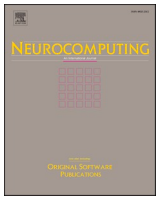
Neurocomputing 651 (2025) 130899

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



Original software publication

PyfUS: Python-based open-source software for the analysis of functional ultrasound imaging data

Théo Lambert [a,b,c,d,1], Clément Brunner [a,b,c,d,1], Gabriel Montaldo [a,b,c,d], Alan Urban [a,b,c,d,*]

[a] Neuro-Electronics Research Flanders, Leuven, Belgium
[b] VIB, Leuven, Belgium
[c] Imec, Leuven, Belgium
[d] Department of Neurosciences, KU Leuven, Leuven, Belgium

ARTICLE INFO

Communicated by Zidong Wang

Keywords:
Functional ultrasound
Whole-brain imaging
Single-voxel clustering
Data analysis
Software

ABSTRACT

Functional ultrasound (fUS) imaging has emerged as an important technology for investigating a multitude of neuro-related research areas in various animal models and patients. Beyond imaging quality and specificity, computational analysis of fUS data sets is critical for accurately and comprehensively characterize brain functions and circuits, under physiological and pathological conditions. To facilitate efficient and reproducible data analysis, we present a python-based open-source software (PyfUS) providing an end-to-end pipeline for registration, signal processing and visualization of fUS datasets. In addition to the conventional analysis - region-based averaging and correlation - we introduce the single-voxel clustering as an alternative analysis method that allows simultaneous spatial and temporal examination of the fUS signals at the finest scale allowed by the fUS. We compare the different strategies for analyzing fUS data and display the results of the analytical pipeline on a dataset comprising awake mice subjected to visual stimulation. In a standard computing environment with 32 GB of memory, a 10-Gb data size of brain-wide fUS images can be loaded in ~1 h and fully processed with the 3 analysis methods in few minutes. The flexibility of the software allows for the easy extension to other animal models and user-developed modules. We deliver a tool providing a convenient access to state-of-the-art analysis methods and an open platform for the development of new processing strategies.

1. Introduction

Over the years, the functional ultrasound (fUS) technology [24,25] has seen an increasing exposure and adoption within the neuroscience community. In short, the fUS neuroimaging modality allows the monitoring of brain hemodynamics at an unprecedented spatio-temporal resolution (~200μm^3, 5 Hz) [27] and faithfully reports on neuronal activity [1,11,21,26,29,40]. This increased attention has been allowed by multiple developments in terms of hardware (e.g., miniaturized [39] and matrix-array transducers[31,7]), experimental tools (e.g., surgery, platform, head-post, synchronous video recordings) [13,14,7,8] and the standardization of the experimental procedures [8]. Comparatively, the data analysis procedures have only marginally evolved since the seminal papers introducing the technology. Indeed, analysis still heavily relies on custom scripts implementing correlation and more recently region-averaging based on existing atlases. We posit that this is due to the lack of an open and standardized framework for the data analysis. Recently, a new analysis method has been developed, known as single-voxel clustering [22], a method provides clearer and more precise maps of brain activity in comparison to conventional methods. The present work aims to provide the community with a Python-based open-source software for fUS data analysis (PyfUS), a modular platform that allows for both the use of conventional strategies in a reproducible fashion and the use of new single-voxel clustering methods for fUS signal processing at the whole-brain scale.

2. Overview

This work presents a flexible approach to fUS data analysis along with practical guidelines. It is composed of 5 procedures. The two first

* Correspondence to: VIB-NERF, Herestraat 49, ON5 Box 602, 3000 Leuven, Belgium.
E-mail address: alan.urban@nerf.be (A. Urban).
[1] These authors contributed equally.

https://doi.org/10.1016/j.neucom.2025.130899
Received 6 May 2025; Accepted 29 June 2025
Available online 15 July 2025

procedures - data loading and data selection - format the data for the application of three different signal processing strategies (Fig. 1.a). These strategies are the conventional correlation [30,40] and atlas-based region averaging [26,7,9], and single-voxel clustering [22] that we introduce in detail in this manuscript. While initially used on a single brain region from a single-plane fUS dataset [22], we have extended the single-voxel clustering method to whole-brain fUS dataset. Once the software is downloaded and installed following instruction detailed in **Software setup**, the user can run the different procedures and replicate the results presented using example datasets (see **Example datasets**).

2.1. Procedure 1: Data formating and preprocessing

The data loading section aims to adequately format and preprocess the data to be included in the later stages of the analysis (Procedures 2–5). This section details how data are averaged, at which hierarchical level of the experiment they are processed (e.g., trial, session, animal, group; see Box 1) and details on the saving/storage of the information.

2.2. Procedure 2: Data selection, a common stage to Procedures 3–5

The data selection stage occurs once the formating and preprocessing are completed and consists of selecting the data to be used in the analysis process. This section presents how to perform the selection and set hyperparameters correctly.

2.3. Procedure 3: Region averaging analysis

The region-based averaging is a widely employed analytical strategy for examining the fUS signal that often relies on brain atlases. The underlying concept is straightforward: the fUS signals of all voxels belonging to a given region are averaged together to produce a single temporal trace, thereby leading to a simple interpretation of the signal change (Fig. 1.b). However, this approach obliterates the spatial component of the response, which could result in the aggregation of signals from different sources, including noisy voxels and signals arising from different phenomena (e.g., loco-regional activity), thus potentially leading to misinterpretation.

2.4. Procedure 4: Correlation

The correlation analysis consists of the computation of the Pearson correlation coefficient between the fUS signal of each voxel and a square window matching the stimulus pattern (Fig. 1.c). This results in a correlation map, and a Fisher transform can be employed to determine whether a voxel is significantly correlated with the stimulus[8]. This approach offers several advantages, including its straightforward implementation and its universal applicability, while allowing for a spatial quantification of the activity. However, this approach does not

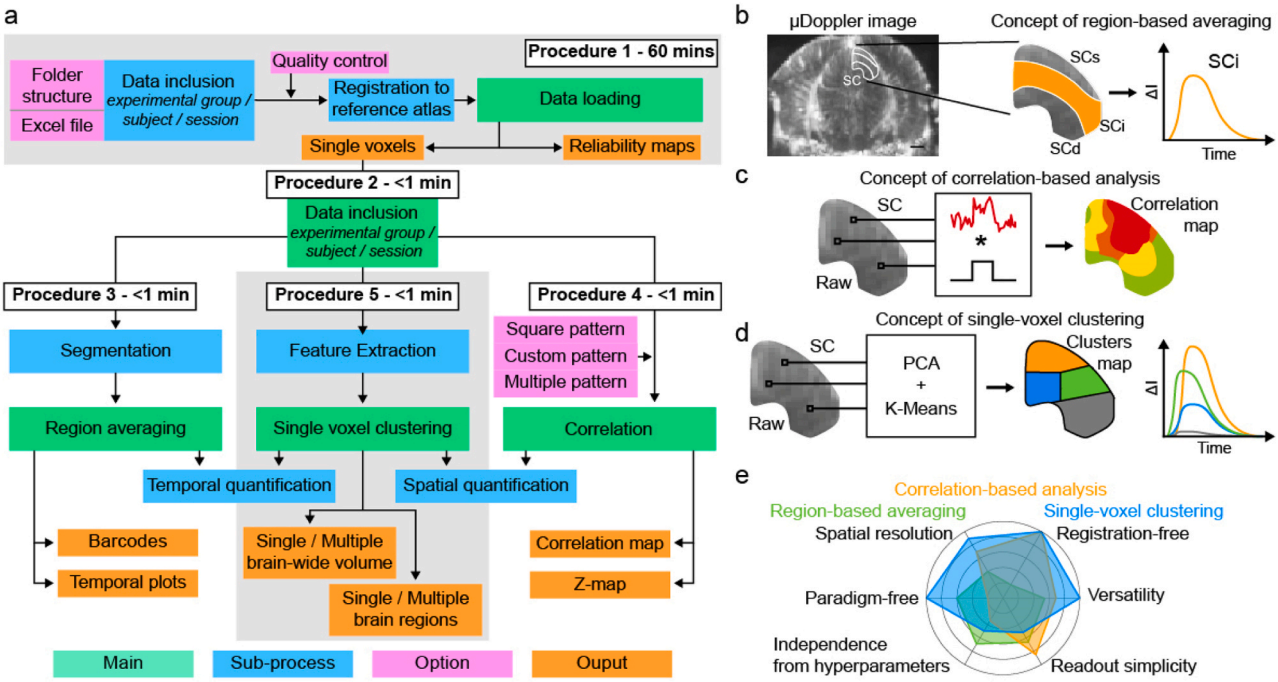


Fig. 1. Workflow and analysis concept, a, Description of the procedures performed by the software to analyze images acquired by fUS imaging. Procedure 1 consists of formating and preprocessing the fUS dataset, including quality control, registration to the reference atlas, data averaging, reliability map generation, and single-voxel signals as the main output. Procedure 2 allows the selection of the data to process. The signal from the single voxel is used as input for procedures 3–5 to perform region-based signal averaging, correlation, and single-voxel clustering, respectively, resulting in dedicated outputs and associated visualizations. The main steps are color-coded in green, sub-processes in blue, options in pink, and outputs in orange. b, *Left;* Example of µDoppler image of mouse brain acquired with linear ultrasound transducer, with the right superior colliculus (SC) outlined in white. Scale bar = 1 mm. *Right;* Concept of region-based averaging performed in Procedure 3. The temporal traces of all voxels belonging to a given region (e.g., SCi, in orange) are averaged to obtain a single trace summarizing the activity in the region of interest. ΔI denotes the variation of signal amplitude. c, Concept of correlation-based analysis performed in Procedure 4. The Pearson correlation coefficient is computed between the temporal trace of each individual voxel and a square pattern replicating the stimulus pattern (or other kind). It results in a correlation map in which each voxel is color-coded based on its correlation coefficient. This map can be further binarized with a Fisher transform to obtain a set of voxels statistically correlated with the stimulus at a given confidence level (z-score map). d, Concept of the single-voxel clustering performed in Procedure 5. Temporal traces from single voxels are grouped together. The dimensionality of the input signal is reduced using a principal component analysis (PCA) and the output fed into a K-means clustering algorithm. The procedure results in a cluster map where each voxel is color-coded based on its cluster attribution, and the temporal traces of single voxels grouped by cluster. e, Qualitative comparison of the three analysis procedures, including region-based averaging (green), the correlation (orange) and the single-voxel clustering (blue).

Box 1
| Which fUS data can be used?.

Data format and metadata requirements. The software is designed to work with input data that are MATLAB files (.mat). The following fields are required:

- "I" contains the data itself, a 4D volume whose first 3 dimensions are space and the last one is time.
- "md" refers to the metadata, organized as follow:
  - "size" as the number of voxels for each dimension
  - "voxelSize" as the voxel size for each spatial dimension
  - "Direction" as the orientation of the data for each spatial dimension, separated by a period (e.g., 'DV.AP.LR' if the first dimension is the dorso-ventral axis, the second the antero-posterior axis and the last the left-right axis).

The transformation matrix, if provided, is expected to have a field 'Transf', which contains the 4×4 transformation matrix in the field 'M'. For the file to be recognized by the software, the only requirement is that 'transf' or 'Transf' is present in the filename, separated from other elements by '_' (examples of valid names: 'Transf.mat', 'transf_mouseA.mat', 'ses1_Transf_mouseB.mat'). Example data are available on the GitHub repository.

Dataset organization. Dataset is organized in a structured and tree-like way:

- experiment ID, providing a unique identifier to the experiment (ex: visual_experiment_01012024)
- one / multiple experimental group(s), defined by a unique identifier (ex: control, treatment)
- one / multiple subject(s), defined by a unique identifier (ex: mouseA, mouseB, mouseC)
- one / multiple session(s) (ex: session1)
  - a folder 'fus', containing one / multiple stimulus / stimuli (ex: checkerboard, gratings)
  - a folder 'other', containing the transformation matrix and other information

The path to the folder containing the datasets is called the *<path to the dataset>* . Below is an example of organization:

- experiment ID: the name of the experiment
  - experimental group 1 (e.g., WT)
    - subject 1 (e.g., mouseA)
      - session 1
        - fus
          - stimulus 1 (e.g., drifting grating orientation 0°)
          - …
          - stimulus T (e.g., drifting grating orientation 180°)
        - other
      - session S
    - subject M
  - experimental group G (e.g., KO)

Compatibility with 2D. The software is fully compatible with data acquired with linear (2D) and matrix array (3D) transducers. However, for 2D single plane images, the data set must be converted to a volume by creating an artificial dimension. For example, a coronal image of size (175,128) with orientation dorso-ventral and left-right (DV.LR) is converted to a volume of size (175,1128) with orientation dorso-ventral, antero-posterior and left-right (DV.AP.LR).

provide any information about the shapes of hemodynamic responses and potentially excludes pre-stimulus (e.g., task preparation, stimulus anticipation) or post-stimulus activities (e.g., non-aligned behavior, delayed neuronal response).

2.5. Procedure 5: Single-voxel clustering

As an alternative of conventional methods (Procedures 3 and 4), we detail the single-voxel clustering method that addresses the drawbacks of both the region-based averaging and correlation approaches. This method involves the clustering of the signal from each voxel individually. It consists of two stages: a feature extraction stage (e.g., with principal component analysis (PCA)) to reduce the dimensionality of the data and a clustering stage, implemented with K-Means (Fig. 1.d). This method allows for joint spatial and temporal analysis of the data with low bias through the resulting cluster maps (in which each voxel is color-coded depending on its cluster attribution) and the traces of each cluster [22]. Importantly, recent research has demonstrated that small groups of voxels as those yielded with single-voxel clustering faithfully report local neuronal activity[21]. Disadvantages include the more complex readout compared to the other approaches, particularly at the brain-wide scale, and some noise sensitivity depending on the feature extraction method. A qualitative comparison of region-based averaging, correlation and single-voxel clustering analyses is presented in Fig. 1.e to help users to select the most appropriate method to their needs.

2.6. Applications

This workflow aims to incorporate conventional and novel approaches for understanding brain function captured by fUS at different scales and in different experimental conditions, i.e., from anesthetized to awake [26,8], physiology [33,5] to pathology [12,17,32,4,6], acute to chronic, and restrained to freely behaving [14,34,35,39,7,9]. The pipeline includes analyses routinely performed in most research papers using the fUS modality (i.e., voxel-to-voxel correlation maps with thresholds of activity, atlas-based segmentation with regional averaging), as well as innovative strategies such as the single-voxel clustering. The latter approach has been developed to fill a gap in spatiotemporal data exploration and to deal with atypical data sets, such as those generated i) in models lacking the detailed and digitized atlas

typically required for reliable data segmentation and (sub-)regional analysis of functions, ii) without prior knowledge of recording localization, or iii) in single trials when reproducibility of experimental procedures is challenging. As the work proposed here may not correspond to specific experimental designs or experimenters wishes, we have opened the software to allow collaborative efforts to add pre-existing but unimplemented approaches or to develop new analysis tools and pipelines.

2.7. Comparison with other approaches: Advantages and limitations

As fUS technology has only recently entered the field of neuroimaging, few software packages for analyzing fUS data have been made available to the community. On the one hand, a commercially available software [2] supports i) registration of the brain to a reference atlas, ii) generation of task-evoked activation maps with associated regional time courses, or iii) seed-based functional connectivity. However, the software is dependent to the acquisition system with restricted access to the script, constrained to mouse model, and has limited analysis options and data visualization. On the other hand, a free software package [8] is available online (https://github.com/nerf-common/whole-brain-fUS). It is suitable for i) manual atlas-based registration and segmentation, ii) the assessment of data stability and motion artifacts, iii) correlation and region-based analyses. While this package is open to fUS users, it is supported by Matlab, an expensive software that limits a broad accessibility and use. It provides sufficient analysis capabilities for highly structured experiments (block design, data averaging, …) but leaves less flexibility for more advanced analysis.

Therefore, to the best of our knowledge, there is no open-source tool that combines high flexibility (at each step and process) and high versatility to perform the main procedures (data formating, registration, signal preprocessing, visualization) from regional to brain-wide scale regardless of paradigm specificity. Furthermore, this fully open software has been designed to promote high modularity, so that the community can easily add new modules and options to the current framework. Importantly, the workflow and procedures are designed to support reproducibility of fUS analysis across users, as the parameters used, and the features extracted can easily be shared and compared.

Currently, we do not present a graphical user interface. While a code-only approach may discourage some users, we provide a fully annotated and operated script that aims to empower the user towards full access and understanding of the analysis procedure. However, we do not exclude the possibility of implementing this framework in a user-friendly graphical interface at a later stage.

2.8. Experimental design

The analysis workflow presented does not put any specific constraint on the experimental design. This include the experimental paradigm (number of stimuli, duration, …), probe specifications (linear array, matrix array,…), probe orientation (sagittal, coronal, with angle) and brain volume covered (single plane, scan, whole brain), based on the requirements of individual experiments. As mentioned above, our approach is suitable for atypical datasets, such as those from animal models lacking brain atlas, or for the analysis of single trial. However, it is important to keep in mind that this tool was primarily design for the analysis of trial-based or trial-like paradigms.

2.9. Subjects

The fUS images presented in this procedure were generated from adult C57bl/6j mice subjected to cranial window, head-post implantation and brain imaging under awake conditions. Animal preparation (i.e. surgery, medication and post-operative care), habituation to the setup and head fixation strictly follow the protocol previously described [8].

2.10. Ultrasound transducer

Parameters such as field of view, spatial resolution, and imaging frequency depend on the transducer and hardware used to acquire brain hemodynamics and function. Due to the novelty of using matrix arrays [31,7], as well as their cost and availability, about 90 % of fUS experiments have been performed with linear arrays to date. To cover most experimental conditions, from single-plane imaging and brain-wide scanning with the linear transducer to whole-brain coverage with the matrix array, we have implemented a flexible approach that allows data analysis from multiple transducer sizes and shapes. However, some requirements (including metadata) must be fulfilled to ensure the adequate data processing regardless of the transducers. Such details are provided in Box 1.

2.11. Experimental datasets

The proof-of-principle validation of the analytical pipeline is based on several dataset covering two stimulation paradigms. It has been collected with a static matrix array aiming at imaging brain-wide functions of awake head-fixed mice as illustrated in Fig. 2a. The dataset consists of a visual stimulus displayed on a large screen aligned to the right eye of the mouse. The stimuli started with a 10 sec of grey background (50 % luminance), followed by a drifting grating stimulus consisting of a full-field sinusoidal grating that drifts in a direction perpendicular to the orientation of the grating (either 0° and 180°) for 20 sec, before switching back to grey background (50 % luminance) for 7 sec until the end of the trial. This stimulation pattern is adapted from Brunner et al.[7]. This dataset is composed of 3 mice, with 9 / 7 / 9 sessions respectively and 5 trials/sessions. Here, matrix transducers were only used for reasons of convenience. As expressed above and discussed in Box 1, the software supports a large diversity of fUS dataset, including single-plane imaging [22] and brain-wide scanning with linear transducers.

2.12. Expertise needed to implement the software

This work is designed to provide neurobiological and medical imaging scientists with full access to and in-depth understanding of the analysis of the fUS dataset they are studying. To install and run PyfUS software, users will need some basic coding skills (ability to use command-line software and knowledge on the base principles of Python). It is advantageous for the users to be familiar with the step-by-step procedure for fUS imaging and registration to a reference atlas. To assist the users in performing Python-based procedures, we have extensively annotated the scripts and provided a clear step-by-step procedure hereafter with several examples and results along the procedure. Additional resources such as a full API documentation are also available online.

3. Material and methods

3.1. Biological materials

Animal models: 3 wildtype C57bl6j mice (~25–30 g of body weight; >8 weeks old; Janvier Labs). All the biological data analyzed in this work has been collected from several projects, but none of it has been generated specifically for it. All experiments presented in this work were approved by the Committee on Animal Care of the Katholieke Universiteit Leuven, following the national guidelines on the use of laboratory animals and the European Union Directive for animal experiments (2010/63/EU). The current approach has so far been applied to male mice; however, the applicability of the method is not sex restricted.

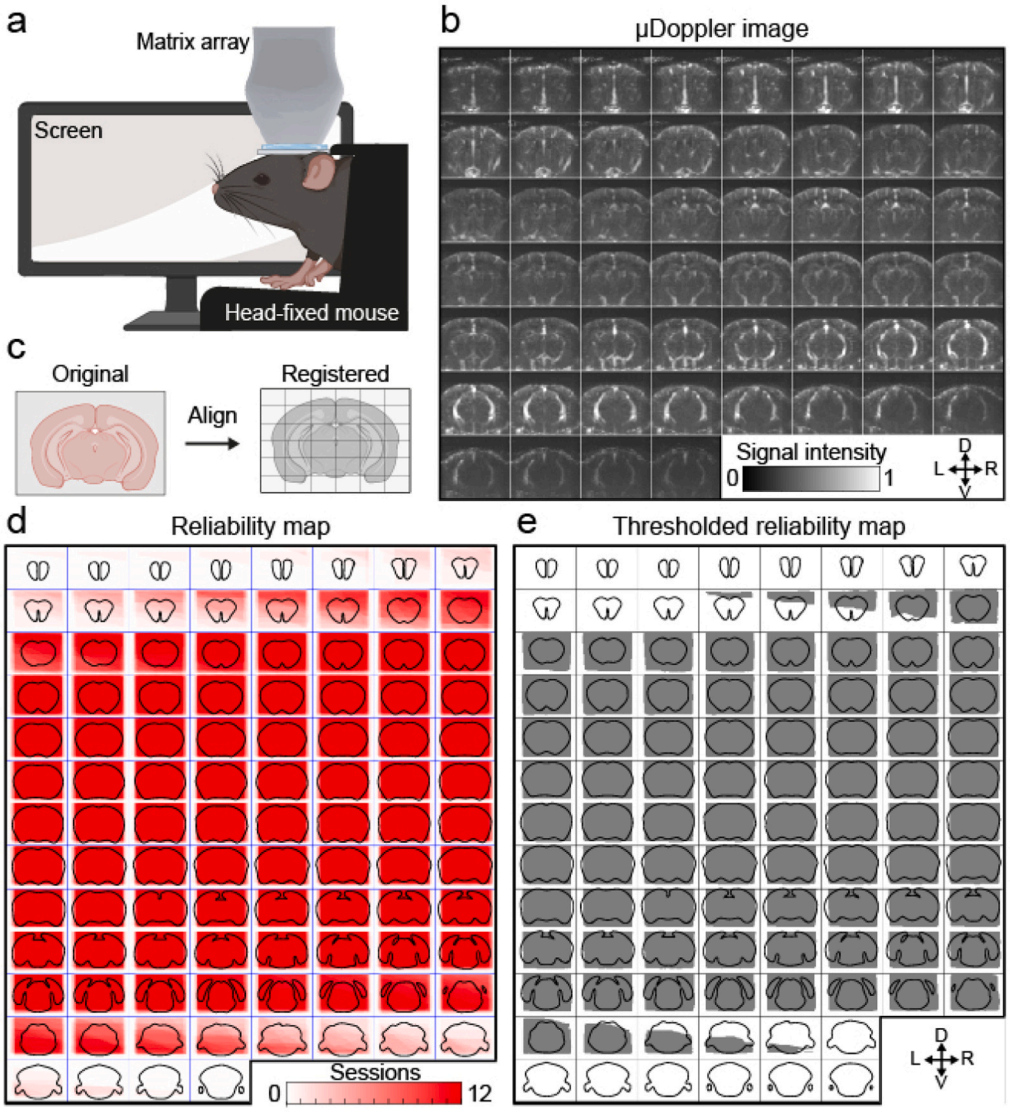


Fig. 2. Experimental dataset and preprocessing, a, Illustration of the experimental setup used for brain-wide fUS imaging of awake mice. The screen displaying visual stimulation is positioned to the right side of the mouse's face, as previously described[26,8]. b, Typical µDoppler image of the mouse brain acquired with the ultrasound matrix array. c, Concept of the registration of the µDoppler image to the reference atlas. d, Reliability map for 10 brain volumes imaged with the ultrasound matrix array from 10 imaging sessions (shades of red), after rigid registration to the reference atlas. e, Reliability map (in grey) in which individual voxels are imaged by a minimum of 70 % of sessions, with in black the outline of the reference atlas. D, dorsal; V, ventral; L, left; R, right.

3.2. Equipment

All required software and data, along with details about how to download all required data for the work are available online at https://github.com/OpenfUS/PyfUS and https://zenodo.org/records/13341387.

3.3. Hardware

- A workstation with 32 GB RAM as minimum requirements.
- ~ 15 GB of storage space as minimum requirements.

3.4. Software

- An up-to-date operating system (Microsoft Win10 Pro 64 bits).
- PyfUS software, download from the GitHub repository at https://github.com/OpenfUS/PyfUS
- PyfUS documentation, available at https://pyfus.readthedocs.io/en/latest
- Mamba https://github.com/mamba-org/mamba
- Miniforge, version 22.3.1 or latest (https://github.com/conda-forge/miniforge)
- Allen Mouse Brain Common Coordinate Framework v3. The CCFv3 used in the PyfUS is embedded within the GitHub repository.
- Atom, version 1.63 (https://atom-editor.cc/), a free and open-source text and source code editor developed by GitHub.
- List of Python packages included in the virtual environment:
  - Python, version 3.9.16 (https://www.python.org/).
  - Numpy, version 1.23.5 (https://numpy.org), a package used for handling arrays and mathematical operations on arrays (e.g., matrix multiplication).
  - Scipy, version 1.10.0 (https://scipy.org/), a package used for interpolation, image transforms, integration, MATLAB files loading and signal filtering.
  - Pandas, version 1.5.2 (https://pandas.pydata.org/), a package used for handling data tables (data frames).
  - Mat73, version 0.63 (https://pypi.org/project/mat73/), a package for loading MATLAB v7.3 files into Python native data types.

- Pynrrd, version 1.0.0 (https://pypi.org/project/pynrrd/), pynrrd is a module for reading and writing NRRD (Nearly Raw Raster Data, http://teem.sourceforge.net/nrrd/) files into and from numpy arrays.
- Scikit-image, version 0.19.3 (https://pypi.org/project/scikit-image/0.19.3/), a package for image processing in Python.
- Scikit-learn, version 1.2.2 (https://scikit-learn.org/stable/whats_new/v1.2.html), a package for using standard machine learning tools, including dimensionality reduction and clustering.
- Seaborn, version 0.12.2 (https://seaborn.pydata.org/), a statistical data visualization library used to generate informative plots.
- Pillow, version 9.3.0 (https://pypi.org/project/pillow/9.3.0/), a package with image processing capabilities.
- Pip, version 23.0.1 (https://pypi.org/project/pip/), a package installer for Python.
- Nibabel, version 5.1.0 (https://pypi.org/project/nibabel/), a package to read and write access to common neuroimaging file format.
- Wheel, version 0.37.1 (https://pypi.org/project/wheel/) a package for built-package format for Python.

The estimated memory needed for running through the procedures is 15–45 GB depending on the parameters of Procedure 1; however, the memory needed stands with the data size analyzed. Moreover, the timings expressed along the procedure are dependent on the hardware, software and dataset specifications detailed above.

3.5. Equipment setup

Download the example dataset used along this study at https://zenodo.org/records/13341387. The provided data or original dataset should follow the organization as detailed in Box 1 to ensure correct data loading and analysis. The processing of the fUS dataset requires the installation of Miniforge Prompt and the creation of a virtual environment following the instructions described in **Software setup.**

Download and install Mamba. Open Miniforge Prompt and create a virtual environment ensuring the smooth functioning of the PyfUS software by executing the command:

mamba create -n pyfus python= 3.9

Type *Y* to proceed to the next step.

Activate the environment using the command:

mamba activate pyfus

To install the library, run:

pip install pyfus_lib

Download the example codes located at: https://github.com/OpenfUS/PyfUS/main/examples. Along the analysis, *<path to the examples>* refers to the physical location of the example files (e.g., C:/Users/OpenfUS/Downloads).

4. Procedures

To get familiar on how to run data loading with the software, follow the code example_data_loading.py. PyfUS only supports Matlab format file (.mat) as fUS data input. Each step of this sequence must be run every time before starting any PyfUS procedure, unless the environment is not closed after a previous analysis.

All commands should be run in the PyfUS directory (i.e., *<path to the examples>*). If the user is not in that directory in the terminal, the first step is to change the current directory to PyfUS and activate the virtual environment using these command lines:

cd <path to the examples>

mamba activate pyfus

4.1. Procedure 1: Data loading and pre-processing of input data

In a code editor, auto-save is not recommended and might not be available. Save any updates made to the script regularly. Check that the changes were saved before executing the script. The software documentation contains detailed information on the formatting requirements and the types of input that are accepted: see *DataLoaderAndPreproc* object in the *data_loading* module.

1. Open with the code editor the example dataset: *example_data_loading.py*, located in *<path to the examples>* . The example code is organized in three sections:
   - o *Hyperparameters* (Lines 7–29), where parameters for data loading are selected.
   - o *Trials preprocessing object* (Lines 31–35), where objects for performing trial selection and/or filtering are defined.
   - o *Data loading* (Lines 37–62), where the code for performing the data loading is executed. This section is not expected to be modified.
2. Adjust the parameters for the data loading. If the provided dataset is used, all parameters can be left as default except for root_folder.

Required parameters:

- Set the variable *root_folder* to the path to the dataset to be used (Line 10, example: *'C:/Users/OpenfUS/Downloads'*). If not existing, two folders are created, one to store the loaded data (*loaded_data)* and a second to store output figures (*output*).

  In a code editor, check that the paths set in the code only '/' as folder separators, as '\' is a Python-reserved character.
- Set the variable *experiment_ID* variable to select the experiment to be loaded (Line 11). If using the example dataset provided, set to *'example_dataset_gratings'*.
- Select the loading mode (Line 13) by setting the variable *mode* to *folder_tree* or *excel*:
  - o *folder_tree* (recommended), the data in the *<root_folder>* / *<experiment_ID>* is expected to be organized as in Box 1. The variable *excel_source* can be ignored here.
  - o *excel,* an excel source file is requested to fetch the selected data per session. Set the variable *excel_source* (Line 14) to the path to the excel file. This option is not recommended since it requires a lot of manual input in the excel source file. However, it offers more flexibility than the *folder_tree* mode in terms of physical location of the data. Find a template of the excel file in Fig. 1 - Supplement Table 1.

- Set the experimental groups, subjects, sessions, and stimuli to load. The input format is the same for all parameters: either a list of the identifiers of the element as strings (e.g., *['mouseA', 'mouseB']*) or *None* if you want to include all elements (Lines 16–19).

Do not use the character '_' in the identifiers as it is used to parse filenames. For example, 'mouseA' is a correct identifier, 'mouse_A' is not.

- Set the reduction method (i.e., averaging), either *'mean'*, *'median'* or *'single_trial'* (Line 21). If *'single_trial'* is selected, data will be processed at the single trial level; thus, the hierarchical level at which the data is averaged (i.e., *level_avg)* will be ignored (Line 22).
- Select the hierarchical level at which the data is averaged (Line 22):
  - o *'session'*, data is averaged at the session level yielding one file per session and stimulus. Format filename is *'<experimental group>_<subject>_<session>_<stimulus>_avg'*.
  - o *'subject'*, data is averaged at the subject level, including all selected sessions and yielding one file per subject and stimulus. Format filename is *'<experimental group>_<subject>_<stimulus>_avg'*.
  - o *'expgroup'*, data is averaged across experimental groups, including all selected sessions and subjects, and yielding one file per group and stimulus. Format filename is *'<experimental group>_<stimulus>_avg'*.

Example, if a dataset contains two experimental groups *'groupA'* and *'groupB'* subjected to *'stimulus1'* and *'stimulus2'*, with an averaging set at the group level, the name of the elements will be: *'groupA_stimulus1_avg', 'groupA_stimulus2_avg', 'groupB_stimulus1_avg', 'groupB_stimulus2_avg'*.

Each generated file will be referred to as an 'element' and 'name' will refer to the filename.

Note that the higher the hierarchical level, the higher the memory consumption is.

- Set the variable *register* to *True* if the data must be registered to the reference atlas, *False* otherwise (Line 24). See Box 2 for more information.
- Define the time domain to which the data is normalized, termed *'baseline'* The baseline is calculated by taking the median of the data over the selected time range, resulting in a 3D matrix. The built-in range function allows a convenient definition of the temporal range, with the first and last parameters being the start and end of the baseline period, respectively (Line 25).

Note that the normalization is performed at the trial level if the averaging level is *single_trial*, and at the session level otherwise (each session is normalized independently). The choice to normalize at the session level is motivated by considerations of baseline consistency. Indeed, different sessions may have slightly different imaging conditions, resulting in different distributions of raw values, and thus different baseline values. Consequently, the average of all baselines may not be appropriate for normalizing individual sessions.

- Set the variable *make_reliability_maps* to *True* to compute the reliability maps (Line 27). Reliability maps are defined as follows: for each voxel, the reliability is the ratio between i) the number of sessions in which this voxel is imaged, ii) the total number of sessions. Therefore, this index ranges from 0 (no sessions in which this voxel is imaged) to 1 (the voxel is imaged in all sessions).
- Set the variable *remove_unreliable* to a float number (Line 28) to ignore voxels whose reliability is below this value during the analysis (Fig. 2d,e). If the variable is set to *None*, no voxel will be excluded.

Note that stimuli cannot be averaged together during this process. Although not recommended, this can be achieved once data loading completed.

Optional parameters:

- Set *frame_removal* to *None* if no preprocessing is desired (Line 34). The default preprocessing is the removal of outlier frames in each trial, as presented in Steps 44–45 of the protocols [8]. For an advanced use, it is possible to provide a custom preprocessing object to the variable *frame_removal* that will be applied on the raw trials before reduction. Such object will be initialized in the data loader and called using the _call_ method on a list of matrix containing the single trials. A list of matrix is expected as output.

Box 2
| Atlas extraction and variations.

Reference brain atlas. The atlas used was extracted from the Allen Mouse Brain Common Coordinate Framework (CCF v3) using the annotated volumes available here: http://download.alleninstitute.org/informatics-archive/current-release/mouse_ccf/annotation/ccf_2017/. The correspondence between anatomical regions and labels in this annotated volumes were determined using the associated ontology available here: http://api.brain-map.org/api/v2/structure_graph_download/1.json. We implemented the 100-μm resolution version of the CCF v3 since it offers the closest resolution to the fUS imaging. The atlas was further simplified by grouping the parts and layers of regions. It contains anatomical structures composed of multiple brain regions, each associated with its acronym and region number. Importantly, each brain region belongs to only one anatomical structure. The list of anatomical structures and acronyms is:

- "CB": Cerebellum,
- "CTXsp": Cortical subplate,
- "HPF": Hippocampal formation,
- "HY": Hypothalamus,
- "Isocortex": Isocortex,
- "MB": Midbrain,
- "MY": Medulla,
- "OLF": Olfactory areas,
- "P": Pons,
- "PAL": Pallidum,
- "STR": Striatum,
- "TH": Thalamus.

By default, the non-vascularized structures including fiber tracts and ventricles have been excluded from the analyses and labelled as belonging to the 'empty' structure in the atlas. If necessary, they can be accessed and used in the analysis. The full atlas or other simplifications can also be used with the software.

Regions and structures input format. To select structures or regions of interest, use the generic format '*<acr>, <hemisphere>*', where '*<acr>*' refers to the acronym of the region or the structure and '*<hemisphere>*' to *'L', 'R', 'LR'* for left, right and both, respectively. The list of regions, and associated acronyms, is included in the atlas is available in the file: *atlases/atlases_lists/regions_ccf_v3_100_nolayersnoparts.txt* on the GitHub repository.

Data Registration. Registration is optional, and both correlation (Procedure 4) and single-voxel clustering (Procedure 5) analyses can be performed on unregistered data. This feature is of interest for processing data when an atlas is not yet implemented in the software or available from the community. However, we strongly recommend using this option for mouse datasets, as it allows for data averaging (group, conditions, regions), spatial quantification, and display of overlays on spatial maps. The registration procedure is implemented in the same way as in Brunner, Grillet et al. Brunner et al., [8] and is merely a Python translation. It is designed to work with transformation matrices estimated using the registration software from Brunner *et al.*[8]. Note that if the dataset has been registered with a reference atlas in some other way, it is still possible to skip the registration step and consider the data registered in the analysis workflow.

3. Run the loading and pre-processing of the selected dataset with the following command (See Table 1 for Troubleshooting):
   cd <path to the examples>
   python example_data_loading.py
4. Type "Y" and "Enter" to proceed to the next step if the data selection is satisfactory. Type "N" and "Enter" if the input must be adjusted (See Table 1 for Troubleshooting).
   The procedure for data loading and pre-processing is time consuming - from few minutes to couples of hours - as it is dependent on the hardware specifications, the volume of data to process and whether the data is to be registered or not. However, once performed, the time to run the following Procedures 2–5 is comparatively very short - in the order of magnitude of seconds up to a few minutes.
5. Check that the following information are displayed and saved:

- In the Miniforge terminal, the name of the folder in which the '*loaded_data*' is stored is displayed. It consists of the following information separated by '_':
  - o experiment ID,
  - o reduction method,
  - o averaging level,
  - o registration status: *reg* or *noreg*, for registered and non-registered respectively,
  - o a unique identifier.

Example <*name of the dataset*> : example_dataset_gratings_median_subject_reg_cqb7h87f.

- In the <*root_folder*> */loaded_data*, a folder is created with a name as described above. It must contain the data averaged at the selected level, termed 'element', dispatched in multiple folders replicating the dataset hierarchy. An "*info.txt*" file is also generated, compiling i) the parameters used to generate the loaded dataset, ii) a list of the files included in the loading process and iii) the number of samples used to generate each averaged file will also be displayed in the file as *n_samples*. See Supplementary Figure 1.
- If the variable *make_reliability_maps* is set to *True:*
  - o In the <*root_folder*> */output*, a folder is created with a name as described above. It must contain the reliability maps. Example of reliability maps are depicted in Fig. 2d,e.
  - o A list of regions that can be excluded as the proportions of reliable voxels within this region is below the selected threshold (set for variable *remove_unreliable*) is available in the "*info.txt*" file for each element under the label "List of regions to exclude for region averaging analysis:" as shown in Supplementary Figure 1. Note that these regions are not automatically excluded but need to be specified in Step 7.
- If all information are well set, the following message is displayed in the Miniforge prompt: "All the data has been processed. Good luck with the analysis!".

4.2. Procedure 2: Data selection - common to procedures 3–5

To get familiar on how to run data analyses with the software, follow the code example_data_analysis.py. It is composed of 4 steps including the i) dataset selection, ii) region averaging, iii) correlation, and the iv) single-voxel clustering analysis.

6. Open the code *example_data_analysis.py* with a code editor, located in <*path to the examples*> .
7. Adjust the parameters of the data selection to process (Lines 10–35).

Required parameters:

- Set the variable *src* to the path of the loaded and preprocessed dataset as <path to the package> /loaded_data/<name of the dataset> , with <path to the package> the path where the dataset is stored and <name of the dataset> the identifier that was output in the terminal during the data loading process (Line 14).
  In a code editor, check that the paths set in the code only '/' as folder separators, as '\' is a Python-reserved character.
- Set the stimuli, experimental groups and subjects to include in the analysis. The input format is the same for all parameters: either a list of the identifiers of the elements as strings (e.g.: *['mouseA', 'mouseB']*) or *'None'* if you want to include all elements (Lines 17–20).
- Set the variable *sampling_rate* to the frequency (in Hz) at which the fUS images were acquired (Line 23).
- Set the variable *registered* to *True* if the data has been registered during the data loading process or to *False* if the data does not need to be registered to the reference atlas. Read the Box 2 for more information (Line 26).
- Set the variable *regions_to_exclude* to a list of acronyms of regions to exclude from the analysis (Line 29). The list is either at discretion of the experimenter or from the *info.txt* file compiling unreliable voxels as defined in Step 5.

8. Select one or several of the three signal processing methods allowed by PyfUS from this step onward by uncommenting the line - i.e., no '#' at the beginning of the line - or commenting it - i.e., add '#' at the beginning of the line -, respectively:

   Region averaging - Lines 42–59 - See Procedure 3
   Correlation - Lines 67–80 - See Procedure 4
   Single voxel clustering - Lines 87–129 - See Procedure 5

4.3. Procedure 3: Region-based analysis

The software documentation contains detailed information on the formatting requirements and the types of input that are accepted: see *RegionAveragingAnalysis* object in the *region_averaging_analysis* module.

9. Adjust the parameters for region-based analysis.

Required parameters for the initialization:

- Method plotting the barcode representation (Line 47):
  - o *names*: if *None*, all the data will be displayed. If a list of strings is provided, only the names specified in it will be displayed.
  - o *separate_plots*: whether to make separate windows for each element to be plotted.
  - o *scale*: min and max of the range for the display (arguments *vmin / vmax* from *plt.imshow*). If set to *None*, auto-scale will be used.
- Method to plot the traces of individual regions (Line 49):
  - o *region*: list of tuple(s) consisting of the acronym and hemisphere of the region to be plotted. See Box 2 for more information.
  - o *scale*: Min and max of the range for the display (arguments *vmin / vmax* from *plt.imshow*). If set to *None*, auto-scale is used.

10. Perform the temporal quantification. This process is optional and can be skipped by commenting lines 56–57.

- The method *get_regions_traces* of the *RegionAveragingAnalysis* object allows for extracting the temporal traces of clusters in a dictionary format for further processing (Line 53).
- Initialization of the object handling the temporal quantification (Line 57). Mandatory input are a dictionary containing temporal traces (variable *res*, Line 53) and a sampling rate value (variable *sampling_rate'*, Line 23).

Table 1
Troubleshooting guide summarizing common issues encountered during analysis with the PyfUS software, their possible causes, and recommended solutions to ensure reliable post-processing and data interpretation.

Step	Problem	Possible reason	Solution
Procedure 1: Data loading and pre-processing of input data			
3	Program terminated with error message: "PermissionError: Acces is denied <path>"	The folder is which the data is located does not allow the creation of new folders.	Run Miniforge Prompt as an administrator.
	Program terminated with error message: "Wrong mode (choose folder_tree or excel) –> failed initialization. Exiting…"	Unrecognized mode selected	The mode selected is incorrect. In Code *example_data_loading.py* - Line 13, set the variable *mode* to *folder_tree* or *excel*
	Program terminated with error message: "Unrecognized averaging level (choose expgroup, subject, session or all) –> failed initialization. Exiting…"	Unrecognized averaging level selected	The averaging level selected is incorrect. In Code *example_data_loading.py* - Line 22, set the variable *level_avg* to *expgroup*, *subject*, *session*, *single_trial* or *all*.
	Program terminated with error message: "Impossible to make reliability maps if data are not registered or if the extraction is done at single trial level"	The required information about data registration is specified incorrectly	The computation of reliability maps can not be executed if the *level_avg* is set to *single_trial* or if the data is not registered. In code *example_data_loading.py:* 1) Compute the reliability maps, set the variable *make_reliability_maps* to *True* (Line 27). 2) Control that the variable *reduction* is not set to *single_trial* (Line 21*)* and *register* is set to *True* (Line 24). If you want to work at the single trial level or don't want to register the data, set the variable *make_reliability_maps* to *False*.
	Warning message: "Expgroup <expgroup> was not found. Skipped."	The requested experimental group <*expgroup*> has not been found at the specified location.	This is a warning message, the process continues if enter *Y* in the Miniforge prompt window or stopped if enter *N*. However, verify that the experimental group <*expgroup*> does exist, that the folder structure is implemented correctly (if applicable) and that the name is spelled without error.
	Warning message: "Subject <subject> was not found for expgroup <expgroup> ."	The requested subject <*subject*> to load for experimental group <*expgroup*> has not been found at the specified location.	This is a warning message, the process continues if enter *Y* in the Miniforge prompt window or stopped if enter *N*. However, verify that the subject <*subject*> does exist, that the folder structure is implemented correctly (if applicable) and that the name is spelled without error.
	Warning message: "Session <session> was not found for subject <subject> <expgroup> . Skipped."	The requested session <*session*> for subject <*subject*> and experimental group <*expgroup*> has not been found at the specified location.	This is a warning message, the process continues if enter *Y* in the Miniforge prompt window or stopped if enter *N*. However, verify that the session <*session*> does exist, that the folder structure is implemented correctly (if applicable) and that the name is spelled without error.
	Warning message: "Stim <stim> was not found for mouse <subject> <expgroup> , session <session> . Skipped."	The requested stimulus <*stim*> for session <*session*> , subject <*subject*> and experimental group <*expgroup*> , has not been found at the specified location.	This is a warning message, the process continues if enter *Y* in the Miniforge prompt window or stopped if enter *N*. However, verify that the stimulus <*stim*> does exist, that the folder structure is implemented correctly (if applicable) and that the name is spelled without error.
	Warning message: "No transformation matrix was found for <session> <subject> <expgroup> . Will be ignored during registration."	The transformation matrix for session <*session*> , subject <*subject*> and experimental group <*expgroup*> was not found while the user required to perform registration. The corresponding data will therefore be ignored during the process.	This is a warning message, the process continues if enter *Y* in the Miniforge prompt window or stopped if enter *N*. However, verify that the matrix is located in the right folder and that its filename fits the requirements of the software (see Box 1).
	Program terminated with error message: "No session was found… please check your paths and requested sessions."	The elements to load cannot be found at the specified location.	If the variable *mode* (Code *example_data_loading.py* - Line 13) is set to: - *folder_tree*, verify that the requested data to be loaded can be found at the path <*root_folder*> / <*experiment_ID*> and are organised following the folder structure defined in Box 1. - *excel,* verify that the excel sheet is located in the specified folder. Find a template of the excel sheet in Supplementary Table 1.
	Program terminated with error message: "Invalid ID: character '_' detected in <elt> . Please remove this character. Exiting."	One or multiple '_' character are present in <elt> , which is an identifer of either experimental group, subject, session or stimulus.	Remove the character '_' from the identifier.
4	Program terminated with error message: "All trials must have the same lengths (issue with data <name>). Exiting"	The different trials do not have the same number of frames and, therefore, cannot be averaged together.	Remove either trials if most trials have the same number of frames, padding/cutting trials to have the same number of frames, or working at the single trial level.
	Program terminated with error message: "Unable to allocate XX GiB for an array with shape XX and data type int64"	The data to be loaded was too large for the RAM of the computer on which the data loading was executed.	This error is mainly raised during trial averaging. Solution are either to: - change the *reduction* variable to *mean* if it was set to *median* (Code *example_data_loading.py* - Line 21), - change the variable *level_avg* to a lower level (e.g., from *expgroup* to *subject* - Line 22),

(*continued on next page*)

Table 1 (*continued*)

Step	Problem	Possible reason	Solution
			- create sub sessions to limit the amount of trials per session if the averaging level is *session,* - perform a temporal downsampling of the trials, - increase the RAM capacity of the computer.
Procedure 3: Region-based analysis			
11	Program terminated with error message: "No file found… Check the path or the IDs you requested."	With the parameters chosen for the data selection, no file was found and the analysis cannot be conducted.	Check that the path to the loaded data is correct (i.e., variable *src* - Code *example_data_analysis.py* - Line 14), and the parameters *stims* (Line 17), *expgroup_ID* (Line 18) and *subject_ID* (Line 19) correspond to existing loaded data.
	Program terminated with error message: "The acronym you provided does not exist"	The acronym used to either the method *plot_region* or *get_region_traces* does not exist in the atlas.	Provide the correct acronym of interest. Refer to the file located at the path defined in variable *regions_info_path* (i.e., variable *src* - Code *example_data_analysis.py* - Line 14) to get the exhaustive list of acronyms available.
	Program terminated with error message: "Please choose an hemisphere between left "L" or right "R", or set to "LR" to display both"	The required information about hemisphere selection is specified incorrectly.	The only accepted input for variable *hemisphere* (Code *example_data_analysis.py* - Line 54) referring to hemispheres is *'L'* (left), *'R'* (right) or *'LR'* (for displaying both hemispheres)
	Program terminated with error message: "You cannot use region averaging analysis if you did not register your data during data loading."	Region averaging analysis relies on the use of an atlas. If the data was not registered during data loading, the atlas and the data will not correspond.	Set the variable *register* (Code *example_data_loading.py* - Line 24) to *True*, if transformation matrices are available, otherwise estimate transformation matrices using the original protocol. Alternatively, you can perform correlation and single voxel clustering analyses without registration.
Procedure 4: Correlation analysis			
14	Program terminated with error message: "No file found… Check the path or the IDs you requested."	With the parameters chosen for the data selection, no file was found and the analysis cannot be conducted.	Check that the path to the loaded data is correct (i.e., variable *src* - Code *example_data_analysis.py* - Line 14), and the parameters *stims* (Line 17), *expgroup_ID* (Line 18) and *subject_ID* (Line 19) correspond to existing loaded data.
	Program terminated with error message: "Wrong type for variable correlation_pattern. Should be list of tuple, np.array or dict[np.array]"	The required information about correlation pattern is specified incorrectly.	The input for variable *correlation_pattern* has specific format requirements (Code *example_data_analysis.py* - Line 69). Verify that the type of the input is either a list of tuples, a numpy array or a dictionary of numpy arrays.
	Program terminated with error message: "Incorrect number of values for n_samples. Either None, one value or same number of values as data_paths."	The required information is specified incorrectly.	The input variable *n_samples is* expected to be either *None*, an integer, or a dictionary whose number of keys equals the number of data paths in variable *data_path* (Code *example_data_analysis.py* - Line 70).
	Program terminated with error message: "Please select a significance threshold in [0.05, 0.01, 0.001]'	The required input for significant threshold is specified incorrectly.	Provide the correct input threshold. The only accepted input for the variable *significance_threshold* is either "0.05", "0.01" or "0.001" (Code *example_data_analysis.py* - Line 71).
	Program terminated with error message: "Missing keys in correlation_pattern. Check that you have: <keys> ."	Issue with the list of keys provided.	If the variable *correlation_pattern* is set to a dictionary of array (i.e., Option 3 - Code *example_data_analysis.py* - Line 71), the input must be a dictionary whose keys corresponding to the element names.
	Program terminated with error message: "Correlation pattern and data lengths do not match'.	The length of the correlation pattern does not match the temporal length of the data.	Verify that the length of the correlation pattern whose equals the length of the dataset.
Procedure 5: Single-voxel clustering			
21	Program terminated with error message: "No file found… Check the path or the IDs you requested."	With the parameters chosen for the data selection, no file was found and the analysis cannot be conducted.	Check that the path to the loaded data is correct (i.e., variable *src* - Code *example_data_analysis.py* - Line 14), and the parameters *stims* (Line 17), *expgroup_ID* (Line 18) and *subject_ID* (Line 19) correspond to existing loaded data.
	Program terminated with error message: "Please set display param to 'all' or 'mean_std'".	Error in the input provided.	The only accepted input for the *plot_signals* method is *all* or *mean_std (*Code *example_data_analysis.py* - Line 111*)*.
	Program terminated with error message: "No list of acronyms provided"	The required information about regions of interest is missing.	A list of region acronyms must be provided if *hemisphere*, *structure* or *multi_region* are selected as argument of the method (Code *example_data_analysis.py* - Line 89). Example can be found in the *info.txt* file.
	Program terminated with error message: "Unrecognized data selection method: choose between brainwide, structure, hemisphere, multi_region."	The required information about method selection is missing or specified incorrectly	The valid input for the method are either *brainwide*, *structure*, *hemisphere* or *multi_region* (Code *example_data_analysis.py* - Line 89).
	Program terminated with error message: "Not enough colors provided for the number of clusters."	Difference between the number of cluster used to process the analysis and the number of colors set for clusters identification	Verify that the number of colors provided (Code *example_data_analysis.py* - Line 107) is equal or higher to the number of clusters (Code example_data_analysis.py - Line 90).
	Program terminated with error message: "Acronym <acr> ' is not correct. Please check the atlas."	The acronym selected is incorrect.	Provide the correct acronym of interest. Refer to the file located at the path defined in variable *regions_info_path* (Code *example_data_analysis.py* - Line XX) to get the exhaustive list of acronyms available.
	Program terminated with error message: "The structure you selected does not exist"	The structure selected is incorrect.	The *structure* selected (Code *example_data_analysis.py* - Line 102) does not exist. Refer to Box 1 for the list of structures that can be used.

(*continued on next page*)

Table 1 (*continued*)

Step	Problem	Possible reason	Solution
	Program terminated with error message: "Hemispheres values should be either 'L' (left), 'R' (right) or 'LR' (both)"	The required information about hemisphere selection is specified incorrectly.	The only accepted input for variable *hemisphere* (Code *example_data_analysis.py* - Line 99) referring to hemispheres is 'L' (left), 'R' (right) or 'LR' (for displaying both hemispheres)
	Program terminated with error message: "All data must have the same number of frames. Check the different stimuli and/ or experimental groups. Exiting…"	The data provided to the clustering algorithm have different number of frames and, therefore, cannot be clustered together in the software.	Check that the number of frames in data across stimuli and experimental groups are the same, and conduct separate analyses for data with different numbers of frames.

- Call to the method for plotting violin plots of the requested metric. The required parameter is the name of the metric to display (Line 58). Box 3 and the software documentation contains detailed information on the formatting requirements and the types of input that are accepted: see *TemporalQuantification* object in the *Quantification* module.

11. Perform the region-based analysis alone by commenting Procedures 3 and 4 (Step 8) and then run the following code in the Miniforge prompt (See Table 1 for Troubleshooting):

python example_data_analysis.py
After this stage, the following displays are generated:

- Pop-up window displaying the barcode plots of the selected elements.
- Pop-up window displaying the temporal traces of the selected regions.
- If the temporal quantification was performed, violin plots with the requested metrics are generated. Each dot represents one element (see required parameters in Step 7).

Barcode plots, temporal traces of regions of interest, and temporal quantification are presented in Figs. 3a to 3c, respectively. Example of raw output are presented in Supplementary Figure 2. Importantly, these figures are not automatically saved. Click on the 'Save' button below the figure to save and store it, see Supplementary Figure 3 for detailed Matplotlib options.

4.4. Procedure 4: Correlation analysis

The software documentation contains detailed information on the formatting requirements and the types of input that are accepted: see *CorrelationAnalysis* object in the *Correlation_Analysis* module.

12. Adjust the parameters for performing the correlation analysis.

Required parameters for the initialization:

- *correlation_pattern* (Line 69).
  - o Option 1 (Default): if list of tuples, first and second elements of each tuple are respectively start and end of a square pulse.
  - o Option 2: if np.array, this array has the same size as the temporal dimension of data. The array is correlated with each element. Option 1 and 2 assume that all elements have same number of frames.

Box 3
| Spatial and temporal quantification.

Temporal quantification. The temporal quantification consists of extracting typical metrics to characterize time series. The object expects as data input a dictionary whose values are time series. Both the region averaging analysis and the single-voxel clustering have methods to get data in such format. The software documentation contains detailed information on the formatting requirements and the types of input that are accepted, see *TemporalQuantification* object in the *Quantification* module.
The current metrics are:

- *'Peak amplitude'*, the maximum amplitude of the time series.
- *'AUC'*, the area under the curve, computed using the Simpson approximation.
- *'Time to peak (s)'*, the time in seconds from the beginning of the signal to reach the maximum value in the time series.
- *'FWHM (s)'*, the width between the first and last points above half of the maximum value.
- *'Time to half max (s)'*, the time in seconds from the beginning of the signal to reach the first value above half the maximum value.

Please note that these values can be affected if the signal has multiple peaks.
Spatial quantification. The spatial quantification consists of characterizing the spatial distribution of groups (significantly correlated voxels, clusters). The object expects as data input a dictionary whose values are spatial maps. Both the correlation analysis (Procedure 4) and the single voxel clustering analysis (Procedure 5) have methods to get data in such a format. Note that the data must be registered to compute these quantifications. A group of voxels present in a spatial map, such as voxels belonging to cluster A or significantly correlated with a pattern, is called a set. The software documentation contains detailed information on the formatting requirements and the types of input that are accepted, see *SpatialQuantification* object in the *Quantification* module.
The current metrics are:

- Quantification per set: print the proportion of voxels in each structure per set (function *print_quantification_per_set*),
- Quantification per hemisphere: print the composition of a hemisphere in terms of sets (function *print_quantification_per_hemisphere*),
- Quantification per structure: print the composition of a structure in terms of sets (function *print_quantification_structure*),
- Quantification per region: print the composition of a region in terms of sets (function *print_quantification_region*).

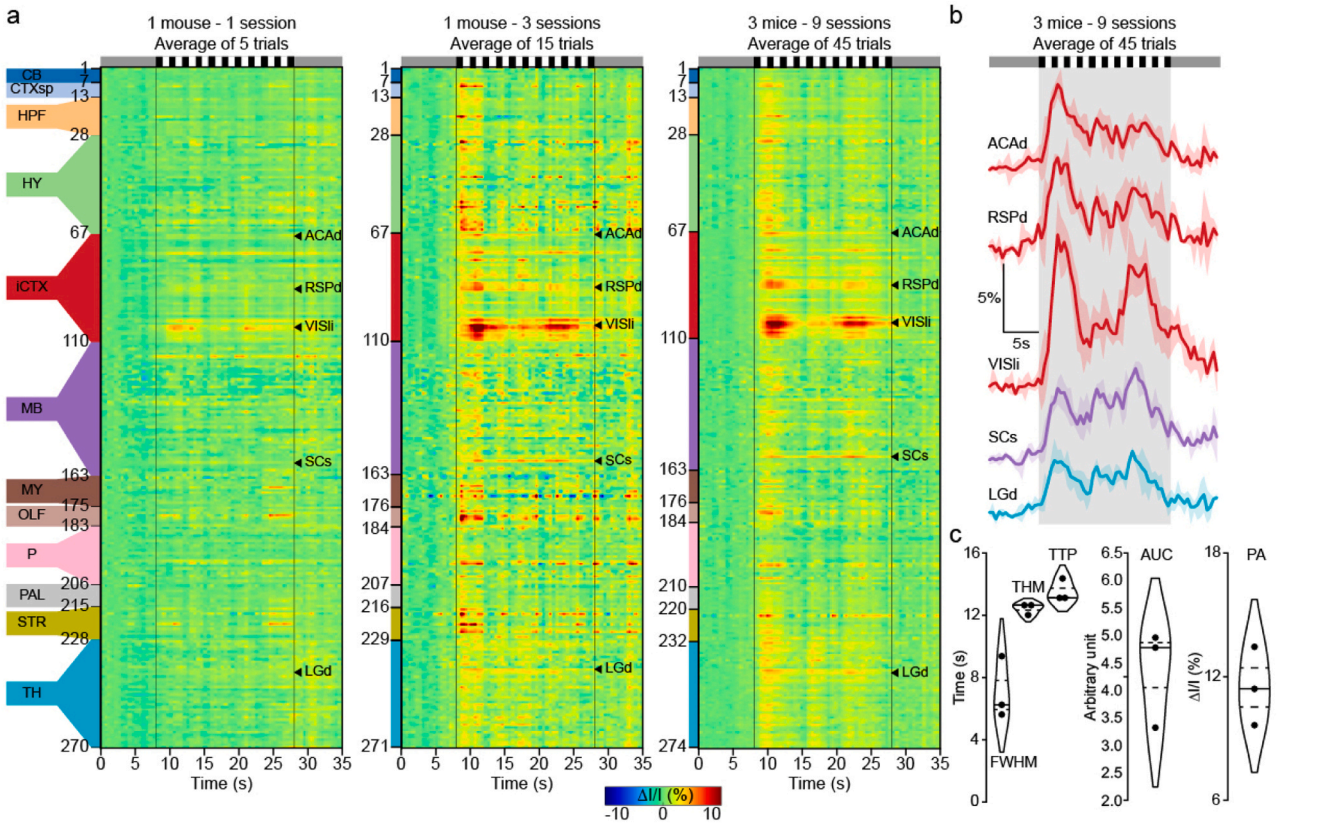


Fig. 3. Visualization and quantification after regional averaging from Procedure 3, a, Barcode visualization of temporal traces of hemodynamic responses (ΔI/I in %) to drifting grating stimuli (orientation 0°) in individual brain regions after regional averaging based on the reference atlas. The left panel shows the barcode for a single imaging session, the middle panel for 3 sessions from the same mouse (average of 5 and 15 trials, respectively), and the right panel for 9 sessions from 3 different mice (average of 45 trials). Vertical black lines: Start and end of the visual stimulation. Brain structures imaged are color-coded (Dark blue: Cerebellum/CB; Light Blue: Cortical subplate / CTXsp; Orange: Hippocampal Formation / HPF; Light green: Hypothalamus / HY; Red: isocortex / iCTX; Purple: Midbrain / MB; Brown: Medulla / MY; Light brown: Olfactory areas / OLF; Pink: Pons / P; Grey: Pallidum / PAL; Dark green: Striatum / STR; Cyan: Thalamus / TH) and listed in Fig. 3 – Supplement Table 1. Black arrowheads point brain regions of interest displayed in b. ACAd: dorsal part of the anterior cingulate area; RSPd: dorsal part of the retrosplenial area; VISli: laterointermediate area of the visual area; SCs: sensory related area of the superior colliculus; LGd: dorsal part of the lateral geniculate complex. b, Set of temporal traces of hemodynamic responses (ΔI/I in %; mean±sd; n = 3 mice, 3 sessions/mouse, 5 trials/session) to drifting grating stimuli (orientation 0°; vertical gray band) in five brain regions after regional averaging: ACAd, RSPd, VISli, SCs, and LGd (from top to bottom). Traces are color-coded according to the color of the brain structures as in (A). **c,** Example of quantification metrics calculated from the time course of a selected region of interest (VISli) of the 3 imaged mice (black dots). Full width at half maximum (FWHM), time to half maximum (THM), and time to peak (TTP) in seconds; area under the curve (AUC) in arbitrary units; and peak amplitude (PA) as change in signal intensity (ΔI/I in %). The vertical black line denotes the mean and vertical dotted black lines the 25/75 % quartiles.

  - o Option 3: if a dictionary containing np.arrays, keys must match the element names (check the info file, key are before *n_samples*) and arrays should have the same length as their associated data.
- *n_samples*: Number of trials used for computing the data reduction (Line 70). Find this value in the *'info.txt' file*, located where the processed data is stored (see Supplementary Figure 1). Set to *None* if not applicable.
  - o set to *None*, no z-score map is computed.
  - o if int, assumes that all data have the same number of samples.
  - o if dict, keys should match the data names.
- *significance_threshold*: The statistical significance threshold used for the computation of the z-score maps (Line 71). Set to 0.05, 0.01 or 0.001.

13. Perform the spatial quantification. This process is optional and can be skipped by commenting Lines 73–78.

- The method *process* of the object *CorrelationAnalysis* allows for extracting the cluster maps of target elements in a dictionary format for further processing (Line 73).
- Initialization of the object handling the spatial quantification (Line 77). Mandatory inputs are a dictionary containing spatial maps (variable *res*, Line 74), the path to the atlas to be used (variable *atlas_path*, Line 35) and the associated regions information path (*regions_info_path*, Line 34).
- The regional quantification is called with the method *print_quantification* (Line 78). The required arguments are the acronym of the region of interest and the hemisphere (e.g., *'VISp', 'L'*). Box 3 and the software documentation contains detailed information on the formatting requirements and the types of input that are accepted: see *SpatialQuantification* object in the *quantification* module.

14. Perform the correlation analysis alone by commenting Procedures 3 and 5 (see Step 8) and then running (See Table 1 for Troubleshooting):

python example_data_analysis.py
After this stage, the following displays are generated:

- Pop-up window displaying the correlation maps of the selected elements, and, if applicable, the associated z-score maps.

- If the spatial quantification was performed, the quantifications will be printed in the Miniforge terminal. In the column *set_ID*, 0 refers to non-significantly correlated voxels and 1 to significantly correlated voxels, respectively.

Correlation map, z-score map and spatial quantification are presented in Figs. 4a to 4**c**, respectively. Example of raw output are presented in Supplementary Figure 4. Importantly, these figures are not automatically saved, thus use the 'Save' button below the figure to save and store it (see Fig. 3-figure supplement 2).

4.5. Procedure 5: Single-voxel clustering

The software documentation contains detailed information on the formatting requirements and the types of input that are accepted: see *SingleVoxelClustering* object in the *clustering* module.

15. Adjust the parameters for performing the single-voxel clustering analysis.

Required parameters for the initialization of the single-voxel clustering object:

- Select the method to determine voxels included in the clustering process (*method* in Line 89):
  - *volume*, voxels from the whole volume. For this option, the registration of the data is not required.
  - *hemisphere*, voxels from one or the two hemispheres (Fig. 5a).
  - *structure*, voxels from a selected anatomical group (Fig. 5b).
  - *multiregion*, voxels from a given set of regions (Fig. 5c).
- Select the number of clusters to process the data (*n_clusters* in Line 90).
- Select the feature extraction method (*fe_method* in Line 91; see Box 4 for further details):
  - *pca*, Principal Component Analysis,
  - *ica*, Independent Component Analysis,
  - *nmf*, Non-negative Matrix Factorization,
  - *custom*, Custom feature extractors.
- Select the feature extraction parameter (*fe_params* in Line 92). Feature extraction methods are implemented using the scikit-learn package. See Box 4 for details about feature extraction.
- Select a noise rejection threshold (*noise_th* in Line 93). Voxels for which the temporal standard deviation is higher than the threshold are set to 0. If set to *None*, no voxel is rejected.

16. Select the process to run by commenting all the lines but the one of interest:

- *brainwide* in Line 98. No argument is expected for the *process* method.
- *hemisphere* in Line 100. Set as argument of the function 'L', 'R' or 'LR' for left, right and both hemispheres, respectively.
- *structure* in Line 102. Set as argument of the function a list of tuples *(<structure>, <hemisphere>)*, where *<structure>* refers to the acronym of the structure (see Box 2) and <hemisphere> to 'L' or 'R' for left and right hemisphere, respectively.
- *multi_region* in Line 104. Set as argument a list of tuples in the format (<region>, <hemisphere>), where <region> refers to the acronym of the region (see Box 2) and <hemisphere> to 'L' or 'R' for left and right hemisphere, respectively.

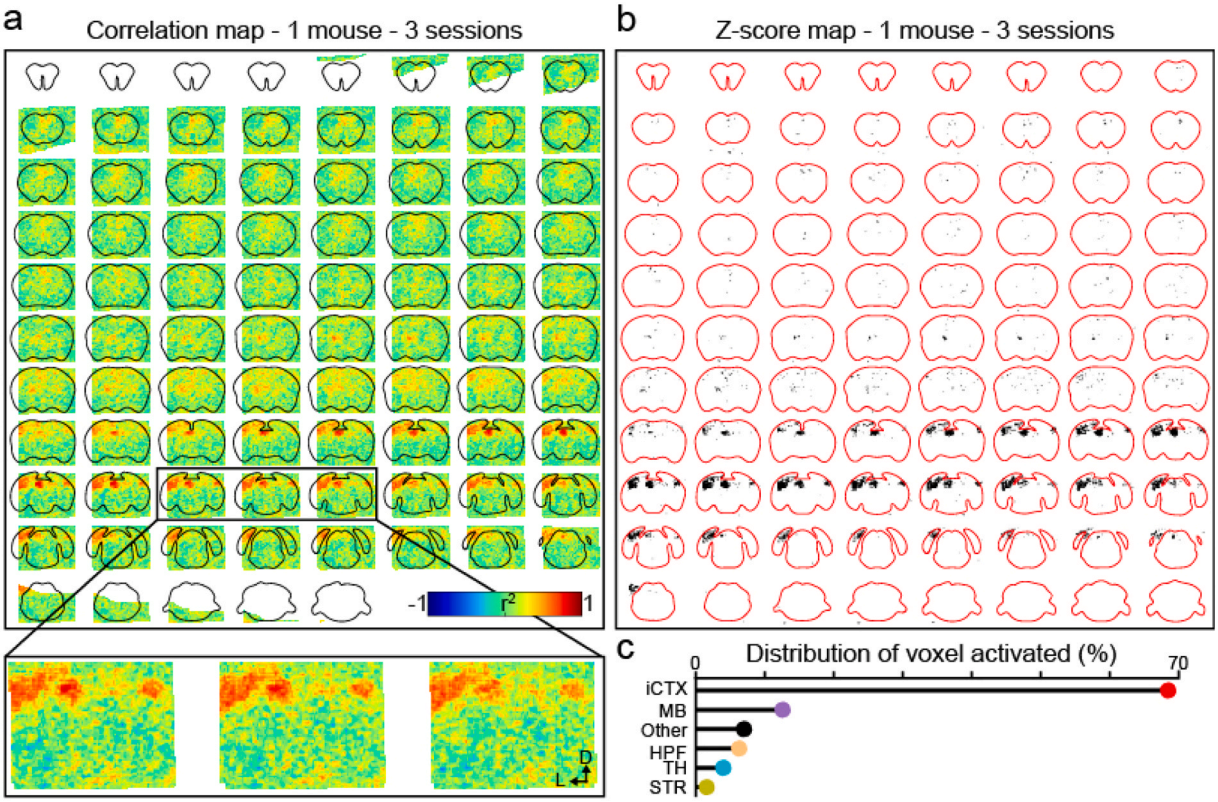
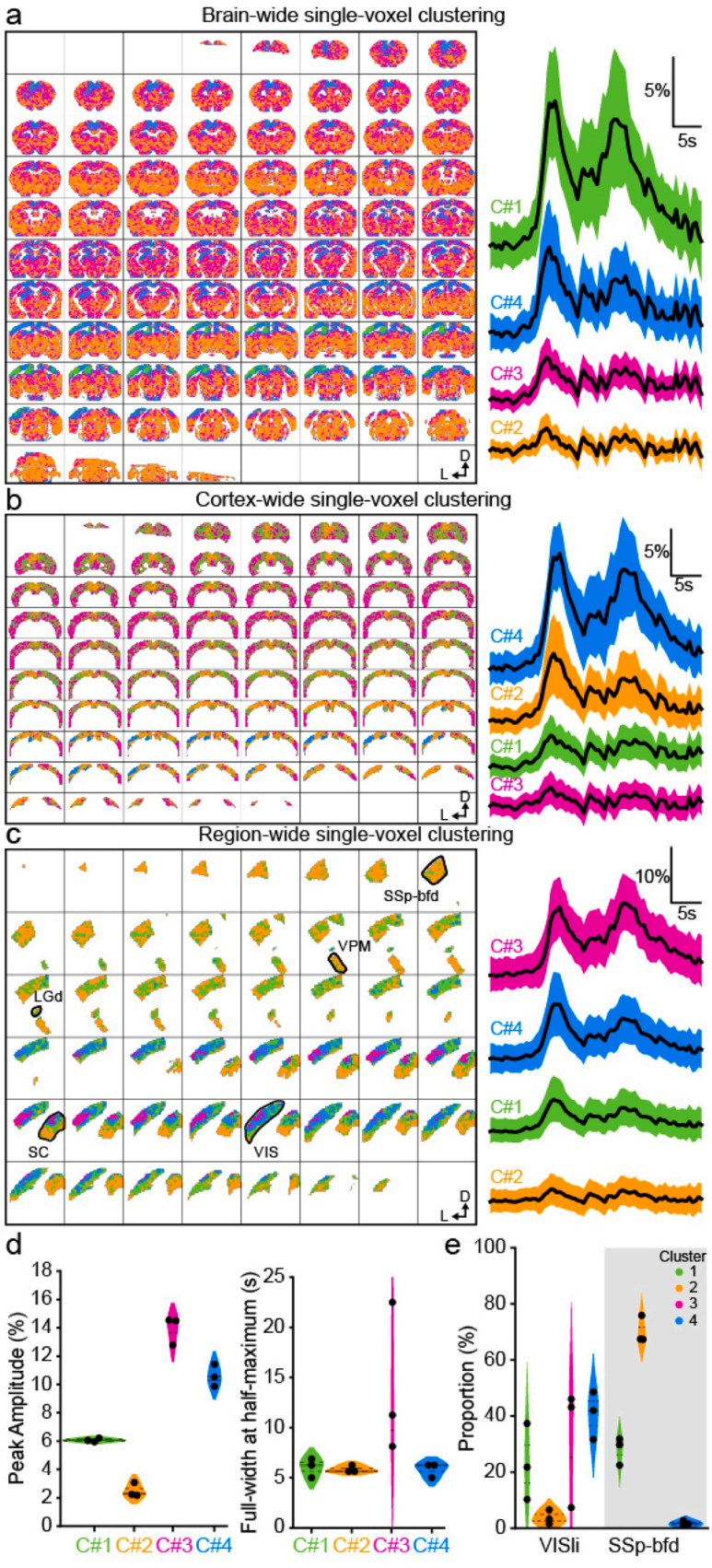


Fig. 4. Visualization and quantification after correlation from Procedure 4. a, Brain-wide color-coded correlation map (r^2) of the average of 3 imaging sessions from a single mouse computed with square shape of the stimulation pattern. In black, the outline of the reference atlas. The bottom panel shows a zoom in of 3 coronal planes. D, dorsal; L, left. b, Z-score map depicting all the voxels activated with a level of significance > 0.001. In red, the outline of the reference atlas. c, Distribution of voxel activated from the z-score map in the brain: isocortex (iCTX), midbrain (MB), hippocampal formation (HPF), thalamus (TH), striatum (STR) and other regions.



(caption on next page)

Fig. 5. Visualization and quantification after single-voxel clustering from Procedure 5. Color-coded single-voxel clustering using 4 clusters (left) and corresponding temporal traces of the voxels allocated to the cluster (mean±sd; right) for the imaged mouse brain (n = 3 mice; a), for the cortex (n = 3 mice; b) and for a set of selected regions of interest (n = 1 mouse; c). D, dorsal; L, left. d, Example of quantification metrics calculated from the temporal traces of the 4 color-coded clusters (C#1 to C#4) of the 3 imaged mice (black dots). *Left*; Peak amplitude as change in signal intensity (ΔI/I in %). *Right;* Full width at half maximum in seconds. The vertical black line denotes the mean and vertical dotted black lines the 25/75 % quartiles. Quantification refers to temporal traces extracted in c. e, Distribution of voxels in the four clusters into the VISli and the SSp-bfd regions of the 3 imaged mice (black dots). The vertical black line denotes the mean and vertical dotted black lines the 25/75 % quartiles. Quantification refers to temporal traces extracted in c.

Box 4
| Clustering hyperparameters.

The single-voxel clustering approach consists of two stages: feature extraction and clustering.

Feature extraction. This stage is key, as the quality of the feature subspace largely determines the quality of the clustering process. In its current version, the PyfUS software offers standard options including Principal Component Analysis (PCA), Independent Component Analysis (ICA) and Non-Negative Matrix Factorization (NMF). Read documentation for PCA, FastICA and NMF in the decomposition module at: https://scikit-learn.org/stable/user_guide.html. The default choice is PCA, which has given satisfactory results throughout our testing. However, these options are limited in terms of representational power, whereas the interesting/discriminative information about fUS signals often lies in their shapes over time. Thus, deep learning approaches may be relevant in this context, especially convolutional autoencoders that can efficiently encode the shape information. In this regard, the software allows the use of user-defined feature extractors. Refer to the documentation of the *FeatureExtractor* object in the *features_extraction* module.

Clustering. The clustering algorithm chosen for single-voxel clustering is K-Means because of its simplicity and computational efficiency. Other clustering methods can be used in the software, but as detailed above, we assume that the feature extraction is of greater consequence than the choice of the algorithm. The main hyperparameter to choose is the number of clusters. Unfortunately, clustering quality metrics under unknown ground truth, such as the silhouette coefficient, are often biased towards convex clusters. Depending on the feature extraction process and the resulting subspace structure, the quantitative and qualitative (expert-based) judgments of the correct number of clusters may differ significantly. Pending a better method, we propose a trial-and-error approach that takes advantage of the speed of clustering (a few seconds).

17. Set a new colormap by changing the variable *custom_cmap* (Line 107). Accepted values are either string corresponding to Matplotlib colormap (e.g., *jet*) or list of color in the RGBA format. Set the variable to *None* to use the default colormap.
18. Select the display mode of the time traces associated with the clusters (Line 110). Set *display* as *all* to plot the temporal traces of all voxels or *mean_std'* to plot the mean and standard deviation instead. Line 112 calls the method for displaying the cluster maps and no argument is required. Check the documentation of these methods for further details (object *SingleVoxelClusteringWrapper*, *clustering* module).
19. Perform the spatial quantification. This process is optional and can be skipped by commenting Lines 115–120:

- The method *get_cluster_maps* of the *SingleVoxelClusteringWrapper* object allows for extracting the cluster maps of target elements in a dictionary format for further processing (Line 116).
- Initialization of the object handling the spatial quantification (Line 119). Mandatory inputs are a dictionary containing spatial maps (variable *res*), the path to the atlas to be used (variable *atlas_path*, line 35) and the associated regions information path (*regions_info_path*, Line 34).
- The regional quantification is called with the method *print_quantification* (Line 120) The required arguments are the acronym of the region of interest and the hemisphere (e.g.: *'VISp', 'L'*). Box 3 and the software documentation contains detailed information on the formatting requirements and the types of input that are accepted: see *SpatialQuantification* object in the Quantification module.

20. Perform the temporal quantification. This process is optional and can be skipped by commenting Lines 123–127.

- The method *get_signals* of the *SingleVoxelClusteringWrapper* object allows for extracting the temporal traces of clusters in a dictionary format for further processing (Line 123).
- Initialization of the object handling the temporal quantification (Line 126). Mandatory input are a dictionary containing temporal traces (variable *res* in Line 123) and a sampling rate value (variable *sampling_rate* in Line 23).
- Call to the method for plotting violin plots of the requested metric. The required parameter is the name of the metric to be displayed (Line 127). Box 3 and the software documentation contains detailed information on the formatting requirements and the types of input that are accepted: see *TemporalQuantification* object in the *Quantification* module.

21. Perform the single-voxel clustering analysis alone by commenting Procedures 3 and 4 (see Step 8) and then running (See Table 1 for Troubleshooting): python example_data_analysis.py

After this stage, the following displays are generated:

- Pop-up window displaying the spatial maps associated with the selected elements.
- Pop-up window displaying the temporal traces associated with each cluster.
- If the spatial quantification was performed, the quantifications will be printed in the terminal. In the column *set_ID*, value 0 correspond to non-clustered voxels and values 1 to *n_clusters* to the cluster identifiers, ordered in the same order as the displays.
- If the temporal quantification was performed, violin plots with the requested metrics are generated. Each dot represents one element (see required parameters in Step 7).

Spatial clustered maps, temporal traces per cluster and spatial

quantification are presented in Figs. 5a to 5e, respectively. Example of raw output are presented in Supplementary Figure 5. Importantly, these figures are not automatically saved, thus use the 'Save' button below the figure to save and store it (see Fig. 3-figure supplement 2).

5. Timings

The timing expressed along the procedure are dependent on the hardware specifications and the volume of data to be processed as detailed in the Equipment section. The provided timing information is based on the example dataset of ~10 Gb and a machine with 64 Gb of RAM equipped with a Intel Xeon E5-2620 v4 CPU (2.10 GHz). The actual timing depends on the resources available, the size of the input dataset and the parameters used for each step.

- Procedure 1: Data loading and pre-processing. Total running time is ~30–60 mins, during which most of the time is spent on Step 4 for averaging and registering the selected data.
- Procedure 2: Data selection - Common to Procedures 3–5. Total running time is less than a minute.
- Procedure 3: Region averaging analysis. Total running time is less than a minute, depending on the selected parameters and data processed.
- Procedure 4: Correlation analysis. Total running time is less than a minute, depending on the selected parameters and data processed.
- Procedure 5: Single-voxel clustering analysis. Total running time is less than a minute, depending on the selected parameters and data processed.

6. Expected results

The work presents a flexible, reliable and open-source software for analyzing brain-activity collected by any fUS system with a full access and control over signal processing and visualization. We provide specific examples of data analysis performed on data displaying evoked brain activity in response to sensory stimuli.

The PyfUS software was designed with flexibility as a core principle. While it was developed for use with volumetric images acquired with a matrix array transducer, it can also be applied to single-plane imaging with a linear transducer. The flexibility of the software also allows it to be used with all mammalian models, even under pathological conditions and without the direct need of digitalized reference atlases, with a wide variety of experimental procedures.

Another key aspect of the software is its ability to provide users with fast and easy access to 3 complementary signal processing strategies: the region-based temporal analysis obtained after registration to a reference atlas (Procedure 3 - Fig. 3), the spatial visualization of brain activity

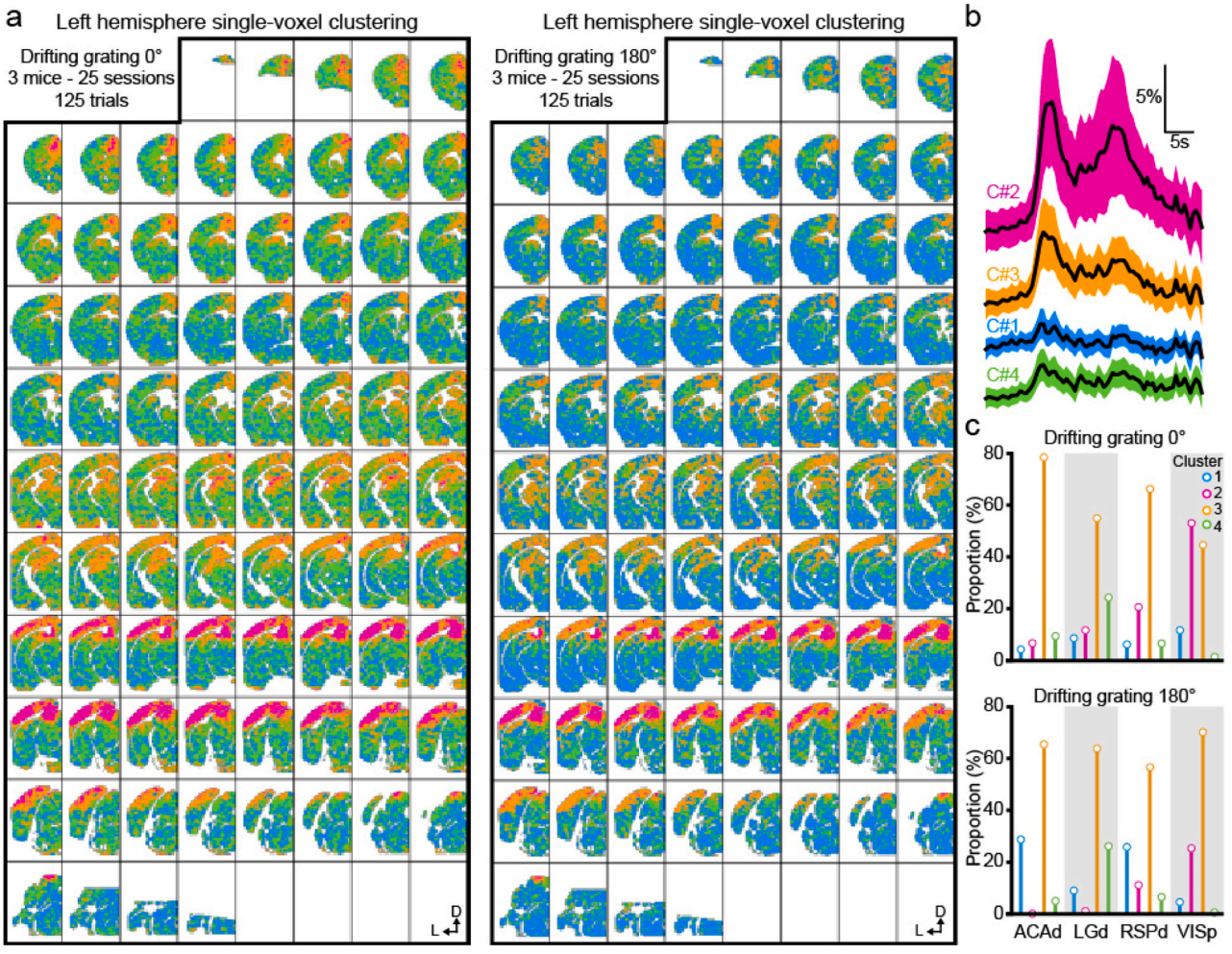


Fig. 6. Expected outcome from different stimuli, a, Comparison of the single-voxel clustering analysis for two drifting grating stimulus with different orientation i.e., 0°(left) and 180°(right) using 4 clusters (n = 3 mice, 25 sessions, 125 trials per condition). D, dorsal; L, left. b**,** Corresponding temporal traces of the voxels allocated to the 4 clusters (mean±sd) for the left hemisphere. **c,** Difference of distribution of the voxels in the 4 clusters into the ACAd, LGd, RSPd and VISp regions (from left to right) based on the orientation i.e., 0°(top) and 180° (bottom).

using the correlation map (Procedure 4 - Fig. 4) and the single-voxel clustering that combines both the spatial and temporal analysis at the smallest scale allowed by fUS measurements (Procedure 5 – Figs. 5 and 6).

While region averaging and correlation analyses can produce interpretable visualizations quickly, they have several limitations. Atlas-based region averaging masks spatial variations in hemodynamic activity, resulting in a mixture of non-active and active voxels, leading to an underestimation of true activity or signals arising from different neural processing, and thus to potential misinterpretation. On the other hand, the outcome of correlation-based analyses is determined by the choice of temporal window, which is biased by expectations on evoked responses and potentially excludes pre-stimulus (e.g., task preparation, stimulus anticipation) or post-stimulus activities (e.g., non-aligned behavior, delayed neuronal response). Furthermore, it does not reflect the temporal dynamics of the fUS signals, since it evaluates how well the signal fits a square pulse without regard for the differences between time-series.

Therefore, there was a need for a new approach that complements region averaging and correlation-based analysis by exploiting both the high spatial and temporal resolution of fUS data in an unbiased manner. We introduced single-voxel clustering to fill this need. The spatio-temporal clustering is a subfield of data mining that has become popular in neuroscience, especially in large-scale functional imaging, due to its exploratory power and broad applicability [15,36,41]. The benefits of this approach include the limited number of assumptions made upon the hemodynamic response properties while avoiding the bias induced by the stimulus pattern. Furthermore, recent research has highlighted that the spread of the hemodynamic activity is of 1–2 voxels at the edge of the active areas, thereby justifying the use of small groups of voxels in the analysis process [21]. Finally, the method allows for a convenient comparison across stimuli along with straighforward quantifications as depicted in Fig. 6.a and b and c respectively. Here, one can directly compare the effect of the orientation of the drifting grating stimulus on the recruitment of some brain regions, and especially integrative ones like the anterior cingulate cortex (ACAd), and the retrosplenial cortex (RSP). Like other analytical approaches, single-voxel clustering has limitations, including sensitivity to the choice of its hyperparameters and the complexity of its readout.

From a more conceptual standpoint, there is no such thing as a perfect analysis method, since every dataset has its own specificity. We therefore encourage users to try multiple analyses and vary hyperparameters to obtain the broadest possible view on the content of their data. For instance, a barcode generated by the region analysis procedure can be used to identify regions requiring further investigation at single voxel scale using the clustering approach, which can also be used to refine the barcode view in large regions such as the caudate putamen.

Regarding potential extensions, providing access to other atlases would be a valuable addition. Indeed, fUS has been applied to multiple animal models, including rats [20,39,40,6], ferrets[23,3], and non-human primates [16,28,37], for which atlases have been created [10,18,19,38]. On another aspect, there is limited research on data preprocessing and no consensus on a standard procedure. This work did not delve deeply into this topic, but the software has been designed to allow the easy implementation and testing of new preprocessing approaches. Finally, as mentioned in the related box, there is an essential work to conduct on the feature extraction for the single-voxel clustering approach. Indeed, if the PCA has provided satisfactory results, resulting subspaces often have a ball-like structure that is not ideal for the clustering. Therefore, it would be beneficial to explore more advanced methods that utilize non-linearities, such as convolutional autoencoders, to extract features from the fUS signals.

In conclusion, this work and software are part of an initiative to promote the adoption of the fUS technology through access to state-of-the-art procedures for data acquisition and analysis. Additionally, we aim to create a community around the development of data analysis techniques. This is made possible by the modular architecture and extensive documentation of the software. In line with this objective, a developer's guide will be made available, with the aim of not only facilitating the use of the software but also encouraging contributions to its ongoing development.

Code availability

PyfUS software can be found and downloaded from the GitHub repository (https://github.com/OpenfUS/PyfUS). The software can be freely used for educational and research purposes.

CRediT authorship contribution statement

Théo Lambert: Formal analysis, Investigation, Validation, Methodology, Conceptualization, Software, Visualization, Writing – original draft. **Gabriel Montaldo:** Conceptualization, Validation, Resources, Writing – original draft, Supervision. **Alan Urban:** Conceptualization, Resources, Project administration, Writing – original draft, Funding acquisition, Supervision, Validation. **Clément Brunner:** Conceptualization, Formal analysis, Validation, Visualization, Software, Writing – original draft, Data curation.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is supported by grants from Fonds Wetenschappelijk Onderzoek (G079623N, G091719N, 1197818N, G0A4F24N, G055124N), ERANET, EU Horizon 2020 (Grant number 964215, UnscrAMBLY), HORIZON-MSCA-2022-DN-01 (Project 101119916 - SOPRANI). Théo Lambert is funded by IMEC PhD Talent grant. Clément Brunner is funded by a FWO Senior Postdoctoral fellowship (12D7523N).

Appendix A. Supporting information

Supplementary data associated with this article can be found in the online version at doi:10.1016/j.neucom.2025.130899.

Data availability

I have shared the link to my data/script in the manuscript file.

References

[1] A.-K. Aydin, W.D. Haselden, Y. Goulam Houssen, C. Pouzat, R.L. Rungta, C. Demené, M. Tanter, P.J. Drew, S. Charpak, D. Boido, Transfer functions linking neural calcium to single voxel functional ultrasound signal, Nat. Commun. 11 (2020) 2954, https://doi.org/10.1038/s41467-020-16774-9.

[2] A. Bertolo, M. Nouhoum, S. Cazzanelli, J. Ferrier, J.-C. Mariani, A. Kliewer, B. Belliard, B.-F. Osmanski, T. Deffieux, S. Pezet, Z. Lenkei, M. Tanter, Whole-brain 3D activation and functional connectivity mapping in mice using transcranial functional ultrasound imaging, J. Vis. Exp. (2021), https://doi.org/10.3791/62267.

[3] C. Bimbard, C. Demene, C. Girard, S. Radtke-Schuller, S. Shamma, M. Tanter, Y. Boubenec, Multi-scale mapping along the auditory hierarchy using high-resolution functional UltraSound in the awake ferret, Elife 7 (2018), https://doi.org/10.7554/elife.35028.

[4] N.F. Binder, M. El Amki, C. Glück, W. Middleham, A.M. Reuss, A. Bertolo, P. Thurner, T. Deffieux, C. Lambride, R. Epp, H.-L. Handelsmann, P. Baumgartner, C. Orset, P. Bethge, Z. Kulcsar, A. Aguzzi, M. Tanter, F. Schmid, D. Vivien, M. T. Wyss, A. Luft, M. Weller, B. Weber, S. Wegener, Leptomeningeal collaterals regulate reperfusion in ischemic stroke and rescue the brain from futile recanalization, Neuron 112 (2024) 1456–1472.e6, https://doi.org/10.1016/j.neuron.2024.01.031.

[5] D. Boido, R.L. Rungta, B.-F. Osmanski, M. Roche, T. Tsurugizawa, D. Le Bihan, L. Ciobanu, S. Charpak, Mesoscopic and microscopic imaging of sensory responses in the same animal, Nat. Commun. 10 (2019) 1110, https://doi.org/10.1038/s41467-019-09082-4.
[6] C. Brunner, N.L. Denis, K. Gertz, M. Grillet, G. Montaldo, M. Endres, A. Urban, Brain-wide continuous functional ultrasound imaging for real-time monitoring of hemodynamics during ischemic stroke, J. Cereb. Blood Flow. Metab. (2023), https://doi.org/10.1177/0271678X231191600.
[7] C. Brunner, M. Grillet, A. Sans-Dublanc, K. Farrow, T. Lambert, E. Macé, G. Montaldo, A. Urban, A platform for brain-wide volumetric functional ultrasound imaging and analysis of circuit dynamics in awake mice, Neuron 108 (2020) 861–875.e7, https://doi.org/10.1016/j.neuron.2020.09.020.
[8] C. Brunner, M. Grillet, A. Urban, B. Roska, G. Montaldo, E. Macé, Whole-brain functional ultrasound imaging in awake head-fixed mice, Nat. Protoc. 16 (2021) 3547–3571, https://doi.org/10.1038/s41596-021-00548-8.
[9] C. Brunner, G. Montaldo, A. Urban, Functional ultrasound imaging of stroke in awake rats, eLife (2023), https://doi.org/10.7554/elife.88919.1.
[10] E. Calabrese, A. Badea, C.L. Coe, G.R. Lubach, Y. Shi, M.A. Styner, G.A. Johnson, A diffusion tensor MRI atlas of the postmortem rhesus macaque brain, Neuroimage 117 (2015) 408–416, https://doi.org/10.1016/j.neuroimage.2015.05.072.
[11] J. Claron, M. Provansal, Q. Salardaine, P. Tissier, A. Dizeux, T. Deffieux, S. Picaud, M. Tanter, F. Arcizet, P. Pouget, Co-variations of cerebral blood volume and single neurons discharge during resting state and visual cognitive tasks in non-human primates, Cell Rep. 42 (2023) 112369, https://doi.org/10.1016/j.celrep.2023.112369.
[12] C. Demené, T. Payen, A. Dizeux, G. Barrois, J.-L. Gennisson, L. Bridal, M. Tanter, 3-D longitudinal imaging of tumor angiogenesis in mice in vivo using ultrafast Doppler tomography, Ultrasound Med. Biol. 45 (2019) 1284–1296, https://doi.org/10.1016/j.ultrasmedbio.2018.12.010.
[13] B.J. Edelman, D. Siegenthaler, P. Wanken, B. Jenkins, B. Schmid, A. Ressle, N. Gogolla, T. Frank, E. Macé, The COMBO window: a chronic cranial implant for multiscale circuit interrogation in mice, PLoS Biol. 22 (2024) e3002664, https://doi.org/10.1371/journal.pbio.3002664.
[14] A. El Hady, D. Takahashi, R. Sun, O. Akinwale, T. Boyd-Meredith, Y. Zhang, A. S. Charles, C.D. Brody, Chronic brain functional ultrasound imaging in freely moving rodents performing cognitive tasks, J. Neurosci. Methods 403 (2024) 110033, https://doi.org/10.1016/j.jneumeth.2023.110033.
[15] J. Gonzalez-Castillo, Z.S. Saad, D.A. Handwerker, S.J. Inati, N. Brenowitz, P. A. Bandettini, Whole-brain, time-locked activation with simple tasks revealed using massive averaging and model-free analysis, Proc. Natl. Acad. Sci. 109 (2012) 5487–5492, https://doi.org/10.1073/pnas.1121049109.
[16] W.S. Griggs, S.L. Norman, T. Deffieux, F. Segura, B.-F. Osmanski, G. Chau, V. Christopoulos, C. Liu, M. Tanter, M.G. Shapiro, R.A. Andersen, Decoding motor plans using a closed-loop ultrasonic brain–machine interface, Nat. Neurosci. (2023), https://doi.org/10.1038/s41593-023-01500-7.
[17] B.-Y. Hsieh, Y.-C.J. Kao, N. Zhou, Y.-P. Lin, Y.-Y. Mei, S.-Y. Chu, D.-C. Wu, Vascular responses of penetrating vessels during cortical spreading depolarization with ultrasound dynamic ultrafast Doppler imaging, Front. Neurosci. 16 (2022) 1015843, https://doi.org/10.3389/fnins.2022.1015843.
[18] E.B. Hutchinson, S.C. Schwerin, K.L. Radomski, N. Sadeghi, J. Jenkins, M. E. Komlosh, M.O. Irfanoglu, S.L. Juliano, C. Pierpaoli, Population based MRI and DTI templates of the adult ferret brain and tools for voxelwise analysis, Neuroimage 152 (2017) 575–589, https://doi.org/10.1016/j.neuroimage.2017.03.009.
[19] H. Kleven, I.E. Bjerke, F. Clascá, H.J. Groenewegen, J.G. Bjaalie, T.B. Leergaard, Waxholm Space atlas of the rat brain: a 3D atlas supporting data analysis and integration, Nat. Methods 20 (2023) 1822–1829, https://doi.org/10.1038/s41592-023-02034-3.
[20] T. Lambert, C. Brunner, D. Kil, R. Wuyts, E. D'Hondt, G. Montaldo, A. Urban, A deep learning classification task for brain navigation in rodents using micro-Doppler ultrasound imaging, Heliyon 10 (2024) e27432, https://doi.org/10.1016/j.heliyon.2024.e27432.
[21] T. Lambert, H.R. Niknejad, D. Kil, C. Brunner, B. Nuttin, G. Montaldo, A. Urban, Functional ultrasound imaging and neuronal activity: how accurate is the spatiotemporal match? bioRxivorg (2024) https://doi.org/10.1101/2024.07.10.602912.
[22] T. Lambert, H.R. Niknejad, D. Kil, G. Montaldo, B. Nuttin, C. Brunner, A. Urban, Spatiotemporal clustering of functional ultrasound signals at the single-voxel level, eNeuro (2025), https://doi.org/10.1523/ENEURO.0438-24.2025.
[23] A. Landemard, C. Bimbard, C. Demené, S. Shamma, S. Norman-Haignere, Y. Boubenec, Distinct higher-order representations of natural sounds in human and ferret auditory cortex, Elife 10 (2021), https://doi.org/10.7554/eLife.65566.
[24] E. Macé, G. Montaldo, I. Cohen, M. Baulac, M. Fink, M. Tanter, Functional ultrasound imaging of the brain, Nat. Methods 8 (2011) 662–664, https://doi.org/10.1038/nmeth.1641.
[25] E. Mace, G. Montaldo, B.-F. Osmanski, I. Cohen, M. Fink, M. Tanter, Functional ultrasound imaging of the brain: theory and basic principles, IEEE Trans. Ultrason Ferroelectr. Freq. Control 60 (2013) 492–506, https://doi.org/10.1109/TUFFC.2013.2592.
[26] É. Macé, G. Montaldo, S. Trenholm, C. Cowan, A. Brignall, A. Urban, B. Roska, Whole-brain functional ultrasound imaging reveals brain modules for visuomotor integration, Neuron 100 (2018) 1241–1251.e7, https://doi.org/10.1016/j.neuron.2018.11.031.
[27] G. Montaldo, A. Urban, E. Macé, Functional ultrasound neuroimaging, Annu. Rev. Neurosci. 45 (2022) 491–513, https://doi.org/10.1146/annurev-neuro-111020-100706.
[28] S.L. Norman, D. Maresca, V.N. Christopoulos, W.S. Griggs, C. Demene, M. Tanter, M.G. Shapiro, R.A. Andersen, Single-trial decoding of movement intentions using functional ultrasound neuroimaging, Neuron 109 (2021) 1554–1566.e4, https://doi.org/10.1016/j.neuron.2021.03.003.
[29] A.O. Nunez-Elizalde, M. Krumin, C.B. Reddy, G. Montaldo, A. Urban, K.D. Harris, M. Carandini, Neural correlates of blood flow measured by ultrasound, Neuron 110 (2022) 1631–1640, https://doi.org/10.1016/j.neuron.2022.02.012, e4.
[30] B.F. Osmanski, C. Martin, G. Montaldo, P. Lanièce, F. Pain, M. Tanter, H. Gurden, Functional ultrasound imaging reveals different odor-evoked patterns of vascular activity in the main olfactory bulb and the anterior piriform cortex, Neuroimage 95 (2014) 176–184, https://doi.org/10.1016/j.neuroimage.2014.03.054.
[31] C. Rabut, M. Correia, V. Finel, S. Pezet, M. Pernot, T. Deffieux, M. Tanter, 4D functional ultrasound imaging of whole-brain activity in rodents, Nat. Methods 16 (2019) 994–997, https://doi.org/10.1038/s41592-019-0572-y.
[32] L. Rahal, M. Thibaut, I. Rivals, J. Claron, Z. Lenkei, J.D. Sitt, M. Tanter, S. Pezet, Ultrafast ultrasound imaging pattern analysis reveals distinctive dynamic brain states and potent sub-network alterations in arthritic animals, Sci. Rep. 10 (2020) 10485, https://doi.org/10.1038/s41598-020-66967-x.
[33] R.L. Rungta, B.-F. Osmanski, D. Boido, M. Tanter, S. Charpak, Light controls cerebral blood flow in naive animals, Nat. Commun. 8 (2017) 14191, https://doi.org/10.1038/ncomms14191.
[34] A. Sans-Dublanc, A. Chrzanowska, K. Reinhard, D. Lemmon, B. Nuttin, T. Lambert, G. Montaldo, A. Urban, K. Farrow, Optogenetic fUSI for brain-wide mapping of neural activity mediating collicular-dependent behaviors, Neuron 109 (2021) 1888–1905.e10, https://doi.org/10.1016/j.neuron.2021.04.008.
[35] L.-A. Sieu, A. Bergel, E. Tiran, T. Deffieux, M. Pernot, J.-L. Gennisson, M. Tanter, I. Cohen, EEG and functional ultrasound imaging in mobile rats, Nat. Methods 12 (2015) 831–834, https://doi.org/10.1038/nmeth.3506.
[36] A. Smolders, F. De Martino, N. Staeren, P. Scheunders, J. Sijbers, R. Goebel, E. Formisano, Dissecting cognitive stages with time-resolved fMRI data: a comparison of fuzzy clustering and independent component analysis, Magn. Reson Imaging 25 (2007) 860–868, https://doi.org/10.1016/j.mri.2007.02.018.
[37] D.Y. Takahashi, A. El Hady, Y.S. Zhang, D.A. Liao, G. Montaldo, A. Urban, A. A. Ghazanfar, Social-vocal brain networks in a non-human primate, bioRxiv (2021), https://doi.org/10.1101/2021.12.01.470701.
[38] X. Tian, Y. Chen, P. Majka, D. Szczupak, Y.S. Perl, C.C.-C. Yen, C. Tong, F. Feng, H. Jiang, D. Glen, G. Deco, M.G.P. Rosa, A.C. Silva, Z. Liang, C. Liu, An integrated resource for functional and structural connectivity of the marmoset brain, Nat. Commun. 13 (2022) 7416, https://doi.org/10.1038/s41467-022-35197-2.
[39] A. Urban, C. Dussaux, G. Martel, C. Brunner, E. Mace, G. Montaldo, Real-time imaging of brain activity in freely moving rats using functional ultrasound, Nat. Methods 12 (2015) 873–878, https://doi.org/10.1038/nmeth.3482.
[40] A. Urban, E. Mace, C. Brunner, M. Heidmann, J. Rossier, G. Montaldo, Chronic assessment of cerebral hemodynamics during rat forepaw electrical stimulation using functional ultrasound imaging, Neuroimage 101 (2014) 138–149, https://doi.org/10.1016/j.neuroimage.2014.06.063.
[41] L. Zhang, M. Guindani, F. Versace, M. Vannucci, A spatio-temporal nonparametric Bayesian variable selection model of fMRI data for clustering correlated time courses, Neuroimage 95 (2014) 162–175, https://doi.org/10.1016/j.neuroimage.2014.03.024.