FiNN: A toolbox for neurophysiological network analysis

Maximilian Scherer[a,b], Tianlu Wang[a], Robert Guggenberger[a], Luka Milosevic[a,b], Alireza Gharabaghi[a§]

[a]Institute for Neuromodulation and Neurotechnology, University Hospital and University of Tübingen, Tübingen, Germany

[b]Krembil Brain Institute, University Health Network, and Institute of Biomedical Engineering, University of Toronto, Toronto, Canada

[§]Corresponding author: Professor Alireza Gharabaghi. Address: Institute for Neuromodulation and Neurotechnology, University Hospital Tübingen, Otfried-Müller-Str. 45, 72076 Tübingen, Germany. Email: alireza.gharabaghi@uni-tuebingen.de

Keywords: connectivity, cross-frequency coupling, neural oscillations, phase-amplitude coupling

Abstract

Recently, neuroscience has seen a shift from localist approaches to network-wide investigations of brain function. Neurophysiological signals across different spatial and temporal scales provide insight into neural communication. However, additional methodological considerations arise when investigating network-wide brain dynamics rather than local effects. Specifically, larger amounts of data, investigated across a higher dimensional space, are necessary.

Here, we present *FiNN* (*Fi*nd *N*europhysiological *N*etworks), a novel toolbox for the analysis of neurophysiological data with a focus on functional and effective connectivity. FiNN provides a wide range of data processing methods, and statistical and visualization tools to facilitate inspection of connectivity estimates and the resulting metrics of brain dynamics. The Python toolbox (https://github.com/neurophysiological-analysis/FiNN) and its documentation (https://neurophysiological-analysis.github.io/FiNN/) are freely available.

We evaluated FiNN against a number of established frameworks on both a conceptual and an implementation level. We found FiNN to require much less processing time and memory than other toolboxes. In addition, FiNN adheres to a design philosophy of easy access and modifiability, while providing efficient data processing implementations. Since the investigation of network-level neural dynamics is experiencing increasing interest, we place FiNN at the disposal of the neuroscientific community as open-source software.

Abbreviations: cfc, cross-frequency coupling; ECG, electrocardiography; EEG, electroencephalography; EMG, electromyography; FIR, finite impulse response; LFP, local field potential; MEG, magnetoencephalography; PAC, phase-amplitude coupling; sfc, same-frequency coupling.

1. Introduction

Analyzing dependence between neurophysiological signals, and the definition of large-scale networks, has become an important field of research that greatly enhances our comprehension of communication between distinct neural structures (Bressler & Menon, 2010; Siegel et al., 2012). Neural connectivity in particular is commonly quantified by estimating the degree to which neural oscillations within the same frequency band or across different frequency bands relate to each other (Fries, 2005). These two types of communication modes are known as same-frequency coupling and cross-frequency coupling, respectively (Friston, 2011; Hyafil et al., 2015).

Neural communication on a network level can be quantified on the basis of neurophysiological data from a wide variety of data sources including electroencephalography (EEG), magnetoencephalography (MEG), and local field potentials (LFPs) (Engel et al., 2013; Ganzetti & Mantini, 2013). An estimation of connectivity across regions and/or frequencies is generally more computationally intensive than the local synchronization of neural activity within a smaller spatial region (He et al., 2019). While the number of power estimates are linearly related to the number of sensors the number of potential connectivity candidates increases in a quadratic order of magnitude. Furthermore, in many applications, the amount of neurophysiological data is rising rapidly, partly due to increasingly dense sensor setups and high sampling rates (Brinkmann et al., 2009; Sahasrabuddhe et al., 2020; Song et al., 2015), as well as to more demanding analysis techniques, such as machine learning. A greater number of samples is therefore required (Glaser et al., 2019; Kus et al., 2004).

In recent years, a number of toolboxes include functions for estimating neural connectivity. However, the majority are either heavyweight frameworks that deeply encapsulate data, making modifications or the integration into an existing pipeline difficult and increasing the time required for memory processing; or frameworks with broad usability, but limited functionality for neuroscientific data analysis and interpretation (see Section 3). Here, we introduce FiNN (*Fi*nd *N*europhysiological *N*etworks), a novel framework written in Python that provides tools to analyze neurophysiological data in a bid to quantify network-wide neural communication within a lightweight and computationally efficient framework. FiNN includes several functions for cleaning and processing neurophysiological data in the context of connectivity. It also includes visualization routines and statistical methods, both of which are useful tools for the analysis of connectivity in large, high-dimensional neurophysiological data sets.

The goal of FiNN is to provide an open-source software toolbox offering easy-to-use and computationally efficient methods for both users and developers. From a user perspective, it is important that the toolbox is accessible and manageable. We therefore designed the functions in FiNN such that they can be readily used without a deep knowledge of the underlying functionality. In addition to an elaborate documentation on the individual functions, we included detailed information on the internal processing functions to promote modifiability. Furthermore, to facilitate the analysis of large datasets across high dimensional spaces, memory and CPU consumption were strictly monitored and reduced, thereby achieving a high level of scalability.

Section 2 describes the functionality of the toolbox, while Section 3 discusses FiNN in relation to the established toolboxes generally used to analyze neurophysiological data. This is followed by an illustration of its performance in comparison to a selection of established toolboxes in Sections 4 and 5.

2. Materials & Methods

2.1. Toolbox documentation and installation

FiNN (v1.0) is freely available to the research community as open-source code on GitHub (https://github.com/neurophysiological-analysis/FiNN). It can be downloaded as a zip file containing the last release, or by cloning the git repository. Documentation is available at https://neurophysiological-analysis.github.io/FiNN/, and includes a detailed description of the functions implemented. Furthermore, FiNN contains a demo folder with several trial scripts. The scripts are intended to provide an introductory demonstration of the functions implemented, but can also be used to gain a deeper methodological understanding of the functions. An exemplary workflow utilizing FiNN for the analysis of EEG data is provided in Table 1. A real-world application of FiNN can be observed in (Scherer, Steiner, et al., 2022) where FiNN was used to quantify phase-amplitude coupling between neuronal spiking and LFP activity. In the scope of this work, FiNN was used for data processing & statistical analysis.

The FiNN toolbox was developed with Python version 3.8 (Van Rossum & Drake Jr, 1995) and requires the following Python packages: numpy (Harris et al., 2020), scipy (Virtanen et al., 2020), PyQt5 , functools, lmfit (Newville et al., 2016), matplotlib (Hunter, 2007), rpy2 . The MNE package (Gramfort et al., 2013) is required if the user wishes to load BrainProducts data (BrainProducts GmbH, Gilching, Germany). The following R (R Core Team, 2021) libraries are required if the user wishes to perform statistical evaluation: Matrix (Bates & Maechler, 2021), lme4 (Bates et al., 2015, p. 4), carData (Fox et al., 2020), and car (Fox & Weisberg, 2019).

Processing step	Module	Function
1. Preparation	file_io	load_brain_vision_data.py or data_manager.py
- Load Data		
2. Pre-processing		
- Artifact rejection	filters	frequency.py
- Bad channel rejection	cleansing	bad_channel_identification.py or channel_restoration
3. Feature extraction		
- Calculate connectivity	sfc	directional_absolute_coherency.py
- Identify faulty samples	cleansing	outlier_removal.py
4. Feature evaluation		
- Statistical analysis	statistics	generalized_linear_model.py
5. Visualization		
- Visualize results	visualization	topoplot.py

Table 1. Illustrative example of a pipeline for processing raw neurophysiological data.

2.2. Organisation of the FiNN Toolbox

FiNN (v1.0) consists of nine modules: *basic processing, artifact rejection, filters, cross-frequency coupling, same-frequency coupling, statistics, visualization, file IO,* and

miscellaneous. A brief description of each module is provided below. Further details can be found in the online documentation (https://neurophysiological-analysis.github.io/FiNN/). Recommendations are also provided as to how to apply these for analysis, where applicable.

2.2.1. Basic processing package

For initial processing, the basic processing package offers the *common average re-referencing (CAR)* and *downsampling* functions. This function subtracts from the data the part that is shared by all channels, since it is presumably the result of activity at the reference electrode, and hence equal in all channels. This procedure is recommended only when a large number of EEG channels (≥ 64) is available and these are equally distributed across the head (Nunez, 2010). The *downsampling* function downsamples a signal from its original sampling frequency to a lower, configurable target sampling frequency. Prior to the application of the *downsampling* function, it is important for the signal to be lowpass filtered below half the target frequency, as aliasing artifacts may otherwise be introduced into the downsampled signal (Shannon, 1948, 1949). Furthermore, we recommend that the target sampling frequency be set as low as possible while maintaining a sufficient level of temporal accuracy to accelerate data evaluation further down the line.

2.2.2. Artifact rejection package

The artifact rejection package contains two functions to detect artifacts, and one function to clean the data. *Bad channel identification* identifies individual channels in which the power in a predefined frequency range deviates by more than three standard deviations (default value; configurable) from the mean power of all given channels as faulty channels, since spectral power is a useful feature for separating artifacts from EEG (Islam et al., 2016). In an *optional* second

step, the function provides a custom-built dialogue window that shows the mean power of each channel (Figure 1A). Channels marked as faulty are automatically highlighted, and the user can visually confirm the selection and make changes accordingly. The outputs of the *bad channel identification* function are a list of non-faulty channels, and a list of faulty channels. Furthermore, it is highly recommended that frequency bands that are part of the subsequent analysis for artifact detection be avoided. This advice should be followed to ensure that the results are not biased, as side effects may arise if the power in the frequency band of interest is used in the artifact rejection.

Once the faulty channels have been identified, an optional follow-up step is to restore them using the *channel restoration* function. This restores faulty channels by averaging the signals from their respective neighbors (Figure 1B, 1C). A default adjacency matrix is provided to the *channel restoration* function. In the event of one or more neighboring channels of a faulty channel being faulty themselves, the channels are iteratively restored, commencing at the channel with the most non-faulty neighbors. Once a channel has been restored during an iteration, it is considered a non-faulty channel during the next iteration of the reconstruction process.

The *outlier removal* function removes any sample within a data set with a z-score higher than two (default value, configurable). This process is repeated iteratively until only those samples with a z-score of less than two remain or until a minimum sample threshold is reached. This function should be applied only to data from a unique, single state (e.g., within the same subject and same condition). It assumes that provided data is from a single process with a Gaussian distribution. Any data segments which fail this assertion are iteratively removed from the provided data. In the event that this assumption cannot be met, the approach presented here may

not apply. The assumption of Gaussianity may be evaluated by either visual inspection or tests for normality.

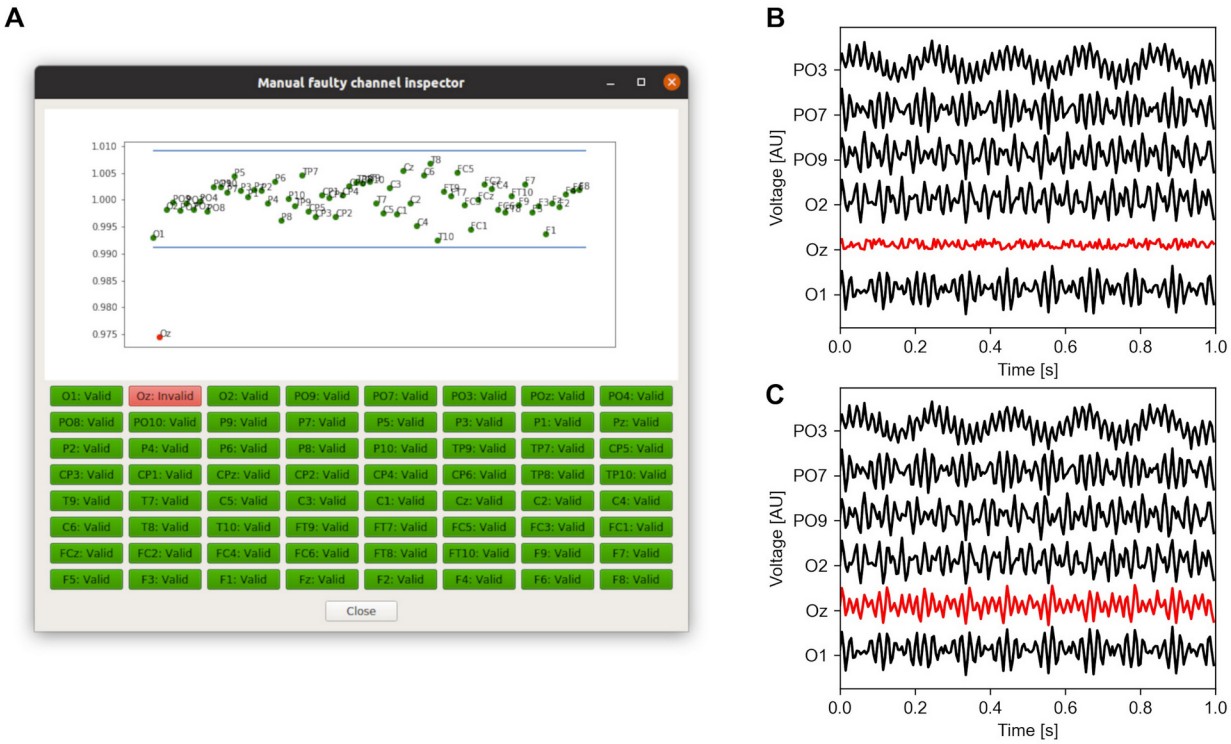

Figure 1. Practical application of the bad channel detection and channel restoration functions. (**A**) Bad channel detection. This figure shows the average power in a predefined frequency band for all included channels. A channel is flagged as bad if the individual power-based z-score is more than 3 standard deviations from the mean. Channels can be manually classified as normal or faulty either by clicking the corresponding dot in the plot, or by pressing the corresponding button. (**B**) Time-series traces at a sub-selection of the EEG electrodes shown in (A) before channel restoration. (**C**) Time-series traces at the same sub-selection of EEG electrodes shown in (B) after channel restoration.

2.2.3. Filters package

The main function in this module is the implemented *finite impulse response (FIR)* filter. The filter is implemented via an overlap add approach (Rabiner & Gold, 1975) to speed up the processing procedure, especially for longer signals. Furthermore, the filter implementation provides a rapid and precise operation mode which initially converts the input data into either 32 bits floats (fast) or 64 bits floats (precise), and subsequently performs all operations with the required precision. The FIR filter can be configured as either a low-pass, high-pass, band-pass or band-stop filter. Furthermore, custom filters can be easily designed and accessed using the functions listed in the main FIR function. Additionally, a wraparound scipy's Butterworth filter is also available.

2.2.4. Cross-frequency coupling (cfc) package

The cross-frequency coupling package currently implements the following phase amplitude coupling (PAC) metrics: direct modulation index (Scherer, Wang, et al., 2022a), modulation index (Tort et al., 2008), phase-locking value (Mormann et al., 2005), and mean vector length (Canolty et al., 2006). A description of the metrics can be found in Table 2. The input signals should already be filtered into the frequency bands of interest, e.g., with the FIR filter implemented in the filters package (Section 2.2.3).

2.2.5. Same-frequency coupling (sfc) package

The same-frequency coupling package currently implements the following metrics: directionalised absolute coherency (Scherer, Wang, et al., 2022b), magnitude squared coherence (Carter et al., 1973), imaginary coherence (Nolte et al., 2004), weighted phase lag index (Vinck et al., 2011), and phase slope index (Nolte et al., 2008). A description of the metrics can be found

in Table 2. It also includes a function for calculating the complex coherency, which can be interpreted as a measure of consistency between two signals with a constant phase shift, at a specific frequency (Shaw, 1984). Complex coherency is implemented as an additional function since it is a common precursor of the magnitude squared coherence, imaginary coherence, and others. The metrics implemented in the sfc module accept input data from the time domain, the frequency domain, and the complex coherency domain. Apart from the most commonly used time domain signals, the additional input data formats (frequency domain and complex coherence domain) were added to support modifiability of the implemented functionality, and to allow more efficient processing of multiple connectivity metrics in parallel when the data is already available in the required format.

Name	Description
Cross-frequency coupling	
Direct modulation index (Scherer et al., 2022a)	The direct modulation index metric is a variant of the modulation index from *Tort et al., 2008* and evaluates phase amplitude coupling (PAC) between two signals. Instead of quantifying PAC using entropy, a sinusoidal slope is fitted to the phase-amplitude histogram. This metric is statically bounded to the interval [0, 1] which allows for the interpretation of absolute PAC changes.
Modulation index (Tort et al., 2008)	The modulation index quantifies PAC as the Shannon entropy (Shannon, 1948) of the phase-amplitude histogram between two

	signals. This metric is not bound to a static interval, and therefore allows only for interpretation of the resulting values in relative changes.
Phase-locking value (Mormann et al., 2005)	The phase-locking value evaluates the average phase lag between the phase of the instantaneous amplitude of a first signal and the phase of a second signal. This metric is not bound to a static interval, and therefore allows only for interpretation of the resulting values in relative changes.
Mean vector length (Canolty et al., 2006)	The mean vector length metric evaluates whether the amplitude of one signal peaks at a specific phase in a second signal.
Same-frequency coupling	
Directional absolute coherency (Scherer et al., 2022b)	The directional absolute coherency utilizes the complex coherency function to calculate the magnitude squared coherence (Carter et al., 1973), phase slope index (Nolte et al., 2008), and the imaginary coherence (Nolte et al., 2004). These three metrics are combined to create a single reliable measure of coherence drawing from their individual strengths without incorporating the individual weaknesses. This connectivity metric is directional, can detect volume conduction, and is statically bound to [-1, 1].
Magnitude squared coherence	Using the complex coherency function, magnitude squared coherence is calculated (Carter et al., 1973) between a source

(Carter et al., 1973)	signal and a target signal. To derive a rational number from the complex output of the complex coherence, the squared absolute of the complex valued complex coherence is calculated. This connectivity metric is directionless, cannot detect volume conduction, and is statically bounded to [0, 1].
Imaginary coherence (Nolte et al., 2004)	The imaginary coherence is calculated using the complex coherency function by taking the imaginary component of the complex coherence. This connectivity metric is directionless, may detect volume conduction, and its output is not statically bound. The latter renders this metric susceptible to bias if the sensors have been moved (e.g., on different measurement days).
Weighted phase-lag index (Vinck et al., 2011)	The weighted phase lag index is an extension of the phase lag index (Stam et al., 2007). This function is directionless, can detect volume conduction (approaches zero in vicinity of 0°/180° phase shift), and is statically bound to [-1, 1].
Phase slope index (Nolte et al., 2008)	The phase slope index is calculated using the complex coherency function, and is normalized by its standard deviation. Due to the additional need for normalization, more data is required than for the other connectivity estimation methods. This metric is directional, can detect volume conduction (approaches zero in vicinity), and is bound to [-1, 1].

Table 2. Overview of the cross-frequency and same-frequency coupling metrics implemented in the FiNN Toolbox.

2.2.6. Statistics package

The statistics package contains the *generalized linear mixed models* function, which allows to employ generalized linear mixed models in the statistical evaluation of an investigation. The rpy2 package is used to wrap around the lme4 package (Bates et al., 2015) in *R (R Core Team, 2021)*. The implementation presented in FiNN provides a complete and easily interpretable output which comprises both the significance values, indicating how reliable an effect is, and the coefficients, allowing for an estimation of how impactful an observed effect is.

2.2.7. Visualization package

FiNN offers the *topoplot* function, with the additional functionality of visualizing various levels of significance. Depending on their statistical significance, individual channels may be marked both before and/or after multiple comparison correction (Figure 2). The *topoplot* function is built on top of Matplotlib (Hunter, 2007), which is a data visualization library in Python. Since this function is solely responsible for visualization, any feature calculation and/or statistical evaluation has to be performed independently.

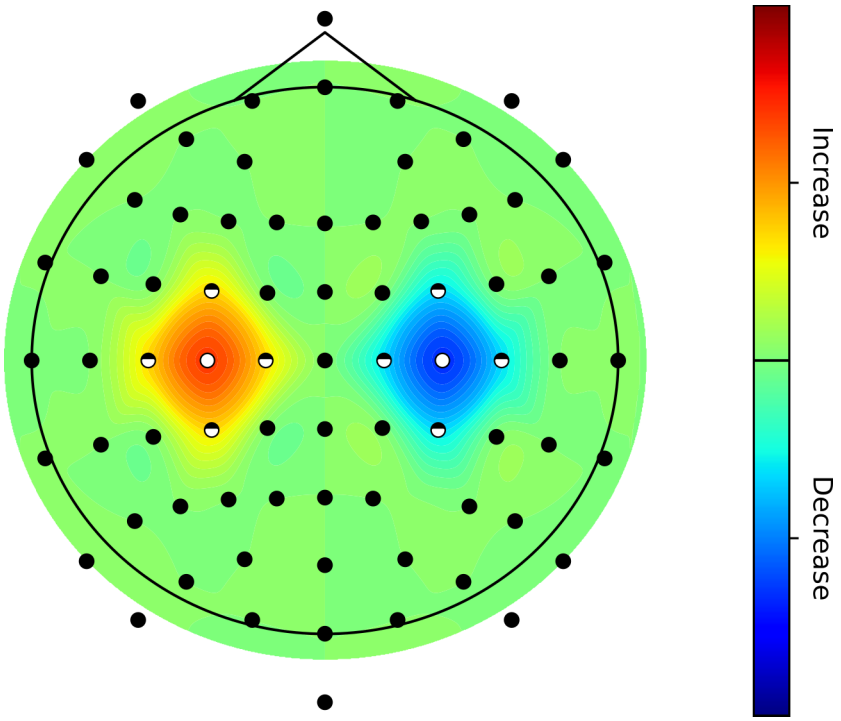

Figure 2. Topography plot of artificial demo data. This figure shows an example of a significantly increased activity above the left motor cortex and significantly decreased activity above the right motor cortex. At the center, the effects were modeled to be significant after multiple comparison correction. This is marked at the electrodes position using full white dots. The outlying areas of these effects were simulated to be also significant, but only prior to multiple comparison correction, this is indicated by half-filled dots at the electrodes location. Finally, non-significant areas are indicated by black dots.

2.2.8. File IO package

When investigating network-wide brain dynamics, larger amounts of data have to be accommodated. FiNN offers the *data manager* module to handle large file sizes. In Python, pickle (.pkl) is a well-known tool for saving arbitrary variables to disk. However, when loading or writing a file, it has the well-known shortcoming of consuming multiple times more memory

than the size of the file itself. This causes memory spikes which may in turn lead to unstable behavior and crashes if not handled carefully. The *data manager* function circumvents this issue by processing a large file into several smaller ones. This increases the stability of the overall processing pipeline and reduces the likelihood of requiring user intervention, particularly if the pipeline is partially or fully automated.

2.2.9. Miscellaneous package

Due to the large size of neuroscientific data sets, it is beneficial to separate the processing into subprocesses and perform operations in parallel. When using the Python native subprocess pool, however, all subprocesses may receive or send their data at the same time. This behavior is similar to issues of pickle-based file reading and writing (see previous section). The FiNN Toolbox therefore offers the *timed pool* function, a custom-built subprocess pool that limits the sending or receiving of data to one subprocess at a time. This implementation substantially decreases the risk of memory spikes for a negligible increase in run time. Additionally, the *timed pool* function has two advantages over the Python native subprocess pool. First, there is an option to delete the original input data immediately after its function is executed, thus releasing additional memory. Second, there is an option to add a configurable life-time duration to each subprocess. When a subprocess does not return a result within the allotted life-time duration, it will be restarted. This behavior is particularly suitable in the event that one or more of the subprocesses do not terminate within a reasonable time, e.g., when running a randomly initialized optimization loop.

2.3. Code testing and validation

System tests have been successfully applied to all the functions implemented in FiNN. Functional tests were performed for mathematically more complex components such as the FIR filter and the complex coherence calculation using scipy as a reference implementation. Automated unit tests were implemented in the framework to verify that its behaviour is as designated.

2.4. Performance evaluation

We evaluated the performance of the FIR filter against the implementation from MNE (a Python based framework for the analysis of electrophysiological data, fully defined in section 3.1) in terms of speed, and the performance of the subprocess pool against both the implementation from *joblib* used in MNE and the default subprocess pool implemented in the multiprocessing package in Python, in terms of RAM consumption. These two functions were selected as they necessitate efficient implementations (both RAM and CPU time-wise) if evaluation is to be rapid. Biological data recorded from the human subthalamic nucleus was used for the evaluation process (Milosevic et al., 2020). During the three subprocesses, signals of different lengths, varying between 30 seconds and 24 hours (artificially expanded), were filtered using both FIR implementations. The data was appended via repetition as required. All parameters were configured equally to achieve a maximum degree of comparability between the two implementations. The scripts of the above evaluations and comparisons are provided in the toolbox (https://github.com/neurophysiological-analysis/FiNN).

3. Results

3.1. Comparison to other frameworks

A large number of open-source toolboxes have been developed to support the processing and analysis of (neurophysiological) signals. Here, we compare FiNN to a selection of existing toolboxes in terms of scope and computational performance (i.e., processing time and memory consumption). These other toolboxes were selected either on account of their reputation in scientific data analysis (e.g., scipy) or as a result of key-word-based searches (e.g., "Python toolbox EEG"). We selected the search engine *startpage.com*, as it delivers Google results while anonymizing search requests, and is therefore not subject to a user-specific filter bubble (Salehi et al., 2018). In the following paragraphs, the selection of frameworks with which the FiNN framework is compared is described in more detail.

Two Python-based frameworks that are widely used in general data processing and analysis are *scipy (scientific python) (Virtanen et al., 2020)* and *scikit-learn (Pedregosa et al., 2011)*. Scipy and scikit-learn are focused on fundamental algorithms (e.g. optimization techniques, basics in digital signal processing, and linear algebra), and machine learning/data analysis, respectively. Currently, scikit-learn provides functionality such as the principal component analysis (Hotelling, 1933; Pearson, 1901) and the independent component analysis (Makeig et al., 1995), which are commonly used for the purpose of artifact identification and removal (Xue et al., 2006).

A major advantage of frameworks such as scipy and scikit-learn is that they are usually heavily modified towards a small memory footprint and limited CPU consumption. Scikit-learn in particular has been optimized for speed, which becomes increasingly important as the

complexity of the applied machine learning algorithm increases. Relative to the other frameworks discussed in this paper, scipy may be categorized as a low-level framework. Generally, scipy and scikit-learn offer a great range of functions which can be – and often have to be – highly customized to a problem at hand. This in turn requires in depth knowledge of the provided methodology in the respective areas. Furthermore, while scipy and scikit-learn excel with regards to the provided functionality (fundamental algorithms and machine-learning/data analysis), on their own, these frameworks do not fully cover all needs of electrophysiological data analysis. For example, scipy/scikit-learn lacks some data analysis functions elementary in many experimental, electrophysiology based neuroscientific analyses, such as connectivity metrics, and proper visualization functions, such as topoplots. Having a different focus, FiNN uses the comparatively low-level functionality provided by scipy (and other toolkits) to provide this functionality for EEG/MEG data analysis.

Another broad Python-based framework specifically developed for the analysis and visualization of neurophysiological data, is *MNE (Gramfort et al., 2013)*. MNE has its roots in the estimation of source-space signals from signal-space signals via EEG or MEG recordings. Unlike scipy and scikit-learn, MNE is a very high-level framework that offers a wide range of functions to analyze neurophysiological data. These methods are often configured in predefined ways, making any deviations from the intended application case rather difficult. MNE focuses strongly on mathematical precision. This, in turn, mandates a high memory consumption and slow data processing speed. MNE follows a fundamentally different design philosophy to FiNN. While FiNN aims to provide methods which can be easily integrated into any workflow, MNE is almost exclusively used if the proposed MNE-specific pipelines are implemented for data processing. This results in a lower degree of customizability when using MNE rather than FiNN. Another

difference is the focus of optimization. MNE focuses on optimal mathematical accuracy, whereas FiNN allows the user to define the trade-off between high mathematical accuracy, high processing speed and efficient resource usage, e.g., in applications where online analysis is necessary.

NeuroKit2 (Makowski, 2016) is another Python-based framework for the analysis of neuroscientific and neurophysiological data. The framework was designed to work with electrocardiography (ECG), electromyography (EMG), and EEG data. It provides an adequate range of functionality with a different focus to FiNN. While NeuroKit2 aims to be a generalist for any kind of neurophysiological signals, FiNN focuses on EEG/EMG/MEG and LFP signals. This difference in focus can be readily observed in the functionality provided. For instance, NeuroKit2 offers methods to analyze heart rate variability via ECG, or the autocorrelation of EEG signals, while FiNN offers metrics for connectivity between EEG channels, or functionality to visualize a topoplot. Neither connectivity nor topoplot functionality are offered in NeuroKit2.

Finally, there is a wide range of Python-based frameworks with limited functionality or rather rigid data processing pipelines. These frameworks include NeuroPycon (Meunier et al., 2020), Plotly (Plotly Technologies Inc., 2015), matplotlib (Hunter, 2007), HEAR (Kobler et al., 2019), Pygpc (Weise et al., 2020), Human Neocortical Neurosolver (Neymotin et al., 2020), Neo (Marcus et al., 2019), nipype (Gorgolewski et al., 2011), ScoT (Billinger et al., 2014), PyEEG (Bao et al., 2011), Gumpy (Tayeb et al., 2018). Due to a too limited overlap with the focus of FiNN, these frameworks were not compared to FiNN.

3.2. Performance comparison between FiNN and MNE

The FIR filter and the multiprocessing pool of FiNN were compared to the same functions when implemented in MNE. The processing times of the FIR filter as implemented in FiNN and MNE are shown in Figure 3. Both configurations in FiNN were substantially faster than their counterparts in MNE, provided that continuous data sets were of 1-hour length or less. When the fast implementation was chosen in FiNN, this implementation of the FIR filter was always executed more quickly than its MNE counterpart. While the difference was up to 670% more rapid for small data sets, the implementation of FiNN remained approximately 10% faster for very long continuous data sets (24 hours) (Figure 3). Performance evaluations were executed using Python v3.9 on a desktop PC (CPU: AMD Ryzen 9 5950x, RAM: 64 GB (2x32 GB DDR4), Mainboard: B550 AORUS ELITE AX V2).

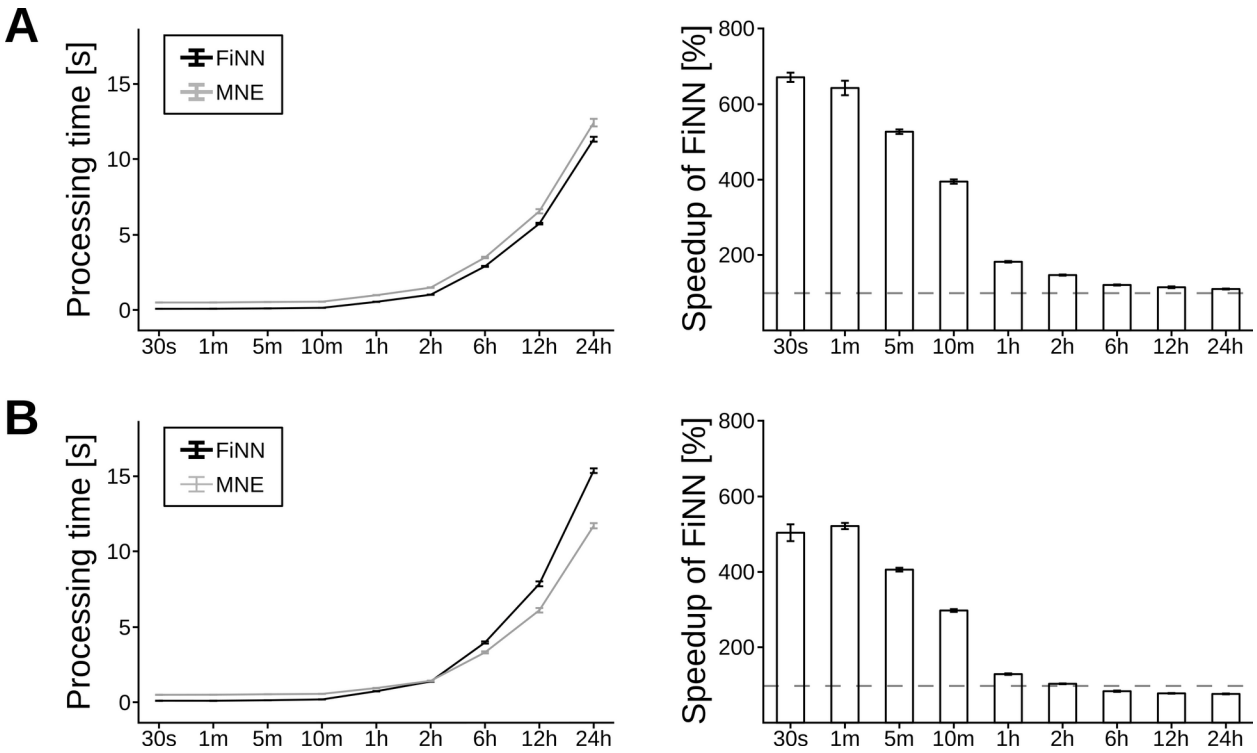

Figure 3. Processing time of the FIR filter implemented in FiNN and MNE. The rows show the processing time (mean and standard deviation) on signals with varying durations in the (**A**) fast mode and (**B**) precise mode. The left column shows the runtime in microseconds. The right column shows the percentage increase in speed of the FIR filter implemented in FiNN relative to the implementation in MNE (mean and standard deviation).

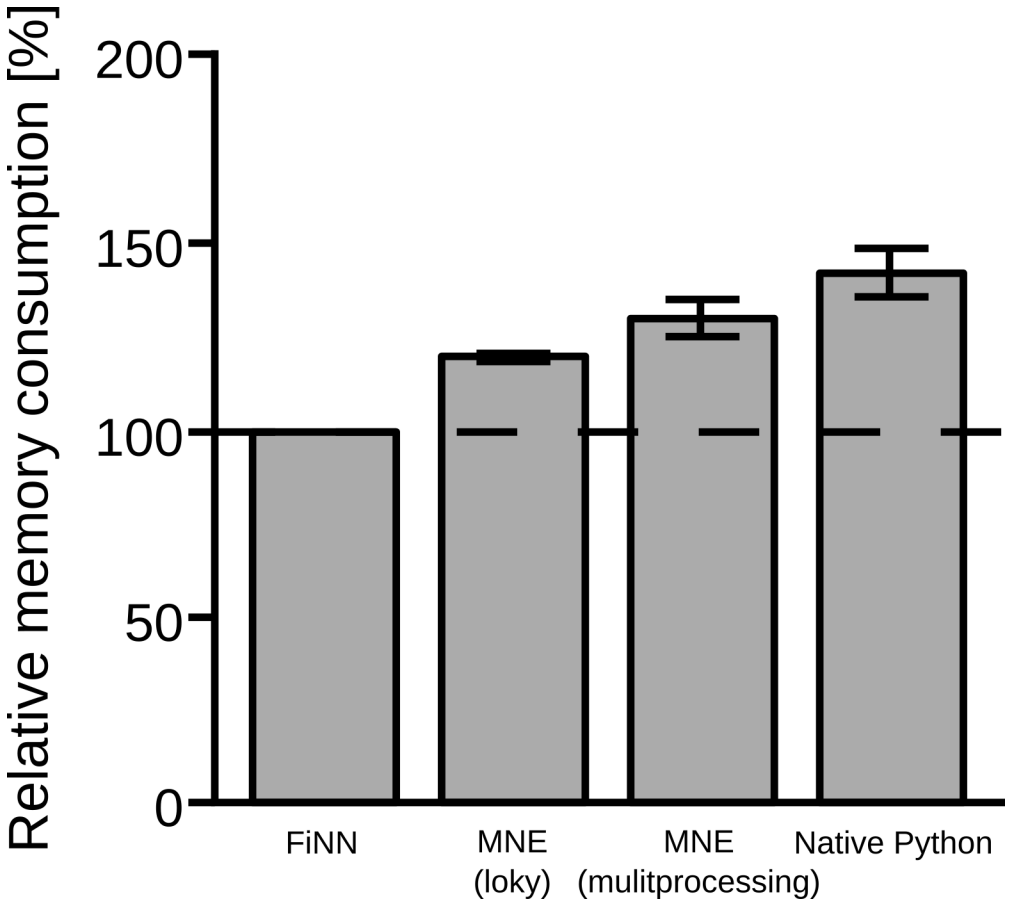

Figure 4. Random access memory of the subprocess pool as implemented in FiNN, MNE, and the native multiprocessing package in Python. For comparison, the relative memory consumption of each pool is shown as a percentage of the memory consumption of the multiprocessing pool implemented in FiNN.

As shown in Figure 4, the multiprocessing pool implemented in FiNN requires less RAM compared to the default multiprocessing package included in the native Python multiprocessing package and MNE with the multiprocessing package backend.

4. Discussion

Neuronal information processing takes place on a local level and on a network level (Reimers, 2011; Thorpe, 1989; Zhang et al., 2016). Therefore, to understand how information is processed, it is also crucial to investigate the non-local components of information processing. To this end, we present FiNN, a Python based framework used to *Fi*nd *N*europhysiological *N*etworks in neurophysiological data and subsequently analyze them. FiNN implements both established and novel metrics for the evaluation of the same frequency coupling (Fries, 2005) and cross frequency coupling (Canolty & Knight, 2010) analyses used to investigate network level information flows. In particular, FiNN offers implementations for the newly proposed connectivity metrics directional absolute coherence (Scherer, Wang, et al., 2022b) and direct modulation index (Scherer, Wang, et al., 2022a).

The amount of data collection in experimental neuroscientific applications is steadily rising with increasingly powerful frequency amplifiers and an ever-growing number of simultaneously recorded channels (Song et al., 2015). Concurrently, the number of features extracted from this raw data also increases (Vaid et al., 2015). For these reasons, efficient data processing is essential. One major benefit of FiNN is its strong optimization towards processing speed and minimal memory consumption. This is reflected not only in the individual functions, which have been optimized to perform as little recalculation as possible, but also in the modules provided.

Exemplary modules for this design philosophy are the custom subprocess pool for parallel processing or the data manager for I/O operations, both of which are included in this framework.

Despite the fact that FiNN includes many elements to assist potential users in the analysis of neurophysiological data, modifiability was still a major concern during development. Although the functionality available offers many parameters to calibrate it to a specific application case, edge cases are difficult to identify. The high level of modifiability provided by FiNN should be most helpful in these cases. Since all functionality is implemented in open-source languages, the programming code can be easily amended to cover specific cases encountered in a user's data analysis. Furthermore, not only the uppermost, but any layer of functionality was fully documented to increase support for potential customization efforts.

Although the main focus of FiNN is on the analysis of network level communication (both same-frequency and cross-frequency) from neurophysiological data, it also provides a full evaluation pipeline for the investigation of local and network level information processing from EEG, MEG, EMG, and/or LFP based data. The pipeline offered by FiNN includes modules for data pre-processing such as semi/fully-automated outlier detection, postprocessing and visualization, and structural functionality to ease parallel processing. Although FiNN was originally developed with EEG and EMG data in mind, the implemented methods are well suited to analyze any kind of neurophysiological signals (e.g., LFP and MEG).

4.1. Outlook

Initially, FiNN was developed for in-house evaluation of experimental paradigms generating neurophysiological data. As the number of paradigms and their subsequent analyses increased, so too did the functionality of FiNN. Meanwhile, FiNN has been developed to the extent where it

not only supports in-house evaluation of neuroscientific investigation, but also external ones. We are therefore pleased to share FiNN as an open-source software with the neuroscientific community.

Credit authorship contribution statement

Maximilian Scherer: Conceptualization, Methodology, Software, Writing – original draft, Writing – review & editing.

Tianlu Wang: Writing – original draft, Writing – review & editing.

Robert Guggenberger: Conceptualization, Methodology, Writing – review & editing.

Luka Milosevic: Conceptualization, Methodology, Writing – review & editing.

Alireza Gharabaghi: Conceptualization, Writing – review & editing, Funding acquisition.

Acknowledgements

This work was supported by the German Federal Ministry of Education and Research [BMBF]. We acknowledge support by the Open Access Publishing Fund of the University of Tübingen.

Declarations of competing interests

The authors declare no conflict of interest.

Code and data availability

The data and code supporting the findings of this study are available on https://github.com/neurophysiological-analysis/FiNN.

References

Bao, F. S., Liu, X., & Zhang, C. (2011). PyEEG: An Open Source Python Module for EEG/MEG Feature Extraction. *Computational Intelligence and Neuroscience*, *2011*, 406391. https://doi.org/10.1155/2011/406391

Bates, D., Mächler, M., Bolker, B., & Walker, S. (2015). Fitting Linear Mixed-Effects Models Using lme4. *Journal of Statistical Software*, *67*(1), 1–48. https://doi.org/10.18637/jss.v067.i01

Bates, D., & Maechler, M. (2021). *Matrix: Sparse and Dense Matrix Classes and Methods*. https://CRAN.R-project.org/package=Matrix

Billinger, M., Brunner, C., & Müller-Putz, G. R. (2014). SCoT: A Python toolbox for EEG source connectivity. *Frontiers in Neuroinformatics*, *8*. https://doi.org/10.3389/fninf.2014.00022

Bressler, S. L., & Menon, V. (2010). Large-scale brain networks in cognition: Emerging methods and principles. *Trends in Cognitive Sciences*, *14*(6), 277–290. https://doi.org/10.1016/j.tics.2010.04.004

Brinkmann, B. H., Bower, M. R., Stengel, K. A., Worrell, G. A., & Stead, M. (2009). Large-scale electrophysiology: Acquisition, compression, encryption, and storage of big data. *Journal of Neuroscience Methods*, *180*(1), 185–192. https://doi.org/10.1016/j.jneumeth.2009.03.022

Canolty, R. T., Edwards, E., Dalal, S. S., Soltani, M., Nagarajan, S. S., Kirsch, H. E., Berger, M. S., Barbaro, N. M., & Knight, R. T. (2006). High Gamma Power Is Phase-Locked to Theta Oscillations in Human Neocortex. *Science*, *313*(5793), 1626–1628. https://doi.org/10.1126/science.1128115

Canolty, R. T., & Knight, R. T. (2010). The functional role of cross-frequency coupling. *Trends in Cognitive Sciences*, *14*(11), 506–515. https://doi.org/10.1016/j.tics.2010.09.001

Carter, G., Knapp, C., & Nuttall, A. (1973). Estimation of the magnitude-squared coherence function via overlapped fast Fourier transform processing. *IEEE Transactions on Audio and Electroacoustics*, *21*(4), 337–344. https://doi.org/10.1109/TAU.1973.1162496

Engel, A. K., Gerloff, C., Hilgetag, C. C., & Nolte, G. (2013). Intrinsic Coupling Modes: Multiscale Interactions in Ongoing Brain Activity. *Neuron*, *80*(4), 867–886. https://doi.org/10.1016/j.neuron.2013.09.038

Fox, J., & Weisberg, S. (2019). *An R Companion to Applied Regression* (Third). Sage. https://socialsciences.mcmaster.ca/jfox/Books/Companion/

Fox, J., Weisberg, S., & Price, B. (2020). *carData: Companion to Applied Regression Data Sets*. https://CRAN.R-project.org/package=carData

Fries, P. (2005). A mechanism for cognitive dynamics: Neuronal communication through neuronal coherence. *Trends in Cognitive Sciences*, *9*(10), 474–480. https://doi.org/10.1016/j.tics.2005.08.011

Friston, K. J. (2011). Functional and Effective Connectivity: A Review. *Brain Connectivity*, *1*(1), 13–36. https://doi.org/10.1089/brain.2011.0008

Ganzetti, M., & Mantini, D. (2013). Functional connectivity and oscillatory neuronal activity in the resting human brain. *Neuroscience*, *240*, 297–309. https://doi.org/10.1016/j.neuroscience.2013.02.032

Glaser, J. I., Benjamin, A. S., Farhoodi, R., & Kording, K. P. (2019). The roles of supervised machine learning in systems neuroscience. *Progress in Neurobiology*, *175*, 126–137. https://doi.org/10.1016/j.pneurobio.2019.01.008

Gorgolewski, K., Burns, C. D., Madison, C., Clark, D., Halchenko, Y. O., Waskom, M. L., & Ghosh, S. S. (2011). Nipype: A Flexible, Lightweight and Extensible Neuroimaging Data Processing Framework in Python. *Frontiers in Neuroinformatics*, *5*. https://doi.org/10.3389/fninf.2011.00013

Gramfort, A., Luessi, M., Larson, E., Engemann, D. A., Strohmeier, D., Brodbeck, C., Goj, R., Jas, M., Brooks, T., Parkkonen, L., & Hämäläinen, M. (2013). MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience*, *7*. https://doi.org/10.3389/fnins.2013.00267

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2

He, B., Astolfi, L., Valdés-Sosa, P. A., Marinazzo, D., Palva, S. O., Bénar, C.-G., Michel, C. M., & Koenig, T. (2019). Electrophysiological Brain Connectivity: Theory and Implementation. *IEEE Transactions on Biomedical Engineering*, *66*(7), 2115–2137. https://doi.org/10.1109/TBME.2019.2913928

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, *24*(6), 417.

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Hyafil, A., Giraud, A.-L., Fontolan, L., & Gutkin, B. (2015). Neural Cross-Frequency Coupling: Connecting Architectures, Mechanisms, and Functions. *Trends in Neurosciences*, *38*(11), 725–740. https://doi.org/10.1016/j.tins.2015.09.001

Islam, M. K., Rastegarnia, A., & Yang, Z. (2016). Methods for artifact detection and removal from scalp EEG: A review. *Neurophysiologie Clinique/Clinical Neurophysiology*, *46*(4), 287–305. https://doi.org/10.1016/j.neucli.2016.07.002

Kobler, R. J., Sburlea, A. I., Mondini, V., & Müller-Putz, G. R. (2019). HEAR to remove pops and drifts: The high-variance electrode artifact removal (HEAR) algorithm. *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. https://doi.org/10.1109/EMBC.2019.8857742

Kus, R., Kaminski, M., & Blinowska, K. J. (2004). Determination of EEG activity propagation: Pair-wise versus multichannel estimate. *IEEE Transactions on Biomedical Engineering*, *51*(9), 1501–1510. https://doi.org/10.1109/TBME.2004.827929

Makeig, S., Bell, A., Jung, T.-P., & Sejnowski, T. J. (1995). Independent component analysis of electroencephalographic data. *Advances in Neural Information Processing Systems*, *8*.

Makowski, D. (2016). Neurokit: A python toolbox for statistics and neurophysiological signal processing (eeg eda ecg emg...). *Memory and Cognition Lab'Day*, *1*.

Marcus, R., Negi, P., Mao, H., Zhang, C., Alizadeh, M., Kraska, T., Papaemmanouil, O., & Tatbul, N. (2019). Neo: A Learned Query Optimizer. *Proceedings of the VLDB Endowment*, *12*(11), 1705–1718. https://doi.org/10.14778/3342263.3342644

Meunier, D., Pascarella, A., Altukhov, D., Jas, M., Combrisson, E., Lajnef, T., Bertrand-Dubois, D., Hadid, V., Alamian, G., Alves, J., Barlaam, F., Saive, A.-L., Dehgan, A., & Jerbi, K. (2020). NeuroPycon: An open-source Python toolbox for fast multi-modal and

reproducible brain connectivity pipelines. *BioRxiv*, 789842. https://doi.org/10.1101/789842

Milosevic, L., Scherer, M., Cebi, I., Guggenberger, R., Machetanz, K., Naros, G., Weiss, D., & Gharabaghi, A. (2020). Online Mapping With the Deep Brain Stimulation Lead: A Novel Targeting Tool in Parkinson's Disease. *Movement Disorders*, *35*(9), 1574–1586. https://doi.org/10.1002/mds.28093

Mormann, F., Fell, J., Axmacher, N., Weber, B., Lehnertz, K., Elger, C. E., & Fernández, G. (2005). Phase/amplitude reset and theta–gamma interaction in the human medial temporal lobe during a continuous word recognition memory task. *Hippocampus*, *15*(7), 890–900.

Newville, M., Stensitzki, T., Allen, D. B., Rawlik, M., Ingargiola, A., & Nelson, A. (2016). LMFIT: Non-linear least-square minimization and curve-fitting for Python. *Astrophysics Source Code Library*, ascl-1606.

Neymotin, S. A., Daniels, D. S., Caldwell, B., McDougal, R. A., Carnevale, N. T., Jas, M., Moore, C. I., Hines, M. L., Hämäläinen, M., & Jones, S. R. (2020). Human Neocortical Neurosolver (HNN), a new software tool for interpreting the cellular and network origin of human MEG/EEG data. *ELife*, *9*, e51214. https://doi.org/10.7554/eLife.51214

Nolte, G., Bai, O., Wheaton, L., Mari, Z., Vorbach, S., & Hallett, M. (2004). Identifying true brain interaction from EEG data using the imaginary part of coherency. *Clinical Neurophysiology*, *115*(10), 2292–2307. https://doi.org/10.1016/j.clinph.2004.04.029

Nolte, G., Ziehe, A., Nikulin, V. V., Schlögl, A., Krämer, N., Brismar, T., & Müller, K.-R. (2008). Robustly Estimating the Flow Direction of Information in Complex Physical

Systems. *Phys. Rev. Lett.*, *100*(23), 234101. https://doi.org/10.1103/PhysRevLett.100.234101

Nunez, P. L. (2010). REST: A good idea but not the gold standard. *Clinical Neurophysiology*, *121*(12), 2177–2180. https://doi.org/10.1016/j.clinph.2010.04.029

Pearson, K. (1901). LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559–572.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825–2830.

Plotly Technologies Inc. (2015). *Collaborative data science*. https://plot.ly

R Core Team. (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Rabiner, L. R., & Gold, B. (1975). Theory and application of digital signal processing. *Englewood Cliffs: Prentice-Hall*.

Reimers, M. (2011). Local or distributed activation? The view from biology. *Connection Science*, *23*(2), 155–160. https://doi.org/10.1080/09540091.2011.575930

Sahasrabuddhe, K., Khan, A. A., Singh, A. P., Stern, T. M., Ng, Y., Tadić, A., Orel, P., LaReau, C., Pouzzner, D., Nishimura, K., Boergens, K. M., Shivakumar, S., Hopper, M. S., Kerr, B., Hanna, M.-E. S., Edgington, R. J., McNamara, I., Fell, D., Gao, P., … Angle, M. R. (2020). *The Argo: A 65,536 channel recording system for high density neural recording in vivo* (p. 2020.07.17.209403). bioRxiv. https://doi.org/10.1101/2020.07.17.209403

Salehi, S., Du, J. T., & Ashman, H. (2018). Use of Web search engines and personalisation in information searching for educational purposes. *Information Research: An International Electronic Journal*, *23*(2), n2.

Scherer, M., Steiner, L. A., Kalia, S. K., Hodaie, M., Kühn, A. A., Lozano, A. M., Hutchison, W. D., & Milosevic, L. (2022). *Single-neuron bursts encode pathological oscillations in Parkinson's disease and essential tremor* (p. 2022.04.05.486956). bioRxiv. https://doi.org/10.1101/2022.04.05.486956

Scherer, M., Wang, T., Guggenberger, R., Milosevic, L., & Gharabaghi, A. (2022a). *Direct Modulation Index: A measure of phase amplitude coupling for neurophysiology data* (p. 2022.02.07.479380). bioRxiv. https://doi.org/10.1101/2022.02.07.479380

Scherer, M., Wang, T., Guggenberger, R., Milosevic, L., & Gharabaghi, A. (2022b). *Directional Absolute Coherence: A phase-based measure of effective connectivity for neurophysiology data* (p. 2022.02.07.479359). bioRxiv. https://doi.org/10.1101/2022.02.07.479359

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(3), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Shannon, C. E. (1949). Communication in the Presence of Noise. *Proceedings of the IRE*, *37*(1), 10–21. https://doi.org/10.1109/JRPROC.1949.232969

Shaw, J. C. (1984). Correlation and coherence analysis of the EEG: A selective tutorial review. *International Journal of Psychophysiology*, *1*(3), 255–266. https://doi.org/10.1016/0167-8760(84)90045-X

Siegel, M., Donner, T. H., & Engel, A. K. (2012). Spectral fingerprints of large-scale neuronal interactions. *Nature Reviews Neuroscience*, *13*(2), 121–134. https://doi.org/10.1038/nrn3137

Song, J., Davey, C., Poulsen, C., Luu, P., Turovets, S., Anderson, E., Li, K., & Tucker, D. (2015). EEG source localization: Sensor density and head surface coverage. *Journal of Neuroscience Methods*, *256*, 9–21. https://doi.org/10.1016/j.jneumeth.2015.08.015

Tayeb, Z., Waniek, N., Fedjaev, J., Ghaboosi, N., Rychly, L., Widderich, C., Richter, C., Braun, J., Saveriano, M., Cheng, G., & Conradt, J. (2018). Gumpy: A Python toolbox suitable for hybrid brain–computer interfaces. *Journal of Neural Engineering*, *15*(6), 065003. https://doi.org/10.1088/1741-2552/aae186

Thorpe, S. (1989). Local vs. Distributed Coding. *Intellectica*, *8*(2), 3–40. https://doi.org/10.3406/intel.1989.873

Tort, A. B., Kramer, M. A., Thorn, C., Gibson, D. J., Kubota, Y., Graybiel, A. M., & Kopell, N. J. (2008). Dynamic cross-frequency couplings of local field potential oscillations in rat striatum and hippocampus during performance of a T-maze task. *Proceedings of the National Academy of Sciences*, *105*(51), 20517–20522.

Vaid, S., Singh, P., & Kaur, C. (2015). EEG Signal Analysis for BCI Interface: A Review. *2015 Fifth International Conference on Advanced Computing Communication Technologies*, 143–147. https://doi.org/10.1109/ACCT.2015.72

Van Rossum, G., & Drake Jr, F. L. (1995). *Python tutorial* (Vol. 620). Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands.

Vinck, M., Oostenveld, R., van Wingerden, M., Battaglia, F., & Pennartz, C. M. A. (2011). An improved index of phase-synchronization for electrophysiological data in the presence of

volume-conduction, noise and sample-size bias. *NeuroImage*, *55*(4), 1548–1565. https://doi.org/10.1016/j.neuroimage.2011.01.055

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … van Mulbregt, P. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Weise, K., Poßner, L., Müller, E., Gast, R., & Knösche, T. R. (2020). Pygpc: A sensitivity and uncertainty analysis toolbox for Python. *SoftwareX*, *11*, 100450. https://doi.org/10.1016/j.softx.2020.100450

Xue, Z., Li, J., Li, S., & Wan, B. (2006). Using ICA to remove eye blink and power line artifacts in EEG. *First International Conference on Innovative Computing, Information and Control-Volume I (ICICIC'06)*, *3*, 107–110.

Zhang, W., Chen, A., Rasch, M. J., & Wu, S. (2016). Decentralized Multisensory Information Integration in Neural Systems. *Journal of Neuroscience*, *36*(2), 532–547. https://doi.org/10.1523/JNEUROSCI.0578-15.2016