature of the brain activity can be understood from the recorded data using multiple analysis methods that are in part complementary and in part overlapping. The interpretation of the results will depend on any strengths or caveats associated with the chosen method[8,9]. One example is the investigation of brain oscillations using the power spectral density (PSD) of the signal recorded by an electrode. There exist a number of distinct algorithms to compute the concept of a PSD estimate from a recorded signal[10–12]. Although they all produce a similar measure (i.e., the power density for specific frequencies in the signal), the interpretation of the values found for one particular frequency will depend on the features of the algorithm, such as resolution and smoother estimates. Second, different software codes implement one specific analysis method. Revisiting the PSD computation as an example, several software toolboxes for data analysis can implement a method such as the one based on the Welch[10] algorithm (Elephant[13], MNE[14], etc.; see

[1]Institute for Advanced Simulation (IAS-6), Jülich Research Centre, Jülich, Germany. [2]RWTH Aachen University, Aachen, Germany. [3]Theoretical Systems Neurobiology, RWTH Aachen University, Aachen, Germany. [4]JARA-Institute Brain Structure-Function Relationships (INM-10), Jülich Research Centre, Jülich, Germany. ✉e-mail: c.koehler@fz-juelich.de

ref. [15] for a review). This may result in subtle differences in the final estimates depending on the toolbox used even though the starting data is the same (see ref. [15] for an example comparing several toolboxes regarding spectral analysis methods). Finally, neuroelectrophysiology recording techniques evolved over the years. Early experiments using coarse field recordings (e.g., EEG and LFP) or isolating a few single neuronal units required simple analysis methods (e.g., simple cross-correlations or spectral measures)[1,16]. Recent technical advances such as multi-electrode arrays and high-density recording systems have allowed for the recording of hundreds of neurons across many areas or for investigating oscillations across large networks[6,17,18]. This progress was accompanied by increased complexity of the analytical methods designed to extract meaningful features from complex and often high-dimensional data (some of the most recent developments are summarized in ref. [19], ref. [20], and ref. [21]). Therefore, gaining insights into the neuroelectrophysiology data analysis requires an unambiguous description of both the specific methodology and the software implementations, and it is essential to systematically describe, standardize, and integrate the diverse and evolving methods used in electrophysiology data analysis. Ambiguities in the description may hinder the interpretation of results and, consequently, the understanding of the underlying neural processes. When methodological choices and the tools used are not transparently reported or standardized, it becomes difficult to determine whether any observed effects reflect neural phenomena or artifacts of the analysis pipeline (e.g., wrong choice of method or parameters or errors in the software code). In addition, the lack of standardization complicates the comparison of results across studies. A clear example of two conceptually similar analysis approaches producing non-comparable results was demonstrated for competing methods used to detect the onset of neuronal slow waves[22]. Overall, ambiguities undermine the reliability and reproducibility of findings, which may limit drawing robust conclusions about the function of the nervous system.

The use of formal ontologies may help to describe all the processes involved in the neuroelectrophysiology data analysis in a manner that facilitates gaining insights into the results[23–26]. An ontology provides a framework to organize the knowledge of a particular domain field by defining concepts and entities without redundancy and ambiguity while providing semantically-enriched relationships[27]. Therefore, if the processes and results associated with the analysis of neuroelectrophysiology data were represented using an ontology, they should be comprehensible and traceable independent of the specific methodology or the analysis software and programming language used. The use of an ontology to describe the analysis brings several advantages: (i) the adoption of a unified vocabulary, (ii) the standardization of descriptions of the processes involved, and (iii) achieving a machine-readable representation of the analysis separate from its realization as software code such that it is possible to query information based on the research questions. The generic description of an analysis result will therefore facilitate the FAIR-ness[28] of analysis results by improving their findability and interoperability. In collaborative scenarios, an ontology will facilitate the knowledge transfer of shared results since each step during the analysis can be annotated to identify their similarities and differences (see ref. [26] and ref. [25] for perspectives on using ontologies in neuroscience and biomedical research).

Several ontologies that could be considered for the description of experimental data analysis are already developed in biomedical sciences and biomedical research in general (OBI[29], OBCS[30], BRO[31], EDAM[32]), or more specifically for neuroscience (NIFSTD[33], CNO[34]) and electrophysiology (NEMO[35], OEN[36], ICEPO[37], OBI_IEE[38], and ref. [39]). Complementing these efforts, the Metadata4Ing[40] and REPRODUCE-ME[41] ontologies provide scaffolds for the description of scientific processing workflows. However, the existing ontologies lack the specificity to describe electrophysiology data analysis, particularly in connecting the analysis methods with their software implementations. They generally focus on broad terms and data collection rather than detailed analysis methods, which allows only a coarse-grained and high-level description of the computational analysis steps. Therefore, those ontologies are not explicitly tailored to describe the workflow required for analyzing neuroelectrophysiology data in a conceptual and semantically rich manner. To address this gap, we present the Neuroelectrophysiology Analysis Ontology (NEAO), which aims to define a unified vocabulary and standardize the descriptions of the processes involved in analyzing neuroelectrophysiology data. We show its application in real-world scenarios where the NEAO was used to annotate the provenance information from different analyses and highlight how it can query information, facilitating finding and obtaining insights on the results.

Results

The Neuroelectrophysiology Analysis Ontology. *Overview of the NEAO model.* The design of NEAO considers that the analysis of neuroelectrophysiology data is composed of a sequence of small atomic steps, each performing one specific action to generate, transform or characterize a piece of data. For example, let us consider a scenario of plotting the PSD of the LFP time series obtained from the recording of one extracellular electrode implanted into a brain area and that was saved into a data file (Fig. 1). First, one may load the raw data from the file into a data structure containing the voltage time series acquired by the recording equipment. The LFP is the low-frequency component of the extracellular signal (here defined as below 250 Hz), and therefore a low-pass filter with a cutoff frequency of 250 Hz is applied to obtain the LFP time series. Finally, the PSD is computed from the filtered data, resulting in an array of values corresponding to the power density estimates for a set of frequency values. This power spectrum may be plotted and saved to a file. In this toy example, each step takes some data as input and produces data as output, and steps may be controlled by one or more parameters (e.g., the cutoff frequency parameter of the filter step controls how the filtering is applied to the raw data).

We propose the NEAO model (Fig. 2a) to describe such a scenario. The NEAO ontology is built on the central **AnalysisStep** class to model the atomic steps of the analysis. It represents any process that generates new data entities (e.g., generating artificial LFP data) or performs specific operations aimed at extracting additional information during the analysis using existing data entities. These include data transformations (e.g., filtering the raw signal into the LFP) or the computation of new, derived data (e.g., obtaining the PSD from the LFP

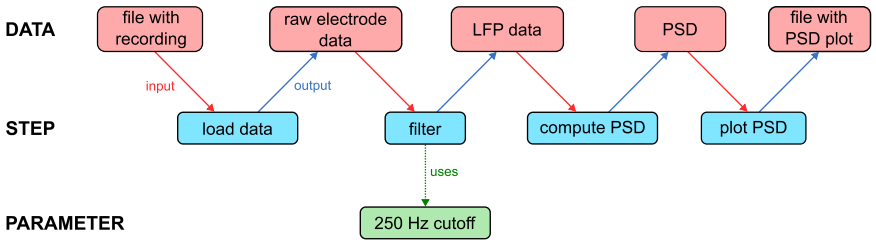

Fig. 1 Conceptual view of an example analysis of neuroelectrophysiology data. The analysis comprises a sequence of atomic steps, each performing one specific action. This example analysis produces a plot of the power spectral density (PSD) of the local field potential (LFP) obtained from a single extracellular electrode recording. A series of four steps (blue rectangles) is executed sequentially, starting from a data file that stores the signal obtained during the recording session. Data is loaded and low-pass filtered to extract the LFP, then the PSD is computed and plotted, and the plot is saved to a file. Each individual step is associated with specific input and output data elements (red rectangles). Notably, the data is transformed throughout the steps, and such transformations may be controlled by specific parameters (green rectangle). In this example, the filtering step used a low pass frequency cutoff parameter, which defines the LFP component of the electrode signal.

signal). In NEAO, many specific classes derived from **AnalysisStep** are defined to describe specific steps in the analysis. The complete specification is available on the documentation page at http://purl.org/neao/steps.

Two further classes complete the core of the ontology model. The **Data** class represents information used during the analysis and constitutes the inputs and outputs of the analysis steps. It can represent either data obtained from biology in an electrophysiology recording (or equivalent data generated by a simulation) or the generated or transformed data resulting from an analysis step. In the example of the computation of the PSD from an electrode signal, the raw signal or filtered LFP time series, the array containing the resulting PSD, and the plot are examples of the **Data** class. The **AnalysisParameter** class represents information that does not provide data to the step but changes its behavior when producing the output, for example, the 250 Hz low-frequency cutoff in the filter step, which sets the bandwidth of the output signal.

Three properties model the relationships between the main classes. **hasInput** and **hasOutput** point to individuals of the **Data** class, representing data that was an input or output of the analysis step. In the example above, the step where the filter was applied will have the raw time series as input and the filtered LFP signal as the output. The **usesParameter** property points to an individual of the **AnalysisParameter** class and corresponds to a parameter used by the analysis step. Several properties may be added to the entity representing a parameter to structure the information of the value. For instance, the parameter indicating the low pass filter setting of 250 Hz could be represented with two properties, one storing the literal integer value *250* and another the literal string with the unit "*Hz*", making the value machine-readable. However, no specific properties for the **AnalysisParameter** class are predefined in NEAO for that purpose, allowing the use of existing ontologies to structure such information when applicable (e.g., QUDT[42] for describing physical quantities).

The OWL source files of NEAO are divided into submodules, each associated with a single namespace. Table 1 describes each module regarding the source file, namespace, and contents. The full documentation can be accessed at http://purl.org/neao.

Solving ambiguities in descriptions with NEAO. Each of the three main NEAO classes represents specific entities in the context of the analysis of neuroelectrophysiology data. However, a frequently encountered situation when referring to the steps, data, or parameters of an analysis is the use of abbreviations or synonyms, which may lead to ambiguity in the meaning of the names. One example is the abbreviation *PSD*, which refers to the term power spectral density but is also often called a power spectrum. First, NEAO establishes a controlled vocabulary when naming the class (e.g., *PowerSpectralDensity* is used to represent the power spectral density data in the analysis). Second, NEAO adopts several annotation properties to structure extended information about the concept modeled by a class. Every class is assigned one label, defined using the Simple Knowledge Organization System[43] (SKOS) annotation *skos:prefLabel* that defines a string literal with a human-readable label that is the chosen term to refer to an individual of that class (e.g., for the *PowerSpectralDensity* class representing the power spectral density data, the label is *power spectral density*). Every class also has one RDFS *rdfs:comment* annotation providing the human-readable description of what the class represents in the context of neuroelectrophysiology data analysis. In addition, one or more string literals may be defined with the SKOS *skos:altLabel* property to provide a set of alternative labels that represent synonyms usually referenced in the literature (e.g., *spectrum*). Finally, one or more string literals defining abbreviations used to refer to an individual of the class may be defined by the **abbreviation** property defined in NEAO (e.g., *PSD* for the power spectral density). By this approach, NEAO allows the use of those annotations to disambiguate the names while structuring and consolidating the diversity of terms that may be present in the literature.

A second source of ambiguity is the implementation of a specific analysis method by different software codes. For example, the Welch method to estimate the PSD is available in several open-source toolboxes to analyze neuroelectrophysiology data (e.g., Elephant[13], MNE[14], NiTime[44], FieldTrip[45], BrainStorm[46], Chronux[47]) or even more general scientific environments or toolboxes such as MATLAB or the SciPy[48] package for Python. Therefore, the description must accommodate the ambiguity in the software implementation of the code associated with the step. NEAO defines two main classes to structure this information: **SoftwareImplementation**

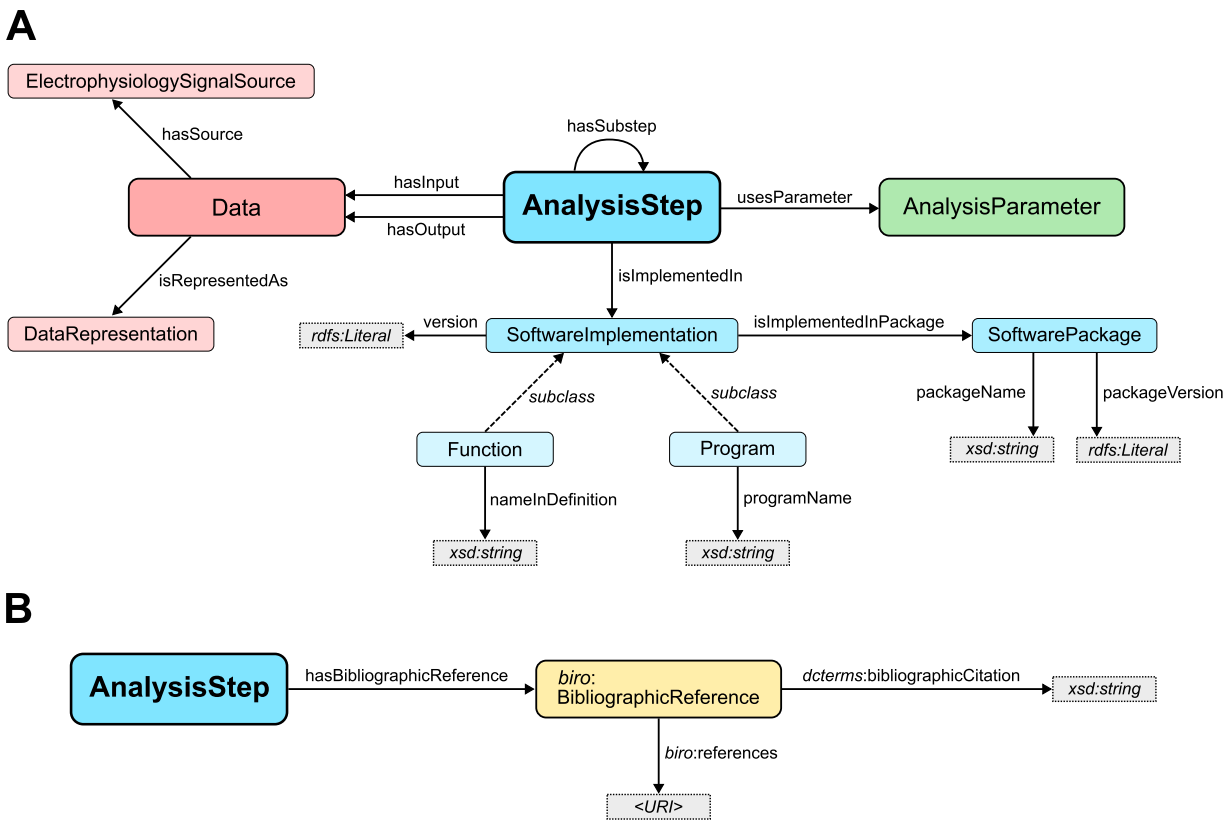

Fig. 2 Core model used by the Neuroelectrophysiology Analysis Ontology (NEAO). (**A**) Each atomic step in the analysis is represented by the **AnalysisStep** class (large blue rectangle). It is bound to data inputs and outputs by the specific properties **hasInput** and **hasOutput** which point to elements of the **Data** class (large red rectangle). The description of parameters that control the behavior of the analysis step is achieved by the **usesParameter** property, which points to elements from the **AnalysisParameter** class (large green rectangle). To describe the software implementation associated with the analysis step, the **Function, Program**, and **SoftwarePackage** classes are used through specific properties, supporting the identification of the code (e.g., strings defining the function, program, and package names) and its version. To add an extended description of the data associated with an analysis step, the **DataRepresentation** class supports detailing how a particular input or output is represented (e.g., array, matrix, scalar value) and the **ElectrophysiologySignalSource** class may be used to identify the source associated with the signal represented by **Data** (e.g., EEG, extracellular recording, extracellular recording from a particular brain area). (**B**) Details about the literature associated with an **AnalysisStep** class are provided by annotations using the **hasBibliographicReference** property, which points to an individual of the BiRO *BibliographicReference* class, whose properties define a string with a textual bibliographic citation, and the URI that allows reaching the bibliographic resource (e.g., a resolvable DOI URL).

and **SoftwarePackage** (Fig. 2a). **SoftwareImplementation** represents the primary code source used to execute the analysis step. It takes any data input, performs the transformations, and generates the outputs. **SoftwarePackage** represents a collection of software and aims to describe the bundling of distinct pieces of code, such as in a toolbox providing multiple functionalities for analyzing neuroelectrophysiology data. The **SoftwareImplementation** comprises two distinct subclasses representing the primary approaches of implementing the code for the analysis step: **Program** and **Function**. **Program** represents a full script or a compiled executable that the operating system can call to perform the analysis step (e.g., an executable that would read a file, perform the computation of the PSD using the Welch method, and save a file with the PSD). **Function** represents a smaller and reusable code that can be used as a building block when writing a more extensive program that executes a sequence of steps in the analysis. For example, to compute a PSD using the Welch method, one could write a Python script that imports the *welch_psd* function from the *spectral* module of the Elephant package, which is executed at some point in the script. However, the script performs several additional steps, cf., Fig. 1. The details of **SoftwareImplementation** and **SoftwarePackage** individuals are provided through a set of properties. For **SoftwareImplementation**, the property **version** defines the version of the program or function. For **Function**, the **nameInDefinition** property defines the name used in the function declaration and that is used within programs that use the function (e.g., the name after *def* in Python functions). For **Program**, the **programName** property defines the program's name as it is published. The **SoftwarePackage** individuals have the **packageVersion** property to define the package version and **packageName** property to define the package name. Finally, the relationship between a **SoftwareImplementation** and a **SoftwarePackage** is established

Module	OWL source	Namespace prefix	Namespace URI	Contents
Root	neao.owl	neao	http://purl.org/neao#	Top-level grouping of the files, importing all modules, and defining the ontology metadata, such as description and version information.
Base	base/base.owl	neao_base	http://purl.org/neao/base#	Top-level classes of the NEAO model that are imported by other modules.
Data	data/data.owl	neao_data	http://purl.org/neao/data#	Subclasses of **Data**, defining specific data entities and their semantic groupings.
Steps	steps/steps.owl	neao_steps	http://purl.org/neao/steps#	Subclasses of **AnalysisStep**, defining specific analysis steps and their semantic groupings.
Parameters	parameters/parameters.owl	neao_params	http://purl.org/neao/parameters#	Subclasses of **AnalysisParameter**, defining specific parameters and their semantic groupings.
Bibliography	bibliography/bibliography.owl	neao_bib	http://purl.org/neao/bibliography#	Define individuals with the bibliographic references used to annotate **AnalysisStep** classes.

Table 1. Modular structure of NEAO. The core ontology model is implemented in a base module, and each defined main class (**AnalysisStep, Data**, and **AnalysisParameter**) is expanded in individual modules. This allows the definition of specific namespaces for the detailed classes derived from each of the three main classes. Bibliographic information is defined in an additional module containing the individuals representing bibliographic citations. A root module provides the main ontology metadata and binds all the modules.

through the property **isImplementedInPackage**, and between the **AnalysisStep** and **SoftwareImplementation** through the property **isImplementedIn** (Fig. 2a).

Another source of ambiguity are analysis steps for which multiple conceptually similar methods are available that produce similar results. This methodological ambiguity can occur in two ways. The first is when a method evolves such that the underlying algorithm or assumptions, and consequently the results, differ. For example, the PSD estimation with the periodogram method has a high variance. To address this, Bartlett[49] introduced an improved approach by dividing the signal into non-overlapping segments, computing the periodogram of each, and averaging the results. This reduced variance at the cost of frequency resolution. Building on Bartlett's method, Welch[10] proposed another refinement by using overlapping segments and applying a window function to each segment before computing their periodograms. This improved the statistical stability and reduced spectral leakage. Therefore, using the periodogram approach, each PSD estimation method allowed for better control over the variance and frequency resolution tradeoff. Typically, the description of an analysis method and its variations is associated with a specific publication. To address this source of ambiguity, NEAO provides the **hasBibliographicReference** annotation property pointing to an individual of the Bibliographic Reference Ontology (BiRO)[50] *biro:BibliographicReference* class that aims to identify the bibliographic resource with the details of the method represented by the class (Fig. 2b). The *biro:BibliographicReference* individuals are defined with the DCMI Terms[51] *dcterms:bibliographicCitation* property providing a string literal with the textual citation of the reference and a BiRO *biro:references* property pointing to another individual with an identifier that allows reaching the resource (e.g., the URL with the DOI or the ISBN for a book). The bibliographic information annotations structure the bibliographic description to allow reaching the detailed and unambiguous definitions of the analysis performed in one step (in our example, disambiguating the Bartlett or the Welch approach to compute a PSD).

A related type of methodological ambiguity is the case where conceptually distinct methods produce a conceptually similar result. One example is the estimation of a PSD using the periodogram, multitaper[11] or wavelet-based[52] methods. Although the former two are based on the Fourier transform of a time series, the multitaper approach introduces additional steps to reduce noise in the estimate. In contrast, wavelet-based methods obtain the estimates by a distinct mathematical formalism. Nevertheless, all three methods will produce similar results, but their differences will translate into strengths and caveats that affect the final results. To address this form of ambiguity, NEAO introduces the concept of groups of similar analysis methods, which will be discussed in greater detail in the following section.

Grouping methods according to semantic meaning. The classes defined by NEAO allow a fine-grained description of the steps, data, and parameters used during the analyses. However, when aiming to gain insights into a given analysis, queries to its description using NEAO may be targeted to answer questions of a more general nature. For example, describing a step as the computation of a PSD using the Welch method provides specific details of what is performed in that analysis step (i.e., the specific method and associated parameters). If one is interested in obtaining information on the existence of any PSD estimate in the analysis description, however, individually querying for (all) specific methods that provide PSD estimates (e.g., multi-taper estimates, wavelet estimates) would not be desirable as it requires expert knowledge regarding the set of all such methods. A

solution to this problem is to implement a class grouping all methods for computing a PSD, representing the PSD estimation as a category of methods. Addressing this type of generalization is accomplished in NEAO using two approaches.

First, the classes are organized in a taxonomy that uses superclasses to group semantically similar steps, data, or parameters. The structure is chosen to maximize the separation between classes at the lowest levels of the hierarchy. For instance, for the PSD estimation (Fig. 3a), there is the *PowerSpectralDensityAnalysis* superclass that groups the atomic steps *ComputePowerSpectralDensityWelch* and *ComputePowerSpectralDensityMultitaper*, each representing its respective method (and described with the proper annotations). This primary taxonomy reflects mainly the algorithmic and technical description of the methods. However, this classification scheme in itself is insufficient as the complexity of describing an analysis method may comprise more than one semantic dimension.

In the following, we demonstrate the nature of such additional semantic dimensions as well as NEAO's proposed solution to introduce additional cross-cutting groups by means of an example. When considering time series with recorded activity from two brain areas, the magnitude of their correlation in the frequency domain can be estimated by computing the coherence[8]. In NEAO, the *CoherenceAnalysis* class groups many methods to compute coherence. As a measure, coherence is part of a broader *SpectralAnalysis* class, as its computation is based on estimations of the cross-power spectral density between two signals (i.e., the grouping is based on the algorithmic approach). Therefore, it is possible to manually assert that *CoherenceAnalysis* is a subclass of *SpectralAnalysis*, and this provides a relationship to the *PowerSpectralDensityAnalysis* that groups methods for PSD estimation in a single time series (Fig. 3b).

However, switching to a different level of semantic description, one potential purpose of performing a coherence analysis is to infer functional connectivity. This purpose can also be pursued using other methods, e.g., cross-correlations in the time domain[8]. Therefore, we provide a class *FunctionalConnectivityAnalysis* to group methods related by their similarity regarding the estimation of functional connectivity. Technically, this is accomplished by employing a normalization with the Rector technique[53] using the special property *hasPurpose*. The normalization approach uses equivalent class axioms, which define classes in terms of necessary and sufficient conditions, often involving property restrictions (Fig. 3c). Property restrictions in OWL are logical constraints that specify conditions such as what kinds or specific values a property must have. In NEAO, we used the value restriction to state conditions where the *hasPurpose* property has particular values. These property values are individuals from the *AnalysisPurpose* class, which is defined outside **AnalysisStep** and represents specific purposes in the analysis. For example, the individual representing the purpose of estimating functional connectivity is *FunctionalConnectivityPurpose*.

To implement the grouping, it is possible to have axioms asserting that *CoherenceAnalysis* is a subclass of *SpectralAnalysis* in the primary taxonomy, but also a subclass of a (logically defined) class where the value of *hasPurpose* is *FunctionalConnectivityPurpose* (i.e., coherence is a method of spectral analysis that is also used to estimate functional connectivity). To complete the normalization, the class *FunctionalConnectivityAnalysis* is defined as a subclass of **AnalysisStep** and equivalent to a class where the value of *hasPurpose* is *FunctionalConnectivityPurpose*. With these logical definitions, the reasoner will automatically infer that the methods grouped within *CoherenceAnalysis* are also subclasses of *FunctionalConnectivityAnalysis*.

The normalization approach facilitates the establishment of many cross-cutting semantic relationships within NEAO. For example, as coherence and cross-correlation are methods related by their potential purpose to estimate functional connectivity, classes such as *CoherenceAnalysis* and *CrossCorrelationAnalysis* are defined in the primary taxonomy to group steps that compute coherence and cross-correlation, respectively (Fig. 3b). However, both are also defined with the property restriction axiom stating that the value of the *hasPurpose* property is *FunctionalConnectivityPurpose* (pink line in Fig. 3c). As the *FunctionalConnectivityAnalysis* class (pink ellipse) is defined as equivalent to a class where the value of *hasPurpose* is *FunctionalConnectivityPurpose*, all the classes representing methods in *CoherenceAnalysis* (dashed red line) and *CrossCorrelationAnalysis* (dashed green line in Fig. 3c) are grouped by inference of the reasoner as classes from *FunctionalConnectivityAnalysis* (Fig. 3c).

In a similar fashion, NEAO introduces additional non-asserted groupings using the normalization technique to identify features common to methods in different branches of the taxonomy (cf., Fig. 3b). This includes time vs. frequency domain, the directionality (directed vs. non-directed), or number of variables involved (bivariate vs. multivariate). Additional properties for other equivalent class definitions are defined. For example, the *isDirected* property defines the classes *DirectedAnalysis* (if the value is true; blue ellipse) and *NonDirectedAnalysis* (if the value is false; orange ellipse in Fig. 3c). These axioms are also inserted in the definition of *CoherenceAnalysis* (a non-directional estimate) and *CrossCorrelationAnalysis* (a directional estimate in the time domain), allowing them to be further used to distinguish coherence from cross-correlation, as the former is inferred as part of *NonDirectedAnalysis* and the latter is inferred as *DirectedAnalysis*. In the end, normalization allows the easy expansion of NEAO to add semantic groupings according to the demands needed to extract relevant insight from the description of the analyses.

Describing analyses composed by multiple substeps. Some analyses might require the completion of a series of smaller steps (substeps) to obtain the final results from the inputs. Each substep is associated with specific parameters that determine the final output. One example is the Analysis of Sequences of Synchronous Events (ASSET)[54] method to detect neuronal activity patterns. ASSET aims to detect activity patterns where groups of neurons fire in sequences that repeat in time (sequences of synchronous events; SSEs). A series of 5 substeps do this: (i) representation of repeated synchronous activation (in the input spike data) as an intersection matrix, (ii) assessment of the significance of matrix entries, (iii) masking of non-significant matrix entries, (iv) clustering of matrix entries using a DBSCAN approach, and (v) identification and extraction of the resulting clusters

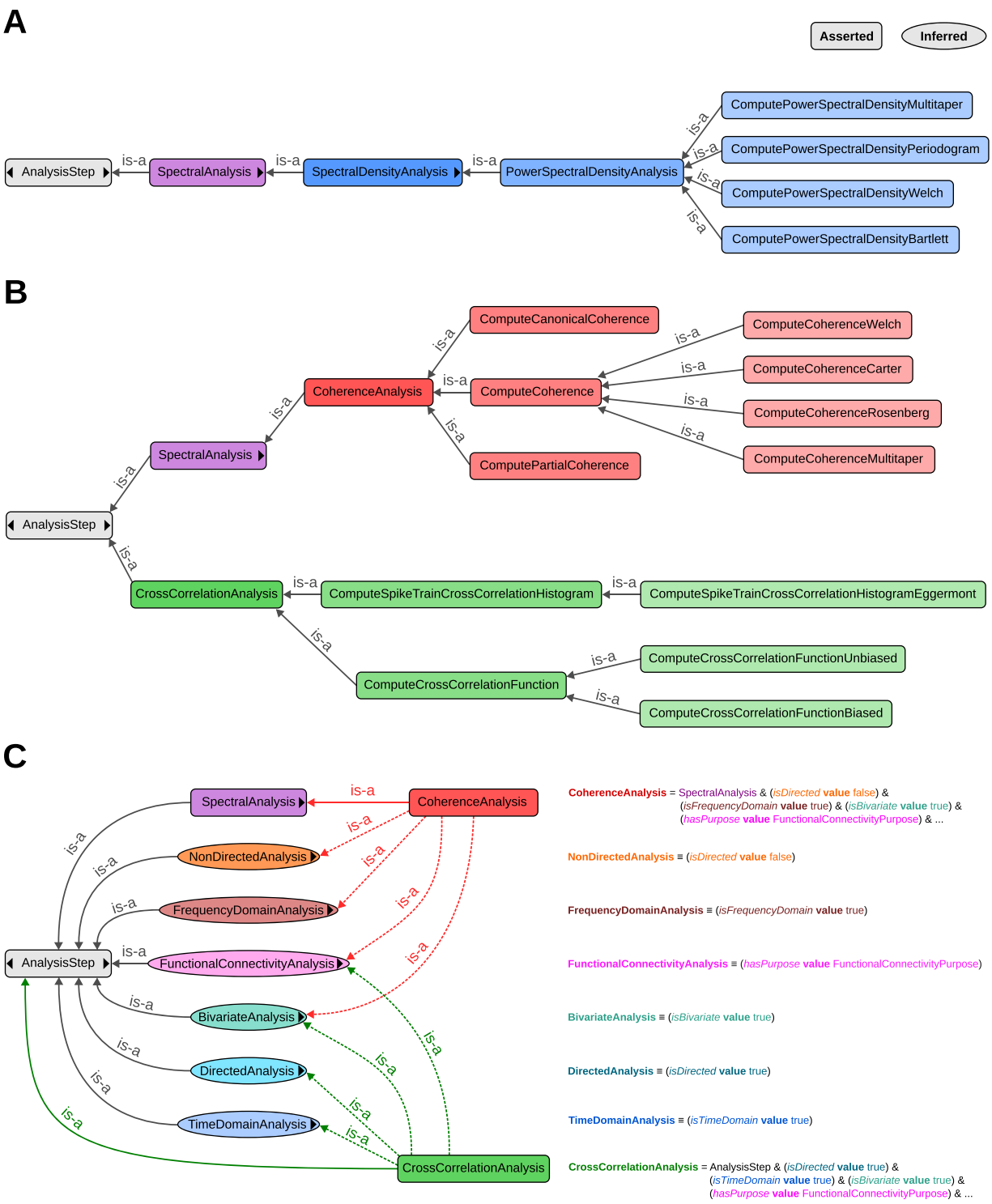

Fig. 3 Grouping of classes describing analysis methods in NEAO. In these diagrams, classes asserted in the main taxonomy are shown as rounded rectangles, and classes inferred using the reasoner are shown as ellipsis shapes. The *is-a* relationship denotes a subclass relationship. The black triangles inside a class shape show missing relationships, not shown for simplicity. (**A**) Example for the specific methods for computing a PSD to illustrate the primary semantic groupings in NEAO. The methods are defined as classes at the lowest levels of a taxonomy starting in **AnalysisStep**, and are described with proper annotations, including the bibliographic resource where the method description was published. They are grouped in the *PowerSpectralDensityAnalysis* class, a subclass of the more general *SpectralAnalysis* class (purple). (**B**) The grouping of coherence and cross-correlation measures exemplifies how NEAO's taxonomy organizes the diversity of analysis methods with a maximum separation regarding their semantic meaning. The classes *CoherenceAnalysis* and *CrossCorrelationAnalysis* define separate branches in the taxonomy. They group methods to compute either coherence measures (red shades) or cross-correlation measures (green shades), respectively. Mathematically, coherence is the normalized magnitude of the cross-power spectral density between two signals. Therefore,

CoherenceAnalysis is a subclass of *SpectralAnalysis* (purple) to express the algorithmic and technical similarities to other methods such as those shown in (**A**). (**C**) Grouping to represent additional semantic dimensions is obtained using the Rector normalization for the examples in panel B. Left: class hierarchy and relationships explicitly defined with subclasses (round rectangles and solid lines) and those inferred with the reasoner (ellipses and dashed lines). Right: definition of each class in terms of axioms. The diagram shows the complementary nature of the classes and relationships asserted in the main **AnalysisStep** taxonomy.

to obtain the SSEs. Each of these substeps is associated with a set of parameters. Therefore, the description of an analysis of spike data using ASSET could be achieved in two levels of granularity. At the highest level, each substep is atomic concerning the model defined in NEAO, with individual data inputs/outputs and parameters for each intermediate step. However, considering only the main inputs (i.e., the spike trains) and outputs (i.e., the list of detected SSEs), the analysis could be described as the neural activity pattern detection method ASSET that used a set of input spike data and produced a set of SSEs as outputs. This poses a challenge in attributing semantic meaning to analysis steps since, for example, the intermediate step of computing a matrix in ASSET is not a neuronal activity detection pattern method, but only the whole process with all the five substeps.

To allow the description of such analyses with multiple substeps while retaining the semantic meaning of the compound process, the **hasSubstep** property is defined in NEAO. **hasSubstep** is used to link two individuals of the **AnalysisStep** class (Fig. 2a). For ASSET, NEAO defines the *ASSETAnalysisSubstep* superclass in the primary taxonomy to group all classes representing the computation of the intermediate substeps (i)–(v) in the ASSET analysis. A distinct class *ExecuteASSETAnalysis* is defined as a subclass of *NeuronalActivityPatternDetectionAnalysis*. The latter is the main superclass in the primary taxonomy to group several semantically related methods to detect patterns in neuronal activity (e.g., SPADE, CAD). The *ExecuteASSETAnalysis* class has a restriction to identify that individuals of this class have elements from *ASSETAnalysisSubstep* as possible values for the **hasSubstep** property. The **hasSubstep** property is also used to define a generic *CompoundAnalysis* class that identifies if any individual of the **AnalysisStep** class represents an analysis process composed by multiple substeps, such as *ExecuteASSETAnalysis*). In this way, complex analyses such as ASSET with multiple intermediate steps and data outputs can be modeled and inferred using NEAO at multiple levels of granularity to retain proper semantic information.

Source information on the data. Although the objective of NEAO is not to model the data acquisition or to provide a more detailed description of the source and format of the data used in the analysis, two classes are defined as abstractions to structure additional information on the entities of the **Data** class (Fig. 2a). The **ElectrophysiologySignalSource** class can be used to define individuals that structure details concerning the data source. For example, this could be used as a base to describe the technique (e.g., EEG or extracellular recording), the recording channel, or the anatomical structures. This could be part of future expansions of NEAO or as a base to align other ontologies suitable for data and metadata descriptions (e.g., EDAM). The **DataRepresentation** class can provide additional information on how the data is structured, which is relevant for interpreting the analysis. For example, computing the Pearson correlation coefficient between a pair of binned spike trains in one analysis step will produce a single scalar value. However, that analysis step might also take a collection of binned spike trains, and the output of the step is the coefficient for all pairwise combinations and outputs the coefficients in the form of a matrix. Therefore, it is possible to use the **DataRepresentation** as a base to structure this additional level in the analysis description.

Competency questions. Several competency questions are addressed with the model NEAO defines and presented above. They are summarized in Table 2.

Example of annotation of RDF using the NEAO. Figure 4 shows an example of how the filtering and PSD computation steps illustrated in Fig. 1 could be described in RDF using NEAO elements, assuming that these steps in the analysis used the implementation available in the software library Elephant version 0.14.0. It is possible to add detailed semantic information to each analysis step with its associated inputs/outputs and to structure the parameter descriptions and the software implementation details.

Practical application of NEAO: annotating provenance information. We considered three representative analysis scenarios as examples to demonstrate how the semantic information provided by NEAO can be used to facilitate describing and understanding the analysis of electrophysiology data. One Python script was implemented for each analysis scenario. These scripts process or generate data and save outputs into a folder. In brief, the analysis scenarios consisted of the following:

- Analysis 1: for each trial in an experimental recording session, plot the power spectral densities of the LFP time series of each recording electrode;
- Analysis 2: for correct trials in a recording session, plot the interspike interval (ISI) histogram (ISIH) of spike train surrogates[55] obtained from selected neurons (single unit activity data obtained after spike sorting) together with the ISIH spread;
- Analysis 3: generate artificial data using either a homogeneous Poisson or a homogeneous gamma process, and plot the ISIHs with a measure of ISI variability.

Class	Example of competency question
AnalysisStep	Which steps were used in the analysis?
	Did the analysis use [specific method] as a step?
	Did the analysis use [category of method] as a step?
Data	What data was input/output to a step in the analysis?
	Did the analysis produce [specific data] as input/output from a step?
	Did the analysis use [category of data] as input/output from a step?
AnalysisParameter	What are the parameters for the steps in the analysis?
	What are the parameters of [specific method] used in the analysis?
	What is the [specific parameter] of [specific method] used in the analysis?
	What are the parameters of [category of method] used in the analysis?
SoftwareImplementation	What software/code implemented a step in the analysis?
	What software/code implements [specific/category of method] used in the analysis?
	What is the version of the software/code of a step in the analysis?
SoftwarePackage	What package contains the software/code of a step in the analysis?
	What package contains the software/code of [specific/category of method] in the analysis?
	What is the package version that contains the software/code of [specific/category of method] in the analysis?
BibliographicReference	What is the bibliographic source of [specific method]?
	What are the bibliographic sources of [category of method]?
ElectrophysiologySource	What neural source does data contain?
	Did a step use data from [specific source]?
DataRepresentation	How is data input/output of a step in the analysis represented?
	Is data input/output of a step represented as [specific representation]?

Table 2. Examples of competency questions addressed by NEAO. The classes and properties defined by the ontology are intended to identify the atomic steps used throughout the analyses, together with their data and parameters. Questions may inquire about *specific* methods, data, or parameters. For example, we can cite the computation of a PSD using the Welch algorithm (specific method), the CV2 interspike variability measure obtained by a corresponding analysis (specific data), and a low-pass cutoff for a filter (specific parameter). The ontology also provides the ability to query about a *category* of methods, data, or parameters. As examples, we can cite PSD analyses (for which the Welch is one of multiple possible), spike interval statistics (for which the CV2 value is one of multiple possible), and filtering parameters (for which a low-pass cutoff is one possibility). Moreover, the ontology intends to support the description of the software implementing each step in the analysis (associated with a specific or a category of methods), with classes and properties to structure the function, program, and software package information (name and versions). NEAO also aims to aid in inquiring about the literature sources associated with a category or specific methods, and the source and representation of data throughout the analysis.

The scripts for Analysis 1 and Analysis 2 were implemented in multiple versions in which one function was changed during the analysis. In the 3 versions of Analysis 1, this was the function to calculate the power spectrum, and in the 2 versions of Analysis 2, this was the method to generate surrogate artificial spiking data. Each script version saved the respective output plots in different subfolders of a main results folder, simulating a situation of a shared folder that collects results from different analyses. This folder is accessible through the repository with the code accompanying this paper (*/outputs/analyses*)[56]. Table 3 summarizes each analysis scenario and the subfolder and file structure used to store the results.

The scripts were instrumented with the software Alpaca to capture detailed provenance throughout the analyses. Alpaca is a Python toolbox that produces a structured record of all the operations performed within the analysis script[57]. The details about the function executions, their parameters, and data inputs/outputs are saved as a graph in RDF using an ontology derived from the W3C PROV-O. The data from the RDF files were inserted into a knowledge graph, allowing the query of the provenance information using the SPARQL graph-based query language[58]. The raw outputs of SPARQL queries presented in this paper are available as CSV files accessible at the Zenodo repository with the code accompanying this paper (*/outputs/query_results*)[56]. All queries are made to the complete graph containing provenance information of Analyses 1–3 (including the 3 versions of Analysis 1 and the two versions of Analysis 2 as described above).

Table 4A presents an example query for the stored provenance information illustrating the type of information that can be extracted without the use of NEAO. This query lists the file paths of all those output files that were derived from a specific input file *i140703-001_no_raw.nix* to the script (going backward through the sequence of functions executed until the plot was saved). Aggregating the table by the folder where the file is stored (Table 4B), it is clear that this corresponds to all files output from Analyses 1 and 2 (in which the experimental data file *i140703-001_no_raw.nix* was used), but not Analysis 3 (where data was artificially generated).

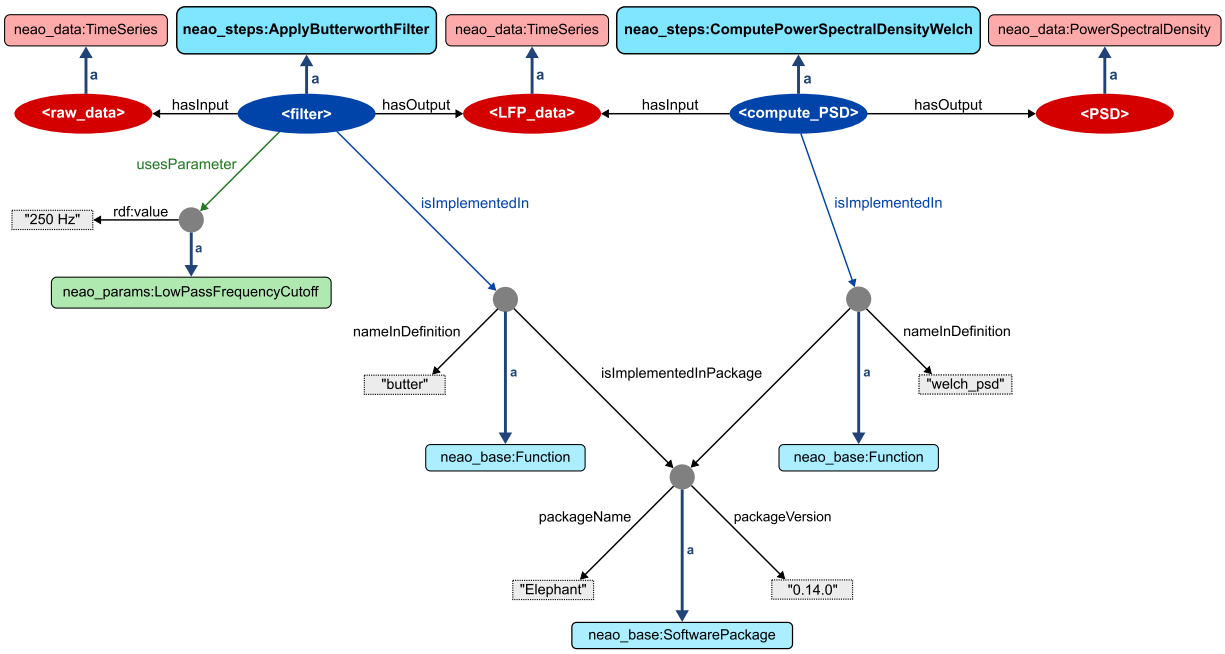

Fig. 4 Using NEAO to describe steps in the analysis of neuroelectrophysiolgy data. In this example, the filtering and power spectral density (PSD) computation steps of the example from Fig. 1 are represented as an RDF graph. The data and analysis step nodes from the example are represented by ellipses (red and blue, respectively), identified by a URI. NEAO properties **hasInput** and **hasOutput** are used to connect each data node to the respective step, and describe it as either an input or output. To add semantic information, each node is associated with a specific **Data** (red rectangles) or **AnalysisStep** class (blue rectangles) defined by NEAO via the *rdf:type* property (thick dark blue arrows, abbreviated as *a*). Note that each analysis step can be easily understood for the method used, as each is associated with a specific class representing the concept of the step in NEAO (i.e., the filter is a Butterworth type filter, and the computation of the PSD used the Welch algorithm). The details of the data transformations are also visible, as the filtering step transformed a time series into another time series. At the same time, the PSD computation generated a new conceptually distinct data entity from the input time series (i.e., the power spectral density), which is represented by a different class. The details of the software implementing the two steps are specified through the **isImplementedIn** property (light-blue arrows). Each used a function, as the nodes are associated with the **Function** class for type description. Both were implemented in the Elephant package version 0.14.0, as the **isImplementedInPackage** property points to a node of the **SoftwarePackage** class, whose properties define the package name and package version (values in the grey rectangles with dashed borders). Finally, the specification of the parameter used by the Butterworth filter is provided by the **usesParameter** property (green arrow). The node is associated with an **AnalysisParameter** class to explicitly define the parameter as a low-pass frequency cutoff (green rectangle), whose value was 250 Hz (grey rectangle with dashed border, defined by the *rdf:value* property). Grey circles represent RDF blank nodes (i.e., nodes not explicitly identified by a URI, but unnamed). Blank nodes can be used to create property values that consist of the information provided by the group of properties defined for the blank node. In this example, the blank node representing Elephant can be interpreted as "a software package whose name is Elephant and version is 0.14.0".

Table 4C shows the result of a different query listing all the files saved by a script (also aggregated by the file folder). This second query identifies all files saved by any of the three analysis scripts.

To incorporate the semantic information introduced by NEAO, additional relationships according to the NEAO model were added to the graph containing the provenance information. These relationships were based on the contents of the provenance RDF triples captured by Alpaca annotated with NEAO classes (details are given in the Methods). In this way, function executions in the script and the associated data input, outputs and parameters are related to specific classes defined in NEAO. For example, the *elephant.spectral.welch_psd* used in Analysis 1.1 and *scipy.signal.welch* used in Analysis 1.3 are associated with the *ComputePowerSpectralDensityWelch* class, while the *elephant.spectral.multitaper_psd* used in Analysis 1.2 is associated with the *ComputePowerSpectralDensityMultitaper* class. By doing so, it is possible to use NEAO classes and relationships in the queries and make inferences on the captured provenance using the extended semantic information provided by NEAO. This will provide a more descriptive and generic representation of the provenance of those files, which will be explored in the following sections.

Overview of the analysis results. SPARQL queries can use the information provided by NEAO to answer overview questions regarding the provenance of the results produced by the three analyses. These queries can either list the analysis steps involved in generating a result file or identify subsets of the results according to specific steps in the analysis. In the following, we list human-comprehensible questions to the knowledge contained

Analysis	Description	Output folder	File name
1.1	PSD computation using the Welch method in the Elephant toolbox	/ *reach2grasp* / *psd_by_trial* / [session]	[trial ID].png
1.2	PSD computation using the multitaper method in the Elephant toolbox	/ *reach2grasp* / *psd_by_trial_2* / [session]	[trial ID].png
1.3	PSD computation using the Welch method in the SciPy toolbox	/ *reach2grasp* / *psd_by_trial_3* / [session]	[trial ID].png
2.1	ISI histograms of spike train surrogates obtained from experimental data using the uniform spike dithering method	/ *reach2grasp* / *surrogate_isih_1* / [session]	[unit ID].png
2.2	ISI histograms of spike train surrogates obtained from experimental data using the trial shifting method	/ *reach2grasp* / *surrogate_isih_2* / [session]	[unit ID].png
3	ISI histograms of spike trains generated by stationary Poisson or gamma processes	/ *isi_histograms*	[spike train index].png

Table 3. Overview of the analysis scenarios presented as use cases for NEAO. Three main analyses were implemented: (1) computation of PSDs of LFPs, (2) ISI histograms (ISIHs) of surrogate spike trains obtained from the data available in one dataset of the Reach2Grasp experiment (*i140703-001_no_raw.nix*), and (3) computation of ISIHs of artificially generated spike trains. Outputs of Analyses 1 and 2 that used the Reach2Grasp experimental dataset were grouped inside the folder *reach2grasp*, while the outputs of Analysis 3 were stored in the separate folder *isi_histograms*. For Analyses 1 and 2, variants of the analysis were implemented (three for Analysis 1 and two for Analysis 2), and each variant stored the results in distinct subfolders (*psd_by_trial** for Analysis 1 and *surrogate_isih_** for Analysis 2). In the folder structure, *[session]* corresponds to the session identifier in the Reach2Grasp experiment (*i140703-001*; subject N, recordings from July 3rd, 2014, first recording session of the day). In the file names of outputs in scenarios 1 and 2, *[trial ID]* is the trial identification number (obtained from the annotations of the behavioral events stored in the data file), and *[unit ID]* is the identifier of a single putative neuron assigned to the data object containing the single-unit neuronal spiking activity after spike sorting. In the file names of outputs in scenario 3, *[spike train index]* is the index of the spike train in the list where they were stored after their generation in the script. All queries presented as use cases are based on the full set of results obtained from Analyses 1–3.

in the provenance information of our three analysis scenarios, and demonstrate how these can be solved via a corresponding query.

Which steps were performed to generate a result file? For each result file, it is possible to identify any function execution in the sequence that generated the data saved in the file. These function executions were annotated with NEAO **AnalysisStep** classes. Table 5A shows the query output, where specific analysis steps represented by classes in NEAO are listed for each file. To get a summary, the resulting table was aggregated to obtain counts of each class per output folder (Table 5B). This shows that: (i) files in the subfolders *psd_by_trial** (Analysis 1) had Butterworth filtering, downsampling, and a step that computed a PSD (although with different methods); (ii) files in subfolders *surrogate_isih_** (Analysis 2) had steps that computed ISIs and ISIHs, calculated sums, means and standard deviations, and generated spike train surrogates (with different methods); and (iii) files in the folder *isi_histograms* (Analysis 3) had steps to generate spike trains (with different methods), compute ISIs and ISIHs, and calculated the interspike interval variability measure CV2. Overall, this query indicates the primary processes used to produce the results stored in each file and corresponds to the overview description presented in Table 3. Knowledge of the functions and programming language used in scripts is not required. In addition, this query does not distinguish the different implementations of the Welch algorithm by different software tools in Analysis 1.

Which files contain PSD results? To identify all files with PSD results, it is possible to execute a query to check which files stored data identified with the NEAO *PowerSpectralDensity* class and which were the output from the execution of a function annotated with a member of the *PowerSpectralDensityAnalysis* grouping class (e.g., *ComputePowerSpectralDensityWelch*). In NEAO, the *PowerSpectralDensityAnalysis* class encompasses all methods to compute a PSD. Table 6A shows the aggregation of the query results by output folder (the results before aggregation are shown in Table S1). Only the folders *psd_by_trial** are shown now, as they store the results from Analysis 1 that performed the PSD analysis. The grouping class *PowerSpectralDensityAnalysis* allowed the identification of the three result sets regardless of the computation method (Welch or Multitaper) used.

Which files contain ISIH results? To identify all files with ISIH results, it is possible to execute a query to check which files stored data identified with the *InterspikeIntervalHistogram* class and which were the output from the execution of a function annotated with the *ComputeInterspikeIntervalHistogram* class. Here, the *ComputeInterspikeIntervalHistogram* class represents specifically the computation of an ISIH in NEAO. Table 6B shows the aggregation of the query results by output folder (non-aggregated results are shown in Table S2). Only the folders *surrogate_isih_** and *isi_histograms* are shown, as they store the results from Analyses 2 and 3, which are the two analysis scenarios computing ISIHs. In contrast to the previous question for the PSD results, this query considers the specific class *ComputeInterspikeIntervalHistogram* representing the step of computing an ISIH instead of a class grouping similar steps, such as *PowerSpectralDensityAnalysis*.

Which files used artificial data? To identify all results that used artificial data as a source for the result, it is possible to execute a query to check if the actual data saved in the file is derived from data that is the output of a function execution that generates artificial data. In NEAO, this is represented by the *ArtificialDataGeneration* class. Table 6C shows the aggregation of the query results by output folder (non-aggregated query result is shown in Table S3). Only the folders starting with *isi_histograms* are shown, as they store the results from Analysis 3, which generated artificial spike trains for the computation of the ISIHs.

A	
Input dataset file path	**Output plot file path**
…/i140703-001_no_raw.nix	…/reach2grasp/psd_by_trial/i140703-001/1.png
…/i140703-001_no_raw.nix	…/reach2grasp/psd_by_trial/i140703-001/10.png
…/i140703-001_no_raw.nix	…/reach2grasp/psd_by_trial/i140703-001/100.png
(omitted 486 lines)	
…/i140703-001_no_raw.nix	…/reach2grasp/surrogate_isih_2/i140703-001/Unit 59001.png
…/i140703-001_no_raw.nix	…/reach2grasp/surrogate_isih_2/i140703-001/Unit 6002.png
…/i140703-001_no_raw.nix	…/reach2grasp/surrogate_isih_2/i140703-001/Unit 7001.png

B	
Output plot root file path	**File count**
reach2grasp/psd_by_trial	160
reach2grasp/psd_by_trial_2	160
reach2grasp/psd_by_trial_3	160
reach2grasp/surrogate_isih_1	6
reach2grasp/surrogate_isih_2	6

C	
Output plot root file path	**File count**
isi_histograms	200
reach2grasp/psd_by_trial	160
reach2grasp/psd_by_trial_2	160
reach2grasp/psd_by_trial_3	160
reach2grasp/surrogate_isih_1	6
reach2grasp/surrogate_isih_2	6

Table 4. The provenance information in the knowledge graph provides a generic overview of the files stored in the analysis output folder. (**A**) Result of a SPARQL query listing files (output plots) that saved data derived from another file (input dataset). The query lists the paths of the input dataset and the output plot. This identifies files for which the experimental dataset *i140703-001_no_raw.nix* was used. The full path strings were truncated to facilitate the visualization, and only 6 of 492 lines are shown. (**B**) Aggregation of table A to show the distribution of files according to the file path root. 160 plots were generated by each of the 3 scripts that implemented a PSD analysis using *i140703-001_no_raw.nix* in Analysis 1 (*psd_by_trial** subfolders), and 6 files were generated by each of the 2 scripts that plotted ISIHs from surrogates obtained from the spike data in *i140703-001_no_raw.nix* in Analysis 2 (*surrogate_isih_** subfolders). All subfolders are stored in *reach2grasp*, as this folder groups all analyses that used the experimental data in *i140703-001_no_raw.nix*. (**C**) Aggregation of the results from a second query identifying any file that contains saved data. This identifies output files from all the scripts (i.e., the files already identified in table B in addition to 200 files produced by the script that plotted ISIHs of artificially generated spike trains in Analysis 3). Note that the ISIH plots derived from artificial data are stored in the separate *isi_histograms* folder.

In-depth queries for Analysis 1. As outlined above, Analysis 1 produced three distinct subsets of results (referred to here as Analysis 1.1, Analysis 1.2, and Analysis 1.3, each stored in a different subfolder inside the main *reach2grasp* folder: *psd_by_trial**; Table 3). From visually inspecting these files (Fig. 5), it is clear that the results produced by Analysis 1.2 are distinct from the results of Analysis 1.1 and 1.3, which themselves appear very similar to each other. We now introduce SPARQL queries building on the query we used previously to identify the results associated with a PSD analysis (Table 6A) in order to unravel the specific differences leading to the three sets of PSD analyses.

Which method was used to compute the PSD plotted in each file? It is possible to query the specific subclass of *PowerSpectralDensityAnalysis* used to annotate the function execution that computed the PSD stored in each file. Table 7A shows the aggregation of the query results by output folder. The query identifies that all files from the folders of Analysis 1.1 and 1.3 used the Welch method for computing the PSD (represented by the class *ComputePowerSpectralDensityWelch* in NEAO), while all files from the Analysis 1.2 folder used the multitaper method (represented by the class *ComputePowerSpectralDensityMultitaper* in NEAO).

In which software package is the method to compute the PSD implemented? To better understand the difference between the results from Analyses 1.1 and 1.3, which both used the Welch method, we use NEAO properties to identify the software package information that is associated with the function used to compute the PSD stored in each file. This information is accessible by the **isImplementedIn** and **isImplementedInPackage** properties. Table 7B shows the aggregation of the results of this query by output folder. The query listed the name and version of the packages using the NEAO properties **packageName** and **packageVersion**. It is apparent that for results produced by Analyses 1.1 and 1.3, although the same PSD computation method was used, the software code containing the implementation of the step differed: Elephant for Analysis 1.1 or SciPy for Analysis 1.3. In addition, we can infer that the multitaper method used in Analysis 1.2 was implemented in Elephant.

A	
File path	**NEAO step class**
…/isi_histograms/1.png	neao_steps:ComputeCV2
…/isi_histograms/1.png	neao_steps:ComputeInterspikeIntervalHistogram
…/isi_histograms/1.png	neao_steps:ComputeInterspikeIntervals
…/isi_histograms/1.png	neao_steps:GenerateStationaryPoissonProcess
…/isi_histograms/10.png	neao_steps:ComputeCV2
(omitted 2302 lines)	
…/reach2grasp/surrogate_isih_2/i140703-001/Unit 7001.png	neao_steps:ComputeInterspikeIntervalHistogram
…/reach2grasp/surrogate_isih_2/i140703-001/Unit 7001.png	neao_steps:ComputeInterspikeIntervals
…/reach2grasp/surrogate_isih_2/i140703-001/Unit 7001.png	neao_steps:ComputeMean
…/reach2grasp/surrogate_isih_2/i140703-001/Unit 7001.png	neao_steps:ComputeStandardDeviation
…/reach2grasp/surrogate_isih_2/i140703-001/Unit 7001.png	neao_steps:GenerateTrialShiftingSurrogate

B						
	File count per root file path					
NEAO step class	**reach2grasp/ psd_by_trial**	**reach2grasp/ psd_by_trial_2**	**reach2grasp/ psd_by_trial_3**	**reach2grasp/ surrogate_isih_1**	**reach2grasp/ surrogate_isih_2**	**isi_ histograms**
neao_steps:ApplyButterworthFilter	160	160	160	0	0	0
neao_steps:ApplyDownsampling	160	160	160	0	0	0
neao_steps:ApplySum	0	0	0	6	6	0
neao_steps:ComputeCV2	0	0	0	0	0	200
neao_steps:ComputeInterspikeIntervalHistogram	0	0	0	6	6	200
neao_steps:ComputeInterspikeIntervals	0	0	0	6	6	200
neao_steps:ComputeMean	0	0	0	6	6	0
neao_steps:ComputePowerSpectralDensityMultitaper	0	160	0	0	0	0
neao_steps:ComputePowerSpectralDensityWelch	160	0	160	0	0	0
neao_steps:ComputeStandardDeviation	0	0	0	6	6	0
neao_steps:GenerateStationaryGammaProcess	0	0	0	0	0	100
neao_steps:GenerateStationaryPoissonProcess	0	0	0	0	0	100
neao_steps:GenerateTrialShiftingSurrogate	0	0	0	0	6	0
neao_steps:GenerateUniformSpikeDitheringSurrogate	0	0	0	6	0	0

Table 5. Annotation of the provenance information with NEAO identifies the main steps used to generate the results in each analysis. (**A**) Result of a SPARQL query listing, for each file saved in the analyses output folder, any class derived from **AnalysisStep** that was used to annotate the execution of a function that was part of the sequence of function executions used to generate the result file. The full path strings were truncated to facilitate the visualization. The prefix of the full IRIs of the NEAO step classes returned by the query was substituted by the namespace according to Table 1. (**B**) Aggregation of table A to show the distribution of result files according to the path root (columns) that used a step identified by a particular class (rows). For each result file set, it is possible to identify particularities and commonalities across the steps taken by each analysis.

A	
Root file path	**File count**
reach2grasp/psd_by_trial	160
reach2grasp/psd_by_trial_2	160
reach2grasp/psd_by_trial_3	160

B	
Root file path	**File count**
isi_histograms	200
reach2grasp/surrogate_isih_1	6
reach2grasp/surrogate_isih_2	6

C	
Root file path	**File count**
isi_histograms	200

Table 6. Annotation of the provenance information with NEAO identifies results with specific content. SPARQL query result rows were aggregated according to the root in the file path. Based on NEAO, each query listed files (output plots) that contain power spectral density estimates (**A**), interspike interval histograms (**B**), or results obtained from artificially generated data (**C**).

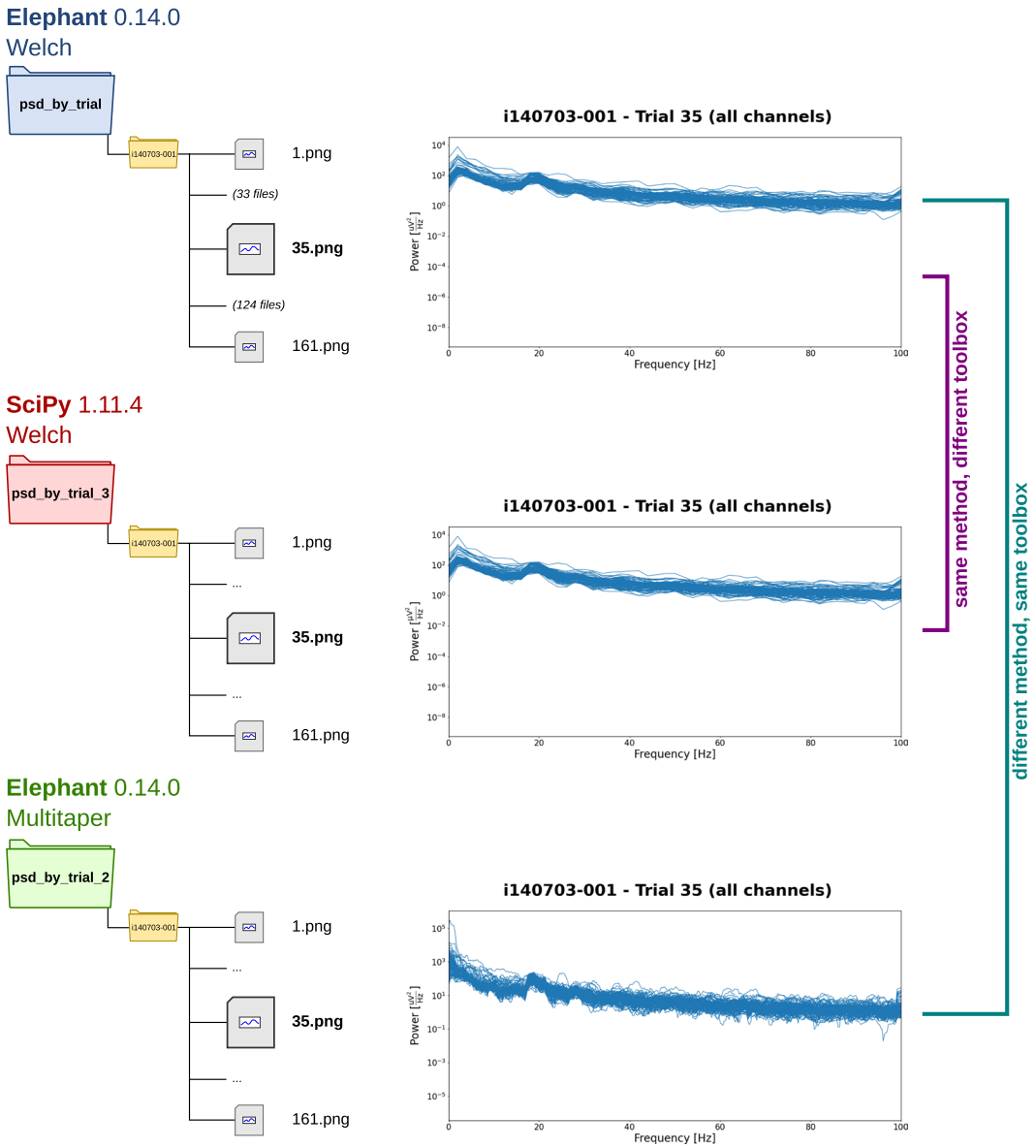

Fig. 5 Analysis 1: Power spectral density (PSD) analysis across trials of a Reach2Grasp recording session. The PSD was computed and plotted (0–100 Hz range) for each trial available in the *i140703-001* session and saved in a PNG file named after the trial ID number. The three versions of the analysis script (i.e., Analysis 1.1, 1.2, and 1.3) store the results in a specific root folder, identified by different colors in the figure. The versions use distinct methods and toolboxes to estimate the PSD: Elephant toolbox with the Welch method (blue), Elephant toolbox with the multitaper method (green), or SciPy using the Welch method (red). Overall, all 160 files are stored in each main output folder for this analysis (trial 142 is ignored). Plots of the same trial across the versions (shown here: trial 35) show that the outputs from the two analyses using the Welch method are visually indistinguishable, contrasting with the multitaper method.

Are the parameters equivalent for plots that used the same PSD method? The software implementations of Analysis 1.1 (Elephant) and 1.3 (SciPy) have distinct parameters to control the application of the Welch algorithm to the input data, such that evaluating the equivalence of the two is not straightforward. For example, the implementation of Welch in Elephant wraps a function from SciPy under the hood, but it provides users alternative parameter specifications. In Elephant, the frequency resolution of the PSD is defined in terms of a frequency value (e.g., 2 Hz) and the degree of overlap between the multiple windows is defined as a fractional value. The Elephant implementation then translates the function call parameters into the corresponding parameters of the more generic SciPy Welch algorithm implementation. This includes specifying the length of the windows used by the algorithm (in samples) and the length of their overlap (in samples) to reflect that frequency resolution and fractional overlap. The description of parameters by NEAO aims to help understand the similarities between Analysis 1.1 and 1.3 by providing classes associated with each possible parameter. Table 7C shows the aggregation of a query that identifies, for the function executions that computed a PSD using the Welch method,

A		
	NEAO PSD computation class	
Root file path	neao_steps:Compute PowerSpectralDensityMultitaper	neao_steps:Compute PowerSpectralDensityWelch
…/psd_by_trial	0	160
…/psd_by_trial_2	160	0
…/psd_by_trial_3	0	160

B				
		File count per root file path		
Package	Version	…/psd_by_trial	…/psd_by_trial_2	…/psd_by_trial_3
Elephant	0.14.0	160	160	0
SciPy	1.11.4	0	0	160

C			
		File count per root file path	
NEAO class	Value	…/psd_by_trial	…/psd_by_trial_3
neao_params:WindowFunction	hann	160	160
neao_params:FrequencyResolution	2.0 Hz	160	0
neao_params:WindowOverlapFactor	0.5	160	0
neao_params:SamplingFrequency	500.0	0	160
neao_params:WindowLengthSamples	250	0	160
neao_params:WindowOverlapSamples	125	0	160

Table 7. NEAO provides specific details for the results of the three PSD analyses. Different SPARQL queries were executed in the knowledge graph to interrogate specific information from the provenance of the files that stored PSD estimates. Aggregations of the query results are presented according to the root in the file path. The main *reach2grasp* folder in the root file path was removed for clarity, and only the names of the subfolders specific to the analysis variants are shown. (**A**) SPARQL query identifying the function used to compute the PSD. It is possible to correctly identify which result file sets contain plots of PSDs computed using the Welch method (all files in the *psd_by_trial* and *psd_by_trial_3* subfolders) or the multitaper method (all files in the *psd_by_trial_2* subfolder). (**B**) SPARQL query result identifying the software package name and version where the function used to compute the PSD was implemented. It is possible to correctly identify which result files used either Elephant (all files in the *psd_by_trial* and *psd_by_trial_2* subfolders) or SciPy (all files in the *psd_by_trial_3* subfolder). (**C**) SPARQL query listing the class and value of the parameters used by executions of a function that computed PSD using the Welch method. The parameter class is derived from **AnalysisParameter**. For the two different result sets that used the Welch method (files in the *psd_by_trial* or *psd_by_trial_3* subfolders), the distinct parameters required by either the Elephant or SciPy software implementations can be identified and compared. In all tables, the prefixes of the full IRIs of the NEAO classes returned by the queries were substituted by the namespaces according to Table 1.

all the parameters used by the function and the specific class derived from **AnalysisParameter**. The query lists the parameter values together with the classes, and the aggregation is performed by the output file folder. The query shows that all results from both Analysis 1.1 and 1.3 used a parameter to define the window function (string "*hann*", which is how a Hanning window is selected in either Elephant or SciPy implementations of Welch). However, each of the two analyses used other distinct parameters when computing the PSD. Analysis 1.1 defined a frequency resolution of *2.0 Hz* and an overlap factor of *0.5*. Analysis 1.3 defined the input time series sampling frequency as the unit-less number *500.0*, the window length as *250* samples, and the length of overlap as *125* samples. With this information, it is possible to conclude that the two scripts performed equivalent PSD estimations, as the overlap factor can be computed from the window and overlap length in samples ($\frac{125}{250} = 0.5$) and the frequency resolution in Hz can be computed from the window length and sampling frequency ($\frac{500\ \text{Hz}}{250} = 2\ \text{Hz}$).

The overview query presented in Table 5B showed that all three variants of Analysis 1 used a Butterworth filtering step. However, we can also use NEAO to create queries that directly ask for details of the filtering used when generating the results of Analysis 1.

Were the PSD results derived from filtered data? For the files that stored the results of a PSD analysis, it is possible to query if any function execution before the computation of the PSD was from the NEAO class *DigitalFiltering* (which groups all filtering methods in the taxonomy). Table 8A shows the query result aggregated by the output folder. This shows that all files from each Analysis 1 implementation computed the PSD using data derived from a filtered time series.

What type of filter was used? We can extend the previous query to identify the subclass of *DigitalFiltering*, which will identify the NEAO class that represents the specific type of filter used. Table 8B presents the query result aggregated by the output folder, which shows that all files from each variant of Analysis 1 used the Butterworth type of filter.

A	
Root file path	**File count**
…/psd_by_trial	160
…/psd_by_trial_2	160
…/psd_by_trial_3	160

B	
Root file path	**neao_steps:ApplyButterworthFilter**
…/psd_by_trial	160
…/psd_by_trial_2	160
…/psd_by_trial_3	160

C				
		File count per root file path		
NEAO class	**Value**	**…/psd_by_trial**	**…/psd_by_trial_2**	**…/psd_by_trial_3**
neao_params:FilterOrder	4	160	160	160
neao_params:LowPassFrequencyCutoff	250.0 Hz	160	160	160

Table 8. NEAO provides details for the filtering step used by the PSD analyses. Different SPARQL queries were executed in the knowledge graph to interrogate specific information from the provenance of the files that stored PSD estimates. Aggregations of the query results are presented according to the root in the file path. The main *reach2grasp* folder in the root file path was removed for clarity, and only the names of the subfolders specific to the analysis variants are shown. (**A**) SPARQL query identifying if any step that performed a filtering operation was executed before the computation of the PSD saved in a result file. All files in each of the three different result subfolders had filtering. (**B**) SPARQL query result identifying the class of the filtering step. All files in each of the result subfolders used a Butterworth-type filter. (**C**) SPARQL query listing the class and value of the parameters used by executions of a function that performed a filtering step. All files in each PSD analysis result set had low-pass filtering with 250 Hz cutoff and used a fourth-order filter. In all tables, the prefixes of the full IRIs of the NEAO classes returned by the queries were substituted by the namespaces according to Table 1.

Which parameters were used for filtering? Besides the type of filter, the choice of filtering parameters determines the final characteristics of the time series used to compute the PSD. We expand the query used to identify any function execution that performed a filtering step (Table 8A) to list all the parameters used by the function and the annotation with the specific class derived from **AnalysisParameter**. The query lists the parameter's value together with the class, and the aggregation is performed by the output folder. Table 8C shows the query result: all three versions of Analysis 1 used a fourth-order filter, and the filter was used to low-pass the input time series with a cutoff frequency of 250 Hz.

In a similar manner, one may also probe for more complex interdependencies of the analysis steps, such as the order of performing the downsampling and filtering steps.

In-depth queries for Analysis 2. Analysis 2 produced two subsets of results (referred to as Analysis 2.1 and Analysis 2.2, each stored in a different subfolder inside the main *reach2grasp* folder: *surrogate_isih_**; Table 3). From visually inspecting these results (Fig. 6), it is apparent that the plots from Analysis 2 show ISI distributions of surrogate spike trains derived from the data of neuronal units identified in the recording session. However, the two results subsets differ in that only for Analysis 2.2 the ISI distribution is preserved by the surrogate procedure. However, the information in the plot does not allow for identifying the exact cause for this discrepancy. Moreover, in the main results folder for all analyses, also other result files store plots of ISIHs (Table 6B). In the following, we show how SPARQL queries can be built upon the generic query used to identify any result with ISIH analyses to answer questions regarding the specific details of the analysis of the ISI of surrogate spike trains.

Were spike train surrogates used in the analysis? To isolate the specific subset of results from Analysis 2, it is possible to query which files stored ISIHs derived from data identified with the NEAO *SpikeTrainSurrogate* class. Table 9A shows the query results aggregated by the output folder. The query correctly identifies all the 6 result files produced by each implementation of Analysis 2.

Which spike train surrogate generation method was used? Several methods exist to compute surrogates from experimental data that either preserve or destroy the ISI distribution of the source data[59]. Inferring the surrogate computation method in Analysis 2.1 and 2.2 based on the ISIH alone is not possible. Instead, we query the function executions that performed spike train surrogate generation using the NEAO *SpikeTrainSurrogateGeneration* class, and obtain the specific method by identifying the subclass. Table 9B shows the query results aggregated by the output folder. This shows that Analysis 2.1 used the uniform spike dithering surrogate generation method, which is expected to distort the ISI distribution. In contrast, Analysis 2.2 used the trial shifting method that is known to preserve the ISI distribution in the generated surrogates[55].

How many spike train surrogates were used? With the previous query, it is also possible to identify the number of outputs from the function executions annotated with the *SpikeTrainSurrogateGeneration* class. The aggregation in Table 9B shows that both implementations of Analysis 2 used 30 surrogates.

What are the parameters used for generating the surrogates? Similarly to queries presented earlier, it is possible to obtain details on the parameters used to generate the surrogates by listing the parameters for the function

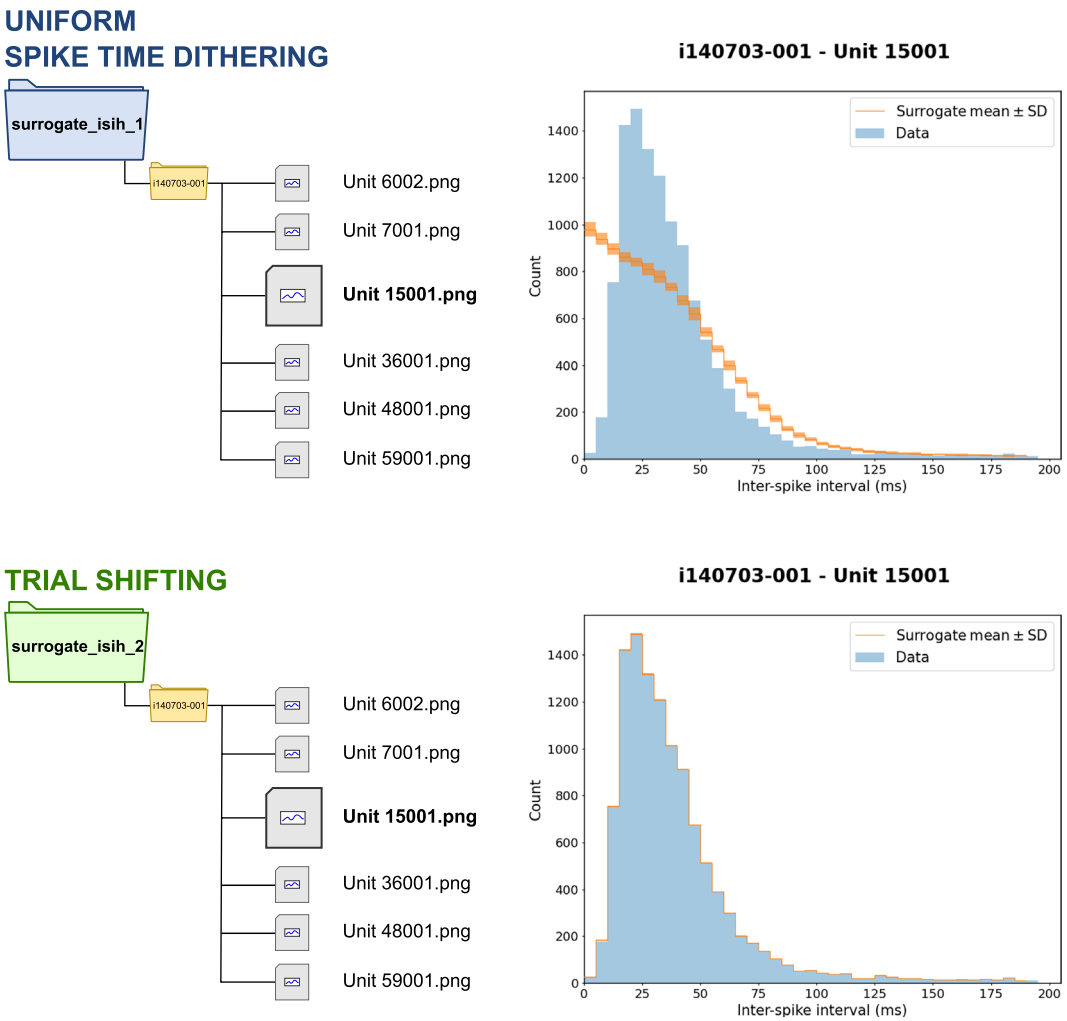

Fig. 6 Analysis 2: Interspike interval histogram (ISIH) analysis of surrogate spike trains generated using spike data from a Reach2Grasp recording session. The ISIH was computed for six neuronal units across all trials where the monkey performed the task correctly in session *i140703-001*. The neuronal units were selected based on the signal-to-noise ratio (SNR, with a value greater than or equal to 5) and mean firing rate in the trial (greater than or equal to 15 Hz in all correct trials in the session). The ISIH obtained from the data is computed using 5 ms bins and plotted as bars. Thirty surrogates were generated from the spike data of each trial, the ISIH across trials was computed similarly as for the spike data, and the mean (orange line) and standard deviation (orange areas) were plotted. Two versions of the analysis script exist (i.e., Analysis 2.1 and 2.2), each storing PNG files with the plots (named after the neuronal unit) into a specific root folder, identified by different colors in the figure. The different versions use distinct methods to generate the surrogates using the Elephant toolbox: uniform spike dithering (blue folder) or trial shifting (green folder). Overall, six files are stored in each main output folder for this analysis. Note that for the plots of the same neuronal unit across the versions (Unit 15001 is shown in the figure), the results obtained using the trial shifting method show that the ISI distribution is preserved, contrasting to the outputs obtained using the uniform spike dithering method. Other than that, the two result sets are visually indistinguishable.

annotated with *SpikeTrainSurrogateGeneration* and the specific classes derived from **AnalysisParameter**. Table 9C shows the result aggregated by the output folder. The query result shows that Analysis 2.1, which used the uniform spike train dithering method, used a dithering time of 15 ms. In contrast, Analysis 2.2, which used the trial shifting method, used a longer dithering time of 30 ms. Both parameters correspond to the maximum time for which either the individual spikes (for uniform spike dithering) or all spikes in the individual spike trains (for trial shifting) are shifted backward or forward in time.

What bin size is used for the ISIH of surrogate spike trains? It is possible to specifically query for the histogram bin size parameter used to compute the ISIHs from data derived from a spike train surrogate (identified by the NEAO *SpikeTrainSurrogate* class) by using the *BinSize* class. Table 9D shows the aggregation of the result by the output folder revealing that all ISIHs from the surrogates were computed using 5 ms bin sizes.

A	
Root file path	**File count**
…/surrogate_isih_1	6
…/surrogate_isih_2	6

B			
		NEAO spike train surrogate generation class	
Root file path	**Number of surrogates**	**neao_steps:GenerateTrialShifting Surrogate**	**neao_steps:Generate UniformSpikeDitheringSurrogate**
…/surrogate_isih_1	30	0	6
…/surrogate_isih_2	30	6	0

C			
		File count per root file path	
NEAO class	**Value**	**…/surrogate_isih_1**	**…/surrogate_isih_2**
neao_params:DitheringTime	25.0 ms	6	0
neao_params:DitheringTime	30.0 ms	0	6

D		
Root file path	**Bin size**	**File count**
…/surrogate_isih_1	5.0 ms	6
…/surrogate_isih_2	5.0 ms	6

Table 9. NEAO provides specific details for the results of the two analyses that computed ISIHs from surrogate spike trains. Different SPARQL queries were executed in the knowledge graph to interrogate specific information from the provenance of the files that stored ISI histograms computed from spike train surrogates. Aggregations of the query results are presented according to the root in the file path. The main *reach2grasp* folder in the root file path was removed for clarity, and only the names of the subfolders specific to the two analysis variants are shown. (**A**) SPARQL query identifying files where the ISIH was computed from data originating from a spike train surrogate. Only files in the *surrogate_isih_** folders are identified, as these are the files with ISIH plots from surrogate spike trains. (**B**) SPARQL query identifying the class and number of outputs of a function executed to generate spike train surrogates, which were used to compute the ISIH saved in the file. The query correctly identifies the use of the uniform spike dithering method in the result files stored in the *surrogate_isih_1* subfolder and trial shifting in the result files stored in the *surrogate_isih_2* subfolder. Both analyses generated 30 surrogates. (**C**) SPARQL query identifying the class and value of the parameters used by executions of a function that generated spike train surrogates. The ISIHs in the files stored in *surrogate_isih_1* (that used the uniform spike dithering surrogate generation method) used a dither time of 25 ms. The ISIHs in the files stored in *surrogate_isih_2* (that used the trial shifting method) used a dither time of 30 ms. (**D**) SPARQL query asking specifically for the bin size parameter during the computation of a ISIH from spike train surrogates. All files in each of the two different result subfolders contain histograms with 5 ms bin size. In all tables, the prefixes of the full IRIs of the NEAO classes returned by the queries were substituted by the namespaces according to Table 1.

In-depth queries for Analysis 3. Inspecting the results from Analysis 3 (Fig. 7), all 200 plots produced from artificial spike trains and stored in *isi_histograms* are visually similar: the histograms show an exponentially decaying ISI distribution, and the variability measure displayed in the plot's title has values close to 1. However, the artificial spike trains used in the plots were generated as two different stationary point processes: Poisson or gamma. Also, several measures exist to investigate the variability of ISIs, each taking into account features of the data such as rate fluctuations[60,61]. NEAO can be used to investigate specific details to understand the provenance of the histogram plots and the computed variability measure.

Which process was used to generate the spike train for each plot? For each file that saved an ISIH that was computed from data output by a function execution that generates artificial data (identified by the *ArtificialDataGeneration* NEAO class), it is possible to query the subclass from **AnalysisStep** used to annotate the function execution. This query will identify one of several steps from the primary NEAO taxonomy that performs data generation. Table 10A shows the aggregation of the query results according to the range of the numbers used in the name of the files stored in the subfolder *isi_histograms* (i.e., 1–100 will correspond to the files generated by the stationary Poisson process, and 101–200 to the files generated by the stationary gamma process; cf., Fig. 7). The query correctly shows that the first 100 files were generated by a stationary Poisson process and the last 100 files by a stationary gamma process.

Which parameters were used to generate the spike train used for each plot? At this point, it is clear that the two groups of plots are different with respect to the process used for data generation, yet they appear similar. Depending on the choice of parameters, it is possible for a gamma process to have statistical properties that are identical to a Poisson process. NEAO allows the identification of the parameters by expanding the previous query to list all the parameters used by the artificial data generation function and the class derived from **AnalysisParameter**. Table 10B shows the result of this query aggregated by the range of the file names. The query result shows that both processes aimed to generate spike trains with a target firing rate of 10 Hz. However,

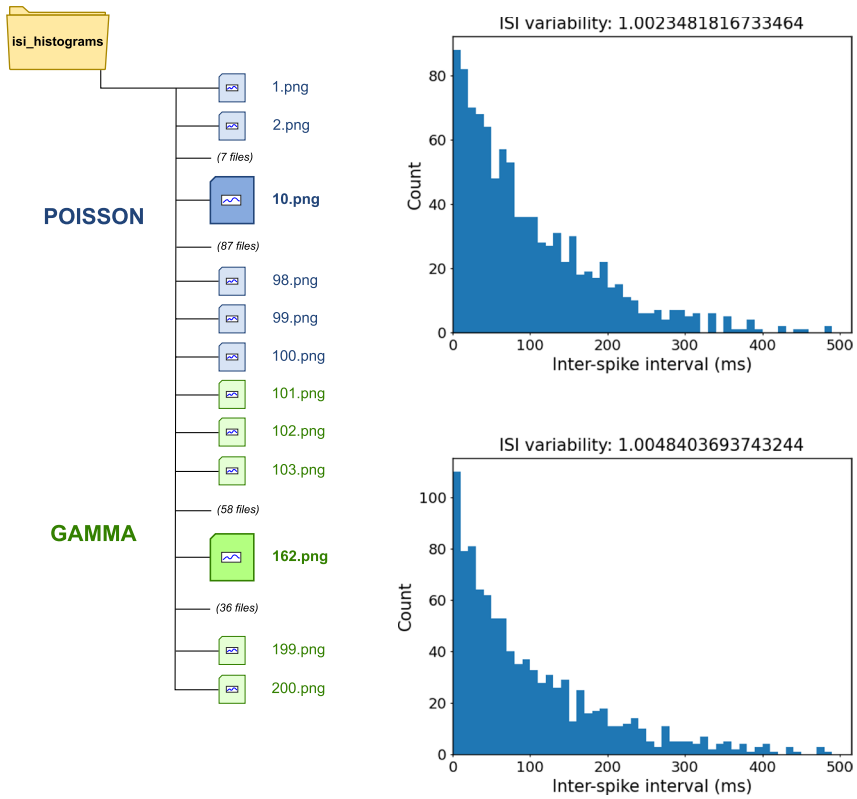

Fig. 7 Analysis 3: Interspike interval histogram (ISIH) analysis of artificially generated spike trains. Two hundred spike trains were generated using either a stationary Poisson or a stationary gamma process using the Elephant package. The ISIH using a 10 ms bin size and the CV2 variability measure were computed for each spike train. The ISIH was plotted with the CV2 value and saved as a PNG file. All files were collected into a single folder. The first one hundred files contain plots obtained from the spike trains generated by the stationary Poisson process (range 1–100, identified with the blue color in the figure), while the last one hundred files contain the plots of the spike trains generated by the stationary gamma process (range 101–200, identified by the green color in the figure). Note that the files from both result sets are indistinguishable (plots for the spike train number 10, generated by a Poisson process, and number 162, generated by a gamma process, are shown). Visually, the plots of both result sets show an exponentially decaying ISI distribution, and all spike trains have an ISI variability measure with values close to 1.

the gamma process generation function (for files 101–200) used the additional parameter shape factor with a value of 1. With this parameter choice, the gamma process employed is mathematically equivalent to a Poisson process[62], and the ISI distribution will have the exponentially decaying shape expected for spike trains generated by stationary Poisson processes.

What bin size is used for the ISIH of artificially generated spike trains? It is possible to use the *BinSize* NEAO class to specifically query for the bin size parameter used by the function that computed ISIHs using data derived from the output of a step that generated artificial data. Table 10C shows the aggregation of the result by the range in the file names. This shows that all ISIHs from artificial data in *isi_histograms* were computed using 10 ms bin sizes.

Which interval variability measure was used? To identify the exact ISI variability measure that is presented in the title of the histograms, it is possible to execute a query to check which files stored data identified with the NEAO *InterspikeIntervalVariabilityMeasure* class, which is the output from the execution of a function annotated with the *InterspikeIntervalVariabilityAnalysis* class. Table 10D shows the aggregation of the query results considering the range in the file names. The query shows that all plots in *isi_histograms* used the same measure CV2, which is a modification of the original method of computing the standard coefficient of variation of ISIs to avoid biased estimations when the firing rate is slowly modulated[61]. However, the interpretation is similar, and spike trains generated by a Poisson process are expected to have values around 1.

Discussion

We introduced the Neuroelectrophysiology Analysis Ontology (NEAO), a novel domain ontology aimed at describing the analysis of data produced by experiments that used electrophysiology to investigate the function of the nervous system, and comparable data resulting from brain simulations. We implemented a model that describes the analysis as a sequence of atomic steps. The analysis steps are associated with specific data inputs and outputs, parameters that control the behavior of the analysis method executed in each step, and the details of its software implementation. The steps are semantically grouped according to the purpose of the analysis and

A

File name range	NEAO spike train generation class	
	neao_steps:GenerateStationaryGammaProcess	neao_steps:GenerateStationaryPoissonProcess
1–100	0	100
101–200	100	0

B

NEAO class	Value	File name range	
		1–100	101–200
neao_params:FiringRate	10.0 Hz	100	100
neao_params:ShapeFactor	1	0	100

C

File name range	Bin size	File count
1–100	10.0 ms	100
101–200	10.0 ms	100

D

File name range	neao_steps:ComputeCV2
1–100	100
101–200	100

Table 10. NEAO provides specific details for the results of the ISIH analysis of artificially generated spike trains. Different SPARQL queries were executed in the knowledge graph to interrogate specific information from the provenance of all the files that stored ISI histograms computed from artificially generated data. Aggregations of the query results are presented according to the range of the numbers in the file names (i.e., 1–100 refers to all consecutive files with the name between "1.png" and "100.png"). (**A**) SPARQL query to identify the step class used to generate the artificial data for which the ISIHs were computed. The first 100 files used spike trains generated by a stationary Poisson process, while the last 100 files used spike trains generated by a gamma process. (**B**) SPARQL query identifying the class and value of the parameters used by executions of a function that generated artificial data. All 200 files used a target firing rate of 10 Hz, but the generation of the spike trains of the last 100 files (by a gamma process) used a shape factor of 1 (equivalent to a Poisson process). (**C**) SPARQL query asking for the bin size parameter during the computation of an ISIH from artificially generated data. All 200 files contain histograms with a 10 ms bin size. (**D**) SPARQL query identifying the class of the step that performed the spike interval variability analysis. All 200 result files used the CV2 statistic. In all tables, the prefixes of the full IRIs of the NEAO classes returned by the queries were substituted by the namespaces according to Table 1.

algorithmic similarity. In addition, the ontology allows for bibliographic references that describe individual steps unambiguously.

Using an ontology to represent data entities is an effective mechanism to represent and expose them according to the FAIR (Findable, Accessible, Interoperable, and Reusable) principles[28]. Firstly, ontologies provide a standardized and formal framework for describing data entities, ensuring their clear and precise representation. By adhering to a common vocabulary, data findability is enhanced, allowing researchers to locate and understand the meaning of specific entities. As presented in the results, using NEAO classes to annotate the entities involved in the analysis of neuroelectrophysiology data fosters findability by allowing human-understandable queries to collections of analysis outcomes. This approach eliminates the reliance on free-text descriptions (e.g., a README file) and the tedious inspection of specific software codes used to generate the data. Such queries may expose answers to questions of increased complexity, for example by requiring a specific sequence of analysis steps or by referencing grouping terms that identify sets of related analysis methodologies. Therefore, the findability of an analysis result is facilitated.

Secondly, ontologies promote interoperability by creating a shared understanding of data semantics. This facilitates seamlessly integrating tools and standards across diverse scientific domains, heterogeneous datasets, and related applications. The analysis of neuroelectrophysiology data often involves complex workflows composed of multiple interconnected steps carried out by distinct software tools and services. As we demonstrated, NEAO is particularly well-suited to represent such intricate workflows and achieve descriptions with a common level of detail. This tool-agnostic framework describes analysis results based on their conceptual content and can be used to identify similar analysis outcomes obtained by different tools. This interoperability also contributes to the reusability of analysis outcomes, as researchers can confidently leverage and combine information from various sources as starting point for further analysis, fostering collaboration and accelerating scientific discovery. The reuse of derived data may be considered as an increasingly important aspect of conceptualizing electrophysiology analysis workflows, given that with the advent of modern recording techniques[7] and simulation technology[63] the analysis of data is prone to require significant amounts of compute resources. In this way, the use of NEAO as an ontology to model analysis outcomes of electrophysiology data aligns with the FAIR principles, offering a robust foundation for enhancing the overall utility of the data.

By addressing the competency questions in Table 2, NEAO facilitates obtaining insights on the analysis results. We implemented example scenarios that address several challenges in identifying information when querying a set of analysis results: heterogeneous data sources (i.e., experimental and artificially generated data); conceptually similar methods leading to different outcomes (i.e., the different approaches for calculating PSDs in Analysis 1, spike train surrogate generation methods in Analysis 2, and spike train generation in Analysis 3); distinct analyses producing conceptually similar results (i.e., the ISIH computation in Analyses 2 and 3); analyses using different software implementations and parameterizations (the PSD computation by the Welch method in Analysis 1); analyses sharing common steps (i.e., common filtering steps in Analysis 1); and analyses using different parameters (i.e., the different bin sizes for the ISIH computations in Analyses 2 and 3 or the different dither times when generating the surrogates in Analysis 2). The use of the common semantic layer provided by NEAO assists in formulating the corresponding queries without the need for knowledge of the analysis code. For example, in a search for results from Analysis 1, the identification of all 3 result sets is possible with a query not containing the Elephant and SciPy function calls "welch_psd," "multitaper_psd," and "welch" as values for the function name. Moreover, only NEAO clearly identifies the similarity of Analyses 1.1 and 1.3 with respect to using the Welch method, which is less apparent from the function names (here, "welch_psd" vs. "welch"). Therefore, using NEAO classes to annotate the steps, data, and parameters involved in the analysis reduces ambiguity and promotes clarity, accessibility, and coherence, especially when considering many complex analyses.

The examples presented highlight the effectiveness and benefits of using NEAO to describe the processes involved in analyzing a neuroelectrophysiology dataset. The approach can unify and semantically enrich the description of distinct analysis workflows. The selected queries demonstrate how the underlying implementation details are abstracted, allowing the researcher to ask questions using human-readable semantics based on the methodological concepts that produced the result. Using machine-readable descriptions that integrate well with the ontology definition in OWL enables users to perform precise queries to gain insights across heterogeneous analyses. The example scenarios illustrated how NEAO effectively bridged gaps caused by differing terminology, parameter choices, and software implementations.

The types of ambiguities we highlighted are frequently found, and we suggest NEAO can help researchers gain a more structured view of available analysis approaches. As an example of a specific method that has considerably evolved in terms of algorithms, assumptions and outcomes is the Spike Pattern Detection and Evaluation (SPADE) method for detecting neuronal activity patterns. The method evolved from identifying patterns of synchronous neuronal spikes[64], to incorporating spatio-temporal (non-synchronous) recurring patterns[65], and finally to refined statistical testing based on the temporal lags (3d-SPADE)[66]. While the conceptual approach to assessing neuronal activity patterns is shared between the three variants, the method's capabilities may differ. Therefore, identifying the precise implementation is crucial for reproducibility. Other conceptually different methods have been published to detect spatio-temporal patterns in spike data (e.g., CAD[67], ASSET[54]; see ref. [9] and ref. [15] for reviews). Despite differences in the algorithmic approach, underlying assumptions and interpretation, such methods share a common semantic quality and data type of their outputs (namely, the identified spike patterns). Using NEAO's cross-cutting groupings, it becomes possible to provide fitting semantic groupings to expose these similarities, while retaining a clear disambiguation between the involved methods and their variants.

To take advantage of NEAO, it must be associated with the data analysis processes. To this end, multiple scenarios can be identified. In this publication, we demonstrated how NEAO can annotate provenance tracks produced while generating a specific data artefact, which were structured using RDF. Here, objects of the **AnalysisStep** class of NEAO are manually associated with individual scripts or functions used to generate the provenance information. This association must also be understood by the tool collecting the provenance information. This lack of automation in our examples is a limitation for using NEAO for the analysis description, as this can be error prone, especially in large-scale or more complex analysis pipelines. Ideally, software tools used in the analysis process would identify their functionality using concepts defined by NEAO and provide semantic annotations without further intervention by the scientist (e.g., the toolbox developers use NEAO to provide RDF descriptions of the functions available in a toolbox). In an alternate second scenario, NEAO could be used to promote a manual classification of analysis results by the scientist, e.g., in a webform filled by the scientist with the matching analysis step(s) upon uploading an analysis result. This scenario is analogous to currently used mechanisms to share primary experimental data, where typically metadata cannot be captured in a fully automatized fashion. A third scenario can be considered, where NEAO is used not to describe a concrete analysis output but to semantically enrich a description of the process implemented in a particular analysis pipeline. In this way, NEAO supports the domain-agnostic characterization, findability, and comparison of process descriptions, which may form the basis for concrete implementations based on a suitable software stack.

Once NEAO enriches the provenance of a data set or a process description in the manner outlined above, scientists may exploit this semantic description in several ways. In the case of provenance tracks stored as RDF data, these could be made available through a suitable knowledge graph that organizes all analysis results obtained by a single scientist or a defined group of people, such as a lab or members of a project. In this way, the researcher's effort to document results is minimized while the findability of results is maximized. This holds particularly true for heterogeneous groups of people, where the abstracted semantics of NEAO help in building successful search queries. Moreover, NEAO's grouping of methodologies by analysis purpose may inspire further investigations and reveal similarities and discrepancies in separate and alternate analysis approaches (e.g., by two researchers working independently on the same data). As NEAO can abstract provenance information or process descriptions, one might also explore the possibility of integrating these semantically enriched descriptions with AI methods. This could lead to the creation of novel descriptions for the analysis (e.g., by automatically generating a textual description of the analysis or by suggesting equivalent code to perform the analysis using a different programming language).

Without annotating provenance or process descriptions using NEAO, the ontology can potentially act as a knowledge organization system through the components **SoftwareImplementation, SoftwarePackage** and **hasBibliographicReference**. These can summarize the capabilities of various software tools available for executing particular analysis methods, as well as relevant literature that interested scientists can refer to and learn about these methods. With this goal in mind, we aim to continuously update and curate NEAO to provide a useful resource for mapping implementations and descriptions of analysis methods. Such a map can help scientists to identify suitable toolboxes for a specific analysis task, and to assist in pinpointing differences between different software solutions and analysis methods. Moreover, for automated AI systems, the formalized representation of such knowledge as a curated ontology may prove a decisive asset in increasing the precision at which these systems can either disambiguate differences between methodologies or associate similar approaches.

The selected use cases demonstrate NEAO's intended role in describing analyses and facilitating the retrieval of relevant knowledge on the results. While these practical examples of NEAO's application demonstrate its ability to answer a large set of competency questions of varying complexity, importantly, they are not designed to empirically benchmark NEAO's ability to improve data sharing and knowledge transfer against other approaches regarding their accuracy and reliability. Nevertheless, we suggest that NEAO might be a formal foundation for designing studies to explore and contrast such different approaches for improving data sharing among researchers, and we can speculate on the added advantages. A straightforward approach is sharing the results with the source code and written documentation (e.g., a README file). This is non-standardized and prone to ambiguities, as outlined. In addition, no direct link between the processes in the pipeline inferred from the code and the actual files is guaranteed. Combined with the unstructured information in the documentation, we hypothesize that this approach will impair the findability and interoperability. As a second approach, the proposal put forth in this study suggests the automatic capture of provenance with manual annotations of specific elements in the pipeline with NEAO. This is expected to provide a direct improvement in findability and interoperability, as run-time information from the analysis process[57] is automatically structured and linked to outputs in a machine-readable form without ambiguities due to NEAO annotations. Nevertheless, among automated solutions to capture the analysis process, we may find that in certain scenarios it is sufficient to use solutions that capture a more coarse-grained (lightweight) view of the analysis provenance. In contrast, a more fine-grained description of the data manipulations may be necessary in other scenarios. A third approach for data sharing might involve using AI tools. For example, a system based on LLMs could support searching for information on the documentation or source code associated with the results. However, fine-tuning these AI models requires domain-specific knowledge, and the information in NEAO could provide a valid source. Therefore, AI-based methods would benefit from a curated ontology, as the formalized representation of neuroelectrophysiology knowledge may increase the precision with which these systems can either disambiguate differences between methodologies or associate similar approaches.

In this work, we also did not attempt comparisons to other ontologies. Direct comparisons to NEAO are difficult since existing ontologies have different scopes, and do not provide elements to describe the analysis of neuroelectrophysiology data in the same depth. Ontologies applicable to data analysis in general (e.g., EDAM, REPRODUCE-ME), workflows (e.g., P-Plan[68], D-PROV[69], ProvONE[70], Wf4Ever[71]), and biomedical sciences (e.g., OBI, OBCS) lack classes describing and organizing the specific neuroelectrophysiology methods and data (e.g., as depicted in Fig. 3). Ontologies for describing software (e.g., SWO[72], Function Ontology[73]) are not focused on the relationship between the specific analysis methods and their implementation by specialized toolboxes in the field, as we introduced in our model (Fig. 2). The ontologies for neuroscience (e.g., NIFTSD, CNO) offer descriptions of broad neuroscience terms. They focus primarily on describing data and its collection, and not specific elements in the analysis of electrophysiology experiments. Finally, the few ontologies implemented for electrophysiology within neuroscience are limited. Some focus on narrow domains, including ion channels (ICEPO) and inner ear (OBI_IEE) electrophysiology. Others provide some methods for the analysis of EEG-related data (NEMO and ref. [39]) or a few typical methods for data analysis (OEN). These ontologies do not cover the many methods for extracellular electrophysiology available in this initial implementation of NEAO. Therefore, it would be difficult to achieve a description of our use cases in the same depth as presented. To facilitate assessing the similarities and differences between NEAO and other ontologies, we provide a comparison in the online documentation (accessible at http://purl.org/neao).

The initial implementation of the NEAO has some limitations. First, although we have defined specific classes for the data and parameters associated with the steps in the analysis, NEAO does not use OWL to formally define the **AnalysisStep** class with respect to its inputs, outputs, and parameters, which reduces the expressivity of the ontology. However, this choice provided the flexibility needed to accommodate distinct code implementations found across various toolboxes. In MNE, for example, the class that computes a PSD allows the generation of either an array with the PSD data (as Elephant or SciPy) or a plot of the PSD. Therefore, placing an OWL restriction on the output of *PowerSpectralDensityAnalysis* to describe it as the *PowerSpectralDensity* class would generate a wrong inference in the latter case.

Second, we captured the concepts describing functionality implemented in common toolboxes that are focused specifically on the analysis of extracellular data. Currently, only 211 classes for specific methods are implemented, to analyze mostly spike activity and LFP data. As demonstrated in our examples, the initial implementation supports typical datasets and analysis methods using the main open source toolboxes in the field. However, we acknowledge that there are still many omissions. For example, although many methods described for LFP analysis are routinely employed with EEG/MEG data, this data type is different and the analysis may involve additional methodologies that are still not covered. Therefore, the ontology needs to be further expanded for the complete coverage of the field and validation with other types of data. This includes concepts more specific to intracellular recordings (e.g., synaptic events and membrane properties), macroscale measurements (e.g., event-related potentials in EEG experiments), and emerging methods specifically developed for

high-dimensional and complex data, which are often provided as standalone toolboxes (e.g., CEBRA[74]). These extensions will allow a more in-depth validation using other datasets.

Third, the curation and releases of NEAO updates need to incorporate novel methods quickly. In this initial version, we defined a framework to incorporate contributions from the community via the code repository on GitHub. This foundation is a starting point for the continuous curation and expansion of the ontology. However, we acknowledge such contributions are not systematic and community engagement may be incomplete in some areas. We consider some strategies to mitigate this limitation. First, we plan to closely collaborate with tool developers and invite them to suggest and enter new methods as functionality changes or new analysis methods are incorporated. Second, for a closer collaboration with the research community, we envision releasing an online directory to collect a library of methods for use by the community. This is intended to foster engagement and accelerate the integration of new methods in use. Finally, we aim to investigate the use of emerging artificial intelligence tools for literature review and synthesis, to optimize and automate the process of collecting the domain knowledge from publications.

Finally, NEAO cannot be directly integrated into the Open Biological and Biomedical Ontologies (OBO) Foundry. The OBO Foundry was created to support biomedical data integration through the development of interoperable ontologies[75,76], and the participating ontologies rely on a set of principles which include the alignment to foundational ontologies such as the Basic Foundational Ontology (BFO)[77] and the Relation Ontology (RO)[78,79]. These are important when aiming to reuse ontologies across different domains, a mission of the OBO. Although it is our interest to integrate NEAO with other ontologies, especially in the scope of the OBO Foundry project, the lack of alignment in this initial release was a design choice to be able to use NEAO quickly and custom-fit to the domain according to the scopes we proposed. In this way, we avoid constraints introduced by ontological commitments to other domains and concentrate on the diversity of domain-specific software that can be used for analysis processes. Nevertheless, NEAO adheres to the OBO Foundry principles of open, accessible, and version-controlled ontologies, and future work towards integration is aligning it with suitable OBO ontologies. The first step is to identify the concepts within the upper BFO classes (i.e., *continuant* and *occurrent*) that are most similar to **AnalysisStep, Data**, and **AnalysisParameter** NEAO classes, adjusting definitions and axioms for the alignment. This is followed by alignment to specific OBO ontologies with overlapping concepts. For example, the *data transformation* class in OBI, defined as "a planned process that produces output data from input data" overlaps with the concept introduced by **AnalysisStep** in NEAO. The redundant concepts in NEAO will be substituted, and the remaining NEAO domain-specific terms adjusted to match the imported classes and taxonomies. As a last step, metadata and identifiers must be harmonized according to the OBO Foundry policies, especially the number-based system for the URIs identifying the classes.

Among the future steps envisioned to expand NEAO, we intend to supply these missing alignments between NEAO and other ontologies. This not only enhances compatibility with OBO Foundry but allows the reuse of concepts that are already structured in well-defined ontologies. For example, the QUDT[42] is a candidate for standardizing the description of physical quantities. QUDT could be employed to specify parameter values (e.g., the frequency resolution in our examples) or to provide an extended description of an output (e.g., the unit of a PSTH depending on the normalization applied). Moreover, the Software Ontology (SWO)[72] provides a model to describe software that could be used to extend the **SoftwareImplementation** concept, and the Ontology of Bioscientific Data Analysis and Data Management (EDAM)[32] has general concepts that could be aligned to NEAO base classes to foster interoperability.

A second future improvement to NEAO is to expand the abstract classes **ElectrophysiologySignalSource** and **DataRepresentation** to allow more specific queries on these semantic dimensions as proposed in Table 2. For this purpose, novel classes need to be developed, potentially leveraging concepts and terms from other ontologies or projects such as InterLex/NeuroLex[80]. For example, the QUDT ontology also describes data types. Therefore, we plan to implement additional modules in NEAO to include more specific definitions for the usual sources for electrophysiology signals and typical data representations.

Finally, in the future, we invite to explore the potential of integrating NEAO with existing analysis toolboxes in an effort to provide more toolbox-centric representations of the available methods and support the automation of the analysis description. For example, an additional ontology module dedicated to the Elephant toolbox could produce a specific subclass of *ComputePowerSpectralDensityWelch*, where the **hasInput, hasOutput** and **usesParameter** properties are modeled restrictions that describe the actual functionality with respect to valid parameters and supported input/outputs for that specific tool. Such toolbox-specific representations would integrate the description with respect to versions and package information, facilitating inference on steps, data, and parameters used. With this integration of NEAO on the level of individual data analysis tools, a toolbox could provide its own ontological representation based on NEAO that could be readily used when describing or annotating an analysis. This prevents the need for users to manually define the classes for function arguments and returns as demonstrated in this study (cf., Fig. 8). In addition, a toolbox could automatically use NEAO classes and properties to provide structured metadata records with the analysis outputs. With this builtin support, even users not familiar with ontological frameworks benefit from the automated and less error-prone standardization achieved with NEAO. We suggest that direct annotations of the Python functions as we used in our examples could already be integrated in the code released and maintained by toolbox developers, facilitating automated tools that aim to describe the analysis based on the code execution. Complementing that effort, toolbox developers could extend existing tools (for example, by using plugins) to use NEAO and provide automated descriptions of computational workflows. In the end, automation is the ultimate goal, and NEAO presents itself as a bridge to support automated systems.

In conclusion, we implemented the Neuroelectrophysiology Analysis Ontology as a new domain-specific ontology aimed to improve the findability, interoperability, and reusability of outcomes produced by the analysis of electrophysiology data in the scope of neuroscience. The semantic framework defined by NEAO facilitates

```
def power_spectral_density(signal, frequency_resolution):

    ...  # process input data in `signal` to obtain the PSD

    return freqs, psd

# To use NEAO to annotate `power_spectral_density` execution
# when capturing provenance with Alpaca, the following
# `__ontology__` dictionary could be inserted into
# the function object:

power_spectral_density.__ontology__ = {
    "function": "neao_steps:ComputePowerSpectralDensityWelch",
    "arguments": {
        "signal": "neao_data:TimeSeries",
        "frequency_resolution": "neao_params:FrequencyResolution"
    },
    "returns": {
        1: "neao_data:PowerSpectralDensity"
    },
    "namespaces": {
        "neao_steps": "http://purl.org/neao/steps#",
        "neao_data": "http://purl.org/neao/data#",
        "neao_params": "http://purl.org/neao/parameters#"
    },
}
```

Fig. 8 Example illustrating an approach to utilize NEAO classes to annotate Python functions in a script. In this example, a Python function to compute the PSD is defined. Although the function internally uses the Welch method, the function name is generic and does not reflect this. The function has two arguments (in italic font) and returns a tuple with two objects. The second object in the return tuple (with index 1) contains the computed PSD. After the function is declared, the special dictionary *__ontology__* is defined as a function attribute. The dictionary keys (bold text) define four main annotations. The *function* annotation is used to specify the URI of the NEAO class that represents the analysis step (in this example, the class for the computation of a PSD using the Welch method). The *arguments* annotation defines a dictionary to associate specific NEAO URIs to each of the function arguments (strings in italic font): *signal* is defined as time series data, and *frequency_resolution* is defined as a parameter that controls the frequency resolution of the PSD estimates. The keys in this dictionary match the names of the arguments in the function definition. Annotations for the individual returns can be defined with the *returns* dictionary element (using integers representing the index in the return tuple). Here, the second function return is mapped to the NEAO class representing PSD data. To use CURIEs, proper namespaces are defined in the *namespaces* dictionary element. This approach to connect software functionality to concepts defined by NEAO is interpreted by the Alpaca software to add URIs from ontologies to the RDF triples of the captured provenance.

the unambiguous description of the heterogeneous processes and elements involved, especially the specific data, parameters, code implementations, and bibliographic references that are associated with the individual steps of the analysis. In the end, this can be used to achieve a solid description of the data analysis process, which not only facilitates querying information about the analysis but also improves its understanding for a reliable interpretation of the findings.

Methods

Implementation of the Neuroelectrophysiology Analysis Ontology. This initial version of NEAO was implemented using the Web Ontology Language (OWL) using the Protégé[81] editor (version 5.5). Classes are named in upper camel case convention (e.g., *ComputeInterspikeIntervals*), and properties in lower camel case (e.g., *hasOutput*). As we implemented a domain-specific ontology, in the initial version we did not align to fundamental ontologies such as the BFO[77] or RO[78]. The rationale was to avoid introducing unnecessary ontological commitments, as the basic competency questions to be addressed do not require the abstract concepts described by these ontologies. We used standard ontologies for the general description of resources, such as Dublin Core DCMI Metadata Terms[51], SKOS[43], and BiRO[50].

We started by defining the schema for an abstract model representing a step in the analysis of data from neuroelectrophysiology experiments. The model is centered on the conceptual definition of an analysis step as a process that transforms/generates data based on specific parameters. The step definition considers that the specific identification of one step uses some well-defined properties (i.e., a particular data input or output, a bibliographic reference, and code implementation). Based on the schema, we defined a set of competency questions that should be addressed to guide the development of the ontology. This resulted in the classes and properties used as upper level for the rest of the ontology (*base* module).

We then used a recent review[15] that compared major open-source toolboxes for the analysis of neuroelectrophysiology data and described common and specific functionalities regarding the analyses that can be performed. This review covers toolboxes with active development (updates in the last five years at the time of its publication) and with a valid link for download. The final list included BrainStorm[46], Chronux[47], Elephant[13], FieldTrip[45], gramm[82], Spyke Viewer[83] and SPIKY[84]. We complemented the information on the

NEAO property	Similar property
hasInput	prov:used
hasOutput	prov:generated
usesParameter	alpaca:hasParameter
isImplementedIn	alpaca:usedFunction
nameInDefinition	alpaca:functionName

Table 11. Mappings of properties from NEAO to the Alpaca provenance model. The Alpaca package uses an ontology derived from the W3C PROV-O ontology to structure the provenance information captured while executing scripts that process data. The generic provenance relationships *prov:wasGeneratedBy* (or its inverse *prov:generated*) and *prov:used* define the input and output data objects for each function execution (a type of *prov:Activity*). These relationships are semantically similar to the NEAO properties *hasInput* and *hasOutput*. The *alpaca:Function* class is used to describe the code of the Python function used in the function execution, and it is semantically similar to the **Function** class in NEAO. Therefore, the *alpaca:usedFunction* property represents a similar relationship to the *isImplementedIn* property from NEAO, and the *alpaca:functionName* property stores the name of the function definition as expected by the *nameInDefinition* NEAO property. This mapping translates the provenance information captured by systems that structure provenance with PROV-O (such as Alpaca) into the model implemented in NEAO. In the table, the prefix *prov:* identifies the namespace of the PROV-O ontology and *alpaca:* the namespace of the Alpaca ontology.

review by manually searching the API description of the toolboxes Elephant[13] (RRID:SCR_003833; https://python-elephant.org), MNE[14] (RRID:SCR_005972; https://mne.tools), and NiTime[44] (RRID:SCR_002504; https://nipy.org/nitime). Therefore, a broad selection of software used for the analysis of data from neuroelectrophysiology experiments was used as the basis for gathering domain-specific knowledge. As a result, the concepts that define specific steps in the analysis of neuroelectrophysiology data were identified.

We defined a controlled vocabulary and descriptions to refer to those concepts, which we termed analysis steps as the atomic elements that compose an analysis process. They were entered as classes in the ontology (*steps* module) with relevant annotations. We then identified the concepts defining the data inputs and outputs of those steps and entered them as classes in the ontology (*data* module). Finally, we started defining specific parameters that define the behavior of an analysis step (*parameters* module). For any class or property, we discussed labels and descriptions.

We defined relevant groupings using the Rector normalization technique[53]. An approach in OWL to provide a hierarchical classification of classes is to manually assert that specific classes are subclasses of more general ones. This may become difficult to manage and prone to errors as the ontology grows and becomes more complex. In NEAO, the complexity is already present because one of the objectives is to provide groupings across many dimensions (e.g., distinct algorithmic implementations or specific purposes in the analysis). The normalization approach constructs inferred classes based on logical definitions rather than manually asserted hierarchies using subclass statements. With normalization, NEAO is modular and the design allows the community to expand and incorporate new functionality easily. For example, consider that a novel method to estimate functional connectivity was developed and a class representing it will be added to the ontology. The new class for the method can be added as a separate branch in the main taxonomy for analysis steps. To define that the new method is suitable to estimate functional connectivity, only an additional property restriction axiom stating that the class has the *FunctionalConnectivityPurpose* as the value for property *hasPurpose* needs to be added. This will allow the reasoner to automatically infer that the new method is also part of the *FunctionalConnectivityAnalysis* class (that groups all methods to estimate functional connectivity). Therefore, expanding the ontology with new methods is straightforward. In addition, consistency is improved as such logical definitions make the criteria for class membership explicit. Finally, the existing definitions (e.g., the class *FunctionalConnectivityAnalysis*) can be reused when adding new methods, making the ontology easier to maintain and avoiding to manually restructure the grouping class hierarchies.

To add analysis methods to NEAO, contributors from the community can submit suggestions through the GitHub repository issue tracker (accessible at http://purl.org/neao/suggestion). A template for the suggestion is defined, asking for detailed information on the method: name, bibliographic reference, abbreviation, and a detailed description of the method's purpose and implementation. This structured process allows NEAO maintainers to review and evaluate the suggestion for inclusion, and to define the implementation of the new classes in OWL. Finally, it is also possible to open issues to discuss misrepresentations and suggest improvements to the existing definitions and references of NEAO classes (accessible at http://purl.org/neao/improvement). For a full description of the contribution process, see the information on the NEAO code repository (http://purl.org/neao/repository).

To use NEAO to describe analysis with newly developed software tools and packages, the current ontology structure allows directly associating the new code with methods represented in NEAO. Software information is represented in RDF as individuals defined with the **SoftwareImplementation** and **SoftwarePackage** classes and properties. This allows structuring the main information of any software associated with an analysis step (e.g., similar to the example in Fig. 4).

Experimental dataset. For the analyses that used experimental data, we used a publicly available dataset containing massively parallel electrophysiological recordings in the motor areas of monkeys during the execution

of an instructed delay reach-to-grasp task[85]. The experiment design, subject details, task protocol, data acquisition setup, and resulting datasets were previously described[86]. Briefly, each subject was implanted with one Utah electrode array (4 × 4 mm, 96 active electrodes) in the primary motor/premotor cortices. During a trial of the task, visual cues were delivered through an LED panel to instruct the monkey to grasp an object using either a precision (PG) or a side grip (SG). After a 1000 ms delay, a new visual cue requested the monkey to pull an object against a load that required either a high (HF) or low (LF) pulling force. Therefore, four possible trial types were defined: SGLF, SGHF, PGLF, or PGHF. If the trial was completed successfully, the monkey received a food reward. A recording session consisted of several repetitions of each trial type that were acquired continuously in a single recording file. Neural activity was recorded using a Blackrock Microsystems Cerebus data acquisition system (raw signals were bandpass-filtered between 0.3 and 7500 Hz at the headstage level and digitized at 30 KHz with 16-bit resolution). The published datasets were extensively annotated with experimental metadata as described in the data publication[86]. Information from different sources (e.g., Utah array datasheets, experimenter records, configuration files of the recording setup) was compiled into a metadata file[87,88] using the odML[89] format (RRID:SCR_001376; https://g-node.github.io/python-odml). The Neo[90] library was used to load the datasets using Python (RRID:SCR_000634; https://neuralensemble.org/neo). Neo introduces a standardized data model and Python objects to handle neuroelectrophysiological data and associated metadata in a format-agnostic manner. A custom Neo interface was implemented to read the raw single-session recording files and offline-sorted spike data in the Blackrock Microsystems formats (NS2, NS5, NS6, NEV) together with the metadata of the odML file. This interface provided Neo data objects with all the relevant metadata as annotations (see ref. [86] for details). In the end, these objects were saved into files using the Neuroscience Information Exchange[91] (NIX) format (RRID:SCR_016196; https://nixio.readthedocs.io). For each subject, a full dataset including the raw electrode data at 30 kHz bandwidth is provided, as well as a reduced dataset with the neural data downsampled to 1000 Hz. In the examples in this paper, we used the reduced NIX dataset of monkey N, identified as *i140703-001_no_raw.nix*, and available in the repository hosting the published dataset (see Data Availability)[85].

Use case analyses. All analyses were implemented as individual Python scripts. Analysis 1 and 2 have multiple implementations to use different methods and/or toolboxes. Unless stated otherwise, the Electrophysiology Analysis Toolkit (Elephant; RRID:SCR_003833) version 0.14.0[92] was used for the analyses.

Power spectral density (Analysis 1). The NIX dataset was loaded and cut into trials using functions provided by Neo. Here, trials were defined as the interval between the task events TS_ON and STOP. These events mark the start and end of a full trial (successful or not successful) in the reach-to-grasp task[86]. For each trial ($N = 161$), the neural data time series was low-pass filtered using a Butterworth filter (fourth order, 250 Hz cutoff) followed by downsampling to 500 Hz. The PSD was computed using one of three possible method/toolbox combinations: Welch method in Elephant, multitaper method in Elephant, or Welch method in SciPy[48] (RRID:SCR_008058). The power estimates for each electrode were plotted between 0 and 100 Hz, and the plot was saved as a PNG file named with the trial ID as defined in the annotations. Therefore, the script output is one PNG file for each trial. One trial was too short to be able to compute a PSD for the requested parameters, and the final plot count for each analysis was 160. Three scripts (Analysis 1.1, 1.2, and 1.3) were implemented with the same data loading, data preprocessing, and plotting steps. Only the steps used to compute the PSD varied. In addition, the plot function was adjusted for the version using SciPy (Analysis 1.3) to manually define the physical quantity of the spectrum, as SciPy outputs NumPy arrays while Elephant outputs arrays with the physical quantities defined (quantities Python package). Each script saved the respective output files in a separate folder.

Surrogate interspike interval histograms (Analysis 2). The NIX dataset was loaded, and data was cut into trials using Neo. Trials were defined similarly as described for the power spectral density analysis above, but only correct trials were considered ($N = 142$). The analysis used spike trains containing the activity of a single neuron (single-unit activity; SUA) if the signal-to-noise ratio (SNR) was equal to or greater than 5 and the neuron had a mean firing rate equal to or greater than 15 Hz in the trial. For inclusion, the neuron must match the criteria in all 142 trials. In the end, six units were selected for the analysis.

The ISIs for the spike train of a single trial were computed, and a histogram of the ISIs was obtained using 5 ms bins. The 142 ISIHs of the neuronal unit were merged to get the final ISIH across trials. In addition, 30 surrogates were generated from each spike train containing the data of a single trial. ISIHs across trials were computed for the surrogates similarly to the experimental data. In the end, 30 surrogate ISIHs were obtained for a single unit. These were averaged, and the standard deviation was computed. The ISIH for the neuronal unit was plotted using bars, and the mean and SD of the 30 surrogate ISIHs were plotted as lines. The plots were saved as a PNG file named with the unit ID in the dataset.

Two scripts were implemented with the same data loading, preprocessing, and plotting steps. Only the function used to compute the surrogates differed between scripts[59]. The first script used uniform spike dithering with a dither time of 25 ms, excluding spikes dithered outside the spike train duration. The second script used trial shifting with a dither time of 30 ms. Each script writes the respective output files in a separate folder.

ISIHs of artificial data (Analysis 3). 200 spike trains (100 s duration) were generated using either a stationary Poisson process ($N = 100$) or a stationary gamma process ($N = 100$) to generate spike trains with a target firing rate of 10 Hz. The gamma process used a shape factor of 1, which is equivalent to a Poisson process and produces results with similar statistical properties. All the spike trains were merged into a single list. For each spike train in the list, the ISIs were computed, and the CV2 measure[61] was calculated to estimate the interval variability. A

histogram of the ISIs was computed (10 ms bin size) and plotted together with the CV2. The plot was saved as a PNG file named with the order of the spike train in the list (range 1 to 200). All plots were saved in a single folder.

Annotation of Python functions with NEAO. In the analysis scripts, a Python decorator was used to embed semantic information provided by the NEAO inside the Python functions used for processing data. The decorator inserted a dictionary as the special *__ontology__* attribute of the function object. An example showing the structure of the *__ontology__* dictionary is shown in Fig. 8. This dictionary can store URIs to classes describing the function, parameters or return objects, and prefixes defining any namespaces in compact URIs (CURIEs). A CURIE is an abbreviated form of a URI, avoiding the repetition of a prefix used repeatedly in the ontology URIs. For instance, the string https://purl.org/neao/base# is used as the prefix of the URIs in the base module of NEAO. Instead of using the full URI <https://purl.org/neao/base#term> for the annotation with *term*, one can associate the prefix with the string *neao_base* and write the annotation as *neao_base:term*. Using this simplified notation, functions in each use case analysis script were annotated with terms defined by NEAO.

The specific annotations used in each script variant from Analyses 1–3 are summarized in the Supplementary Text. The examples in these use cases relied on the Automated Lightweight Provenance Capture (Alpaca; RRID:SCR_023739) Python toolbox[57] to capture provenance enriched with the semantic information defined by those annotations. This approach automatically provides a detailed description of the analysis as an RDF graph consisting of the data flow and the sequence of functions executed throughout the script. However, NEAO is not dependent on Alpaca and can be used to describe an analysis using different approaches. For example, the analysis scripts might have been implemented to manually insert text metadata in the result PNG plots, where terms of NEAO act as keyword identifiers. This metadata could comprise a structured record describing inputs, steps, and outputs using the URIs defined in NEAO. Alternatively, the script could have used NEAO to manually represent the analysis as an RDF graph to be added as metadata to the results (e.g., similar to Fig. 4). Another use case scenarios may involve the use of NEAO to create a machine-readable and software-independent process description of a proposed analysis plan.

Provenance capture. We used Alpaca version 0.2.0[93] to instrument each script used for a particular use case analysis to capture provenance information[57]. Alpaca uses a function decorator to identify inputs, outputs, and parameters of the functions executed inside a script that processes data. Additional details on the data objects are also captured (e.g., object attributes such as array shapes or annotations in Neo objects). At the end of the script execution, Alpaca saves provenance data as RDF in a sidecar file to the results, using an ontology derived from PROV-O[57]. This work used the Turtle serialization format to store the captured provenance information. Thus, each analysis script saves a file with TTL extension and PNG files in the folder storing the outputs. Alpaca can read semantic information embedded into function and data objects as a dictionary defined in the *__ontology__* attribute and add this information to the RDF output. In the end, Alpaca annotated the provenance information of the use case analyses with the classes defined by NEAO.

Knowledge graph and SPARQL queries. To demonstrate how NEAO is used to query information regarding the performed analyses, we used the Ontotext GraphDB Free database (Desktop installation) running as a local RDF triple store. The GraphDB Free RDF triple store database was obtained from the Ontotext website accessible at https://www.ontotext.com/products/graphdb. After running the scripts of the use case analyses, RDF data in the TTL files with provenance information and the OWL files defining NEAO, PROV-O, and the Alpaca ontologies were inserted into an empty repository using the importrdf utility tool (the *OWL2-RL* rule set was used). The GraphDB Desktop application was started, and a local SPARQL endpoint was accessible.

Several SPARQL update queries added additional triples to the graph. These triples map the model utilized by Alpaca for describing function execution provenance in RDF to the model defined by NEAO to describe the analysis steps. Table 11 shows the similar properties.

First, for the analysis steps, if a function execution captured by Alpaca was annotated with a class defined by NEAO (i.e., **AnalysisStep**), and one of the properties defined in the Alpaca/PROV-O ontologies pointed to an individual of a class also defined by NEAO (i.e., **Data** or **AnalysisParameter**), the triple using the appropriate NEAO property was added. For parameters annotated with **AnalysisParameter** classes, the actual value used in the function execution can be retrieved by the *alpaca:pairValue* property, which Alpaca uses to structure the name and values of Python function execution parameters when serializing the provenance to RDF.

Second, to use NEAO to describe the software implementation of the step, information about the function code captured by Alpaca (e.g., version, name, and source module) was transformed into individuals of the **Function** and **SoftwarePackage** NEAO classes and their appropriate relationships.

Finally, some functions used in the analyses might produce outputs that were grouped into containers (e.g., Python lists). This is the case, for instance, of the generation of surrogates by the trial shifting method. As the input to the method is a collection of spike trains (the multiple trials), the actual output of the surrogate generation analysis step is the collection with all the trial spike trains dithered. In this case, Alpaca uses the PROV-O *prov:hadMember* property to describe the container membership of the data objects returned by the function. As the elements of these containers were annotated with **Data** classes defined by NEAO to attach the proper semantic meaning, an additional query was executed to properly map these special container outputs to the analysis step using the **hasOutput** property.

The SPARQL update queries to insert all these complementary triples are available in Fig. S1–S3 and the code repository accompanying this paper (see Code availability below). In the end, when the provenance information had semantic annotations using NEAO classes, a mapping of the PROV-O and Alpaca ontology relationships to the ones defined by NEAO was created. This produced a knowledge graph with all the provenance of the analysis output files linked to NEAO definitions.

We used the Python gastrodon library to execute multiple SPARQL queries in the knowledge graph to answer specific questions regarding the analyses. gastrodon can connect to the GraphDB SPARQL endpoint, execute a SPARQL query, and format the query results as Pandas DataFrames, allowing easy formatting and output aggregations (e.g., pivot tables). Each query was run using a Python script that saved a raw result table as a CSV file. These CSV files were loaded into Pandas DataFrames and transformed into descriptive LaTeX tables for reporting.

Data availability

The GitHub repository containing the NEAO OWL sources is freely accessible at https://purl.org/neao/repository that points to https://github.com/INM-6/neuroephys_analysis_ontology at the time of publication. The ontology documentation can be accessed at http://purl.org/neao. The dataset used in the use-case analyses is publicly available in a repository[85] hosted on the GIN service. Instructions for downloading the dataset are provided at the repository, and it can also be directly downloaded at the permanent link address https://gin.g-node.org/INT/multielectrode_grasp/src/a6d508be099c41b4047778bc2de55ac216f4e673/datasets_nix/i140703-001_no_raw.nix. The output files produced by the three analyses, the raw results of the SPARQL queries as CSV files, and the presented result tables are available in a freely accessible Zenodo repository[56].

Code availability

The Zenodo repository with the code implementing the use case analyses, the interface to the local knowledge graph, the SPARQL queries, and scripts to generate the presented result tables are freely accessible[56]. The open-source toolbox Alpaca that was used to capture the provenance information annotated with NEAO is accessible at https://alpaca-prov.readthedocs.io and can be installed from PyPI (https://pypi.org/project/alpaca-prov) or the code repository (https://github.com/INM-6/alpaca). All software and codes were run using Ubuntu 18.04.6 LTS 64-bit and Python 3.9. Specific details of the execution environment are described in the code repository of the use cases.

Received: 15 January 2025; Accepted: 15 May 2025;
Published online: 29 May 2025

References

1. Huang, Z. Brief History and Development of Electrophysiological Recording Techniques in Neuroscience. In Li, X. (ed.) *Signal Processing in Neuroscience*, 1–10, https://doi.org/10.1007/978-981-10-1822-0_1 (Springer, Singapore, 2016).
2. Buzsáki, G., Anastassiou, C. A. & Koch, C. The origin of extracellular fields and currents–EEG, ECoG, LFP and spikes. *Nat Rev Neurosci* **13**, 407–420, https://doi.org/10.1038/nrn3241 (2012).
3. Lalley, P. M. Intracellular recording. In Binder, M. D., Hirokawa, N. & Windhorst, U. (eds.) *Encyclopedia of Neuroscience*, 2019–2026, https://doi.org/10.1007/978-3-540-29678-2_2554 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2009).
4. Subash, P. *et al*. A comparison of neuroelectrophysiology databases. *Scientific Data* **10**, 719, https://doi.org/10.1038/s41597-023-02614-0 (2023).
5. Hodgkin, A. L. & Huxley, A. F. Action Potentials Recorded from Inside a Nerve Fibre. *Nature* **144**, 710–711, https://doi.org/10.1038/144710a0 (1939).
6. Buzsáki, G. Large-scale recording of neuronal ensembles. *Nat Neurosci* **7**, 446–451, https://doi.org/10.1038/nn1233 (2004).
7. Siegle, J. H. *et al*. Survey of spiking in the mouse visual system reveals functional hierarchy. *Nature* **592**, 86–92, https://doi.org/10.1038/s41586-020-03171-x (2021).
8. Bastos, A. M. & Schoffelen, J.-M. A tutorial review of functional connectivity analysis methods and their interpretational pitfalls. *Frontiers in Systems Neuroscience* **9**, 175, https://doi.org/10.3389/fnsys.2015.00175 (2016).
9. Grün, S., Quaglio, P., Stella, A. & Torre, E. Statistical evaluation of spatio-temporal spike patterns. In Jaeger, D. & Jung, R. (eds.) *Encyclopedia of Computational Neuroscience*, 1–4, https://doi.org/10.1007/978-1-4614-7320-6_100702-1 (Springer New York, New York, NY, 2020).
10. Welch, P. The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics* **15**, 70–73, https://doi.org/10.1109/TAU.1967.1161901 (1967).
11. Thomson, D. Spectrum estimation and harmonic analysis. *Proceedings of the IEEE* **70**, 1055–1096, https://doi.org/10.1109/PROC.1982.12433 (1982).
12. Percival, D. B. & Walden, A. T. *Spectral Analysis for Physical Applications* (Cambridge University Press, Cambridge, 1993).
13. Denker, M., Yegenoglu, A. & Grün, S. Collaborative HPC-enabled workflows on the HBP Collaboratory using the Elephant framework. In *Neuroinformatics 2018*, P19, https://doi.org/10.12751/incf.ni2018.0019 (2018).
14. Gramfort, A. *et al*. MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience* **7**, 267, https://doi.org/10.3389/fnins.2013.00267 (2013).
15. Unakafova, V. A. & Gail, A. Comparing open-source toolboxes for processing and analysis of spike and local field potentials data. *Frontiers in Neuroinformatics* **13**, 57, https://doi.org/10.3389/fninf.2019.00057 (2019).
16. Grün, S. & Rotter, S. (eds.) *Analysis of Parallel Spike Trains* (Springer, 2010).
17. Muller, L., Chavane, F., Reynolds, J. & Sejnowski, T. J. Cortical travelling waves: Mechanisms and computational principles. *Nature Reviews Neuroscience* **19**, 255–268, https://doi.org/10.1038/nrn.2018.20 (2018).
18. Dann, B., Michaels, J. A., Schaffelhofer, S. & Scherberger, H. Uniting functional network topology and oscillations in the fronto-parietal single unit network of behaving primates. *eLife* **5**, e15719, https://doi.org/10.7554/eLife.15719 (2016).
19. Brown, E. N., Kaas, R. E. & Mitra, P. P. Multiple neural spike train data analysis: state-of-the-art and future challenges. *Nature Neuroscience* **7**, 456–461, https://doi.org/10.1038/nn1228 (2004).
20. Semedo, J. D., Gokcen, E., Machens, C. K., Kohn, A. & Yu, B. M. Statistical methods for dissecting interactions between brain areas. *Current Opinion in Neurobiology* **65**, 59–69, https://doi.org/10.1016/j.conb.2020.09.009 (2020).
21. Stringer, C. & Pachitariu, M. Analysis methods for large-scale neuronal recordings. *Science* **386**, eadp7429, https://doi.org/10.1126/science.adp7429 (2024).
22. Gutzen, R. *et al*. A modular and adaptable analysis pipeline to compare slow cerebral rhythms across heterogeneous datasets. *Cell Reports Methods* **4**, 100681, https://doi.org/10.1016/j.crmeth.2023.100681 (2024).
23. Bernabé, C. H. *et al*. The use of foundational ontologies in biomedical research. *Journal of Biomedical Semantics* **14**, 21, https://doi.org/10.1186/s13326-023-00300-z (2023).

24. Kaplan, R. M. & Beatty, A. S. (eds.) *Ontologies in the Behavioral Sciences: Accelerating Research and the Spread of Knowledge* (National Academies Press, Washington, D.C., 2022).
25. Hoehndorf, R., Schofield, P. N. & Gkoutos, G. V. The role of ontologies in biological and biomedical research: a functional perspective. *Briefings in Bioinformatics* **16**, 1069–1080, https://doi.org/10.1093/bib/bbv011 (2015).
26. Larson, S. & Martone, M. Ontologies for neuroscience: What are they and what are they good for? *Frontiers in Neuroscience* **3**, 60–67, https://doi.org/10.3389/neuro.01.007.2009 (2009).
27. Studer, R., Benjamins, V. R. & Fensel, D. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering* **25**, 161–197, https://doi.org/10.1016/S0169-023X(97)00056-6 (1998).
28. Wilkinson, M. D. *et al*. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data* **3**, 160018, https://doi.org/10.1038/sdata.2016.18 (2016).
29. Bandrowski, A. *et al*. The Ontology for Biomedical Investigations. *PLOS ONE* **11**, e0154556, https://doi.org/10.1371/journal.pone.0154556 (2016).
30. Zheng, J. *et al*. The Ontology of Biological and Clinical Statistics (OBCS) for standardized and reproducible statistical analysis. *Journal of Biomedical Semantics* **7**, 53, https://doi.org/10.1186/s13326-016-0100-2 (2016).
31. Tenenbaum, J. D. *et al*. The Biomedical Resource Ontology (BRO) to Enable Resource Discovery in Clinical and Translational Research. *Journal of biomedical informatics* **44**, 137–145, https://doi.org/10.1016/j.jbi.2010.10.003 (2011).
32. Ison, J. *et al*. EDAM: An ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics* **29**, 1325–1332, https://doi.org/10.1093/bioinformatics/btt113 (2013).
33. Bug, W. J. *et al*. The NIFSTD and BIRNLex Vocabularies: Building Comprehensive Ontologies for Neuroscience. *Neuroinformatics* **6**, 175–194, https://doi.org/10.1007/s12021-008-9032-z (2008).
34. Le Franc, Y. *et al*. Computational Neuroscience Ontology: A new tool to provide semantic meaning to your models. *BMC Neuroscience* **13**, P149, https://doi.org/10.1186/1471-2202-13-S1-P149 (2012).
35. Frishkoff, G., LePendu, P., Frank, R., Liu, H. & Dou, D. Development of Neural Electromagnetic Ontologies (NEMO): Ontology-based Tools for Representation and Integration of Event-related Brain Potentials. *Nature Precedings* 1–1, https://doi.org/10.1038/npre.2009.3458.1 (2009).
36. Le Franc, Y. *et al*. Describing Neurophysiology Data and Metadata with OEN, the Ontology for Experimental Neurophysiology. In *Front. Neuroinform. Conference Abstract: Neuroinformatics 2014*, https://doi.org/10.3389/conf.fninf.2014.18.00044 (Frontiers, 2014).
37. Hinard, V. *et al*. ICEPO: The ion channel electrophysiology ontology. *Database* **2016**, baw017, https://doi.org/10.1093/database/baw017 (2016).
38. Farrell, B. & Bengtson, J. Scientist and data architect collaborate to curate and archive an inner ear electrophysiology data collection. *PLOS ONE* **14**, e0223984, https://doi.org/10.1371/journal.pone.0223984 (2019).
39. Štebeták., J. & Moucek., R. Ontology based Description of Analytic Methods for Electrophysiology. In *Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2016) - HEALTHINF*, 420–425, https://doi.org/10.5220/0005814004200425. INSTICC (SciTePress, 2016).
40. Arndt, S. *et al*. Metadata4Ing: An ontology for describing the generation of research data within a scientific activity. *Zenodo* https://doi.org/10.5281/zenodo.5957103 (2024).
41. Samuel, S. & König-Ries, B. End-to-End provenance representation for the understandability and reproducibility of scientific experiments using a semantic approach. *Journal of Biomedical Semantics* **13**, 1, https://doi.org/10.1186/s13326-021-00253-1 (2022).
42. FAIRsharing Team. FAIRsharing record for: QUDT; Quantities, Units, Dimensions and Types, https://doi.org/10.25504/FAIRsharing.d3pqw7.
43. Miles, A. & Bechhofer, S. SKOS Simple Knowledge Organization System Reference. *W3C Recommendation* http://www.w3.org/TR/2009/REC-skos-reference-20090818 (2009).
44. Gorgolewski, K. *et al*. Nipype: A Flexible, Lightweight and Extensible Neuroimaging Data Processing Framework in Python. *Frontiers in Neuroinformatics* **5**, 13, https://doi.org/10.3389/fninf.2011.00013 (2011).
45. Oostenveld, R., Fries, P., Maris, E. & Schoffelen, J.-M. FieldTrip: Open Source Software for Advanced Analysis of MEG, EEG, and Invasive Electrophysiological Data. *Computational Intelligence and Neuroscience* **2011**, 156869, https://doi.org/10.1155/2011/156869 (2011).
46. Tadel, F., Baillet, S., Mosher, J. C., Pantazis, D. & Leahy, R. M. Brainstorm: A User-Friendly Application for MEG/EEG Analysis. *Computational Intelligence and Neuroscience* **2011**, 879716, https://doi.org/10.1155/2011/879716 (2011).
47. Bokil, H., Andrews, P., Kulkarni, J. E., Mehta, S. & Mitra, P. Chronux: A Platform for Analyzing Neural Signals. *Journal of neuroscience methods* **192**, 146–151, https://doi.org/10.1016/j.jneumeth.2010.06.020 (2010).
48. Virtanen, P. *et al*. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17**, 261–272, https://doi.org/10.1038/s41592-019-0686-2 (2020).
49. Bartlett, M. S. PERIODOGRAM ANALYSIS AND CONTINUOUS SPECTRA. *Biometrika* **37**, 1–16, https://doi.org/10.1093/biomet/37.1-2.1 (1950).
50. FAIRsharing Team. FAIRsharing record for: Bibliographic Reference Ontology, https://doi.org/10.25504/FAIRSHARING.99DA5F.
51. DCMI Usage Board. DCMI Metadata Terms. *Dublin Core Metadata Initiative* http://dublincore.org/specifications/dublin-core/dcmi-terms/2020-01-20 (2020).
52. Percival, D. B. & Walden, A. T.*Wavelet Methods for Time SeriesAnalysis* (Cambridge University Press, Cambridge, 2000).
53. Rector, A. L. Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In *Proceedings of the 2nd International Conference on Knowledge Capture*, K-CAP '03, 121-128, https://doi.org/10.1145/945645.945664 (Association for Computing Machinery, New York, NY, USA, 2003).
54. Torre, E. *et al*. Asset: analysis of sequences of synchronous events in massively parallel spike trains. *PLoS Comp. Biol.* **12**, e1004939, https://doi.org/10.1371/journal.pcbi.1004939 (2016).
55. Grün, S. Data-Driven Significance Estimation for Precise Spike Correlation. *Journal of Neurophysiology* **101**, 1126–1140, https://doi.org/10.1152/jn.00093.2008 (2009).
56. Köhler, C. & Denker, M. Neuroelectrophysiology Analysis Ontology (NEAO) Use Case. *Zenodo* https://doi.org/10.5281/zenodo.14288030 (2024).
57. Köhler, C. A., Ulianych, D., Grün, S., Decker, S. & Denker, M. Facilitating the sharing of electrophysiology data analysis results through in-depth provenance capture. *eNeuro***11**, ENEURO.0476–23.2024, https://doi.org/10.1523/ENEURO.0476-23.2024 (2024).
58. Harris, S. & Seaborne, A. SPARQL 1.1 Query Language. *W3C Recommendation* https://www.w3.org/TR/sparql11-query/ (2013).
59. Stella, A., Bouss, P., Palm, G. & Grün, S. Comparing surrogates to evaluate precisely timed higher-order spike correlations. *eNeuro* **9**, ENEURO.0505–21.2022, https://doi.org/10.1523/ENEURO.0505-21.2022 (2022).
60. Shinomoto, S. *et al*. Relating neuronal firing patterns to functional differentiation of cerebral cortex. *PLoS Comput Biol* **5**, e1000433, https://doi.org/10.1371/journal.pcbi.1000433 (2009).
61. Holt, G. R., Softky, W. R., Koch, C. & Douglas, R. J. Comparison of discharge variability *in vitro* and *in vivo* in cat visual cortex neurons. *Journal of neurophysiology* **75**, 1806–1814, https://doi.org/10.1152/jn.1996.75.5.1806 (1996).
62. Cox, D. R. & Lewis, P. A. W.*The Statistical Analysis of Series of Events*. Methuen's Monographs on Applied Probability and Statistics (Methuen, London, 1966).
63. Jordan, J. *et al*. Extremely Scalable Spiking Neuronal Network Simulation Code: From Laptops to Exascale Computers. *Frontiers in Neuroinformatics* **12**, 2, https://doi.org/10.3389/fninf.2018.00002 (2018).

64. Torre, E., Picado-Muiño, D., Denker, M., Borgelt, C. & Grün, S. Statistical evaluation of synchronous spike patterns extracted by frequent item set mining. *Frontiers in computational neuroscience* **7**, 132, https://doi.org/10.3389/fncom.2013.00132 (2013).
65. Quaglio, P., Yegenoglu, A., Torre, E., Endres, D. M. & Grün, S. Detection and evaluation of spatio-temporal spike patterns in massively parallel spike train data with spade. *Frontiers in computational neuroscience* **11**, 41, https://doi.org/10.3389/fncom.2017.00041 (2017).
66. Stella, A., Quaglio, P., Torre, E. & Grün, S. 3d-SPADE: Significance evaluation of spatio-temporal patterns of various temporal extents. *Biosystems* **185**, 104022, https://doi.org/10.1016/j.biosystems.2019.104022 (2019).
67. Russo, E. & Durstewitz, D. Cell assemblies at multiple time scales with arbitrary lag constellations. *Elife* **6**, e19428, https://doi.org/10.7554/eLife.19428 (2017).
68. Garijo, D. & Gil, Y. Augmenting PROV with Plans in P-PLAN: Scientific Processes as Linked Data. In Kauppinen, T., Pouchard, L. C. & Keßler, C. (eds.) *Proceedings of the Second International Workshop on Linked Science 2012 - Tackling Big Data* (CEUR Workshop Proceedings, Boston, 2012).
69. Missier, P., Dey, S., Belhajjame, K., Cuevas-Vicenttin, V. & Ludäscher, B. D-PROV: Extending the PROV Provenance Model with Workflow Structure. In *5th USENIX Workshop on the Theory and Practice of Provenance (TaPP 13)* (2013).
70. Bettivia, R., Cheng, Y.-Y. & Gryk, M. R. ProvONE. In Bettivia, R., Cheng, Y.-Y. & Gryk, M. R. (eds.) *Documenting the Future: Navigating Provenance Metadata Standards*, 41–56, https://doi.org/10.1007/978-3-031-18700-1_4 (Springer International Publishing, Cham, 2022).
71. Belhajjame, K. *et al.* Using a suite of ontologies for preserving workflow-centric research objects. *Journal of Web Semantics* **32**, 16–42, https://doi.org/10.1016/j.websem.2015.01.003 (2015).
72. Malone, J. *et al.* The Software Ontology (SWO): A resource for reproducibility in biomedical data analysis, curation and digital preservation. *Journal of Biomedical Semantics* **5**, 25, https://doi.org/10.1186/2041-1480-5-25 (2014).
73. De Meester, B., Dimou, A., Verborgh, R. & Mannens, E. An Ontology to Semantically Declare and Describe Functions. In Sack, H. *et al.* (eds.) *The Semantic Web*, 46–49, https://doi.org/10.1007/978-3-319-47602-5_10 (Springer International Publishing, Cham, 2016).
74. Schneider, S., Lee, J. H. & Mathis, M. W. Learnable latent embeddings for joint behavioural and neural analysis. *Nature* **617**, 360–368, https://doi.org/10.1038/s41586-023-06031-6 (2023).
75. Smith, B. *et al.* The OBO Foundry: Coordinated evolution of ontologies to support biomedical data integration. *Nature Biotechnology* **25**, 1251–1255, https://doi.org/10.1038/nbt1346 (2007).
76. Jackson, R. *et al.* OBO Foundry in 2021: Operationalizing open data principles to evaluate ontologies. *Database* **2021**, baab069, https://doi.org/10.1093/database/baab069 (2021).
77. Arp, R., Smith, B. & Spear, A. D.*Building Ontologies with Basic Formal Ontology* (Massachusetts Institute of Technology, Cambridge, Massachusetts, 2015).
78. Mungall, C. *et al.* Oborel/obo-relations: February 2023 release 2 (Major Pipeline Change). *Zenodo* https://doi.org/10.5281/zenodo.7665156 (2023).
79. Smith, B. *et al.* Relations in biomedical ontologies. *Genome Biology* **6**, R46, https://doi.org/10.1186/gb-2005-6-5-r46 (2005).
80. Larson, S. D. & Martone, M. NeuroLex.org: An online framework for neuroscience knowledge. *Frontiers in Neuroinformatics* **7**, 18, https://doi.org/10.3389/fninf.2013.00018 (2013).
81. Musen, M. A. The protégé project: A look back and a look forward. *AI Matters* **1**, 4–12, https://doi.org/10.1145/2757001.2757003 (2015).
82. Morel, P. Gramm: Grammar of graphics plotting in Matlab. *Journal of Open Source Software* **3**, 568, https://doi.org/10.21105/joss.00568 (2018).
83. Pröpper, R. & Obermayer, K. Spyke Viewer: A flexible and extensible platform for electrophysiological data analysis. *Frontiers in Neuroinformatics* **7**, 26, https://doi.org/10.3389/fninf.2013.00026 (2013).
84. Kreuz, T., Mulansky, M. & Bozanic, N. SPIKY: A graphical user interface for monitoring spike train synchrony. *Journal of Neurophysiology* **113**, 3432–3445, https://doi.org/10.1152/jn.00848.2014 (2015).
85. Brochier, T. *et al.* Massively parallel multi-electrode recordings of macaque motor cortex during an instructed delayed reach-to-grasp task. *G-Node* https://doi.org/10.12751/g-node.f83565 (2017).
86. Brochier, T. *et al.* Massively parallel recordings in macaque motor cortex during an instructed delayed reach-to-grasp task. *Scientific Data* **5**, 180055, https://doi.org/10.1038/sdata.2018.55 (2018).
87. Zehl, L. *et al.* Handling Metadata in a Neurophysiology Laboratory. *Frontiers in Neuroinformatics* **10**, 26, https://doi.org/10.3389/fninf.2016.00026 (2016).
88. Denker, M., Grün, S., Wachtler, T. & Scherberger, H. Reproducibility and efficiency in handling complex neurophysiological data. *Neuroforum* **27**, 27–34, https://doi.org/10.1515/nf-2020-0041 (2021).
89. Grewe, J., Wachtler, T. & Benda, J. A bottom-up approach to data annotation in neurophysiology. *Frontiers in Neuroinformatics* **5**, 16, https://doi.org/10.3389/fninf.2011.00016 (2011).
90. Garcia, S. *et al.* Neo: an object model for handling electrophysiology data in multiple formats. *Frontiers in Neuroinformatics* **8**, 10, https://doi.org/10.3389/fninf.2014.00010 (2014).
91. Stoewer, A., Kellner, C. J., Benda, J., Wachtler, T. & Grewe, J. File format and library for neuroscience data and metadata. In *Front. Neuroinform. Conference Abstract: Neuroinformatics 2014*, https://doi.org/10.3389/conf.fninf.2014.18.00027 (Frontiers, 2014).
92. Denker, M., Köhler, C., Kern, M. & Kleinjohann, A. Elephant 0.14.0. *Zenodo* https://doi.org/10.5281/zenodo.10075775 (2023).
93. Köhler, C. A. Alpaca 0.2.0. *Zenodo* https://doi.org/10.5281/zenodo.10277861 (2023).

Acknowledgements

This work was performed as part of the Helmholtz School for Data Science in Life, Earth and Energy (HDS-LEE) and received funding from the Helmholtz Association of German Research Centres. This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Specific Grant Agreement No. 945539 (Human Brain Project SGA3), the European Union's Horizon Europe Programme under the Specific Grant Agreement No. 101147319 (EBRAINS 2.0 Project), the Ministry of Culture and Science of the State of North Rhine-Westphalia, Germany (NRW-network "iBehave", grant number: NW21-049), and the Joint Lab "Supercomputing and Modeling for the Human Brain." Open access publication funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 491111487.

Author contributions

C.A.K., S.G., and M.D. conceived the study. C.A.K. and M.D. designed the ontology model. C.A.K. implemented the ontology, the example analyses, the knowledge graph, and queries. C.A.K. created figures and tables and C.A.K. and M.D. wrote the initial draft of the paper. C.A.K., S.G., and M.D. reviewed and edited the manuscript. M.D. and S.G. supervised the study. All authors read and approved the final version of the manuscript.

Funding
Open Access funding enabled and organized by Projekt DEAL.

Competing interests
The authors declare no competing interests.

Additional information
Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41597-025-05213-3.

Correspondence and requests for materials should be addressed to C.A.K.