A library for fMRI real-time processing systems in python (RTPSpy) with comprehensive online noise reduction, fast and accurate anatomical image processing, and online processing simulation

Masaya Misaki[1*], Jerzy Bodurka[1], Martin P Paulus[1]

[1]Laureate Institute for Brain, Tulsa, OK, USA

*** Correspondence:**
Masaya Misaki
mmisaki@laureateinstitute.org

Keywords: real-time fMRI, neurofeedback, online noise reduction, python library, fast segmentation.

Abstract

We introduce a python library for real-time fMRI (rtfMRI) data processing systems, Real-Time Processing System in python (RTPSpy), to provide building blocks for a custom rtfMRI application with extensive and advanced functionalities. RTPSpy is a library package including 1) a fast, comprehensive, and flexible online fMRI denoising pipeline comparable to offline processing, 2) utilities for fast and accurate anatomical image processing to define a target region on-site, 3) a simulation system of online fMRI processing to optimize a pipeline and target signal calculation, 4) interface to an external application for feedback presentation, and 5) a boilerplate graphical user interface (GUI) integrating operations with RTPSpy library. Since online fMRI data processing cannot be equivalent to offline, we discussed the limitations of online analysis and their solutions in the RTPSpy implementation. We developed a fast and accurate anatomical image processing script with fast tissue segmentation (FastSeg), image alignment, and spatial normalization, utilizing the FastSurfer, AFNI, and ANTs. We confirmed that the FastSeg output was comparable with FreeSurfer, and could complete all the anatomical image processing in a few minutes. Thanks to its highly modular architecture, RTPSpy can easily be used for a simulation analysis to optimize a processing pipeline and target signal calculation. We present a sample script for building a real-time processing pipeline and running a simulation using RTPSpy. The library also offers a simple signal exchange mechanism with an external application. An external application can receive a real-time neurofeedback signal from RTPSpy in a background thread with a few lines of script. While the main components of the RTPSpy are the library modules, we also provide a GUI class for easy access to the RTPSpy functions. The boilerplate GUI application provided with the package allows users to develop a customized rtfMRI application with minimum scripting labor. Finally, we discussed the limitations of the package regarding environment-specific implementations. We believe that RTPSpy is an attractive option for developing rtfMRI applications highly optimized for individual purposes. The package is available from GitHub (https://github.com/mamisaki/RTPSpy) with GPL3 license.

1 Introduction

Online evaluation of human brain activity with real-time functional magnetic resonance imaging (rtfMRI) expands the possibility of neuroimaging. Its application has been extended from on-site quality assurance (Cox et al., 1995), brain-computer-interface (BCI) (Goebel et al., 2010), brain self-regulation with neurofeedback (Sulzer et al., 2013), and online optimization in brain stimulation (Mulyana et al., 2021). Nevertheless, a complex system setup specific to an individual environment and noisy online evaluation of neural activation due to a limited real-time fMRI signal processing have hindered the utility of rtfMRI applications and reproducibility of its result (Thibault et al., 2018). Indeed, the significant risk of noise contamination in the neurofeedback signal has been demonstrated in recent studies (Weiss et al., 2020;Misaki and Bodurka, 2021). These issues have been addressed with a community effort releasing easy-to-use rtfMRI frameworks (Cox et al., 1995;Goebel, 2012;Sato et al., 2013;Koush et al., 2017;Heunis et al., 2018;MacInnes et al., 2020) and consensus on reporting detailed online processing and experimental setups (Ros et al., 2020).

As one of the contributions to such an effort, we introduce a software library for rtfMRI; fMRI Real-Time Processing System in python (RTPSpy). This library does not aim to replace the existing rtfMRI software. Instead, the goal of the RTPSpy is to provide building blocks for making a highly customized and advanced rtfMRI system. RTPSpy is a library package including 1) a fast and comprehensive online fMRI denoising pipeline comparable to offline processing (Misaki and Bodurka, 2021), 2) utilities for fast and accurate anatomical image processing to define target regions on-site, 3) a simulation system of online fMRI processing to optimize processing pipeline and target signal calculation, 4) interface to an external application to exchange real-time brain activation signals, and 5) a boilerplate graphical user interface (GUI) application integrating operations with RTPSpy. We believe that the independent processing components provided by RTPSpy can be more useful for developing a custom rtfMRI system than a tightly packed framework.

RTPSpy is a python library in which each process is implemented in independent class modules. The processing pipeline can be created easily by calling a few class methods. The library provides brain anatomical image processing tools for fast and accurate tissue segmentation and anatomical target identification. This enables the quick import of predefined target brain regions on a template into an individual participant's brain space. The library also provides an online processing simulation system. Simulation analysis is vital to building an optimal processing pipeline (Ramot and Gonzalez-Castillo, 2019;Misaki et al., 2020;Misaki and Bodurka, 2021). Our previous report (Misaki and Bodurka, 2021) performed simulations to evaluate the impact of each online processing module on the signal quality using this system. The present report introduces a detailed implementation and usage of the system as well as other utility functions. Although the main components of RTPSpy are the independent processing modules, it also provides a GUI class integrating the RTPSpy modules. We also shared a sample script of neurofeedback presentation using PsychoPy (Peirce, 2008) to demonstrate how the RTPSpy interfaces to an external application. A user can develop a custom neurofeedback application with minimum modification on this example script.

This paper is organized as follows. The next section introduces the methodological details of the online fMRI denoising process in RTPSpy. We discussed the issues and caveats in online fMRI data analysis and how they are addressed in RTPspy. The third section describes fast and accurate anatomical image processing. We developed a new script to perform anatomical image processing quickly. We also evaluated the accuracy of tissue segmentation and the quality of tissue-based noise regressors compared to FreeSuefer segmentation. The fourth and fifth sections illustrate the usage of library methods to build a processing pipeline and run a simulation analysis. The GUI

implementation is shown in the sixth section. The last section discusses the system components that are not provided with RTPSpy but are required for a complete system depending on an individual environment. The RTPSpy can be obtained from GitHub (https://github.com/mamisaki/RTPSpy) with GPL3 license.

2 Implementations of the online fMRI denoising process

RTPSpy includes six online fMRI processing modules; a real-time data loader (RTP_WATCH), slice-timing correction (RTP_TSHIFT), motion correction (RTP_VOLREG), spatial smoothing (RTP_SMOOTH), noise regression with signal scaling (RTP_REGRESS), and an application module (RTP_APP) receiving a processed fMRI volume and extracting a target signal to send it to an external application (Table 1).

These processes can be completed in less than 400 ms for the whole-brain voxels on a current PC with a graphics processing unit (GPU) (Misaki and Bodurka, 2021). Thus, computation time does not limit today's real-time fMRI processing with a decent personal computer. However, the limited number of sample points available online poses a challenge for online processing. Some denoising processes rely on the time series of the signal, but online analysis can only access a limited number of past time points and cannot use future ones. This issue is critical for slice timing correction, signal scaling, and online noise regression. This section describes the methods used in the RTPSpy modules to address this issue. Table 1 summarizes the process in the RTPSpy module and how it differs from the offline process. The class library files for these modules can be found in the 'rtpspy' directory of the package.

RTP_WATCH finds a newly created file in a watching directory in real-time, reads the data, and sends it to the next processing module. The watchdog module in python (https://pythonhosted.org/watchdog/index.html) is used to detect a new file creation. RTP_WATCH uses the NiBabel library (https://nipy.org/nibabel/) to read the file. The currently supported file types are NIfTI, AFNI's BRIK, and Siemens' mosaic dicom. Technically, this module can handle all file types supported by NiBabel, so a user can easily extend the supported files as needed. The data is retained in the NiBabel Nifti1Image format throughout the RTPSpy pipeline. The observer function used in RTP_WATCH (PollingObserverVFS) is system-call independent and can work with various file systems. However, polling may take a long time if many files are in the monitored directory, hindering real-time performance. Thus, it is recommended to clean the watching directory before a session. Such a cleaning function is provided in the module.

Although RTP_WATCH offers a simple interface to read a data file in real-time, how the MRI scanner saves the reconstructed image files varies across the manufacturers and sites. RTPSpy does not provide a universal solution for that. A user may need another package to send data to the watched directory or modify the script file (rtp_watch.py) to adjust for each environment. We will discuss this limitation of the environment-specific issues in the last section.

RTP_TSHIFT performs a slice-timing correction by aligning the signal sampling times in multiple slices to the same one with temporal interpolation. Because we cannot access the future time point in real-time, the online process cannot be equivalent to offline. RTPSpy aligns the sampling time to the earliest slice for avoiding extrapolation. RTPSpy implements two interpolation methods, linear and cubic. The linear interpolation uses only the current and one past time point so that it is equivalent to an offline process. The cubic interpolation uses four time-points; two from the past, the present, and one future. RTPSpy puts a pseudo future point with the same value as the present one to perform the

cubic interpolation. We have confirmed that this pseudo cubic method has a higher correlation with a high-order interpolation method (e.g., FFT) than a linear method (Misaki et al., 2015). By default, RTPSpy uses cubic interpolation.

RTP_VOLREG performs motion correction by registering volumes to a reference one. RTPSpy uses the C implementation in AFNI's 3dvolreg (https://afni.nimh.nih.gov/pub/dist/doc/program_help/3dvolreg.html). The AFNI's functions are compiled into a C shared library file and accessed via ctypes interface in python. The online process is equivalent to the offline one. RTPSpy implements all the interpolation methods for volume resampling used in 3dvolreg. By default, RTPSpy uses cubic interpolation for a faster process. Because volume resampling is the most time-consuming part of online processing (Misaki and Bodurka, 2021), the choice of resampling method is critical for real-time performance.

RTP_SMOOTH performs spatial smoothing by convolving a Gaussian kernel within a masked region. RTPSpy uses the C implementation in AFNI's 3dBlurInMask (https://afni.nimh.nih.gov/pub/dist/doc/program_help/3dBlurInMask.html), compiled into a C shared library file and accessed via ctypes interface in python. This process has no difference between online and offline processing.

RTP_REGRESS performs a signal scaling and noise regression analysis. Since the regression needs enough sample points, at least the number of regressors, the module waits until receiving enough volumes before starting the process. The signal scaling is done with the average signal in this waiting period and converts a signal into percent change relative to the average in each voxel. We note that this scaling is not equivalent to the offline process using an average of all time points in a run so that the absolute amplitude cannot be comparable between the online and offline processing. We also note that the volumes before the start of regression are processed retrospectively so that the saved data includes all volumes.

RTPSpy can include regressors of high-pass filtering (Legendre polynomials), motion parameters (three shifts and three rotations), their temporal derivatives, global signal, mean signals in the white matter and ventricle regions, and cardiac and respiration noise models with RETROICOR (Glover et al., 2000). The motion parameters were received from the RTP_VOLREG module in real-time. The global signal and the mean white matter and ventricle signals are calculated from the unsmoothed data, which is also received from the RTP_VOLREG module. The order of polynomial for high-pass filtering is adjusted according to the data length at each volume with 1 + int(d/150), where d is the scan duration in seconds (the default in AFNI).

RTPSpy uses cumulative GLM, which performs regression with all samples at each time, rather than incremental GLM, which updates only the most recent estimates based on previous estimates (Bagarinao et al., 2003). We have confirmed that cumulative GLM has the advantage of being able to recalculate regressors (such as high-pass filtering and RETROICOR) at each volume, thereby improving the regressors' quality (Misaki and Bodurka, 2021). The regression is done with an ordinary least square (OLS) approach using the PyTorch library (https://pytorch.org/) to allow a seamless switching of CPU and GPU usage according to the system equipment. The residual of regression is obtained as a denoised signal. A user can also include any pre-defined time-series such as a task design as a covariate in the regressors.

The RETROICOR regressors are calculated with another module, RTP_RETROTS. This module implements the same functions as the AFNI's RetroTS.py script, making four cardiac and four

respiration noise basis regressors. While the respiration volume per time (RVT) regressors (Birn et al., 2008) can also be made, we do not recommend using them in online processing. A previous study indicated that the online evaluation of RVT regressors could not be accurate, and its usage could introduce an artifactual signal fluctuation in real-time processing (Misaki and Bodurka, 2021). The RTP_RETROTS should receive cardiac and respiration signals in real-time. The RTP_PHYSIO module (rtp_physio.py) does this recording, but its implementation may depend on the environment of each site and is described in the last section.

RTP_APP receives the processed image and calculates the neurofeedback signal from it. The default implementation extracts the average signal in an ROI mask, defined in the 'do_proc' method of the rtp_app.py file. This method is provided as a prototype and can be customized according to the need for individual applications. Supplementary figure S1 shows an example of a customized method. The RTPSpy noise reduction is performed for the whole-brain voxels, which is advantageous in calculating the feedback signals from multiple regions, such as the functional connectivity and decoding neurofeedback (Watanabe et al., 2017). Misaki and Bodurka (2021) demonstrated the advantage of comprehensive noise reduction in connectivity neurofeedback signals. The calculated signal can be sent to an external application through a socket (Supplementary figures S1 and S2). The RTP_APP class also implements general application control methods, including anatomical image processing described in the next section.

3 Anatomical image processing with fast and accurate tissue segmentation

Anatomical image processing is often required in a rtfMRI session. If the target brain region is defined in the template brain with a group analysis, we need to warp the region mask into the participant's brain. The noise regressions with the white matter and ventricle mean signals also require tissue segmentation masks on an individual brain image. Although there are many tools for brain tissue segmentation using a signal intensity, they are prone to an image bias field and often need a manual correction. Another approach for tissue segmentation uses anatomical information to segment the regions in addition to the signal intensity, such as FreeSurfer (https://freesurfer.net/). FreeSurfer usually offers more accurate and robust segmentation than using only the signal intensity, but its process takes hours or longer to complete, inhibiting its use in a single visit rtfMRI session. Recently, an alternative approach of brain image segmentation using a deep neural network has been released as FastSurfer (Henschel et al., 2020). FastSurfer uses a U-net architecture (Ronneberger et al., 2015) trained to output a segmentation map equivalent to the FreeSurfer's volume segmentation from an input of anatomical MRI image. FastSurfer can complete the segmentation in a few minutes with GPU. We developed a script called FastSeg utilizing the advantage of FastSurfer to extract a brain mask (skull stripping), gray matter, white matter, and ventricle segmentations. FastSeg is implemented as part of the RTPSpy anatomical image processing pipeline and also released as an independent tool (https://github.com/mamisaki/FastSeg).

3.1 FastSeg

FastSeg uses the first stage of the FastSurfer process to make a volume segmentation map (DKTatlas+aseg.mgz). Then, all the segmented voxels are extracted as the brain mask with filling holes. The white matter mask is made with a union of the white matter and corpus callosum segmentations. The ventricle mask is made with the lateral ventricle segmentation. We did not include small ventricle areas because the mask is used only for making a regressor for online fMRI denoising. For removing the partial volume voxels, the white matter mask is eroded two voxels, and the ventricle mask is eroded one voxel.

Although the anatomical segmentation by FastSurfer is not exactly the same as FreeSurfer (Henschel et al., 2020), the effect of the discrepancy on making the noise regressors in functional image resolution could be minor. We confirmed this by comparing the FastSeg segmentations and the noise regressors to those made from FreeSurfer segmentation. The comparison was made for resting-state data of 87 healthy participants (age 18-55 years, 45 females) used in our previous study (Misaki and Bodurka, 2021). Figure 1 shows the dice coefficients of the segmentation masks in anatomical images (Fig. 1, left panel) and correlations of the mean white matter and ventricle fMRI signals (Fig. 1, right panel) between the FastSeg and FreeSurfer. The correlation was nearly 1.0 and higher than 0.98, even at the minimum between the participants. This indicates that the effect of segmentation difference between the FastSeg and FreeSurfer was minor on the mean white matter and ventricle signals in using them as noise regressors for fMRI signals.

3.2 Anatomical image processing pipeline

RTPSpy offers a simple function interface to run a series of anatomical image processing, the 'make_masks' method in RTP_APP class. Figure 2 shows the processing pipeline in this method. The method receives filenames of a reference function image (func_orig), anatomical image (anat_orig), template image (template, optional), and a region of interest (ROI) image in the template space (ROI_template, optional). The process includes the following five steps.

1) Extracting the brain (skull stripping), white matter, and ventricle regions using FastSeg.

2) Aligning the extracted brain image to a reference function image using AFNI align_epi_anat.py.

3) Aligning and resampling the brain mask into the reference function image space using the parameters made in step 2, and making a signal mask of the function image using 3dAutomask in AFNI. The union of these masks is made as a real-time processing mask (RTPmask.nii.gz), used at spatial smoothing in RTP_SMOOTH and defining the processing voxels in RTP_REGRESS. The intersect of these masks is also made as a global signal mask (GSRmask.nii.gz), used in RTP_REGRES.

4) If the template image and the ROI mask on the template are provided, the template brain is warped into the participant's anatomical brain image using the python interface of ANTs registration (https://github.com/ANTsX/ANTsPy). Then, the ROI mask on the template is warped into the participant's brain anatomy image and resampled to the reference function image to make an ROI mask in the functional image space. This mask will be used for neurofeedback signal calculation.

5) Eroding the white matter (two voxels) and ventricle (one voxel) masks, and aligning them to the functional image space using the alignment parameters (affine transformation) estimated at step 2. These masks will be used for the white matter and ventricle average signal regression.

These anatomical image processing could be completed in less than a few minutes in our computational environment (Ubuntu 16.04 LTS with dual Intel Xeon CPUs [Gold 6126, 2.6 GHz, 12 cores for each], 256 GB RAM, and an NVIDIA TITAN V GPU [5120 CUDA cores with 12 GB memory]). We note that the FastSurfer process in FastSeg (step 1) took very long (about an hour) if GPU was not available. Thus, RTPSpy also offers another processing stream that does not use FastSeg. This stream uses AFNI's 3dSkullStrip instead of FastSeg for brain extraction in step 1 and warping the template white matter and ventricle masks into an individual brain in step 4, which are used in step 5.

4 RTPSpy usage

A detailed RTPSpy installation procedure is described on the GitHub site (https://github.com/mamisaki/RTPSpy). The RTPSpy has been developed on a Linux system (Ubuntu) and can also be run on Windows with the Windows Subsystem for Linux (WSL) and Mac OS X, while GPU support on WSL and OS X is limited. Even without GPU, the online noise reduction process can be fast enough (< 600 ms), as shown in Misaki and Bodurka (2021).

4.1 Building a processing pipeline

Figure 3 shows a snippet of an example script to run a real-time processing pipeline. Refer also to the system check script in the package (rtpspy_system_check.py) for a complete script. The pipeline creation consists of making an instance of RTP_APP class (Fig. 3A), calling the 'make_masks' method to make masks for RTP (Fig. 3B), giving processing parameters to the 'RTP_setup' method (Fig. 3E), and calling the 'ready_to_run' method (Fig. 3F). The processing options can be modified by changing rtp_params properties. This example uses dummy physiological signals (Fig. 3C) to simulate the cardiogram and respiration signal recordings. If these signals are not available, set the 'phys_reg' property of 'REGRESS' to 'None' (Fig. 3D). All the properties and possible parameters are described in the script files of each module (see Tab. 1 for the filenames). If the mask files are made by the 'make_masks' method, they are automatically set in the pipeline. The process can be started by calling the 'manual_start' method (Fig. 3G), and then the WATCH module starts watching a new file in the watched directory. The start of the process can also be triggered by a TTL signal implemented in the RTP_SCANONSET module (rtp_scanonset.py file). To close the pipeline, call the 'end_run' method (Fig. 3H), then the WATCH module stops monitoring and the online processed data is saved in a file.

To customize the feedback signal calculation, you can modify the 'do_proc' method in RTP_APP (rtp_app.py). The RTP_APP module works as a pipeline terminal, receiving the processed data, extracting the neurofeedback signal, and sending the signal to an external application. By default, it extracts the mean signal in the ROI mask, but it can be overridden according to the individual application need. An example way to make a customized application class is shown in the Supplementary materials (Supplementary figure S1).

4.2 Example rtfMRI session

We present an example procedure of a real-time fMRI (rtfMRI) session in Figure 4. The rtfMRI session using RTPSpy should start with an anatomical image scan and a reference functional image scan to make the mask images. We usually perform a resting-state scan after an anatomical scan, and an initial image of the resting-state scan acquired in real-time is used as the reference function image. The mask creation using the 'make_masks' method in RTP_APP can be finished during the resting-state scan so that no waiting time is required for a participant. Then, you can set the RTP parameters, run the 'RTP_setup' and 'ready_to_run' methods, and start the neurofeedback scan. A boilerplate application example offers graphical user interfaces for these operations (see Section 6).

A critical RTP property related to the task design is the 'wait_num' in RTP_REGRESS. This property determines how many volumes the module waits before starting the regression. Note that this number does not include the initial volumes discarded before the fMRI signal reaches a steady state. The 'wait_num' must be larger than the number of regressors at least, but the just enough number is not enough because the regression with small samples overfits to data, resulting in very small variance in the denoised output (Misaki et al., 2015). The actual number of required samples

depends on the number of regressors and the target region, and the simulation analysis would help determine the optimal parameter. The task block should start after this burn-in time. We note that RTPSpy processes the volumes during the burn-in time retrospectively so that these volumes can be used, for example, for the baseline calculation to scale the neurofeedback signal amplitude (Zotev et al., 2011;Young et al., 2017).

5 Simulating real-time fMRI processing

One of the most effective ways to examine the integrity of real-time signal calculation is to simulate online processing and neurofeedback signal calculation using previously obtained fMRI data (Ramot and Gonzalez-Castillo, 2019;Misaki et al., 2020;Misaki and Bodurka, 2021). Assuring the integrity of online noise reduction is critical for neurofeedback training. If the noise reduction is insufficient, other factors than brain activation could confound the training effect (Weiss et al., 2020). Not only for the denoising process, but the feedback signal calculation also can be unique in the online analysis, for example, in the connectivity neurofeedback. The online connectivity calculation should use a short window width for a timely feedback signal reflecting the current brain state, and the optimal window width for the neurofeedback training would be specific to the target region and the task design (Misaki et al., 2020). In addition, simulating the signal processing is useful to evaluate the level of the actual feedback signal. For example, when the baseline level of the neurofeedback signal is adjusted by a mean signal in the preceding rest block (Zotev et al., 2011;Young et al., 2017), simulating such signal calculation could help to estimate a possible signal range to adjust a feedback presentation.

The modular library design of the RTPSpy helps perform a simulation with a simple script. The simulation can be done by copying the data volume-by-volume into the watched directory. Also, you can skip the RTP_WATCH module and inject the data directory into the pipeline for a faster process. An example simulation script is provided as example/Simulation/rtpspy_simulation.py file in the package. Figure 5 shows a snippet of the script. The pipeline creation is the same as shown in Fig. 3 except for disabling the RTP_WATCH module (Fig. 5A) and getting the pipeline object returned from the 'ready_to_run' method (Fig. 5B). The simulation can proceed with feeding the Nibabel NiftiImage object to the 'do_proc' method of the pipeline (Fig. 5C). This method receives a volume image, image index, and the end time of the previous process. Calling the 'end_run' method closes the pipeline and returns the saved filenames (Fig. 5D). The output files include the parameter log (text file), ROI signal time-series (csv file), and the denoise image saved as a NIfTI file. You can modify the neurofeedback signal calculation by overriding the do_proc method as explained in Supplementary materials, 'ROI-NF application.'

6 GUI application integrating the RTPSpy modules

RTPSpy offers a graphical user interface (GUI) class (RTP_UI, rtp_ui.py) for easy access to the module functions. The example GUI application is provided in the example/ROI-NF directory in the package. Calling the script 'run_RTPSpyApp.py' in this directory starts a GUI application (Figure 6). This application allows a user to perform the RTPSpy operations with several button presses. The script file ROI-NF.py defines the application class, ROI_NF, inherited from RTP_APP. This example script can be a boilerplate showing how to extract the neurofeedback signal and to send the signal to an external application (Supplementary figure S1). By editing this script, a user can customize the neurofeedback signal calculation.

The example starts an external application, NF_pyspy.py, a PsychoPy application script for neurofeedback presentation. This sample application demonstrates how to get the neurofeedback signal in real-time and present it in the PsychoPy window (Supplementary figure S2). The sample application just shows the received value on the screen, but users should develop a customized presentation according to their needs. The more detailed ways of customization and operations are presented in the Supplementary materials, 'ROI-NF application.' Using these example scripts, a user can develop an easy-to-use and highly customized rtfMRI application with minimum scripting labor. We also provide a full-fledged application of the left-amygdala neurofeedback session (Zotev et al., 2011;Young et al., 2017) in the 'example/LA-NF' directory, which is explained in the Supplementary materials, 'LA-NF application.'

7 Limitations and environment-specific issues

While the RTPSpy provides general-use libraries for rtfMRI data processing, it still is not a complete toolset for all environments. There could be several site-specific settings that a general library cannot support. One of the first critical settings is to obtain a reconstructed MRI image in real-time. The image format, the saved directory, and how to access the directory (e.g., network mount or copying to the processing PC) could differ across manufacturers and sites. The RTP_WATCH detects a new file in the watched directory, but setting up the environment to put an fMRI file to an accessible place in real-time is not covered by the library. Specifically, our site uses AFNI's Dimon command (https://afni.nimh.nih.gov/pub/dist/doc/program_help/Dimon.html) running on the scanner console computer and receives the data sent by Dimon with AFNI's realtime plugin on a rtfMRI operation computer. This may not be possible for all if one cannot install additional software on the scanner console. Users may have to set up real-time access to the reconstructed image according to their environment.

Another caveat of environment-specific implementation is physiological signal recording. One of the significant advantages of the RTPSpy is its ability to run a physiological noise regression with RETROICOR in real-time. However, the equipment for cardiogram and respiration signal recording could vary across the sites and manufacturer. In our site, we measure a cardiogram using a photoplethysmography with an infrared emitter placed under the pad of a participant's finger and respiration using a pneumatic respiration belt. These are equipped with the MRI scanner, GE MR750, and the signal is bypassed to the processing PC via serial port. RTPSpy provides an interface for these signals in the RTP_PHYSIO module. This implementation, however, is highly optimized to our environment, and a user may need to develop a custom script to replace the RTP_PHYSIO module adjusting to the individual environment. Similarly, TTL signal detection in the RTP_SCANONSET module is device-dependent and needs to be implemented by a user according to the user's device.

8 Conclusions

RTPSpy is a library for real-time fMRI processing, including comprehensive online fMRI denoising, fast and accurate anatomical image processing, and a simulation system for optimizing neurofeedback signals. RTPSpy is focused on providing the building blocks to make a highly customized rtfMRI system. It also provides GUI applications wrapped around RTPSpy modules, allowing rapid application development with minimal modification. Although library packages requiring scripting skills may not be easy to use for everyone, we believe that RTPSpy's modular architecture and easy-to-script interface will benefit developers who want to create customized rtfMRI applications. With its rich toolset and highly modular architecture, RTPSpy must be an attractive choice for developing optimized rtfMRI applications.

9 Conflict of Interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

10 Author Contributions

Masaya Misaki: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Writing - original draft, Writing - review & editing. Jerzy Bodurka: Conceptualization, Funding acquisition, Investigation, Methodology, Software, Project administration, Supervision. Martin Paulus: Conceptualization, Funding acquisition, Project administration, Supervision, Writing - review & editing.

11 Funding

This research was supported by the Laureate Institute for Brain Research (LIBR) and the William K Warren Foundation. The funding agencies were not involved in the design of the experiment, data collection and analysis, interpretation of results, and preparation and submission of the manuscript.

12 Data Availability Statement

The RTPSpy package is available from https://github.com/mamisaki/RTPSpy with GPL3 license.

13 References

Bagarinao, E., Matsuo, K., Nakai, T., and Sato, S. (2003). Estimation of general linear model coefficients for real-time application. *Neuroimage* 19**,** 422-429.

Birn, R.M., Smith, M.A., Jones, T.B., and Bandettini, P.A. (2008). The respiration response function: the temporal dynamics of fMRI signal fluctuations related to changes in respiration. *Neuroimage* 40**,** 644-654.

Cox, R.W., Jesmanowicz, A., and Hyde, J.S. (1995). Real-time functional magnetic resonance imaging. *Magn Reson Med* 33**,** 230-236.

Glover, G.H., Li, T.Q., and Ress, D. (2000). Image-based method for retrospective correction of physiological motion effects in fMRI: RETROICOR. *Magn Reson Med* 44**,** 162-167.

Goebel, R. (2012). BrainVoyager--past, present, future. *Neuroimage* 62**,** 748-756.

Goebel, R., Zilverstand, A., and Sorger, B. (2010). Real-time fMRI-based brain-computer interfacing for neurofeedback therapy and compensation of lost motor functions. *Imaging in Medicine* 2**,** 407-415.

Henschel, L., Conjeti, S., Estrada, S., Diers, K., Fischl, B., and Reuter, M. (2020). FastSurfer - A fast and accurate deep learning based neuroimaging pipeline. *Neuroimage* 219**,** 117012.

Heunis, S., Besseling, R., Lamerichs, R., De Louw, A., Breeuwer, M., Aldenkamp, B., and Bergmans, J. (2018). Neu(3)CA-RT: A framework for real-time fMRI analysis. *Psychiatry Res Neuroimaging* 282**,** 90-102.

Koush, Y., Ashburner, J., Prilepin, E., Sladky, R., Zeidman, P., Bibikov, S., Scharnowski, F., Nikonorov, A., and De Ville, D.V. (2017). OpenNFT: An open-source Python/Matlab

414 framework for real-time fMRI neurofeedback training based on activity, connectivity and
415 multivariate pattern analysis. *Neuroimage* 156**,** 489-503.

416 Macinnes, J.J., Adcock, R.A., Stocco, A., Prat, C.S., Rao, R.P.N., and Dickerson, K.C. (2020).
417 Pyneal: Open Source Real-Time fMRI Software. *Front Neurosci* 14**,** 900.

418 Misaki, M., Barzigar, N., Zotev, V., Phillips, R., Cheng, S., and Bodurka, J. (2015). Real-time fMRI
419 processing with physiological noise correction - Comparison with off-line analysis. *J*
420 *Neurosci Methods* 256**,** 117-121.

421 Misaki, M., and Bodurka, J. (2021). The impact of real-time fMRI denoising on online evaluation of
422 brain activity and functional connectivity. *J Neural Eng* 18.

423 Misaki, M., Tsuchiyagaito, A., Al Zoubi, O., Paulus, M., Bodurka, J., and Tulsa, I. (2020).
424 Connectome-wide search for functional connectivity locus associated with pathological
425 rumination as a target for real-time fMRI neurofeedback intervention. *Neuroimage Clin* 26**,**
426 102244.

427 Mulyana, B., Tsuchiyagaito, A., Smith, J., Misaki, M., Kuplicki, R., Soleimani, G., Rashedi, A.,
428 Shereen, D., Bergman, T.O., Cheng, S., Paulus, M., Bodurka, J., and Ekhtiari, H. (2021).
429 Online Closed-Loop Real-Time tES-fMRI for Brain Modulation: Feasibility, Noise/Safety
430 and Pilot Study. *bioRxiv***,** 2021.2004.2010.439268.

431 Peirce, J.W. (2008). Generating Stimuli for Neuroscience Using PsychoPy. *Front Neuroinform* 2**,** 10.

432 Ramot, M., and Gonzalez-Castillo, J. (2019). A framework for offline evaluation and optimization of
433 real-time algorithms for use in neurofeedback, demonstrated on an instantaneous proxy for
434 correlations. *Neuroimage* 188**,** 322-334.

435 Ronneberger, O., Fischer, P., and Brox, T. (2015). "U-Net: Convolutional Networks for Biomedical
436 Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention –*
437 *MICCAI 2015,* eds. N. Navab, J. Hornegger, W.M. Wells & A.F. Frangi. (Cham: Springer
438 International Publishing), 234-241.

439 Ros, T., Enriquez-Geppert, S., Zotev, V., Young, K.D., Wood, G., Whitfield-Gabrieli, S., Wan, F.,
440 Vuilleumier, P., Vialatte, F., Van De Ville, D., Todder, D., Surmeli, T., Sulzer, J.S., Strehl,
441 U., Sterman, M.B., Steiner, N.J., Sorger, B., Soekadar, S.R., Sitaram, R., Sherlin, L.H.,
442 Schonenberg, M., Scharnowski, F., Schabus, M., Rubia, K., Rosa, A., Reiner, M., Pineda,
443 J.A., Paret, C., Ossadtchi, A., Nicholson, A.A., Nan, W., Minguez, J., Micoulaud-Franchi,
444 J.A., Mehler, D.M.A., Luhrs, M., Lubar, J., Lotte, F., Linden, D.E.J., Lewis-Peacock, J.A.,
445 Lebedev, M.A., Lanius, R.A., Kubler, A., Kranczioch, C., Koush, Y., Konicar, L., Kohl, S.H.,
446 Kober, S.E., Klados, M.A., Jeunet, C., Janssen, T.W.P., Huster, R.J., Hoedlmoser, K.,
447 Hirshberg, L.M., Heunis, S., Hendler, T., Hampson, M., Guggisberg, A.G., Guggenberger, R.,
448 Gruzelier, J.H., Gobel, R.W., Gninenko, N., Gharabaghi, A., Frewen, P., Fovet, T.,
449 Fernandez, T., Escolano, C., Ehlis, A.C., Drechsler, R., Christopher Decharms, R., Debener,
450 S., De Ridder, D., Davelaar, E.J., Congedo, M., Cavazza, M., Breteler, M.H.M., Brandeis, D.,
451 Bodurka, J., Birbaumer, N., Bazanova, O.M., Barth, B., Bamidis, P.D., Auer, T., Arns, M.,
452 and Thibault, R.T. (2020). Consensus on the reporting and experimental design of clinical and
453 cognitive-behavioural neurofeedback studies (CRED-nf checklist). *Brain* 143**,** 1674-1685.

454 Sato, J.R., Basilio, R., Paiva, F.F., Garrido, G.J., Bramati, I.E., Bado, P., Tovar-Moll, F., Zahn, R.,
455 and Moll, J. (2013). Real-time fMRI pattern decoding and neurofeedback using FRIEND: an
456 FSL-integrated BCI toolbox. *PLoS One* 8**,** e81658.

457 Sulzer, J., Haller, S., Scharnowski, F., Weiskopf, N., Birbaumer, N., Blefari, M.L., Bruehl, A.B.,
458 Cohen, L.G., Decharms, R.C., Gassert, R., Goebel, R., Herwig, U., Laconte, S., Linden, D.,
459 Luft, A., Seifritz, E., and Sitaram, R. (2013). Real-time fMRI neurofeedback: progress and
460 challenges. *Neuroimage* 76**,** 386-399.

461 Thibault, R.T., Macpherson, A., Lifshitz, M., Roth, R.R., and Raz, A. (2018). Neurofeedback with
462 fMRI: A critical systematic review. *Neuroimage* 172**,** 786-807.

463 Watanabe, T., Sasaki, Y., Shibata, K., and Kawato, M. (2017). Advances in fMRI Real-Time
464 Neurofeedback. *Trends Cogn Sci* 21**,** 997-1010.

465 Weiss, F., Zamoscik, V., Schmidt, S.N.L., Halli, P., Kirsch, P., and Gerchen, M.F. (2020). Just a very
466 expensive breathing training? Risk of respiratory artefacts in functional connectivity-based
467 real-time fMRI neurofeedback. *Neuroimage* 210**,** 116580.

468 Young, K.D., Siegle, G.J., Zotev, V., Phillips, R., Misaki, M., Yuan, H., Drevets, W.C., and Bodurka,
469 J. (2017). Randomized Clinical Trial of Real-Time fMRI Amygdala Neurofeedback for Major
470 Depressive Disorder: Effects on Symptoms and Autobiographical Memory Recall. *Am J*
471 *Psychiatry* 174**,** 748-755.

472 Zotev, V., Krueger, F., Phillips, R., Alvarez, R.P., Simmons, W.K., Bellgowan, P., Drevets, W.C.,
473 and Bodurka, J. (2011). Self-Regulation of Amygdala Activation Using Real-Time fMRI
474 Neurofeedback. *PLoS ONE* 6**,** e24522.

475

Table 1. Summaries of RTPSpy real-time processing modules and their differences from the offline process.

Class module (library file)	Process	Difference from an offline process
RTP_WATCH (rtp_watch.py)	Monitoring a new fMRI volume file creation to read and convert the data as NiBabel Nifti1Image.	N/A
RTP_TSHIFT (rtp_tshift.py)	Slice-timing correction with temporal interpolation and resampling time points.	Cubic interpolation uses a pseudo future point with the same value as the present one.
RTP_VOLREG (rtp_volreg.py)	Motion correction with registration to a reference volume.	None.
RTP_SMOOTH (rtp_smooth.py)	Spatial smoothing by convolving a Gaussian kernel inside a brain mask.	None.
RTP_REGRESS (rtp_regress.py)	Signal scaling and noise regression. Regressors include high-pass filtering, motion parameters and their temporal derivatives, global signal, mean signals in the white matter and ventricle regions, and cardiac and respiration noise models with RETROICOR (Glover et al., 2000).	The regression starts after acquiring enough number of volumes. The signal scaling uses the average signal before starting the regression. High-pass filtering and RETROICOR regressors are updated at each time (Misaki and Bodurka, 2021).
RTP_APP (rtp_app.py)	Calculating the neurofeedback signal from the processed image to send the signal to an external application. General application controls, including anatomical image processing, are also performed.	N/A

N/A: Not applicable to an offline process. The library files can be found in the 'rtpspy' directory of the RTPSpy package.

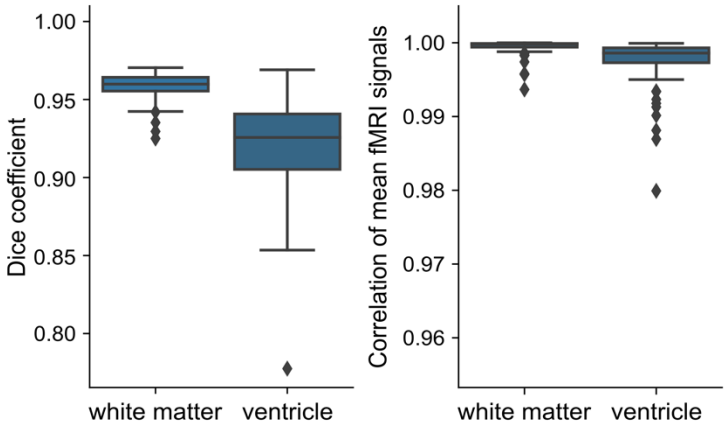

Figure 1. Comparisons between the FastSeg and FreeSurfer segmentation for the white matter and ventricle masks in the anatomical image (left, Dice coefficients) and the mean signals in functional images (right, a correlation between the mean fMRI signals).

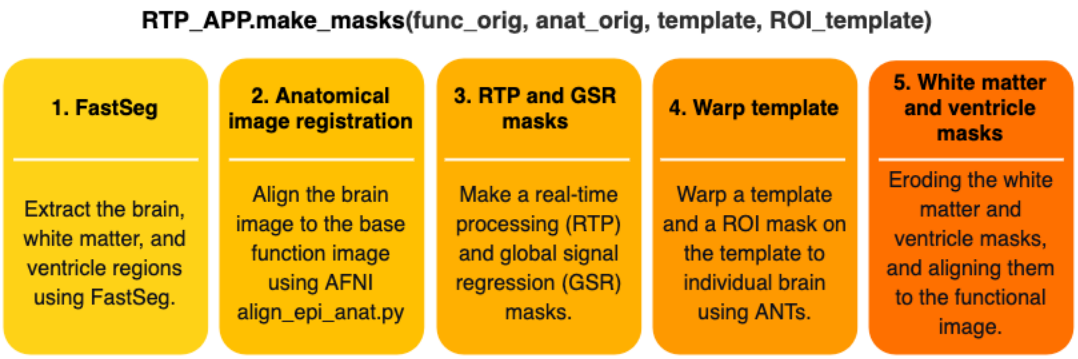

Figure 2. Procedures of creating image masks in the make_mask method of RTP_APP.

```
# --- Create RTP_APP instance ---
rtp_app = RTP_APP(work_dir=work_dir)                                    (A)

# --- Make mask images ----
rtp_app.make_masks(func_orig, anat_orig, template, ROI_template)        (B)

# --- Dummy physio recording for a test ---
rtp_app.rtp_objs['PHYSIO'] = RTP_PHYSIO_DUMMY(                          (C)
    ecg_f, resp_f, sample_freq, rtp_app.rtp_objs['RETROTS']
)

# --- RTP parameters ---
rtp_params = {'WATCH':  {'watch_dir': watch_dir,  'watch_file_pattern': r'nr_\d+.*\.nii'},
              'TSHIFT':  {'slice_timing_from_sample': testdata_f, 'method': 'cubic',
'ignore_init': 3},
              'VOLREG':  {'regmode': 'cubic'},
              'SMOOTH': {'blur_fwhm': 6.0},
              'REGRESS': {'mot_reg': 'mot12', 'GS_reg': True, 'WM_reg': True,
                          'Vent_reg': True,
                          'phys_reg': 'RICOR8',  # set 'None' if no physio is available   (D)
                          'wait_num': 40}}
# --- RTP setup ---
rtp_app.RTP_setup(rtp_params=rtp_params)                                (E)

# --- Ready to run the pipeline ---
rtp_app.ready_to_run()                                                  (F)

# --- Start ---
scan_onset_time = rtp_app.manual_start()                                (G)
…

# -- End ---
rtp_app.end_run()                                                       (H)
```

Figure 3. RTPSpy example script.

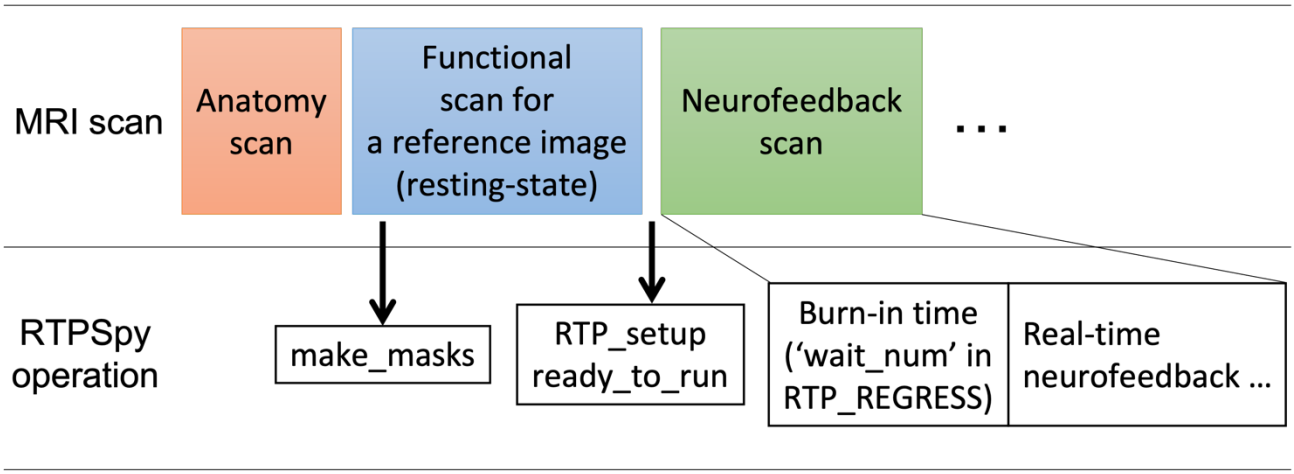

Figure 4. Example real-time fMRI session procedure with RTPSpy.

```
# --- Create RTP_APP instance ---
rtp_app = RTP_APP(work_dir=work_dir)

# --- Make mask images ----
rtp_app.make_masks(func_orig, anat_orig, template, ROI_template)
…

# --- RTP parameters ---
rtp_params = {'WATCH':  {'enabled': False},                                  (A)
              'TSHIFT':  {'slice_timing_from_sample': testdata_f,
                          'method': 'cubic', 'ignore_init': 3},
              'VOLREG':  {'regmode': 'cubic'},
              'SMOOTH': {'blur_fwhm': 6.0},
              'REGRESS': {'mot_reg': 'mot12', 'GS_reg': True, 'WM_reg': True,
                          'Vent_reg': True,
                          'phys_reg': 'RICOR8',  # set 'None' if no physio is available
                          'wait_num': 40}}
# --- RTP setup ---
rtp_app.RTP_setup(rtp_params=rtp_params)

# --- Ready to run the pipeline ---
proc_chain = rtp_app.ready_to_run()                                          (B)

# --- Simulate scan ---
img = nib.load(testdata_f)
fmri_data = np.asanyarray(img.dataobj)
N_vols = img.shape[-1]
for ii in range(N_vols):
   fmri_img = nib.Nifti1Image(fmri_data[:, :, :, ii], affine=img.affine)
   proc_chain.do_proc(fmri_img, ii, time.time())                             (C)
…
saved_fnames = rtp_app.end_run(scan_name='sim_test')                         (D)
```

Figure 5. RTPSpy simulation example script.

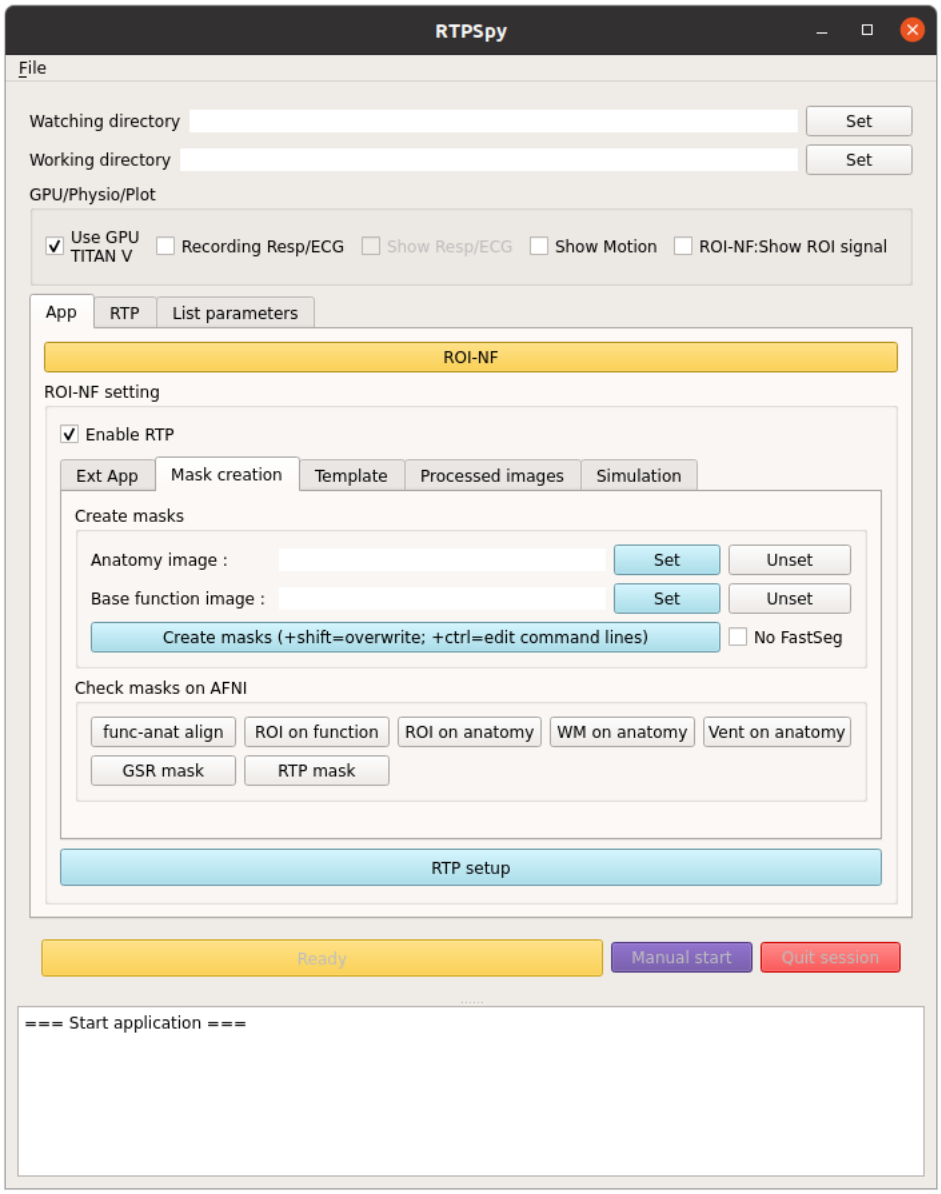

Figure 6. RTPSpy ROI-NF GUI application. The figure presents the 'mask certation' tab to run the make_mask process with GUI. The application also offers graphical interfaces to almost all parameters in RTPSpy (see Supplementary materials).