OPPNI-fMRI User Manual

(Version 10.0, Feb 2020)

If you find omissions, errors, or sections of the manual that need revision, please contact the current developer Mark Prati (mprati@research.baycrest.org)

If you have a scientific question, please contact
Prof. Stephen Strother (sstrother@research.baycrest.org)

Note:

Pay attention to the details provided in paragraphs in red color.

Features marked in blue are beta and/or under development e.g. BIDS support.

1. Introduction

This manual provides instructions for running **OPPNI-fMRI** (**O**ptimization of **P**reprocessing Pipelines for **N**euro**I**maging-**fMRI**), which does fast optimization of preprocessing pipelines for BOLD fMRI (Blood Oxygenation Level Dependent functional MRI). This software package identifies the set of preprocessing steps ("pipeline") specific to each scanning session/run, which optimizes quality metrics of Prediction and Reproducibility for post-processing analysis results (Strother et al., 2002, 2004; see Appendix F for more information about these metrics). This procedure has been shown to significantly improve signal detection, reliability of brain activations, and sensitivity to brain-behaviour correlations (Churchill et al., 2012a, 2012b). The pipeline software can also be used for simple automated batch-processing of fMRI datasets if no appropriate analysis model is currently available to do optimization (e.g. some resting-state connectivity studies).

What can OPPNI do for you?

> Clean up your data: it will automatically output data that has been optimally processed to control for noise and artifact (e.g. due to motion, physiology, scanner noise), which you can then use for further analysis. Automated de-noising is done based on replicable, statistical criteria; no more tedious batch scripting, or hand-selection of ICA components!
>
> Analyze your data: as part of optimization, OPPNI creates Z-scored maps of brain activity for each scanning session/run. You can directly report these results, or take individual activation maps and do standard group-level analysis. OPPNI can perform univariate or multivariate analysis of brain activity. This can be done for a variety of different task designs, including event-related and block-design tasks. Recent additions also include seed-based connectivity and component modelling, i.e., selecting an optimal PCA subspace.
>
> Run batched pipelines: if you cannot analyze your data (or OPPNI does not have the appropriate analysis tools), you can still process it using OPPNI without doing optimization. Our scripts make it straightforward to choose the pipeline steps that you want, and perform automatic batch processing of large datasets.

Preprocessing and analysis scripts are coded in Matlab, and functions are called and managed using Python scripts. The code tests all possible combinations of different preprocessing steps, to identify the optimal pipeline for each fMRI dataset that is being tested. Current preprocessing options include AFNI utilities (Analysis of Functional NeuroImaging; https://afni.nimh.nih.gov/afni), along with a set of functions developed in-house. All steps are widely used in the fMRI literature, or demonstrated to be important in prior studies of pipeline optimization (e.g. Tegeler et al., 1999; La Conte et al., 2003; Shaw et al., 2003; Strother et al., 2004; Zhang et al., 2009; Churchill et al., 2012a, 2012b).

Pipeline optimization is analysis-driven: it evaluates the quality of analysis results for each pipeline, via Prediction and Reproducibility metrics, and selects the pipeline that gives highest-quality outputs. Analysis techniques are "modular" – you choose the task design and analysis model you wish to optimize, from a list of available models. The pipeline software also includes procedures for automated spatial normalization of subjects to an anatomical template, via FSL utilities (https://fsl.fmrib.ox.ac.uk/fsl/fslwiki). This enables users to run group-level analysis of preprocessed data across subjects and task runs.

2. Overview of OPPNI Framework

The pipelines are run from a single tool: **oppni**

Which manages a series of calls to a set of Matlab (Octave) functions, along with pre-compiled AFNI and FSL functions.

This pipeline automatically tests a set of different preprocessing algorithms in a fixed order, and the effects of including or excluding these steps during preprocessing of fMRI data. The steps are listed in the table below, in the order in which they are performed. Note that multiple steps require the AFNI software package (afni.nimh.nih.gov/afni); spatial normalization of brain volumes also requires the FSL software package (fsl.fmrib.ox.ac.uk/fsl); if you are running pipelines locally, make sure the packages are in your path. Both packages are freely available online; refer to **Appendix B** for installation tips.

Table 1. List of currently available pipeline steps

PIPELINE STEP	DESCRIPTION	SOFTWARE	OPTIONS
Estimating minimum displacement volume	Initial coarse masking of brain volumes, converts 4D fMRI data into spatiotemporal matrix, then identifies volume closes to medioid in PC-space coordinates	AFNI's 3dAutomask, 3dmerge and Matlab script	ON
Motion correction (**MOTCOR**)	Performs rigid-body alignment of individual fMRI volumes to a "reference" volume, in order to minimize confounds due to head motion[1].	AFNI's 3dvolreg	OFF/ON
Brain masking	Construct mask to segment brain from non-brain tissue, with initial spatial smoothing 6 mm FWHM to improve mask edges	AFNI's 3dmerge and 3dAutomask	ON
Censoring outlier volumes; also does more aggressive component-based filtering (**CENSOR**)	<mask description needed> Basic censoring: identifies significant outlier volumes in fMRI time series, which are discarded and replaced with interpolated values from neighbouring volumes. Aggressive censoring: uses PCA[2] (default) or ICA[2] to remove components with significant temporal spiking, or significant edge artefact. **[*Note: "aggressive" censoring has not been extensively validated. Use with extreme caution.*]**	Matlab script developed in-house (Campbell et al.,2013)[3]	OFF / BASIC / AGGRESSIVE-PCA or AGGRESIVE-ICA
Physio. correction; external measures (**RETROICOR**)	Performs retrospective correction of physiological noise, by regressing out BOLD signal that is correlated with cardiac and respiratory phase data, which ***must be acquired during the scanning session.***	AFNI's 3dretroicor	OFF/ON
Slice-Timing Correction (**TIMECOR**)	Performs temporal interpolation of fMRI voxel values, so that values at all brain slices correspond to the same time-point. Corrects for echo-time (TE) delays between axial slices in standard EPI BOLD. **[Note: the order (e.g, ascending interleaved) must be verified and entered explicitly. We do not trust this information extracted from DICOM headers]**	AFNI's 3dTshift	OFF/ON

Spatial Smoothing (**SMOOTH**)	Decreases spatial noise variance by convolving the fMRI brain volume with a 3D isotropic Gaussian kernel. This step results in some signal bias (decreased resolution)	AFNI's 3dmerge	Full width at half max (FWHM) of the kernel can be varied
Subject-specific non-neuronal tissue mask	Generates a data-driven, subject-specific mask of predominantly nonneuronal tissue voxels (vasculature, sinuses and ventricles), which should be excluded from subsequent processing. Otherwise, these voxels produce false-positive activations, and biased estimates of spatial reproducibility. * NB: step cannot be optimized directly but can be manually turned on/off in command line	Matlab script developed in-house (Churchill and Strother, 2013)	OFF/ON
Temporal Detrending (**DETREND**)	Corrects for low-frequency noise effects, by fitting Legendre polynomials of order [0…*k*] at each voxel, and regressing them out of the timeseries	Matlab script developed in-house	Maximum polynomial order *k* can be varied
Motion Parameter Regression (**MOTREG**)	Performs PCA decomposition on the head movement parameters from MOTCOR, and regresses out PCs that account for >85% of motion parameter variance. This corrects for residual motion artifact.	Matlab script developed in-house	OFF/ON
Task Covariate regression (**TASK**)	Includes task design as a covariate when regressing out DETREND, MOTREG and GSPC1 (see below). This step protects against over-regression of task-related BOLD	Matlab script developed in-house	OFF/ON
Global Signal Removed Using PC#1 (**GSPC1**)	Estimates and regresses out the 1st PC of the fMRI data. It is used to control for global signal modulations, which predominantly appear in PC#1 (Carbonell et al., 2011).	Matlab script developed in-house	OFF/ON[4]
Physio. correction; data-driven (**PHYPLUS**)	With subject-specific, non-neuronal tissue masking already performed uses the data-driven PHYCAA+ algorithm to estimate and remove physiological noise components. Uses a multivariate component model (similar to PCA or ICA) to identify noise with a strong vascular component.	Matlab script developed in-house (Churchill and Strother, 2013)	OFF/ON
Regress out mean time-series in a user defined mask. (**CUSTOMREG**)	Removes the mean time-series of a set of areas specified by a NIFTI mask file . It can be used to remove the white matter mean time-series, i.e. suggested by many studies.	Matlab script developed in-house	OFF/ON[4]
Low-pass filtering (**LOWPASS**)	Uses a linear filter to suppress BOLD frequencies above 0.10 Hz, which contain a relatively large amount of physiological noise power[5].	Matlab script developed in-house	OFF / ON

[1] The reference volume for alignment is automatically chosen by identifying the volume with estimated minimum overall head displacement in the scanning run, defined as the volume with minimum Euclidean distance from the median coordinates in Principal Component (PCA) space of the 4D fMRI data set.
[2] Full pipeline optimization including PCA censoring is quite slow, even using high performance computing resources with a scheduler; using ICA instead of PCA is even slower. See section 3(ii) for details on how to specify censoring options.

[3] Only includes description of basic despiking in this paper (CENSOR=1).

[4] If you have both GSPC1 and CUSTOMREG set to ON or OFF the IND optimization results produce 4 GSPC1 output values of 0-3 in the QC outputs. This occurs because they code for the 4 possible combinations of GSPC1 ON or OFF and CUSTOMREG ON or OFF. 0 is both OFF, 1 is GSPC1=ON and CUSTOMREG=OFF, 2 is GSPC1=OFF and CUSTOMREG=ON, and 3 is both ON.

[5] This is primarily used for resting-state data, to minimize potential physio. noise (as functional connectivity measures may be more sensitive than task-based analysis). We recommend applying with caution as it is a very conservative step, and there is some evidence of BOLD response power >0.1 Hz that may be relevant to cognitive function.

This software package allows users to test all possible combinations of choices for 12 of these 13 pipeline steps. For every tested pipeline, the dataset of interest is preprocessed and analyzed (using your chosen analysis model and task design), which produces performance metrics of (Prediction, Reproducibility). The script then compares across all tested pipelines, and identifies the pipeline that optimizes these performance metrics (see Appendix F for further details on these metrics).

Pipeline Optimization Approaches

We provide results for three different pipeline optimization approaches, ordered by increasing model flexibility:

Standard, conservative pipeline (CON): this approach applies a single "conservative" set of preprocessing steps to each fMRI dataset, designed to minimize potential noise confounds without concern for signal optimization; this is a (somewhat enhanced) version of standard processing options in fMRI literature, included for comparison purposes. The steps included are: motion correction with optimal reference volume selection, basic outlier censoring, RETROICOR if external physiological measures are provided, slice-timing correction, spatial smoothing (if multiple kernels are tested, it selects the one closest to 6 mm FWHM), subject-specific non-neuronal tissue masking, motion parameter regression, and linear detrending (if multiple orders are tested, it selects the one closest to AFNI's heuristic criterion; afni.nimh.nih.gov/pub/dist/doc/ program_help/3dDeconvolve.html).

Fixed optimization (FIX): selects the single fixed set of pipeline steps, applied across all subject's session/run datasets, that gives highest average (Prediction, Reproducibility).

Individual optimization (IND): selects the combination of pipelines' steps that optimizes (Prediction, Reproducibility) specific to each subject's session/run dataset. In cases where there are multiple task runs or contrasts of interest, the user may choose the pipeline that gives best average performance across runs and/or contrasts (this is explained in further detail in Section 5 below).

3. Running Pipeline Optimization

This section lists requirements for running OPPNI. This includes fMRI data, optional T1 anatomical and physiological data, and some user-supplied text files, which specify the data and analysis of interest. We also provide tutorial instructions of how to create input files and submit jobs to OPPNI, with examples.

i. Requirements

1. 4D fMRI dataset, NIFTI format (i.e. with a '.nii' extension)
2. T1 anatomical volume, NIFTI format

   (this is optional –only required if you want to normalize all data to a common template space)
3. Cardiac and Respiratory data, converted into .1D design files; must be formatted as {name*}.puls.1D and {name}.resp.1D file extensions (respectively).

   (this is optional –only required if you want to run RETROICOR step in pipelines)
4. A set of formatted textfiles, specifying (a) the name and location of your data, (b) pipeline steps you want to test, and (c) task information, condition onsets and duration:

   a. File #1: {subject_input}.txt
   b. File #2: {pipeline_list}.txt
   c. File(s) #3: {task_information}.txt

   If you do not have a BIDS compliant dataset, you will have to write these yourself, but they have a simple format with step-by-step instructions below (Sections **3.ii** and **3.iii**). Once written, you can also re-use them for different analyses and pipeline choices.

*Curly brackets {} denote variable names that must be specified by the user

ii. Creating Pipeline Input Text Files

File #1: {subject_input}.txt

If you are providing a reference to a BIDS compliant data set via the --bids_dir parameter, the subject_input file will be automatically created as an intermediate input file for OPPNI during processing.

If your data is not BIDS compliant you must manually create the subject_input file for OPPNI as described here:

Each line in this text file corresponds to a different fMRI dataset being processed. For example, if there are 10 subjects with 2 runs each, there will be twenty lines in the text file, one/run to be processed. The fields in each line are separated by spaces and described below. ***Note that fields 1-4 are mandatory***. The rest depends on if you want to test specific options.

1. **IN**={ (path)/(input nifti file) }
   a. Name and location of unprocessed fMRI data you wish to optimize
2. **OUT**={ (path)/(output prefix) }*
   a. Name and location for final processed & optimized outputs
   b. This output prefix must be unique among all the subjects in the given input file. It is usually the file name of the input nifti file specified in IN step without the nii extension, but it doesn't have to be.
   c. In order to assist the users in organizing their processing for commonly used variations of processing in a given dataset, OPPNI would automatically add an

additional level of hierarchy based on the name of the input file (say young-TMT-20subjects.txt), analysis model (say LDA) and the contrast chosen (say task_A-baseline). For example, for a run denoted by the prefix sub1_run1, if OUT=/disk/fMRI/my-results/sub1_run1, OPPNI would interface with the codebase to place the results in /disk/fMRI/my-results/young-TMT-20subjects-LDA-task_A-baseline/sub1_run1 . This frees the user from the burden of having to manually reorganize files for different versions of preprocessing.

d. If you use the OPPNI -o folder_prefix command line option the (path) should contain a leading "/" for final output folder to resolve to: folder_prefix/(path). If the "/" is omitted it will resolve to: folder_prefix/(absolute_path)

*** WARNING: Care should be taken to ensure that final path names will not exceed the maximum 256 characters. This includes output paths that are used for intermediate processing steps.**

3. **DROP**=[{# scans to drop at start},{# scans to drop at end}]
   a. Discards scan volumes at the start/end of the run, which may be non-equilibrium or contain task instructions

   ex. DROP=[2,2] discards the first two volumes and last two volumes of the time series

   ex. DROP=[0,0] does not discard any volumes

4. **TASK**={(path)/(task information)}
   a. Name and location of formatted text file, describing the experimental paradigm for this dataset, including task onsets and duration. You must create this file yourself, with step-by-step instructions below.
   b. In the {task information file}, onsets are 1-relative (i.e. first volume=1), and indexed relative to the first scan volume before any volumes are dropped.
   *** If datasets have the same task paradigm (e.g. fixed onsets and durations), you can re-use the same (task information) file for all of these datasets.**

5. **PHYSIO**={ (path)/(physiological data prefix) }
   a. Name and location of Cardiac and Respiratory data, converted into .1D design files. must be formatted as (path)/(physiological data prefix).puls.1D and (path)/(physiological data prefix).resp.1D, respectively

   *** PHYSIO** field can be omitted if RETROICOR is not being tested in pipeline

   *** If RETROICOR is being tested and some subjects are missing physio files, those subjects must be processed in a separate '{subject input}.txt' file**

6. **STRUCT**={ (path)/(T1 anatomical nifti file) }
   a. Name and location of subject's 3D structural brain image, which is used to co-register data to a common template space, if desired. Typically subject's T1 MRI but can be an EPI

   *** STRUCT** field can be omitted if you do not plan to warp data to a common template

7. **CUSTOMREG**={(path)/(nifti file)}

a. Name and location of binary mask, where 1=brain regions containing noise confounds (e.g. white matter). The mean time-series of this ROI is computed and regressed out of all voxels in the brain by the CUSTOMREG pipeline step.

   * This volume must have the same dimensions as the user's **STRUCT** file as the mask is assumed to live in the subject's T1 space. If the EPI to T1 subject transformations exist it will use the inverse transforms to put the mask on the EPIs. If the inverse transforms don't exist it estimates them using FSL FLIRT and uses these. If you want to perform CUSTOMREG, you will need to create the binary ROI mask yourself, e.g. using one of the neuroimaging packages that allows you to manually edit images (e.g. fslview, mricron, afni). For advanced users only!

   * **CUSTOMREG** field can be omitted if you do not want to perform this step.

Tips for {subject input}.txt file:

- For **IN, OUT, PHYSIO,** list the directory path/filename for the data being preprocessed
  - Ex. IN=mydirectory/subject1_directory/subject1_data.nii
- Each field is separated with a space in the line. **All** mandatory fields must be included in every line (i.e. for every dataset being optimized). Optional fields should be consistently included/excluded from all lines in the textfile.
- You choose how you want to analyze your data (e.g. task onsets and analysis model). See section ***2(iii)*** below for more information on task design and analysis models

Steps for {subject_input}.txt file to use CBRAIN:

1. Make sure all of you paths have a common directory

   a. Ex. IN=/**mydirectory**/input_session/subject1_data.nii
   b. Ex. OUT=/**mydirectory**/outputs/subject1_data.nii
   c. Ex. STRUCT=/**mydirectory**/struct_files/subject1_data.nii … and so on

2. Replace your common director with /CBRAIN_PROCESSING_LOCATION.

   a. Ex. IN=**/CBRAIN_PROCESSING_LOCATION**/input_session/subject1_data.nii
   b. Ex. OUT=**/CBRAIN_PROCESSING_LOCATION**/outputs/subject1_data.nii
   c. Ex. STRUCT=**/CBRAIN_PROCESSING_LOCATION**/struct_files/subject1_data.nii …

   The OPPNI CBRAIN task will adjust these paths for you depending on the HPC system and location it selected to run you data on.

3. Rename your file to be *"**user_input_file.txt**"*, this is a special name that cbrain will be able to find, adjust and rename to "user_input_file_cbrain.txt".

Example Text for File#1:

For 1 run from each of 5 different subject datasets, you would produce a textfile with the 5 lines below. Note that "CUSTOMREG" field is omitted, meaning that this pipeline step will be fixed OFF.

```
IN=my_dir/subj1_task.nii OUT=new_dir/subj1 DROP=[2,1] TASK=taskInfo_subj1.txt PHYSIO=my_dir/physio/subj1 STRUCT=my_dir/subj1_T1.nii
IN=my_dir/subj2_task.nii OUT=new_dir/subj2 DROP=[2,1] TASK=taskInfo_subj2.txt PHYSIO=my_dir/physio/subj2 STRUCT=my_dir/subj2_T1.nii
IN=my_dir/subj3_task.nii OUT=new_dir/subj3 DROP=[2,1] TASK=taskInfo_subj3.txt PHYSIO=my_dir/physio/subj3 STRUCT=my_dir/subj3_T1.nii
IN=my_dir/subj4_task.nii OUT=new_dir/subj4 DROP=[2,1] TASK=taskInfo_subj4.txt PHYSIO=my_dir/physio/subj4 STRUCT=my_dir/subj4_T1.nii
IN=my_dir/subj5_task.nii OUT=new_dir/subj5 DROP=[2,1] TASK=taskInfo_subj5.txt PHYSIO=my_dir/physio/subj5 STRUCT=my_dir/subj5_T1.nii
```

Example Text for File#1:

For 1 run from each of 5 different subject datasets, you would produce a textfile with the 5 lines below. Note that "CUSTOMREG" field is omitted, meaning that this pipeline step will be fixed OFF. This file will not run normally since /CBRAIN_PROCESSING_LOCATION does not exist, but this placeholder will be replaced by CBRAIN to a viable path when the task is run.

```
IN=/CBRAIN_PROCESSING_LOCATION/subj1_task.nii OUT=/CBRAIN_PROCESSING_LOCATION/subj1 DROP=[2,1] TASK=taskInfo_subj1.txt
    PHYSIO=/CBRAIN_PROCESSING_LOCATION/physio/subj1 STRUCT=/CBRAIN_PROCESSING_LOCATION/subj1_T1.nii
IN=/CBRAIN_PROCESSING_LOCATION/subj2_task.nii OUT=/CBRAIN_PROCESSING_LOCATION/subj2 DROP=[2,1] TASK=taskInfo_subj2.txt
    PHYSIO=/CBRAIN_PROCESSING_LOCATION/physio/subj2 STRUCT=/CBRAIN_PROCESSING_LOCATION/subj2_T1.nii
IN=/CBRAIN_PROCESSING_LOCATION/subj3_task.nii OUT=/CBRAIN_PROCESSING_LOCATION/subj3 DROP=[2,1] TASK=taskInfo_subj3.txt
    PHYSIO=/CBRAIN_PROCESSING_LOCATION/physio/subj3 STRUCT=/CBRAIN_PROCESSING_LOCATION/subj3_T1.nii
IN=/CBRAIN_PROCESSING_LOCATION/subj4_task.nii OUT=/CBRAIN_PROCESSING_LOCATION/subj4 DROP=[2,1] TASK=taskInfo_subj4.txt
    PHYSIO=/CBRAIN_PROCESSING_LOCATION/physio/subj4 STRUCT=/CBRAIN_PROCESSING_LOCATION/subj4_T1.nii
IN=/CBRAIN_PROCESSING_LOCATION/subj5_task.nii OUT=/CBRAIN_PROCESSING_LOCATION/subj5 DROP=[2,1] TASK=taskInfo_subj5.txt
    PHYSIO=/CBRAIN_PROCESSING_LOCATION/physio/subj5 STRUCT=/CBRAIN_PROCESSING_LOCATION/subj5_T1.nii
```

File #2: {pipeline_list}.txt

This is a text file listing the choices for each of the 12 pipeline steps that you want to test during pipeline optimization. Individual fields are formatted as {pipeline step}=[{options}], where {options} is a comma-separated list of choices that you are testing.

Table 2. List of pipeline input options

Entry in File	Pipeline Step	Options
MOTCOR=[{options}]	Rigid-body motion correction	"0","1" or "0,1" (0=OFF, 1=ON)
CENSOR=[{options}]	Censoring outlier spikes in data	Comma-separated list of 0,1,2,3 options (0=OFF, 1=BASIC, 2=AGGRESSIVE PCA, 3=AGGRESSIVE ICA[1])
RETROICOR=[{options}]	Physiological noise correction #1	"0","1" or "0,1" (0=OFF, 1=ON)
TIMECOR=[{options}]	Slice-timing correction	"0","1" or "0,1" (0=OFF, 1=ON)
SMOOTH = [{options}]	Gaussian spatial smoothing	Comma-separated list of FWHM smoothing kernels (mm), e.g. "0,6,8"
DETREND=[{options}]	Temporal detrending	Comma-separated list of Legendre polynomial detrending order, e.g. "1,2,3"
MOTREG=[{options}]	Motion parameter regression	"0","1" or "0,1" (0=OFF, 1=ON)
TASK=[{options}]	Include task design covariate	"0","1" or "0,1" (0=OFF, 1=ON)
GSPC1=[{options}]	Global signal corrected with PC#1	"0","1" or "0,1" (0=OFF, 1=ON)
PHYPLUS=[{options}]	Physiological noise correction #2	"0","1" or "0,1" (0=OFF, 1=ON)
CUSTOMREG=[{options}]	Regression of noise ROI	"0","1" or "0,1" (0=OFF, 1=ON)
LOWPASS=[{options}]	Low-pass filtering	"0","1" or "0,1" (0=OFF, 1=ON)

[1]***"aggressive" censoring has not been extensively validated. Use with extreme caution.***

Tips for {pipeline_list}.txt file:

- All entries must be placed within square brackets with **no spaces**
  Ex. MOTCOR=[0,1]
- Comma-separated list specifies which options are being tested. For example with MOTCOR:
  - MOTCOR = [0] → do not apply to any datasets (forced off)
  - MOTCOR = [1] → apply to all datasets (forced on)
  - MOTCOR = [0,1] → do not apply and apply to all datasets (test on AND off)
- Note that DETREND, MOTREG and GSPC1 are regressed as nuisance covariates in a general linear model (GLM). If TASK=1, we include the task design as an additional GLM covariate, to protect against over-regression of the task signal.
- **We recommend testing as many pipeline options as possible, as they may all have a significant impact on analysis results, with the caveat that fully optimized testing of all steps in many 10s of sessions will likely be very slow (i.e., days) without significant high performance computing resources.**

Example Text for File #2

If you wanted to test MOTCOR, CENSOR, LOWPASS and DETREND (orders 0 to 3), and leave all other steps OFF (with a fixed spatial smoothing scale of 6mm), you would have a text file as follows:

```
MOTCOR=[0,1]
CENSOR=[0,1]
RETROICOR=[0]
TIMECOR=[0]
SMOOTH=[6]
DETREND=[0,1,2,3]
MOTREG=[0]
TASK=[0]
GSPC1=[0]
PHYPLUS=[0]
CUSTOMREG=[0]
LOWPASS=[0,1]
```

File #3: {task_information}.txt

If you are providing a reference to a BIDS compliant data set via the --bids_dir parameter, the task_information file will be automatically created as an intermediate input file for OPPNI during processing.

If your data is not BIDS compliant you must manually create the task_information file for OPPNI as described here:

This is a text file (or set of text files) specifying details of your fMRI task paradigm. For each line of the {subject inputs} file described above, the field **TASK**=(…) points to the corresponding {task information} file for that dataset, so that OPPNI knows how to analyze it. If all of your datasets have the same task design (e.g. a fixed block paradigm), you will only need to create a single {task

information} file. However if onsets are randomized across subjects, different files must be created.

As with File #2, this file will have a series of entries {task parameter}=[{options}], that describe important details about these task. You will need to include all of the following "general" inputs (except the "SEED" parameter, which is only required if you are doing seed-based connectivity):

Table 3. List of "general" task parameters (only need to specify once in file)

Entry in File	Entry	Options
UNIT=[{options}]	Unit in which task onset/duration is measured	"TR", "sec" or "msec" (#scan volumes, seconds or milliseconds)
TR_MSEC=[{options}]	Time between scan volumes (repetition time)	Integer value, in milliseconds
TYPE=[{options}]	Type of task paradigm being processed/analyzed	"block", "event" or "nocontrast" (see the next section for discussion of task types, if you are unsure)
SEED=[{options}]	Name of ROI brain mask, for seed-based connectivity analysis. *only required for SCONN analysis (details in next section). Proceed with caution if using	Should give {path/filename} of 3D binary brain volume (1=seed ROI/0=non-ROI). Volume should be a NIFTI file in coordinate space of the input fMRI data (i.e. subject-specific masks).

Now, for each task condition in the fMRI run that you want to include in your analyses, you must specify the following three fields:

Table 4. List of parameters for each condition (need to specify for each condition being analyzed)

Entry in File	Entry	Options
NAME=[{options}]	Name of this task condition	Anything you like, but avoid special characters e.g. "+-[]". The name "baseline" is reserved for the scans in which the brain is in the "REST" condition (See below for details).
ONSETS =[{options}]	List of onset times, **in time-units specified by UNIT field**	Must be a comma-separated list of numbers, e.g. "ONSETS=[1,10,30]"
DURATION=[{options}]	List of condition durations corresponding to the above onsets, **in time-units specified by UNIT field**	Must be a comma-separated list of numbers, e.g. "DURATION=[9,10,8]"

Tips for creating your {task_information} files:

- Defining conditions: for every new condition, you will need to specify a NAME, ONSETS and DURATION field. Within each task condition, the number of ONSET entries must match the number of DURATION entries.
- Defining onsets: onset times are relative to the start of the raw data run, i.e. before any scans have been dropped using the DROP field in the {subject input} file. Also, it is assumed that scanning time starts at TR=1, and time sec=0 (and msec=0).

- Multiple conditions: You can specify as many conditions as you like, prior to optimization. Block design requires a minimum of 2 conditions; event-related analysis only requires 1 condition
- Event-related analysis: if data is event-related, all "DURATION" fields should be 0.
- Baseline condition: The baseline condition has to be defined if the two following conditions are met: 1) the data is a block design (i.e. TYPE=[BLOCK]); 2) the preprocessing step TASK is set to ON [1], or ON AND OFF [0,1], or the analysis model is GLM (See the next subsection for the available analysis models). In this case, the users have to specify the onset and duration of those blocks during which the brain is in the resting-state condition (i.e. the participants did not perform any particular task). The name of this condition has to be "baseline" (NAME=[baseline]). This condition will be treated differently in the code; there will not be a regressor associated with this condition in the GLM model or when TASK is ON. It is better to define the baseline condition as the last condition in the task information file (See below for the example). Note that for other cases (the remaining analysis models and TASK=[0]) defining the baseline condition is not necessary and does not have any effect on the final results.
- IMPORTANT NOTE: If 1) the data is a block design (i.e. TYPE = [BLOCK]); AND 2) the preprocessing step TASK is set to ON [1], or ON AND OFF [0,1], or the analysis model is GLM, all conditions in the paradigm MUST be specified, even if they are not going to be analyzed. In addition, one condition must be defined as a 'baseline' condition (see above). For example, if your dataset has four conditions A,B,C,D and you want to contrast A vs. C, conditions B and D still need to be defined in the {task information}.txt file (i.e. the file will contain onset information for Conditions A,B,C,D), with one condition defined as the 'baseline' condition. For example, the conditions can be defined as 'A','B','C', 'baseline'.

Example Text for File #3

In the example below, we have a block-design task with a TR of 2000 ms. It has three task conditions presented sequentially (A-B-C-A-B-C), each 10 TR in length. In this example, we discard the first 2 TR of each block (hence DURATION values of 8); due to the sluggish nature of the BOLD hemodynamic response, there is a 4-6 second delay before reaching peak activation.

```
UNIT=[TR]
TR_MSEC=[2000]
TYPE=[BLOCK]

NAME=[task_A]
ONSETS=[13,43]
DURATION=[8,8]

NAME=[task_B]
ONSETS=[23,53]
DURATION=[8,8]

NAME=[task_C]
ONSETS=[33,63]
DURATION=[8,8]
```

```
NAME=[baseline]
ONSETS=[1,71]
DURATION=[12,12]
```

If we saved this file as "design.txt", in the directory "mydir", then in File #1, you would point to this file by typing **TASK=mydir/design.txt**.

iii. Specifying your Analysis

Pipeline optimization is done to maximize the Prediction and Reproducibility of your analysis results, so it is important to decide what sort of analysis you are conducting, see Appendix F for more detail. You have control over your analyses in two ways: (1) what task conditions you want to analyze (specified inside of a {task information} file, described in the last section), and (2) what kind of analysis model you want to use (which is specified when you submit your jobs to OPPNI). This section provides some information to help guide your choice of analysis model. Once you have decided the analysis model you want to use from Table 3, you can go to the next section and start running pipelines!

First, you need to decide how the fMRI data will be analyzed. This depends on your experimental design. Broadly speaking, are three types of fMRI task design:

Block design: a fixed stimulus is presented for an extended duration, usually longer than the time-to-peak for a standard HRF (>6-10 sec.). A block design analysis looks for difference in "mean" signal between different experimental stimulus blocks (for example, task and baseline). If you have runs with multiple conditions, make sure to optimize pipelines specifically for the contrast you are interested in studying. In general, different task contrasts in the same fMRI run require different pipeline choices for optimal signal detection (Churchill et al., 2012a).

Event-Related design: a brief stimulus event is presented, of shorter duration than HRF time-to-peak. These analyses estimate the dynamic BOLD response curve itself, along with the associated brain pattern(s). Unlike Block design, you can capture information about the shape of the HRF; this comes at the expense of decreased estimation power, and a more complex, potentially unstable signal space.

No-Condition design: studies which only expose subjects to a single stimulus type. The most common example is "resting-state" (i.e. no overt task demands), but other examples may occur (e.g. movie viewing, performing mental arithmetic, enforced rumination). Because there is no structured temporal design, these analyses focus on coherent brain dynamics. **Although models are available in OPPNI, (SCONN, and gPCA), Single-Condition optimization properties are less well understood and the subject of ongoing research, so use with caution!**

No Analysis Model: if there are no appropriate models for the analysis you want to do (e.g. if you are preprocessing resting-state data) and cannot do optimization, you can specify a single pipeline, or small set of pipelines, and just generate the preprocessed data.

The table below tells you which analysis models are recommended for pipeline optimization. It depends on which of the above 3 categories your task design falls into (Block, Event-Related, Single-Condition), and whether you want to conduct univariate or multivariate analysis. Univariate models measure regional BOLD activations and tend to be less sensitive to pipeline choices (good and bad); multivariate models account for functional connectivity between brain regions and tend to be more sensitive to pipeline choices (good and bad). Neither approach is "right" or "wrong", it just depends on what you are interested in measuring.

Table 5. Available analysis models in the OPPNI framework

	Univariate, Independent Voxels	Multivariate, Covarying Voxels
Block Design TYPE=[block]	**GNB** (Gaussian Naïve Bayes) predictive general linear model of difference between 2 conditions	**LDA** (Linear Discriminant Analysis) predictive multivariate model of differences between 2 conditions
Event-Related TYPE=[event]	**erGNB** (Event-Related GNB) predictive general linear model, treating each time-point in HRF as a different condition	**erCVA** (Event-Related CVA) predictive multivariate model, treating each time-point in an HRF window as a different condition
No Contrast TYPE=[nocontrast]	**SCONN** (Seed-based Connectivity) measures pairwise correlations of all voxels, relative to average signal in a seed Region of Interest (ROI)	**gPCA** (Generalized PCA) Principal Component Analysis decomposition of fMRI dataset, identifying most consistent subspace
No Analysis Model TYPE=[none]	**NONE** generates preprocessed data without any analysis model. This will generate data from all tested pipelines, so make sure you only select a small number of pipeline combos or you will quickly run out of disk space.	

*** as noted above, the optimization properties of "No Contrast" analyses (e.g. for resting-state) are less well understood. Use with caution!!**

* The OPPNI code also has standard General Linear Model (GLM) and event-related GLM (erGLM) models available, but we do not recommend them: they can only be used to measure Reproducibility, whereas Prediction accuracy cannot be computed. If you want to analyze your data assuming independent voxels, as in GLM, we recommend using **GNB** or **erGNB**, which are the predictive versions of these models.

*NB: The baseline condition has to be defined, when the GLM analysis is used (See the previous subsection).

iv. Running Individual Pipelines

Once you have located the raw fMRI data or BIDS dataset, created your input files and chosen your analysis model, you are ready to begin pipeline optimization.

OPPNI is designed to run using in a high-performance environment . This is a distributed resource manager, which allows datasets to be submitted as individual jobs to a computing cluster, greatly accelerating computational speed. This is all executed by running the wrapper "**oppni**". The command line structure is defined below, along with important outputs (see Appendix ?? for further details). You can still run OPPNI without access to HPC clusters in (almost) any unix-like

environment, but it will be much slower. This may be a significant problem with modest computational resources and anything but small numbers of subjects with small numbers of scans.

Oppni is also provided as a tool on CBRAIN. Users will upload their data to the CBRAIN data providers, and be able to select their analysis using a GUI. The CBRAIN Portal may be easier to use for beginners but has the disadvantage of running considerably slower. Unlike proper HPC use, CBRAIN will run your subjects one after another rather than in parallel unless a BIDS compliant dataset can be provided . This method should only be used for small numbers of subjects with small number of scans. More information can be found in the Appendices.

Preparatory details:

- Check that both AFNI and FSL software packages are in your path (type "which afni", "which fsl" in the terminal console, and it should return a directory location).
- Make sure OPPNI folders are in the current working directory along with {subject_input} and {pipelines} files
- Confirm raw fMRI data, physio data, and {task_information} files are in the directories listed in your {subject_input} file

When the scripts are run on HPC system, each line in the {subject input}.txt file is submitted as a job.. Every job has an ID number, used to track progress of the scripts. These job numbers appear on the screen as soon as the script is run. Once the system begins to process the specific job, a text file is created in the working directory, labeled with the wrapper name and job number, which tracks the job. If you see an error (or data appears to be missing), you can check these log files to see what happened. Typing (SLURM command) "squeue –u $USER" in the command line will show jobs that are currently being processed as well as pending jobs, identified by job number.

Input commands:

Once OPPNI software is setup and you have created your {subject_input}.txt and {pipeline_list}.txt files, along with the necessary "split_info".mat files for each dataset, you can initiate the pipeline by typing the following command:

oppni.py –i {subject_input}.txt –c {pipeline_list.txt} –a {analysis_model} –r {reference_image.nii} [other arguments]

Or for BIDS compliant data sets, {subject_input}.txt are generated automatically for the complete dataset

```
oppni.py --bids_dir {dataset path} --output_dir {output path} –c
{pipeline_list.txt} –r {reference_image.nii} –a {analysis_model}
[other arguments]
```

Or using BIDSApp style (participant/group) parameter options

```
oppni.py --bids_dir {dataset path} --output_dir {output path} --
analysis_level {participant or group} [--participant_label
{participant}] -p PART –c {pipeline_list.txt} –r
{reference_image.nii} –a {analysis_model} [other arguments]
```

NOTE: Arguments enclosed in [ ] are optional.

***Note*: For detailed instructions for installing and setting up OPPNI, refer to APPENDIX A.**

We have already reviewed how to write the {subject_input}.txt and {pipeline_list}.txt files in section 2(iii). You will want to specify additional flags, the most important being **–a** and **–r**, but refer to Tables 5 & 6 for special options that may be useful if you want to do advanced analysis.

The flag **"-a {analysis model}"** specifies the chosen analysis method that will be applied to all data sets specified in the {subject input}.txt (e.g. "**-a LDA**" performs linear discriminant analysis). Options are described in detail in the previous section, and include **GNB**, **LDA**, **erGNB**, **erCVA**, **SCONN** and **gPCA**. If you don't want to (or can't) do analysis, type "**-a NONE**"; this will just output all processing pipeline combinations specified in {pipeline_list.txt}. **WARNING: if you choose this option, it will create a preprocessed dataset for <u>every</u> pipeline combination. Make sure it is a short list of options, as this will quickly eat up all your available disk space!**

The flag "**-r {reference_image.nii}**" specifies the path+name of a reference template volume (e.g. MNI, ICBM or Talairach atlas); all optimized results will be spatially normalize to match this volume. **If you omit this flag, or don't include "STRUCT=..." fields in the {subject_input.txt} file, spatial normalization step will be skipped.**

The **[other arguments]** allows you set other optional flags for OPPNI, giving you greater control over pipeline inputs/outputs. Refer to the tables 6 & 7 below to see what is available.

Output results:

After completing pipeline optimization, you will get a set of folders containing intermediate pipeline results, and final optimized outputs. If your {subject_input.txt} has output fields specified as follows: **OUT**={outdir}/{outname}, ***you will see the following output directory structure in directory "outdir"***:

intermediate_processed	→ contains semi-processed data plus other selected outputs
afni_processed	→ data processed using AFNI algorithms
diagnostic	→ diagnostic metrics used to identify outlier volumes
masks	→ functional brain masks
mpe	→ head motion parameter estimates from AFNI: - 6 rigid-body parameters {outname}_mpe, and max. displacement at each time point {outname}_maxdisp
spat_norm	→ parameters required to do spatial normalization
split_info	→ stores copies of split_info data, describing task structure
intermediate_metrics	→ contains metrics of data quality and analysis results
res0_params	→ misc. parameters, e.g. analysis model type, artifact measures
res1_spms	→ activation maps for all pipelines (stored in 2D matrix)
res2_temp	→ BOLD time series of activation maps (in 2D matrix)
res3_stats	→ statistics of pipeline quality (Prediction, Reproducibility)
regressors	→ task and nuisance covariates for each pipeline model
optimization_results	→ contains optimally-processed fMRI data and brain maps
processed	→ processed 4D fMRI runs (for STD, FIX and IND pipelines)

spms → optimal activation maps (STD, FIX and IND concatenated)
matfiles → activation maps and quality metrics, stored in .mat files

All important outputs are highlighted in red; the rest are just intermediate results that are used by the OPPNI code during pipeline optimization. If you are pressed for disk space after optimization, you can delete the "intermediate_processed" and "intermediate_metrics" folders.

The results of optimization are created as follows. If, for example, you have a dataset with the following output field in {subject_input}.txt : "**OUT**=new_dir/subj_1", you get the following output:

new_dir/optimization_results/processed
Proc_ subj_1_STD.nii
Proc_ subj_1_FIX.nii
Proc_ subj_1_IND.nii
Proc_ subj_1_STD_sNorm.nii *
Proc_ subj_1_FIX_sNorm.nii *
Proc_ subj_1_IND_sNorm.nii *

new_dir/optimization_results/spms
rSPM_ subj_1_STD_FIX_IND.nii
rSPM_ subj_1_STD_FIX_IND_sNorm.nii *

Where "Proc…" represents the preprocessed 4D fMRI datasets, under STD (standard), FIX (fixed) and IND (individually optimized) pipelines. The "rSPM…" is a set of reproducible Z-scored SPMs produced by STD, FIX and IND pipelines, concatenated along the time (4^{th}) dimension.
Outputs with a '*' are only produced if spatial normalization is included. Datasets without "sNorm" are in native subject space, while "sNorm" indicates the corresponding dataset, warped into common template space.

If you run spatial normalization, a set of group-level masks are also created in a "GroupMasks" folder; it is placed in the output path of the first dataset in {subject input}.txt. This includes:

group_consensus_mask.nii → a binary mask of brain tissue voxels
group_mean_NN_WM.nii → concatenated probabilistic tissue masks: non-neuronal (NN), white matter (WM)

More information on the output files can be found in the Appendices.

The Tables below displays all arguments available when running OPPNI. This includes some options that can be applied to all task types (Table 6), and options that may only apply to specific analysis models (Table 7). We also have a few options that are available for group-level analysis (Table 8). **Only arguments in grey (Table 6) are critical (unless requesting a status update on previous processing) for running pipelines. All others are optional, and can be ignored unless you have specialized analysis plans.**

Table 6. Arguments available when running pipelines

Switch	Description
-h, --help	show help message and exit
-i FILE.txt, (--inputdata=FILE.txt)	Path and name of the {subject input}.txt file Optional for BIDS data sets , intermediate files will be automatically created.
-o output_folder --output_prefix output_folder	Output folder prefix for storage of all the processing and results. If you specify this, OUT=/path in input files maps to OUT=output_prefix/path.
-c PIPELINE.txt, (--pipeline PIPELINE.txt)	Path and name of PIPELINE.txt, with list of pipeline steps being tested
-a ANALYSIS, (--analysis=ANALYSIS)	Chosen analysis model (LDA, GNB, GLM, erCVA, erGNB, erGLM, SCONN, NONE)
-r REFERENCE, (--reference=REFERENCE)	Path and name of reference volume used in (optional) spatial normalization step. This volume must be in the same orientation as the input images and ideally be skull-stripped.
--TPATTERN	Defines axial slices acquisition order for slice-timing correction - this must be specified whenever slice timing correction is requested. Options include: 'altplus' (alternating, + direction – standard interleaved protocol) 'alt+z2' (alternating, starting at slice #1 instead of #0) 'altminus' (alternating, negative direction) 'alt-z2' (alternating, starting at slice #nz-2 instead of #nz-1) 'seqplus' (sequential, positive direction) 'seqminus' (sequential, negative direction) 'auto_hdr' (expert option 3dTshift automatically reads slice timing from header)
--output_dir output_folder	This is equivalent to BIDSApp output_dir when processing BIDS data sets
--analysis_level=value {participant or group}	BIDSApp options can only be used with BIDS compliant datasets, By default PART 1 of the pipeline will be run on the subject, when analysis_level is participant: --participant_label parameter is required. When analysis_level is group, by default PART 2 the of pipeline is run on participants that have completed PART 1 processing.. (see option -p)
--participant_label subject	This BIDSApp option can only be used with BIDS compliant datasets,
--taskname task	BIDS datasets - String specifying the name of the fMRI task you want to analyze
-p PART (--part PART)	Only do specific parts of the full optimization pipeline. 0 = do all three steps (this is the default) 1 = run all pipelines and produce metrics 2 = select optimal pipelines based on metrics (must have already run part-1) 3 = do spatial normalization of optimized results (must have already run 1,2) 4=run QC only (choose this only when you have existing processing that was successfully completed from earlier runs, and also supply the flag --use_prev_processing_for_QC) Use in conjunction with BIDSApp option --analysis_level options to select pipeline parts to run, NOTE: -p 0 is not permitted as a BIDSApp option.
--min_subjects #	(optional) Override default(4), specify minimum of unique subjects required
--convolve VALUE	VALUE=Binary value, determines whether user-provided design matrix (in split_info) should be convolved with the canonical HRF modeled by two gamma

	functions (AFNI's "SPM1" function), for TASK pipeline step and GLM analysis. 0 = do not convolve and 1 = perform convolution (default =0)
-m METRIC, (--metric=METRIC)	Metric used to choose optimal pipelines standard options include 'R'= reproducibility 'P'= prediction 'dPR= combined prediction & reproducibility (this is the default)
--contrast=CONTRAST	Task contrast being analyzed and optimized for. Necessary when more than two conditions are defined in the split_info files. Syntax: --contrast "CON1-CON2,CON2-CON3", where CON1, CON2 are condition names (The condition number may be used instead, e.g. –contrast 1-2,2-3). Default is --contrast 1-2. (See Subsection 4.ii for more details) If you are doing a single condition 'nocontrast', this is omitted. NOTE: event-related analysis is *always* relative to baseline, e.g. --contrast "CON1-baseline", as it is modelling HRF shape relative to baseline BOLD signal.
--vasc_mask {0,1}	Toggles estimation of subject-specific vascular mask that would be excluded prior to analysis (0: disable, 1: enable). Default enabled.
--num_PCs NUMBER	NUMBER = total number of principal components to retain
-k KEEPMEAN, (--keepmean=KEEPMEAN)	KEEPMEAN=binary value that determines whether the output nifti files have temporal means re-added to each voxel after processing (default KEEPMEAN=1, not remove the mean scan), or the mean is removed and not replaced (KEEPMEAN=0).
-v VOXELSIZE, (--voxelsize=VOXELSIZE)	Determines the output voxel size of nifti file. Syntax: -v "[3.0 3.0 5.0]" gives 3x3x5 mm voxels as final results. Default is to keep output voxels the same size as input.
--BlurToFWHM {0,1}	This option will enable adaptive spatial smoothing to equalize smoothing across multiple sites. Default 0.
--DEOBLIQUE	Corrects for raw data that are at an oblique angle relative to the cardinal scanning axes, using AFNI's 3dWarp program. May improve the quality of registration
--control_motion_artifact {yes, no}	Control for motion artifact. Default yes.
--control_wm_bias {yes,no}	Control for white matter bias using spatial priors. Default no.
--output_nii_also {1,0}	Whether to output spms for all the optimal pipelines.
Convenience methods:	
--status FOLDER or -s FOLDER	Request for a status update on previously submitted processing. Supply the output directory where previous processing has been submitted to.
--print_options_in FOLDER -po FOLDER	Prints (dumps) all options used in launching the previous processing in the folder
--validate INPUT_FILE_PATH	Performs a basic validation of an input file (existence of file paths listed in the file and consistency across different subjects/runs etc).
HPC Cluster related arguments	
-e ENVIRONMENT	(optional) This determines which software used to run the code: matlab, octave or compiled(default). The "compiled" argument uses pre-compiled OPPNI code and does not require a matlab license to run, while "matlab" requires a license. The octave option (still in development) also does not require a license.
--account ACCTNAME	(optional) If you have multiple allocations on the cluster, you must specify the account to be used.
-q QUEUE, (--queue=QUEUE)	HPC queue name, default is "None" (For HPCVL SGE, use –q abaqus.q).

--cluster clustertype **--cluster cluster**	**Please specify the clustertype OR cluster you're running the code on.** **clustertype: SLURM, SGE (SUNGRID), TORQUE (PBS)** **Cluster: FRONTENAC, CEDAR, GRAHAM, QUEENU, BAYCREST, ROTMAN**
--memory MEMORY	**(optional) determine the minimum amount RAM needed for the job, e.g. --memory 8 (in gigabytes)! Default is 6GB**
--numcores #	**(optional) Default = 1**
-pe PARALLEL_ENV or --parallel_env PARALLEL_ENV	**(optional) Name of the parallel environment under which the multi-core jobs gets executed. This must be specified explicitly when numcores > 1.**
--run_locally	**Run the pipeline on this computer without using HPC. The number of cores will currently be forced to 1, processes are run sequentially.**
--force_rerun	**Cleans up existing processing and forces a rerun. Be absolutely sure this is what you want to do.**
--dry_run	**Generates job files only, but not run/submit them. This option can help in various scenarios such as debugging.**

Table 7. Other arguments that are specific to individual analysis models.

	Switch	Descriptions
GLM		This model does not have any specific switch
GNB	--decision_model MODEL	MODEL is a string specifying type of decision boundary. Either: 'linear' for a pooled covariance model or 'nonlinear' for class-specific covariances (default is 'linear')
LDA	--drf FRACTION	FRACTION is a scalar value of range (0,1), indicating the fraction of full-date PCA subspace to keep during PCA-LDA analysis (default is 0.5)
erGLM		This model does not have any specific switch
erGNB	--Nblock NUMBER	NUMBER is the number of equal-sized blocks to break the data into, to perform time-locked averaging. Must be at least 4 (the default) to obtain robust covariance estimates.
	--WIND SIZE	SIZE is the window size to average on, in TR (usually in range 6-10 TR). Default is 10.
erCVA	--Nblock NUMBER	NUMBER is the number of equal-sized blocks to break the data into, to perform time-locked averaging. Must be at least 4 (the default) to obtain robust covariance estimates.
	--WIND SIZE	SIZE is the window size to average on, in TR (usually in range 6-10 TR). Default is 10.
	--drf FRACTION	FRACTION is a scalar value of range (0,1), indicating the fraction of full-date PCA subspace to keep during PCA-LDA analysis (default is 0.5)
	--subspace COMP	COMP is a string specifying either: 'onecomp' = only optimize on CV#1 or 'multicomp' = optimize on full multidimensional subspace. Default is 'onecomp'
SCONN	--spm FORMAT	FORMAT is a string specifying format of output SPM for seed-based connectivity analysis (SCONN). Options include 'corr' (map of voxelwise seed correlations) or 'zscore' (Z-scored map of reproducible correlation values). Default is 'zscore' *See analysis section and use with caution

Examples of Pipeline Jobs

1. Let's assume we have four conditions in the data ATT, DMS, PMT, and RT. The goal is to run pipeline optimization with LDA for two task contrasts: 1) ATT and DMS vs PMT, 2) DMS vs RT. The syntax for submitting all three parts is as follows:

```
oppni -i /example/input.txt -c /example/param.txt -a LDA -r
/example/MNI152_1mm.nii -v 4.0 --contrast 'ATT+DMS-PMT,DMS-RT'
```

The pipeline will optimize for both ATT and DMS vs PMT, and DMS vs RT task contrasts. Note that we used a special option of "-v 4.0" (Table 6) which forces the spatially normalized fMRI data to have voxel resolution of 4x4x4mm. This would be omitted if we wanted to keep the same voxel size as in the raw data. Alternatively, say you did not specify a contrast:

2. Let's assume the goal is to optimize the preprocessing of an event-related dataset using the erCVA analysis model, we are modeling the HRF for a condition GO, and we do not want to perform spatial normalization. The syntax is as follows:

```
oppni –i /example/input.txt –c /example/param.txt –a erCVA --WIND 6 --contrast 'GO-baseline'
```

Unlike in example #1, we did not specify the reference image using `–r`. Then the program skips this step, and only produces optimized data in the subjects' native space. Note that we again used a special option, of "`--WIND 6`" (Table 7). This changes the window size of the estimated HRF in erCVA analysis, to 6TR. If we omitted this option, it would give the default of 10TR.

4. Quality Control

As part of pipeline optimization, OPPNI generates a series of diagnostic plots which can be used to evaluate the overall quality of preprocessing pipelines and SPMs generated by OPPNI. They are divided into two sets of outputs, described below. See ***Appendix E*** for example QC output plots.

Quality Control-1: this step evaluates overall quality of data, across all tested preprocessing pipelines. **QC1** produces output plots showing: (1) head motion stats, (2) potential SPM artifact (e.g. motion, global signal, white matter), (3) the sensitivity of performance metrics to pipeline choice, and (4) preprocessing steps that have the greatest impact on results. Output is saved to "QC1_results" folder, in output path of first dataset in {subject input}.txt

Quality Control-2: this step specifically compares the results of different optimization schemes (CON, FIX, IND), including group-level activation maps. **QC2** produces output plots showing: (1) optimal pipeline choices, (2) optimal performance metrics; (3) statistics on quality of spatial normalization; (4) between-subject SPM similarity; (5) group-level PCA results for each pipeline. Output is saved to "QC2_results" folder, in output path of first dataset in {subject input}.txt

5. Optimizing for Multiple Task Contrasts

Independently Optimizing for each task contrast increases computational time, and the required storage space. In addition, it results in several sets of preprocessed data, which make it difficult for researcher to generate a consensus dataset. OPPNI provides the capability of optimizing for several task contrasts simultaneously. To use this feature, all the desired task contrasts have to be defined in the command line using the "--contrast" option. This is specific to block design as event-related design must be contrasted with baseline.

The pipeline gives best average (Prediction, Reproducibility) over all contrasts, using the "`--contrast`" option, in Table 6). For example, if you have 3 conditions CON1,CON2,CON3 and you just wanted to optimize CON1 vs. CON2, you would use the option:
"`--contrast CON1-CON2`"

If you wanted to optimize for all possible paired contrasts (CON1 vs CON2, CON1 vs CON3, CON2 vs CON3), you would instead use the option:
"`--contrast CON1-CON2,CON1-CON3,CON2-CON3`"

which would gives a single preprocessed dataset, along with optimized SPMs for each of the 3 different contrasts in your outputs.

6. Checklist for Running OPPNI

1) Install your code correctly, See APPENDIX A
2) Make sure your data fits in with the requirements
3) Format all of your documents properly

a. Subject_input.txt

```
IN=my_dir/subj1_task.nii OUT=new_dir/subj1 DROP=[2,1] TASK=taskInfo_subj1.txt PHYSIO=my_dir/physio/subj1 STRUCT=my_dir/subj1_T1.nii
IN=my_dir/subj2_task.nii OUT=new_dir/subj2 DROP=[2,1] TASK=taskInfo_subj2.txt PHYSIO=my_dir/physio/subj2 STRUCT=my_dir/subj2_T1.nii
IN=my_dir/subj3_task.nii OUT=new_dir/subj3 DROP=[2,1] TASK=taskInfo_subj3.txt PHYSIO=my_dir/physio/subj3 STRUCT=my_dir/subj3_T1.nii
IN=my_dir/subj4_task.nii OUT=new_dir/subj4 DROP=[2,1] TASK=taskInfo_subj4.txt PHYSIO=my_dir/physio/subj4 STRUCT=my_dir/subj4_T1.nii
IN=my_dir/subj5_task.nii OUT=new_dir/subj5 DROP=[2,1] TASK=taskInfo_subj5.txt PHYSIO=my_dir/physio/subj5 STRUCT=my_dir/subj5_T1.nii
```

b. Pipeline_list.txt

```
MOTCOR=[0,1]
CENSOR=[0,1]
RETROICOR=[0]
TIMECOR=[0]
SMOOTH=[6]
DETREND=[0,1,2,3]
MOTREG=[0]
TASK=[0]
GSPC1=[0]
PHYPLUS=[0]
CUSTOMREG=[0]
LOWPASS=[0,1]
```

c. Task_information.txt

```
UNIT=[TR]
TR_MSEC=[2000]
TYPE=[BLOCK]
NAME=[task_A]
ONSETS=[13,43]
DURATION=[8,8]
NAME=[task_B]
ONSETS=[23,53]
DURATION=[8,8]
NAME=[task_C]
ONSETS=[33,63]
DURATION=[8,8]
NAME=[baseline]
ONSETS=[1,71]
DURATION=[12,12]
```

4) Specify your analysis: block, event, no-contrast, no-analysis
5) Make sure your data is in the correct location according to your input file
6) Compose your command line – make sure you have it properly set up and correctly choose options from Table 6. Example:

   ./oppni –i subject_input.txt –c pipeline_list.txt –a GNB –r MNI152_T1_1mm_brain.nii –m dPR –memory 8 --e matlab -v “3.125 3.125 5.0” --decision_model linear --TPATTERN auto_hdr --contrast “task-A-task_B” --cluster SGE

7) Run your command and make sure all jobs are submitted correctly
8) Check that you have all of your results..

7. Results and Simple Debugging

To check that all submitted jobs have been processed by issuing the following (SLURM) command:

squeue -u $USER"

Any jobs showing on the resulting list will not have been completed. Once all jobs have completed you can issue the following command:

```
oppni --status (or -s)  [outdir folder]
```

Once you have typed the command above you should see an output similar to:

```
Now checking the outputs on disk ...
input1: preproc : Done. ✓  Metrics : Done. ✓  spat.norm : Done. ✓
input2: preproc : Done. ✓  Metrics : Done. ✓  spat.norm : Done. ✓
input3: preproc : Done. ✓  Metrics : Done. ✓  spat.norm : Done. ✓
input4: preproc : Done. ✓  Metrics : Done. ✓  spat.norm : Done. ✓

...

Summary:
# subjects:  24
P1 : all finished.
P2 : all finished.
SPNORM : all finished.
GMASK:   done.
QC1: figures done.
QC2: figures done.

Estimated processing time for whole workflow:
        102 minutes 56.183 seconds.
```

The above output shows that all steps of OPPNI (P1, P2, SPNORM, GMASK, QC1 and QC2) are successfully completed. If there was a problem with any of these steps you would see an output similar to below (depending on which parts have failed for which subjects):

```
Now checking the outputs on disk ...
input1: preproc : Done. ✓  Metrics : Incomplete ✘  spat.norm : Incomplete ✘
input2: preproc : Done. ✓  Metrics : Incomplete ✘  spat.norm : Incomplete ✘
input3: preproc : Done. ✓  Metrics : Incomplete ✘  spat.norm : Incomplete ✘
input4: preproc : Done. ✓  Metrics : Incomplete ✘  spat.norm : Incomplete ✘

...

Summary:
# subjects:  24
P1 : all finished.
```

```
P2 : Incomplete
# sbujects whose stats need to be computed: 24
SPNORM : incomplete     # subjects/runs failed: 24 / 24 (100%)
       resubmit list :  [outdir folder]/input_file-resubmit-spnorm.txt

GMASK:   failed.
QC1: Some/all of the visualizations have NOT been exported as images.
QC1: diagnostic data (path below) doesn't exist or is empty.
      [outdir folder]/QC1_results/output_qc1.mat
QC2: Some/all of the visualizations have NOT been exported as images.
QC2: diagnostic data (path below) doesn't exist or is empty.
      [outdir folder]/QC2_results/output_qc2.mat

Processing time could not be estimated.
Previous processing is incomplete.
Would you like to resubmit jobs for failed subjects/runs?  y / [N] :
```

If some of the jobs are not complete, as the example above you will be asked whether you would like to resubmit the failed jobs.

(…. Still have to add different cases of failure)

USEFUL SLURM COMMANDS

Other useful commands when debugging on a SLURM scheduler are:

```
squeue -u $USER
```

squeue shows the status of current jobs, i.e. running or waiting, or if they have a dependency. If the jobs are not shown, they will have completed running either successfully or failed for some reason. Note that if a job fails, all of its dependencies will be canceled.

```
sacct -u $USER
```

sacct will show all of the current jobs but will also show your job history. This display also specifies the running node,running time and memory usage. It also has a state section which will tell you if the job completed or failed. This can provide additional detail in conjunction with oppni status checks.

LOG FILES

Each job that starts on the scheduler or running on a local machine will have a log file associated with it. This file can be found in the job_files/ folder. The text in this file is the output that would have been in the command terminal if you where running each job individually. This log file can be very useful for debugging when jobs don't complete no matter the environment. Use a text editor, or use a “tail -n50” command on the log file to see the end, where errors are usually displayed. Depending on the pipeline process there may be a message at the end stating that the job completed to completion.
When running jobs locally using multiple computer cores, you wont get any displaces in your command terminal. You will have to look at the log file if you want to moniteur a modules progress.

8. References

Campbell K et al. (2013). Age Differences in the Intrinsic Functional Connectivity of Default Network Subsystems. Frontiers in Human Neuroscience. 5:73

Churchill NW et al. (2012a): Optimizing Preprocessing and Analysis Pipelines for Single-Subject fMRI: 2. Interactions with ICA, PCA, Task Contrast and Inter-Subject Heterogeneity. PLoS One. 7(2):e31147

Churchill NW et al. (2012b): Optimizing Preprocessing and Analysis Pipelines for Single-Subject FMRI. I. Standard Temporal Motion and Physiological Noise Correction Methods. Human Brain Mapping 33:609–627

Churchill NW, Strother SC (2013): PHYCAA+: an optimized, adaptive procedure for measuring and controlling physiological noise in BOLD fMRI. NeuroImage 82:306-325.

Cox RW (1996): AFNI: Software for analysis and visualization of functional magnetic resonance neuroimages. Computers and Biomedical Research, an International Journal, 29(3): 162-173.

Glover GH, et al. (2000): Image-based method for retrospective correction of physiological motion effects in fMRI: RETROICOR. Magnetic Resonance in Medicine: Official Journal of the Society of Magnetic Resonance in Medicine / Society of Magnetic Resonance in Medicine. 44(1): 162-167.

LaConte S et al. (2003): The Evaluation of Preprocessing Choices in Single-Subject BOLD fMRI Using NPAIRS Performance Metrics. NeuroImage, 18(1):10-27

Schmah T, et al. (2010): Comparing classification methods for longitudinal fMRI studies. Neural Comput, 22:2729-62.

Shaw ME et al. (2003): Evaluating subject specific preprocessing choices in multisubject fMRI data sets using data-driven performance metrics. NeuroImage. 19(3):988-1001.

Strother SC, Anderson J, Hansen LK, Kjems U, Kustra R et al. (2002): The Quantitative Evaluation of Functional Neuroimaging Experiments: The NPAIRS Data Analysis Framework. NeuroImage 15:747–771

Strother S et al. (2004): Optimizing the fMRI data-processing pipeline using prediction and reproducibility performance metrics: I. A preliminary group analysis. NeuroImage. 23:S196-S207.

Strother S et al. (2014): Stability and Reproducibility in fMRI Analysis, in Practical Applications of Sparse Modeling, I. Rish, G. A. Cecchi, A. Lozano, and A. Niculescu-Mizil, Eds., pp. 99-121, Boston: MIT Press.

Tegeler C et al. (1999): Reproducibility of BOLD-based functional MRI obtained at 4 T. Hum. Brain Mapp. 7:267–283

Zhang J et al. (2009). Evaluation and optimization of fMRI single-subject processing pipelines with NPAIRS and second-level CVA. Magn. Reson. Imag. 27:264–278

9. Citation

If you use OPPNI, please cite the software using the following details:

Raamana, Pradeep Reddy, Churchill, Nathan W., & Strother, Stephen C.. Optimization of Preprocessing Pipelines for NeuroImaging (OPPNI) for fMRI preprocessing (Version v0.7.3.1_06JUL2017). Zenodo. http://doi.org/10.5281/zenodo.3662956

Appendix A: OPPNI Installation & setup

Step 1:

- Get a Compute Canada Account from here
- Find your username - let's assume it as DrFMRI

Step 2:

- Login to cedar with this command :
  ssh DrFMRI@cedar.computecanada.ca

Step 3:

- cd scratch
- salloc --ntasks=1 --mem=6000M
- cd **..**

Step 4:

- source /home/raamana/software/oppni/cPRONTO/simple_init_oppni_setup.sh
- exit

A1 Additional details on what's happening in the "simple install script" above: Setting up your OPPNI environment

OPPNI can be setup to operate from your terminal in most environments by running the setup script: "setup_oppni.py" Follow the steps below

Step 1.

If you are going to be using OPPNI with Octave on an HPC cluster setup_oppni.py will need to be able to find Octave. To make Octave accessible, enter the following:

```
]$  module load nixpkgs/16.09 gcc/7.3.0 octave/4.4.1
```

Note: For local or custom setups you will have to have installed a local copy of Octave 4.4.1+ OR Matlab..

Step 2.

Python version 3.5 or newer is required. On an HPC cluster , enter the following:

```
]$ module load python/3.7
```

To run the setup script enter the following:

```
]$ cd ~
]$ python [PATH]/setup_oppni.py  -env [ENVIRONMENT]
```

where is ENVIRONMENT is one of : “CAC” , “CC” , ”FRONTENAC”, “CEDAR”, “GRAHAM”
and PATH is the directory where “setup_oppni.py” is located

The setup script will first check and display the current environment (if any) and prompt you to continue. Example output::

The following OPPNI setup PATH environment variable exist:

OPPNI_PATH = /home/mprati/oppni_git
AFNI_PATH = /software/afni
FSL_PATH = /software/fsl/bin
OCTAVE_PATH = /software/octave/versions/4.2.1/bin

bash profile: /home/mprati/.bash_profile
octaverc: /home/mprati/.octaverc

Do you want to overwrite the existing environment [Yes]/No:

The setup script can be used to also set a “CUSTOM” OPPNI environment. Allowing custom paths to be specified. Run setup_oppni with -h parameter to view all available options.

```
]$ python setup_oppni.py  -h
```

A more detailed explanation of what changes setup_oppni.py makes to your initialization files and environment variables can be found in sections A2 and A3 below.

NOTE: During the installation of Octave support packages you may receive multiple warning messages similar to the one shown below. You can ignore these messages.

Example warning message:

```
In file included from __control_slicot_functions__.cc:48:0:
sl_mb05nd.cc: In function ‘octave_value_list F__sl_mb05nd__(const octave_value_list&, int)’:
sl_mb05nd.cc:96:13: warning: ‘f77_exception_encountered’ is deprecated: [4.4]: this variable is obsolete and
should not be needed [-Wdeprecated-declarations]
      if (f77_exception_encountered)
          ^~~~~~~~~~~~~~~~~~~~~~~~~~
```

NOTE: Pay attention to versions of (paths to) AFNI and FSL that setup_oppni configures to ensure the versions meet your requirements. If necessary you can manually modify these paths in the “.bash_profile” that is created.

NOTE: setup_oppni.py will always create a backup of your existing “.bash_profile”.
Example: .bash_profile_backup_20190905-T134153 where the suffix is date-UID.

A2. Manual installation of OPPNI on CAC (Frontenac before 2018)

Instructions below give some important details for installing OPPNI, in the pre-compiled form (which does not require a matlab license). Note that when submitting jobs, you can choose between

using compiled or matlab versions of the software using the "-e ENVIRONMENT" flag described in Table 6. However, we only test the compiled version regularly and do not support the license-requiring version.

Note: Pre-installed versions of OPPNI, AFNI and FSL will be used and do not have to be separately installed.

We recommend you use the pre-setup installation by following the steps from Step 5 onwards. But if you are an expert user, who would like to use your own release of OPPNI (which we discourage), you need to follow steps 1 through 4, and replace the path for OPPNI in step 7 with the path to OPPNI in your home folder on CAC e.g.

```
export OPPNI_PATH=/global/home/USERNAME/oppni/cPRONTO
```

NOTE: Steps 1-4 are for the core developers only. Regular users start from step 5.

The following steps will bring the latest OPPNI release to your home folder:

1. Download the release found at: https://github.com/raamana/oppni/releases/tag/v0.7.3.1_06JUL2017
2. Say it is downloaded to: ~/Downloads/oppni2017.zip
3. Unzip it and rename the folder as ~/Downloads/oppni
4. Copy this folder over to your HPC home folder using the following command:

```
scp -r ~/Downloads/oppni USERNAME@HOST:/global/home/USERNAME/
```

The following steps will setup your CAC environment:

5. Using the emails from CAC, note down your username and password
6. To access your CAC environment from your terminal, enter the following command:

```
ssh USERNAME@HOST
```

   replace USERNAME and HOST with yours, i.e. hpcxxx@login.cac.queensu.ca you will then be prompted to enter your password.

7. (Refer to section A1 for automated setup) or once logged in, your user environment needs to be set-up for OPPNI by inserting a few predefined environment variables in your login script (usually .bash_profile).

   Environment variables AFNI_PATH, FSL_PATH, MCR_PATH, OPPNI_PATH set the root paths to the pre-installed software components required to run OPPNI. (AFNI, FSL, Matlab Component Runtime and OPPNI folders respectively).

   a. Copy / Paste the commands shown below into terminal
   b. And press enter to make sure the last command is executed

```
echo "export AFNI_PATH=/u1/work/hpc3194/apps/afni/" >> ~/.bash_profile
echo "export FSL_PATH=/u1/work/hpc3194/apps/fsl/bin/" >> ~/.bash_profile
echo "export FSLDIR=/u1/work/hpc3194/apps/fsl" >> ~/.bash_profile
```

```
echo "export FSLOUTPUTTYPE=NIFTI_GZ" >> ~/.bash_profile
echo "export MCR_PATH=/u1/work/hpc3194/apps/mcr_install/v80/" >> ~/.bash_profile
echo "export OPPNI_PATH=/u1/work/hpc3194/oppni-latest/cPRONTO" >> ~/.bash_profile
echo "export PATH=\${AFNI_PATH}:\${FSL_PATH}:\${MCR_PATH}:\${OPPNI_PATH}:\$PATH" >>
  ~/.bash_profile
```

10. To check whether the above commands set your environment correctly, do the following:
    a. Log out of CAC terminal by running `exit` command (or press Ctrl-D)
    b. Log back in (Step 6 above).
    c. Now run the following commands, they should output the paths we setup earlier:

```
 i.   echo $AFNI_PATH
ii.   echo $FSL_PATH
iii.  echo $MCR_PATH
iv.   echo $OPPNI_PATH
```

    d. Now run OPPNI without any command line arguments:

```
[swlogin1 ~]$ oppni
```

    Help text should be displayed indicated paths have be set correctly.

11. If the output from step 10 does not make sense, or if running the above commands fail for some reason, please contact one of the OPPNI developers (by emailing Pradeep at praamana@research.baycrest.org) for assistance.

Version Check:

If there is a concern about software versions, or the difference in versions contributing to change in results, you can quickly find out what software versions you are using, with the following command:

```
[swlogin1 ~]$ oppni --validate_user_env

You should see output similar to the following:

AFNI_PATH: /home/hpc2590/afni/
FSL_PATH: /home/hpc2590/fsl/bin/
MCR_PATH: /u1/work/hpc2590/Applications/mcr_install/v80
Version AFNI_2011_12_21_1014
FLIRT version 6.0
MELODIC Version 3.13
Versions (major) match those tested.
[swlogin1 ~]$
```

This check happens automatically when you run OPPNI, and warns the user when versions differ from known tested versions. When you explicitly make the check, you can be informed about what versions are being used.

A3: Manual installation of OPPNI on ComputeCanada Clusters CEDAR / GRAHAM

(new from 04/2019)

Instructions below give some important details for installing OPPNI, in the source-code form (requiring a matlab license or optionally using Octave). Note that when submitting jobs, you need to choose matlab environment using "-e matlab" flag described in Table 6.

Octave compatibility is supported. Referto extra set up steps in APPENDIX I for Octave setup. Once available use you can choose octave environment using "-e octave" . Note: Octave version 4.4.1 or higher is required.

We recommend you use the pre-setup installation by following the steps from Step 5 onwards. But if you are an expert user, who would like to use your own release of OPPNI (which we discourage), you need to follow steps 1 through 4, and replace the path for OPPNI in step 7 with the path to OPPNI in your home folder on CC e.g.

```
export OPPNI_PATH=/home/USERNAME/oppni/cPRONTO
```

Note: Steps 1-4 are for the core developers only. Regular users start from step 5.

The following steps will bring the latest OPPNI release to your home folder on a CC cluster:

1. Download the latest release listed on https://github.com/raamana/oppni/releases
2. Say it is located at ~/Downloads/oppni2017.zip
3. Unzip it and rename the folder as ~/Downloads/oppni
4. Copy this folder over to your home folder on CC using the following command

   Example Compute Canada CEDAR:
   ```
   scp -r ~/Downloads/oppni USERNAME@cedar.computecanada.ca:/home/USERNAME/
   ```

The following example steps will setup your HPC environment.

5. Using the emails from Compute Canada, note down your username and password
6. To access your HPC environment from your terminal, enter the following command:

   ```
   ssh USERNAME@HOST
   ```

   replace USERNAME and HOST with yours, you will then be prompted to enter your password

7. (Refer to section A1 for automated setup) or once logged in, your user environment needs to be set-up for OPPNI by inserting a few predefined environment variables in your login script (usually " .bash_profile").

   Environment variables AFNI_PATH, FSL_PATH, MCR_PATH, OPPNI_PATH set the root paths to software components required by OPPNI. (AFNI, FSL, Matlab Component Runtime and OPPNI folders respectively):

   The login script can be modified by entering the following commands in the terminal:

Directly copy / paste the commands into your terminal, then press enter to make sure the last command is executed.

```
echo "module load matlab" >> ~/.bash_profile
echo "export SWPATH=/home/raamana/software/" >> ~/.bash_profile
echo "export AFNI_PATH=\${SWPATH}/afni" >> ~/.bash_profile
echo "export FSL_PATH=\${SWPATH}/fsl/bin/" >> ~/.bash_profile
echo "export FSLDIR=\${SWPATH}/fsl/" >> ~/.bash_profile
echo "export FSLOUTPUTTYPE=NIFTI_GZ" >> ~/.bash_profile
echo "export OPPNI_PATH=\${SWPATH}/oppni/cPRONTO" >> ~/.bash_profile
echo "export OPPNI_PATH_MATLAB_ORIG=\${SWPATH}/oppni/cPRONTO" >> ~/.bash_profile
echo "export PATH=\${AFNI_PATH}:\${FSL_PATH}:\${OPPNI_PATH}:\$PATH" >> ~/.bash_profile
```

These paths will reference a standard version of all software components needed for OPPNI from the Strother Lab group.

8. Matlab startup scripts needs to be created/modified so that matlab recognizes the paths to source code for OPPNI. Your matlab startup script is usually in a folder called `matlab` with the filename "`startup.m`". The Matlab startup script can be created by entering the following commands in the terminal:

```
mkdir -p ~/matlab
touch ~/matlab/startup.m
echo "display('adding OPPNI related paths to matlab path')" >> ~/matlab/startup.m
echo "addpath(genpath('/home/raamana/oppni/scripts_matlab'))" >> ~/matlab/startup.m
echo "addpath(genpath('/home/raamana/oppni/extra'))" >> ~/matlab/startup.m
```

9. If you plan on using Octave you will need run the octave_setup.py script. This will load the HPC octave module, set paths, download Octave packages and create your .octaverc startup file. Make sure call this from your home folder. (NOTE: Only required the first time)

```
cd /home/USERNAME
python /home/raamana/oppni/Octave/octave_setup_cc.py
```

10. To check whether the above commands set your environment correctly, do the following:

    a. Log out of cluster by running `exit` command (or press Ctrl-D)
    b. Log back in to your CC host the same way as before. Step 6
    c. Run the following commands, they should output the paths we setup earlier

```
 i.   echo $AFNI_PATH
ii.   echo $FSL_PATH
iii.  echo $OPPNI_PATH
```

    d. Now run OPPNI without any command line arguments:

```
[swlogin1 ~]$ oppni
```

    Help text should be displayed indicated paths have be set correctly.

11. The commands from step 10 don't make sense or if running the above commands fail for some reason, please contact one of the OPPNI developers (by emailing Pradeep at praamana@research.baycrest.org) for assistance.

12. If the installation is successful, use this page https://cac.queensu.ca/wiki/index.php/SLURM to learn how to use the SLURM scheduler.

Version Check:

If there is a concern about software versions, or the difference in versions contributing to change in results, you can quickly find out what software versions you are using, with the following command:

```
[user@cedarX]$ oppni --validate_user_env
```

Note: the output shown here references the CC stack versions of AFNI,FSL,MCR rather then the Strother Lab versions.

AFNI_PATH: /cvmfs/soft.computecanada.ca/easybuild/software/2017/avx2/Compiler/gcc7.3/afni/20180404
FSL_PATH: /cvmfs/soft.computecanada.ca/easybuild/software/2017/avx2/Compiler/intel2016.4/fsl/5.0.11/fsl/bin
MCR_PATH: /cvmfs/restricted.computecanada.ca/easybuild/software/2017/Core/matlab/2018b
Version AFNI_18.1.01
FLIRT version 6.0
MELODIC Version 3.13
Versions (major) match those tested.
[user@cedarX]$

This check happens automatically when you run OPPNI, and warns the user when versions differ from those tested. When you explicitly make the check, you can be informed about what versions are being used.

Appendix B: Tips for Installing AFNI and FSL

These packages may already be installed and available. Check with the systems administrator of the system you are running on before (re?)-installing them. For example, AFNI and FSL are already available on CAC and Compute Canada clusters (CEDAR & GRAHAM) and may be accessed by setting up the correct paths as detailed in Appendix A. Note the the default installation of OPPNI will by default use a specific preinstalled version of AFNI and FSL on FRONTENAC, GRAHAM, CEDAR .

AFNI Installation

AFNI: you will need to install this package as it is required for multiple preprocessing steps, including motion correction, slice-timing correction, and spatial smoothing. This can be obtained from the installation website, which gives helpful step-by-step instructions:

afni.nimh.nih.gov/pub/dist/HOWTO/howto/ht00_inst/html/linux_inst_basic.html

Although you can download and compile AFNI locally, it is often easier to just get the pre-compiled binaries. This can be done as follows:

1. First determine what OS you are running, as different AFNI binaries are compiled for different OSes. Use command "uname -m" in the unix terminal where you will be running pipelines. You'll get something like:

   "Linux [...] x86_64 GNU/Linux"

   This tells you it is a 64-bit Linux OS (x86_64 = 64-bit, anything else is usually 32-bit)

2. Now you have to identify and download the correct package from afni.nimh.nih.gov/pub/dist/tgz/. For example we might want the "linux_xorg7_64.tgz" package (standard 64-bit linux one). Type:

   ```
   wget http://afni.nimh.gov/pub/dist/tgz/linux_xorg7_64.tgz
   ```

   Wait for download to terminate

3. Now unzip/untar the downloaded file, and move into directory of choice. Make sure it is in path. e.g. if we have moved it to folder "abin" type

   ```
   export PATH=$PATH:~/abin
   ```

This must be put into your ~/.bashrc or .bash_profile file, to set the path for every job

CAUTION: Specific versions of AFNI may affect your results.

FSL Installation

FSL: you will need to install this package in order to use their suite of registration tools, required to spatially normalize data into a common template space. They also have helpful instructions at:

fsl.fmrib.ox.ac.uk/fsldownloads/fsldownloadmain.html

You can either (1) download their "installer" Python script, or (2) download the full FSL package (quite large, ~1.7 Gb in size). Option (1) is rapid and should automatically configure your environment to run FSL. However it requires administrator privileges to install and configure, and may throw unexpected errors. If you go by option (2), use the "download" tab to choose the appropriate build (e.g. Linux CentOS5 64-bit works for many servers).

As with AFNI, if you choose (2) you will have to unzip the file, and move to your chosen directory. You will also have to set your terminal environment so that all FSL software is in your path. For example in bash shell, you will add the following to your .bashrc or .bash_profile (you may have to create it):

```
FSLDIR=/usr/local/fsl
. ${FSLDIR}/etc/fslconf/fsl.sh
PATH=${FSLDIR}/bin:${PATH}
export FSLDIR PATH
```

where you would replace "/usr/local/fsl" with the directory your have installed FSL into. Refer to fsl.fmrib.ox.ac.uk/fsl/fslwiki/FslInstallation/ShellSetup for further instructions and details for other OSs.

Appendix C: Pipeline Optimization Script Details

There are three key scripts that are called during pipeline optimization:

1. Pipeline_PART1_afni_steps.m
2. Pipeline_PART1.m
3. Pipeline_PART2.m

The following sections describe in more detail how each underlying script functions.

Pipeline_PART1_afni_steps.m: this script is able to perform the following steps:

Fixed steps

1. Remove non-equilibrium/instructional scans, specified in {subject input}.txt
2. Generate coarse brain mask on unregistered data via3dAutomask (AFNI)
3. Identify minimum-displacement fMRI volume using PCA (i.e. the volume with minimum average Euclidean distance from all others in the run), then run MOTCOR using 3dvolreg (AFNI), with this volume chosen as the reference. This produces motion-corrected fMRI data and Motion Parameter Estimates (MPEs)
4. Run diagnostic script on fMRI data (with and without MOTCOR), producing .mat files containing diagnostic information, including indices of potential head motion spikes

Optional steps, specified in {pipeline list}.txt

5. Correct for oblique scan-volumes using 3dWarp (AFNI)
6. Retain data, with and without MOTCOR applied
7. Remove head motion spikes, including full volumes, and component-based (CENSOR)
8. Perform RETROICOR using 3dRetroicor (AFNI)
9. Perform TIMECOR using 3dTshift (AFNI)
10. Perform SMOOTH for specified kernel(s) using 3dmerge (AFNI)

Pipeline_PART1.m: this script is able to perform the following steps:

Fixed Steps

1. Obtain maps of brain edges, used to detect motion artifact
2. Generate mask of non-neuronal (vascular/ventricle regions) and downweight them

Optional steps, specified in {pipeline list}.txt

3. Run specified DETREND orders
4. Perform MOTREG using the MPE estimates from "Pipeline_PART1_afni_steps.m"
5. Include task design as part of the GLM model (TASK)
6. Estimate and remove Principal Component #1, a potential global signal confound (GSPC1)
7. Physiological regression using PHYCAA+ (PHYPLUS)
8. Low-pass filtering of BOLD time-series (LOWPASS)

Fixed Final Step:

Run split-half NPAIRS analysis on preprocessed data, generating brain maps and performance metrics

Pipeline_PART2.m: this script performs the following steps:

This script produces optimized analysis results under three different optimization strategies. Two pipelines are fixed (i.e. the same pipeline across all subjects), and one pipeline is individually optimized:

1. **CON:** "conservative" standard fixed pipeline; all steps except PHYCAA+ GSPC1, TASK turned on; detrending order is set using AFNI's heuristic; smoothing is 6mm FWHM
2. **FIX:** highest-ranked fixed pipeline
3. **IND**: individually optimized subject pipelines

This script performs the following steps:

1. Loads all subjects "METRIC_set" data, and concatenated values
2. Performs pipeline optimization, selecting the three different optimal pipeline strategies mentioned above (CON/FIX/IND)
3. Acquires optimal pipeline SPMs and time-series data
4. Performs spatial similarity testing of selected pipeline statistical parametric maps (SPMs)
5. Re-generates the optimally preprocessed 4D data

Note that this script does optimization based on the metric you choose to optimize, which means you can choose from all fields in the "METRIC_set" output. For example, 'LDA' analysis model has options of {'P','R',dPR}, where 'dPR' is the default option (see Table 6).

In addition, pipeline optimization is able to automatically exclude pipelines with excess motion artifact or excess global signal confound. The current default is to exclude motion artifact, but **not** control for global signal, as it is unclear whether this should be removed in all cases. This is controlled using the 2D binary input **mot_gs_control** (see Table 6), defined as:

a. **[0 0] = no artifact control**
b. **[1 0] = control motion artifact only**
c. **[0 1] = control global signal bias only**
d. **[1 1] = control both**

Appendix D: Spatial Normalization Script Details

(this is being replaced with generation of an ANTs custom EPI template and alignment of the template to a T1 reference template)

This section describes how to use the **spatial_normalization**.m script used to co-register all subject data into a common anatomical space. Inter-subject alignment is performed using a 2-stage transformation process:

1. Non-rigid (12-parameter affine) warp of subject anatomical volumes (T1) to an anatomical reference template (e.g. Talairach, MNI or ICBM)
2. Rigid-body within-subject alignment of functional (EPI) to their anatomical (T1) data. This is used for group-level analysis across SPMs

Spatial normalization should only run on the optimally processed datasets, as it is a highly compute-intensive process.

Spatial normalization steps:

1. Mask subject T1 brain volumes using bet (brain extraction tool; FSL)
2. Compute 12-parameter affine transformation of each T1 volume to the user-specified group template using flirt algorithm (FSL), producing affine transformation matrix
3. Compute the 7-parameter affine (rigid-body + scaling) alignment of fMRI data to the subject's corresponding T1 volume, producing affine transformation matrix
4. Compute the net transformation matrix from the above two steps (fMRI → T1 → template), by multiplying these matrices
5. Spatially transform 4D fMRI dataset with basic preprocessing (called "baseproc") using transform. matrix of Step #4, in order to build group-level functional brain masks
6. Spatially transform optimized SPMs, and optimally processed fMRI datasets, with transform. matrix computed in Step #4
7. Produce downsampled structural T1, to use as a reference.

Appendix E: Sample Quality Control Figures

i. Quality Control-1

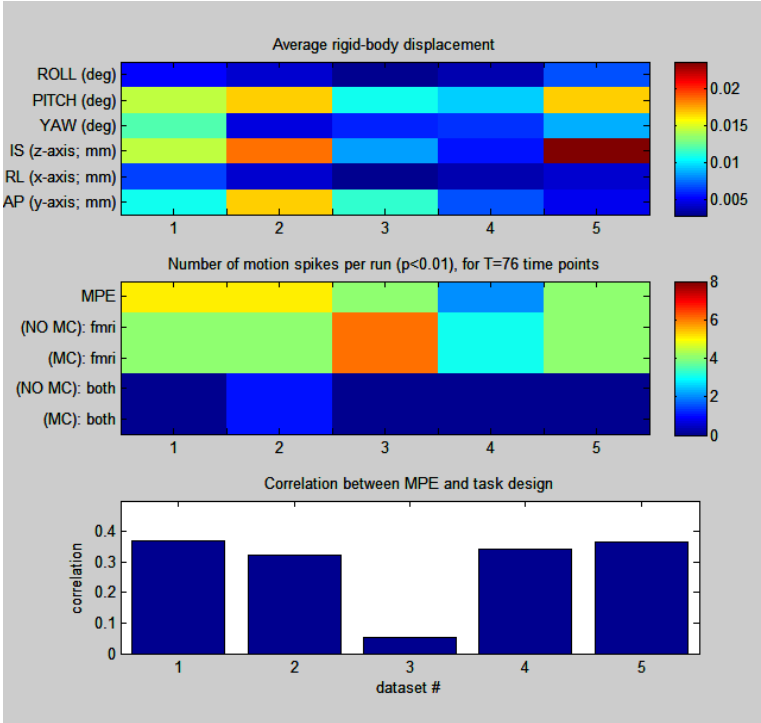


FIG1_motion_statistics: This plot provides some statistics on head motion, for each dataset being optimized (columns correspond to lines in your {subject input} file. Top: average displacement on the 6 rigid-body motion parameter estimates. Middle: the number of volumes identified as head motion spikes during CENSOR pipeline step for MPE (motion parameter estimates), fMRI data (with/without motion correction (MC)), and number of volumes that are spikes in both MPE and fMRI data. Bottom: correlation between task paradigm and first PC of motion parameters – determines if motion is task-coupled.

* If you notice pipelines with consistently high motion amplitude (e.g. >0.05 mm/deg) and strong task correlations (e.g. >0.3), or lots of head motion spikes (e.g. in both MPE and fmri, for >10% of timepoints) examine their data carefully to make sure there are no persistent motion artifacts.

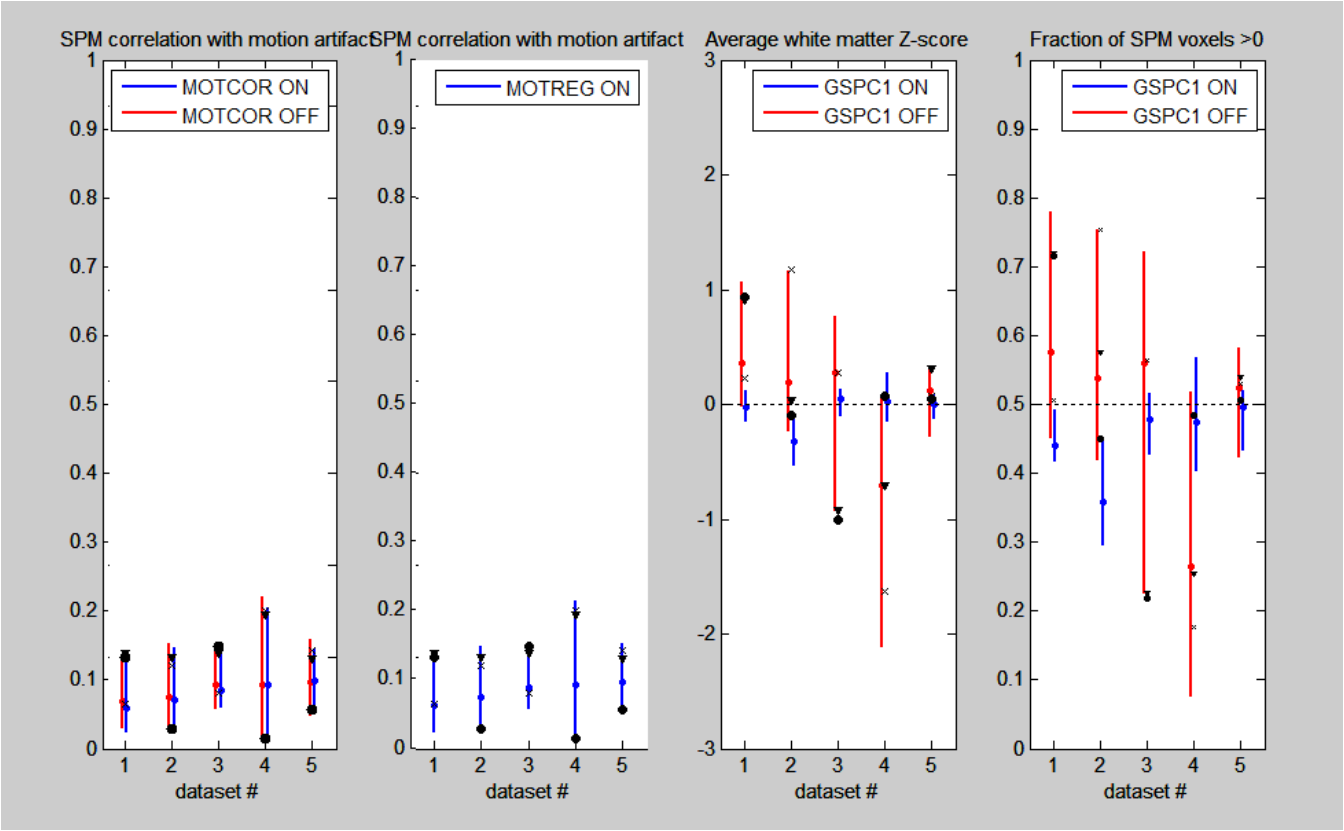


FIG2_SPM_artifact: This plot shows potential artifact present in SPMs, for each dataset being optimized. Vertical bars show distribution across all tested pipelines, per dataset. Going left to right, Panel-1 shows correlation with motion artifact, estimated by spatial gradient maps (for pipelines with/without MOTCOR), and Panel-2 shows the same (for pipelines with/without MOTREG). Panel-3 plots average Z-score in white matter tissues and Panel-4 plots fraction of signal >0 ("globalness" of signal). The latter two panels are shown for pipelines with/without GSPC1 which can sometimes control for these confounds.

* note that if you leave MOTCOR, MOTREG or GSPC1 fixed (on or off) you will only see a single set of bars. In this example, we fixed MOTREG on, which you can see in Panel-2.
* Panels 1,2 should give you an idea of how much motion artifact is present in your SPMs, and whether motion correction has a significant impact on the SPM patterns of your data.
* Panels 3,4 will help to determine if there is persistent global/white matter signal in your data, and whether the current pipelines can fully correct it

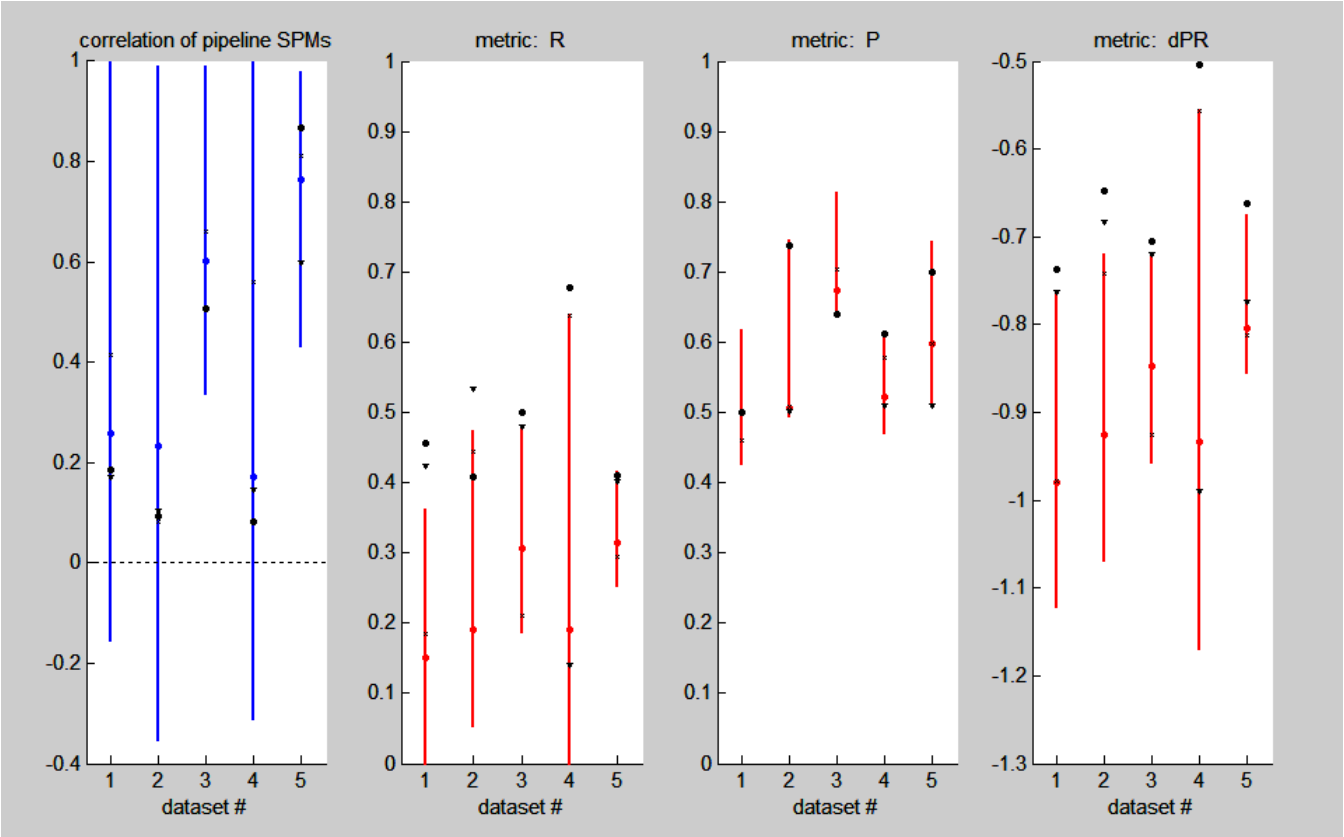


FIG3_pipeline_similarity_by_dataset: This plot shows how much results are altered by different pipeline choices, for each dataset being optimized. Vertical bars show the 95% CI distribution across all tested pipelines, per dataset. Left: pair-wise correlation between all pipeline SPMs. This indicates how similar the SPMs are across pipelines. For example, SPMs of dataset#4 are highly sensitive to pipeline choice (wide range of correlation values, many are negative), whereas SPMs of dataset#5 are relatively insensitive to pipeline choice (consistently high correlations).
All other panels show distribution of NPAIRS metrics across pipelines, with black icons indicating CON ('X'), FIX (triangle) and IND (circle).

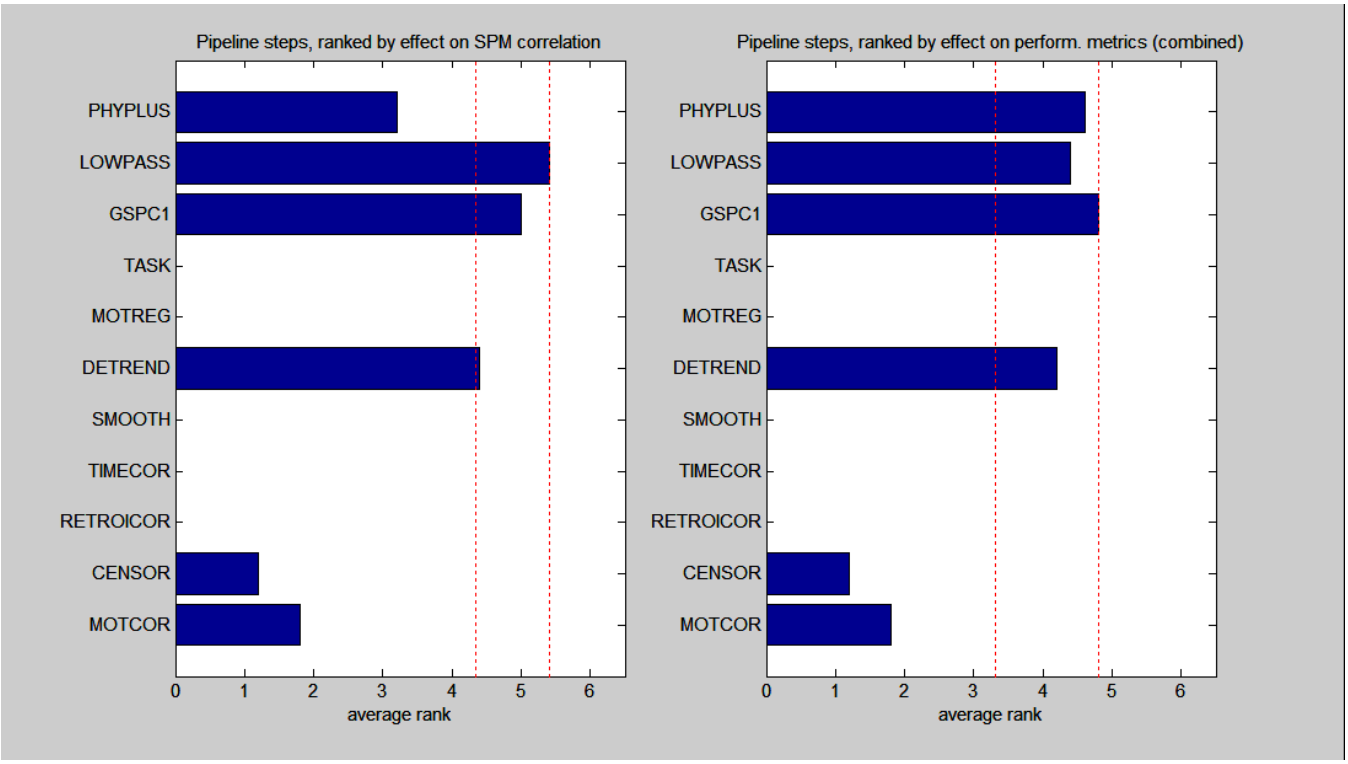


FIG4_effects_of_pipeline_steps: This bar chart depicts the relative influence of individual pipeline steps on results, with higher rank indicating a more influential step; bars that fall between the dashed red lines are the most influential steps for this dataset (Friedman rank test). Panel-1 indicates effect on SPM pattern (based on correlation distance between pipelines with/without each step). Panel-2 indicates effect on performance metrics (absolute change in (Prediction, Reproducibility) between pipelines with/without each step).

ii. Quality Control-2

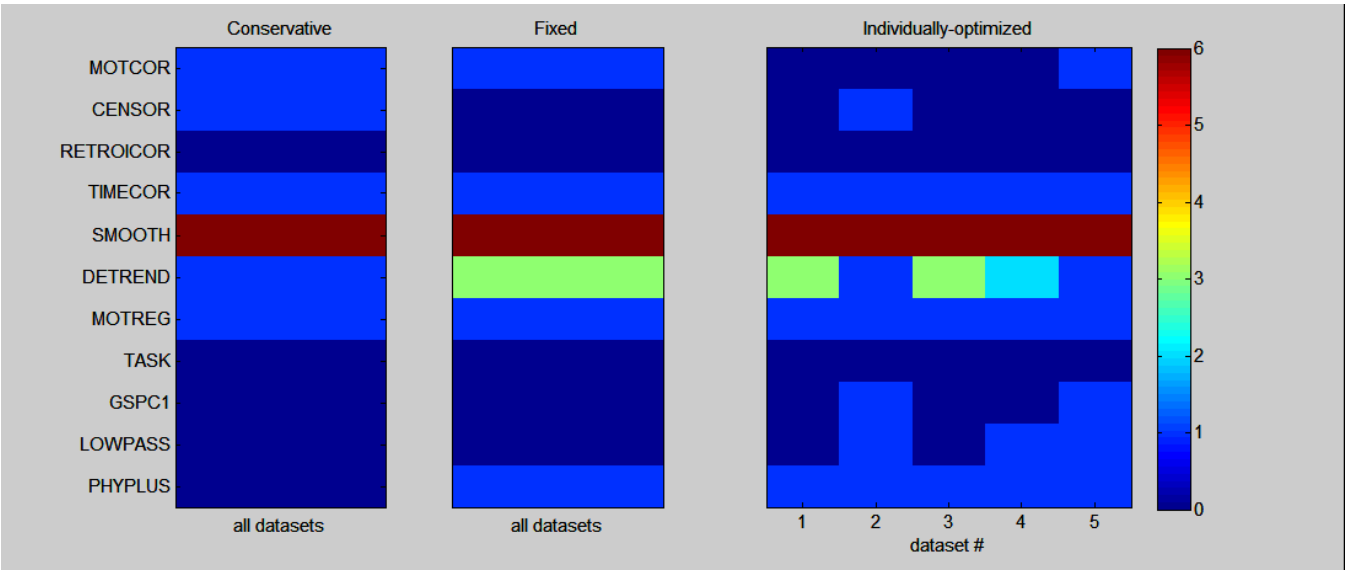


FIG1_optimized_pipeline_steps: List of pipeline choices for each optimization strategy; for the IND pipelines (right panel), pipeline choices are listed for each dataset. For 8/11 pipeline steps, 1=ON 0=OFF. Current exceptions are CENSOR (0=none, 1=basic, 2=aggressive(PCA), 3=aggressive(ICA)), SMOOTH (=kernel FWHM) and DETREND (=maximum polynomial order *k*).

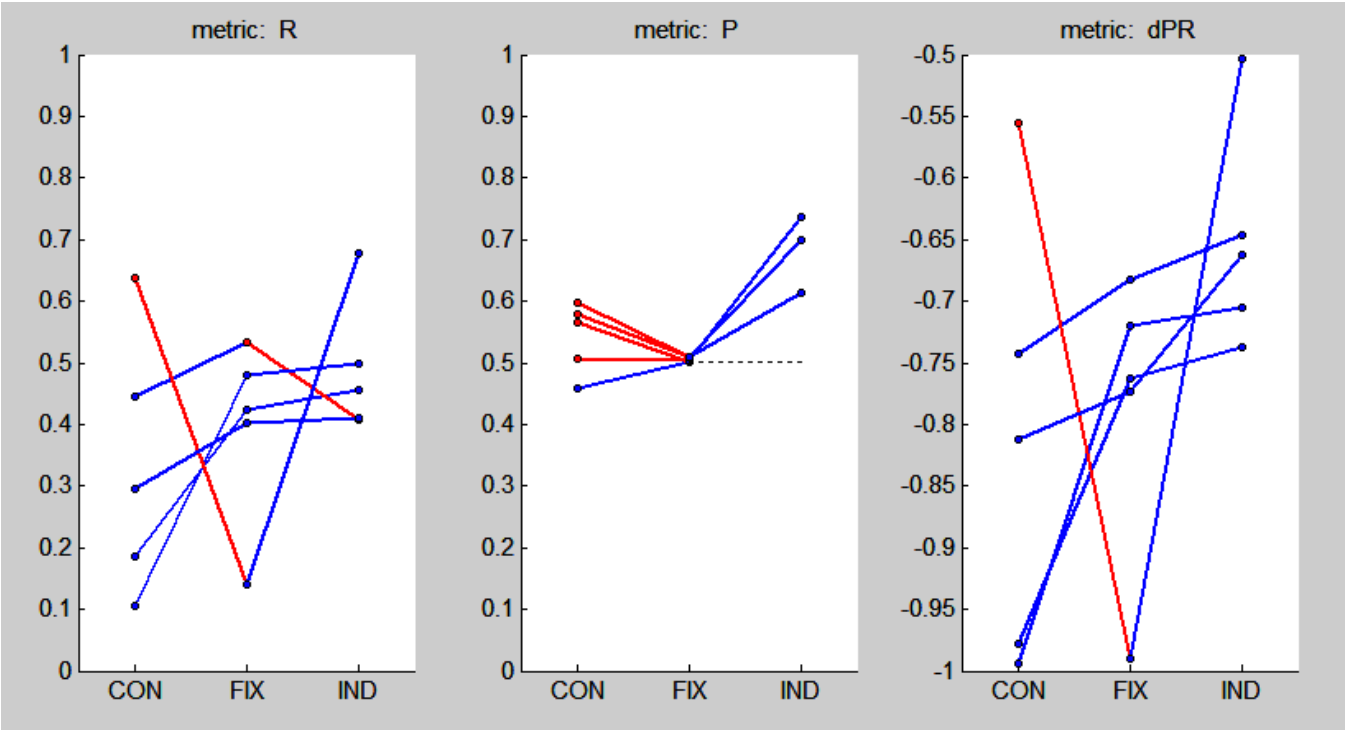


FIG2_optimized_performance_metrics: Performance metrics as a function of pipeline optimization method (STD, FIX, IND), plotted for each dataset. Blue bars indicate improved performance with increasing pipeline flexibility; red bars indicate the converse.

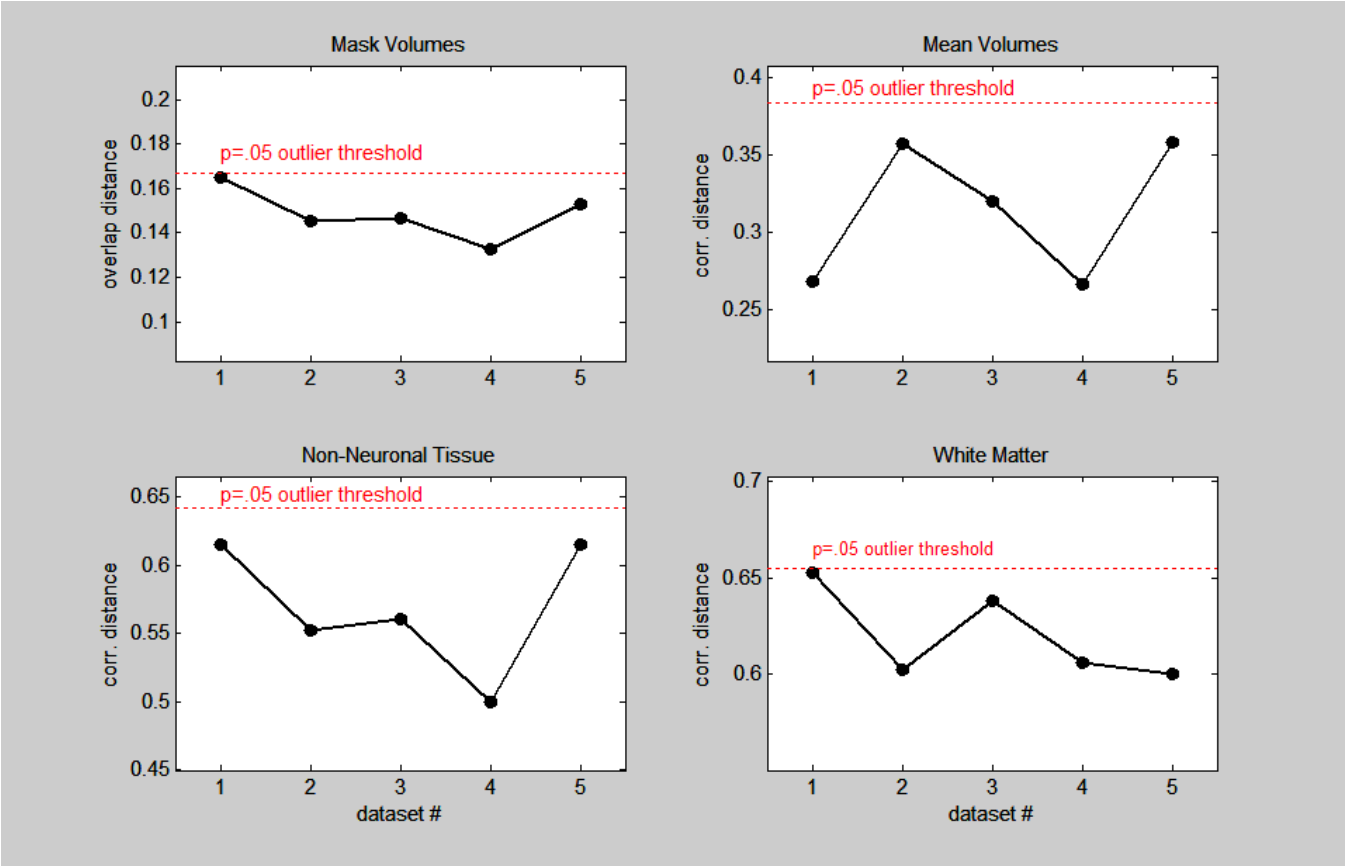


FIG3_spatial_norm_statistics: Metrics reflecting the quality of spatial normalization across subject datasets. Plots indicate mean overlap/correlation distance of each dataset, relative to all others. Dashed red line indicates threshold for significant outliers (p<.05 significance).

* Some further details:

-for generating masks and spatial transformation matrices, I use a single basic pre-processing pipeline (motion correction, slice timing, 6mm smoothing), with is produced as part of the intermediate pipeline outputs (has a "baseproc" suffix).

-we generate masks by applying 3dAutomask to the transformed EPI "baseproc" data from each subject, and mean volumes" are subjects' masked "baseproc" pipelines.

- “overlap distance” is the average (1 – [jaccard overlap]) of each subject map relative to all others

- “correlation distance” is the average (1 – [correlation]) of each subject map relative to all others

- significance is estimated by fitting a maximum-likelihood Gamma distribution to the data, and identifying and subjects with $p<.05$. This is going to be a bit conservative, since the distribution is fitted to the subjects themselves, but if there is an outlier, you can be more confident that there is a potential abnormal normalization result that should be checked out

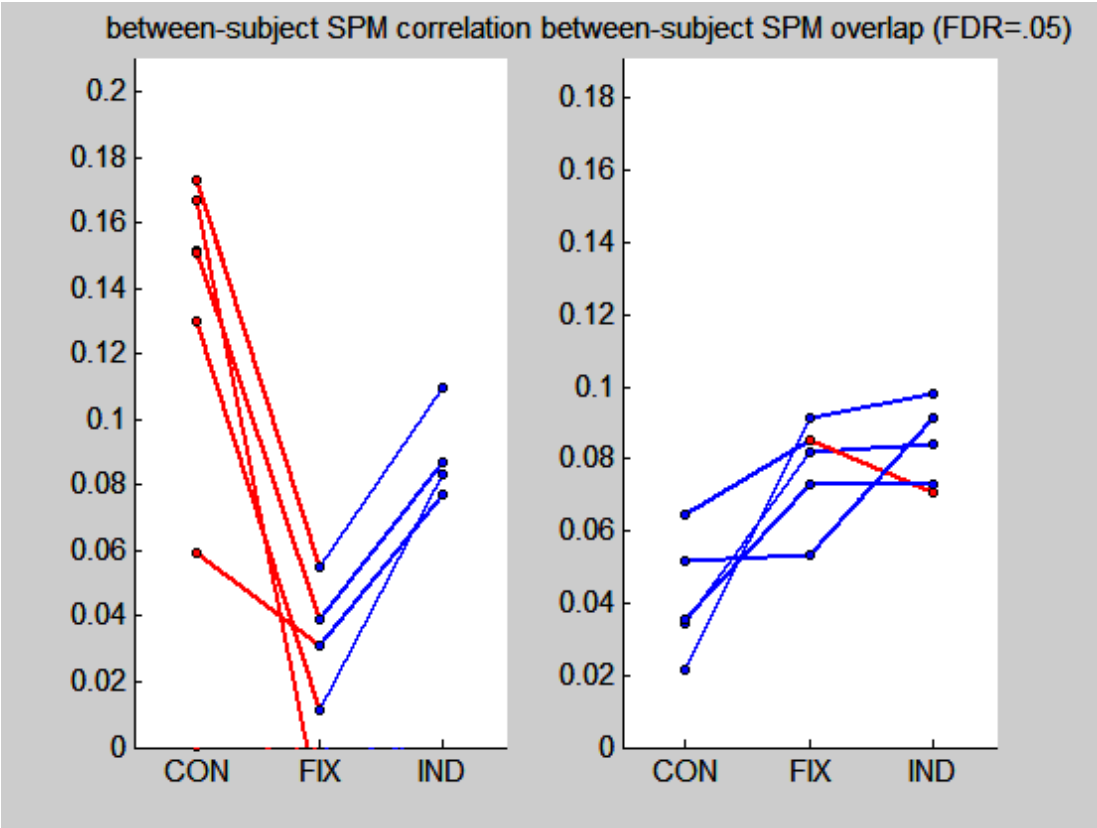
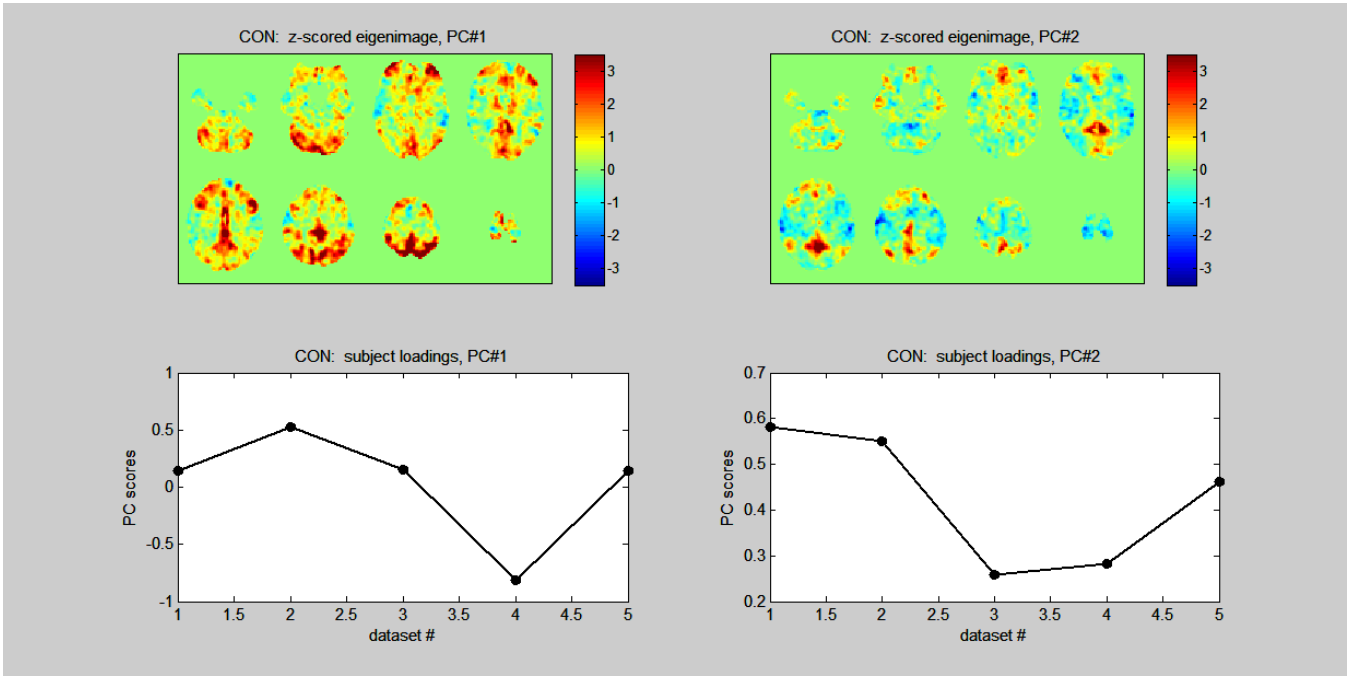
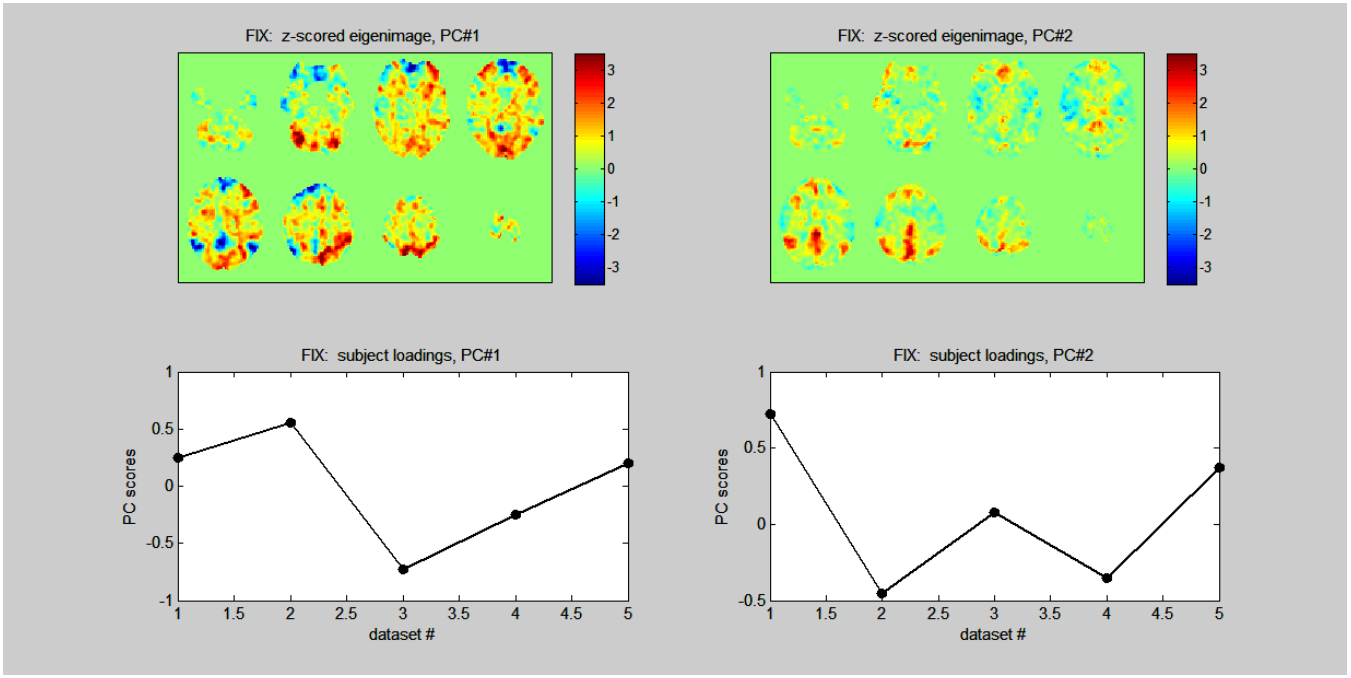
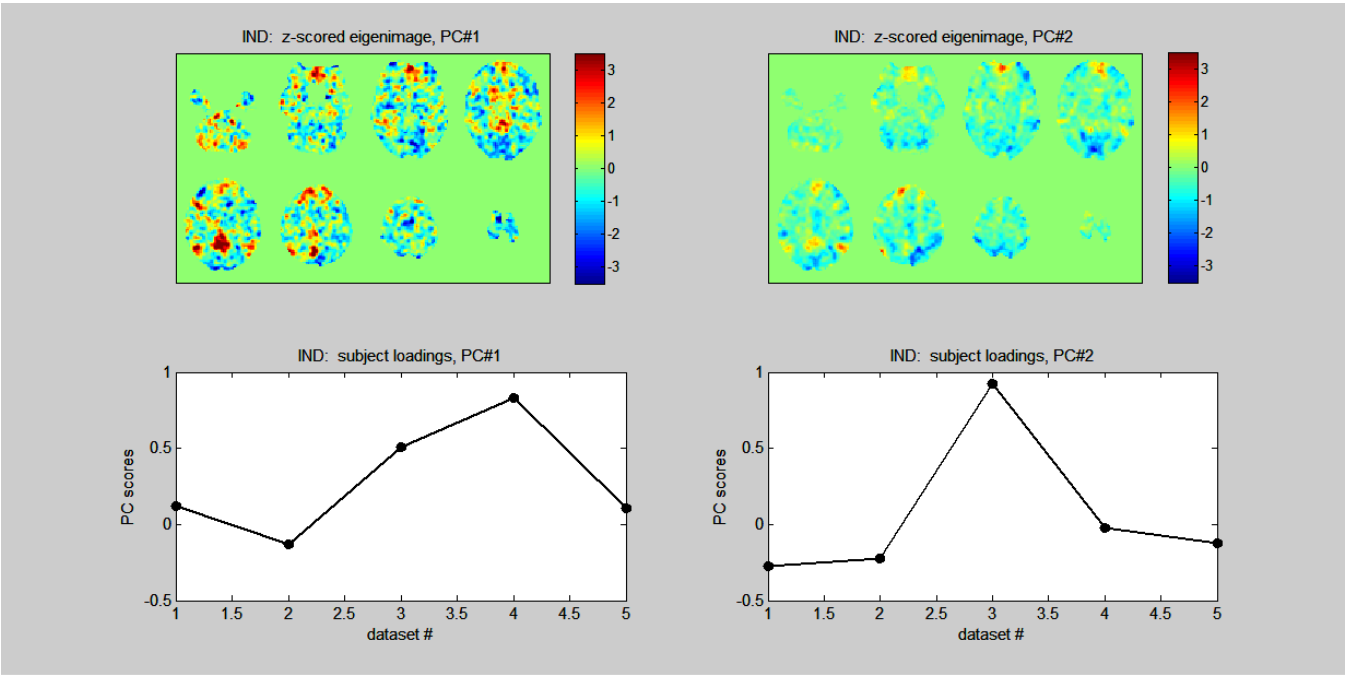


FIG4_inter_subject_SPM_similarity: inter-subject reliability metrics of (left) SPM correlation and (right) overlap of thresholded SPMs, as a function of pipeline optimization method, plotted for each dataset. SPMs are thresholded at False-Discovery Rate FDR=.05

FIG5_group_pca_{STD/FIX/IND}_pipeline: plots showing Principal Components (PCs) #1,2 on subject SPMs, as a function of pipeline optimization methods. Brain maps are spatially Z-scored, and subject loadings are plotted below.
* The PCA results show "networks" of brain regions that tend to be found in subject SPMs. Subject loadings indicate how strongly this network is present in each subject.


CON: z-scored eigenimage, PC#1
CON: z-scored eigenimage, PC#2
CON: subject loadings, PC#1
CON: subject loadings, PC#2
PC scores
dataset #


FIX: z-scored eigenimage, PC#1
FIX: z-scored eigenimage, PC#2
FIX: subject loadings, PC#1
FIX: subject loadings, PC#2
PC scores
dataset #


IND: z-scored eigenimage, PC#1
IND: z-scored eigenimage, PC#2
IND: subject loadings, PC#1
IND: subject loadings, PC#2
PC scores
dataset #

Appendix F: Details of Split-half Pipeline Optimization

Every single pipeline for all subjects and all runs are put through the NPAIRS (Nonparametric Prediction, Activation, Influence, and Reproducibility reSampling) framework (Strother et al., 2002), using the user-selected analysis model. NPAIRS uses a method called split-half resampling, where the scans from one half of a run are compared to its other half. The splits are independent of each other; one is the 'training' set and the other is the 'test' set. The user-chosen analysis method is applied to each split to generate two spatially z-scored SPMs.

NPAIRS calculates two quantitative statistics that reflect the consistency of the data across splits without having to set an SPM detection threshold, see Fig. E-1. The first is reproducibility, which is a measure of similarity between two SPMs. Reproducibility (R) in NPAIRS is obtained by measuring the correlation between all pairs of unthresholded brain voxel values between the z-scored SPMs of the two split groups. Prediction (P) is calculated as the median rate of using the first split's analysis model to correctly predict the class of each scan in the second split with posterior Bayesian probability, and vice versa. P ranges from 0.5→1 and R ranges from 0→1, with perfect P and R both at 1.

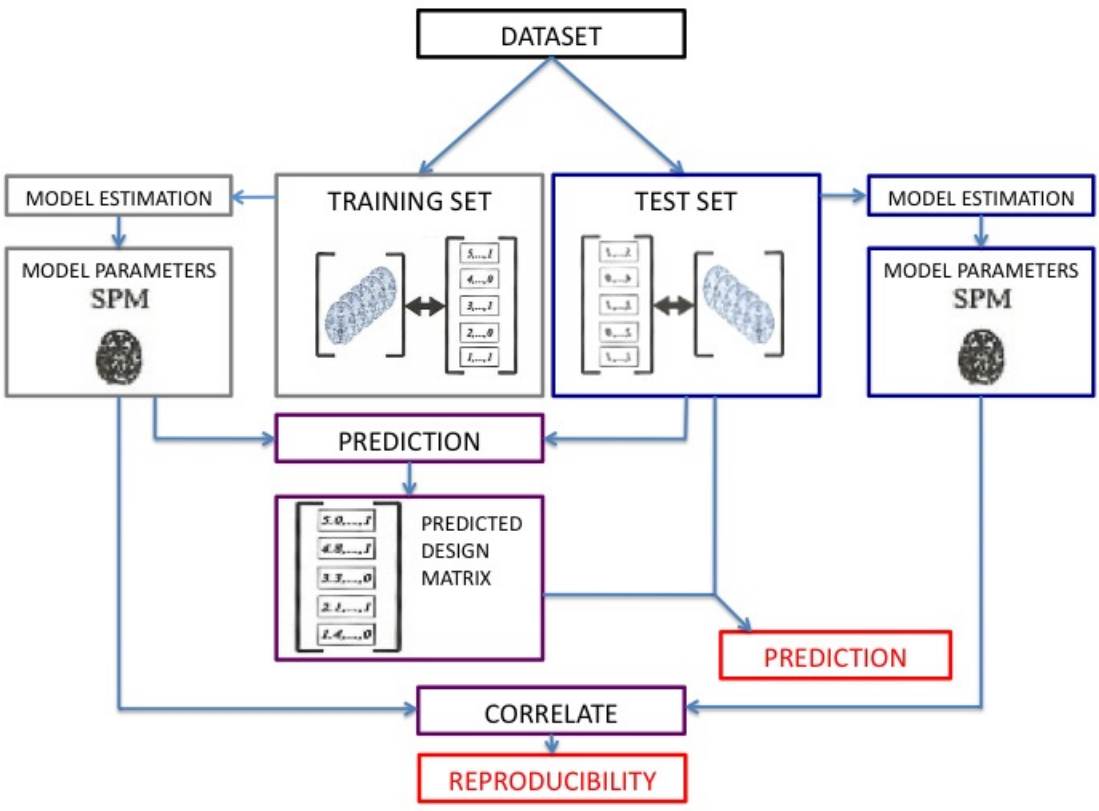


Figure E-1 Reproducibility and Prediction in NPAIRS
Illustration of how reproducibility and prediction metrics are calculated by split-half resampling in the NPAIRS framework.

i. Determining the Optimal Fixed Pipeline (FIX)

An optimal fixed pipeline is the "best" fixed pipeline on average across all subjects. Following NPAIRS analysis, the R and P values are obtained and used for determining the optimal fixed pipeline. The optimal fixed pipeline is chosen using the procedure outlined by Churchill et al. (2011), see Fig. E-2:

1. Each subject's pipeline performance is assessed via D, quantifying how close performance is to having the perfect model (R=1, P=1), see Fig. E-2A.
2. Pipelines per subject are ranked by arranging the D's in a matrix and assigning the smallest D (closest to perfect) as the highest rank, which forms a matrix of ranks, see Fig. 2.5B.
3. The median of each subject's ranking matrix is calculated for each pipeline across subjects. This was the 'median rank metric', see Fig. 2.5C.
4. Significance of the optimal fixed pipeline is tested using a Friedman test.

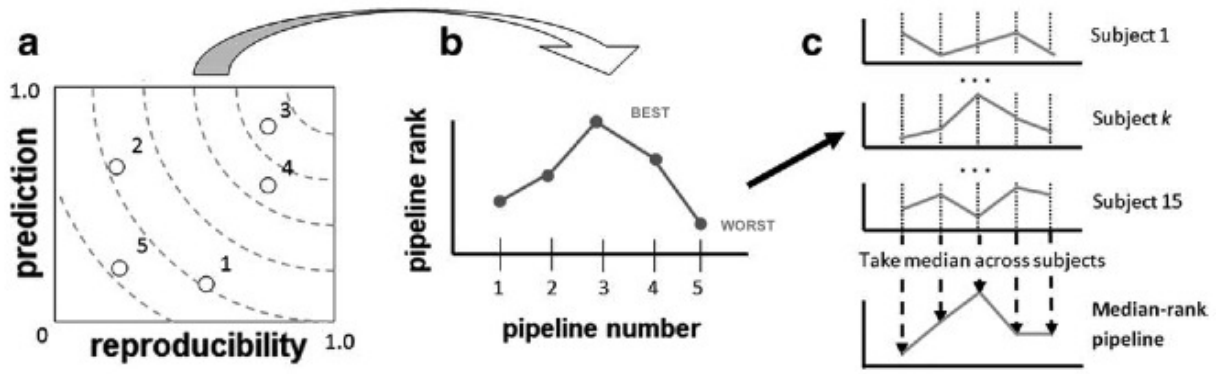


Figure E-2 Median Rank Metric
The procedure for identifying the optimal fixed pipeline across subjects in a 5 pipeline example. Euclidean distance, D, is obtained per pipeline to measure the pipeline closest to perfect R and P (A), ranked from least optimal (lower score) to most optimal (higher score) (B) for all subjects, and then median rank is obtained for each pipeline across all subjects, producing a median-rank profile (C). Adapted from Churchill et al. (2011) (Churchill et al., 2011).

ii. Determining the Optimal Individual Pipeline

Pipelines that are optimal for each individual subject usually have increased R and/or P relative to most of the other pipelines, and the preprocessing steps included generally differ from the fixed pipeline (Churchill et al., 2011). The optimal individual pipeline is chosen by calculating D (mentioned above) per pipeline per subject and selecting the pipeline per subject, without reproducible spatial artifact, that produces the lowest D-value.). Individual subjects may be prone to task-coupled motion, which generates artifact with high spatial Reproducibility. For this reason, IND optimization includes an addition step (established in Chuchill et al. (2012a)) to automatically reject pipelines corrupted with motion artifact before choosing the optimal one.

Appendix G: Compilation Instructions

The OPPNI software package is mainly written in Matlab. If your computing environment has installed Matlab and you have a license, you can run directly using the "-e matlab". You can alternately use "-e octave" (if Octave has been installed) argument when submitting jobs; see Table 6).

Many computing environments do not have Matlab licenses, including most large HPC clusters. In this case, you will need to run OPPNI using compiled software (using the "-e compiled" argument). While OPPNI comes with pre-compiled software, you may want to recompile yourself -- **note that this is for experienced users only, otherwise you should use the pre-compiled software**.

If you do want to recompile, you will require an environment with a Matlab license and Matlab Component Runtime (MCR) to create the files, which can then be run in a non-Matlab environment. As an example, the **QC script** can be compiled using the following command:

```
cd <Path-To-OPPNI-Root-Folder>
mcc -v -d extra/compiled -o QC
-I scripts_matlab -I scripts_other -I extra
-I scripts_matlab/NIFTI_tools -I scripts_matlab/optimization
-R -singleCompThread
-m extra/QC_wrapper.m
```

This would create an executable called QC in the `extra/compiled` folder. The qc_wrapper.py utilizes this (when requested via -e standalone option) to run the QC computations and generate the necessary visualizations and diagnostic outputs.

Appendix H: Pipeline Stage Outputs and Descriptions

Outlined in this section are the files produced by oppni in each stage of analysis and a description of what they are.

Note that the <path>/directory is the <path> you specified in your input file "OUT=" plus the additional processing level directory added in by the python wrapper. The new directory always starts with "processing" and contains details based on the parameters you selected to be used in the analysis.

More information can be found in this Document:
https://docs.google.com/document/d/1ln5RfksY20O1PcZS87U5E9cb1eAwTXUmO6mVK7PSeAk/edit

Table 1:OPPNI Output Structure		
Directory	**Description**	**Created By:**
<path>/intermediate_processed		
afni_processed	files produced by combinations of AFNI steps (MOTCOR,CENSOR,TIMECOR,RETROICOR,SMOOTH)	
diagnostic	files of diagnostic data for quality control (also used in CENSOR)	Part1-FSL melodic
masks	subject brain masks (produced by *3dAutomask*)	Part1-AFNI 3dAutomask
mpe	rigid-body motion parameter estimates (produced by MOTCOR, used in MOTREG)	Part1-AFNI 3dvolreg
split_info	data from subject input files, reformatted as .mat files	Part 1
<path>/intermediate_metrics		
regressors	contains all design matrices used for different processing pipelines (e.g. task+noise covariates)	Part 1
res0_params		Part 1
res1_spms	contains some parameters of task design	Part 1
res2_temp	contains activation times series of *all* tested pipelines	Part 1
res3_stats	contains NPAIRS metrics (P,R) of *all* tested pipelines	Part 1
<path>/optimization_results		

matfiles	.mat files containing individual summary info from optimized subject pipelines	Part 2
	A summary of all the .mat files combined into a single structure.	Part 2
processed	optimally preprocessed subject datasets	Part 1 Part 2
spms	analysis SPMs from optimally preprocessed subject data	Part 1 Part 2
<path>/GroupMasks		
		wrapper
<path>/QC1_results		
		QC1
<path>/QC2_results		
		QC2
<path>/input_files		
*.input.txt	These files are the string split version of the main input file generated by the wrapper. There should be one for each part of the pipeline for each subject.	wrapper
<path>/job_files		
.job files	A .job file is generated for every pipeline section for each participant.The files contain the command that is executed in the terminal that executes the .m file. It specifies the program to use matlab/octave and the .m file to run.	wrapper
.m files	A .m file is generated for every pipeline section for each participant.The files contain the code that is run in matlab/octave which usually contains a path set-up as well as a call to the main processing function with the desired settings.	wrapper
.log files	A .log file is generated for every pipeline section for each participant. The log file contains the output of the matlab/octave display. It is useful for debugging when files do not complete, or for monitoring a file's progress. Rerunning files will not overwrite the logs.	wrapper
_all_subjects	A single file for each of the above extensions is generated for the Part2, Gmask, QC1 and QC2 sections of the pipeline, since these require the data from all the subjects.	wrapper

Miscellaneous Lose Files		
hpc_config.json	A JSON file creates if HPC resources are being used. File contains info for the parameters used on hpc including memory, cores an job IDs for each job requested..	wrapper
pronto_job_ids_by_step_prefix.json	File is same as above but only contains the job ID numbers for each job requested.	wrapper
input_file.txt	A copy of the input file used to call the wrapper, but with the updated OUT paths adjusted to include the unique processing directory added by the wrapper.	wrapper
-resubmit-	These files are similar to the input_file but are specific to the files that were selected for a rerun of failed jobs. Only the failed jobs will appear in this file.	wrapper
Miscellaneous Lose Files (CBRAIN)		
Task_Complete_CBRAIN.txt	This file is generated upon a successful local run of the pipeline. Its sole purpose is to alert CBRAIN of a successful completion of the task. Absence of file during a local run indicates that one or more of the pipeline modules was not successful.	wrapper

Appendix I: Running OPPNI with Octave

Octave is a free programming language that can read MATLAB syntax. This allows OPPNI's code to run without the need to compile or have a MATLAB license.

Below see additional steps you will need to do to set up the Octave environment for OPPNI.
This assumes you have already read and set up OPPNI as described in APPENDIX A.

If you haven't already use these lines into your HPC command terminal.

```
echo "export OPPNI_PATH_MATLAB_ORIG=/home/raamana/software/oppni/cPRONTO" >>
~/.bash_profile
cd home/USERNAME
python /home/raamana/software/oppni/octave_setup_cc.py
```

These lines will do a few important things:

1. First sets up an Environment variable allowing you to reference the MATLAB scripts , since the new default oppni will be running Octave.
2. Second calls a script that contains a script developed to take care of the rest. It does the following:

    a. Load the Octave module of the HPC system
    b. Installs Octave Packages(toolboxes) into your home directory. And sets their path.
    c. Generates a .octaverc startup file, which will load you packages and set Octave setting to match MATLAB's each time Octave is used.

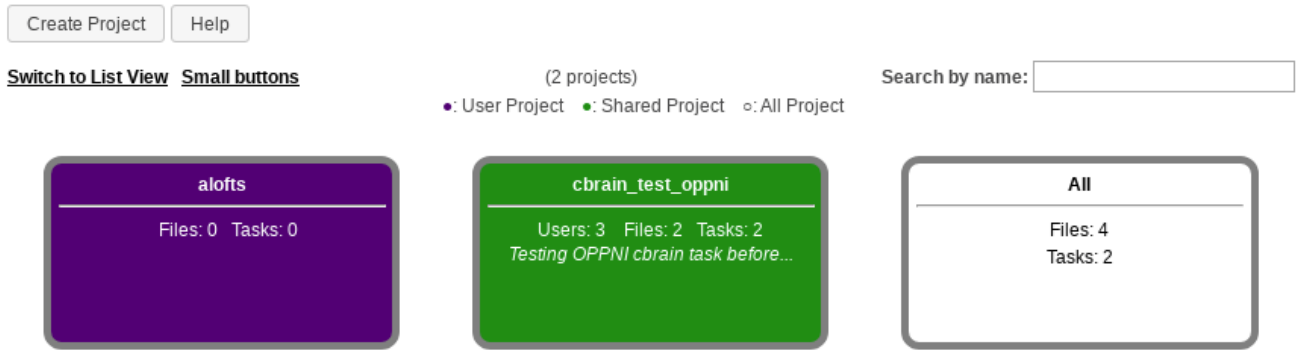
Appendix J: Running OPPNI via CBRAIN

CBrain is an online portal that resources can use to gain access to HPC for specific tasks such as oppni. The Oppni CBRAIN tool can be accessed from the CBRAIN Portal and allows users with limited HPC and unix experience to use our pipeline. Environment setup is done automatically and analysis options can be entered using a GUI. However this should only be used for studies with a small number of subjects as the processing time of the tool has to do each subject one by one. Follow the following steps to set up your CBRAIN account, adjust oppni data, upload projects and run the pipeline.

Getting Started:

You will need a cbrain account which you can get at:
https://mcin-cnim.ca/technology/cbrain/

Create a project folder. Highlight the project and click the settings icon, add any other users ass necessary.

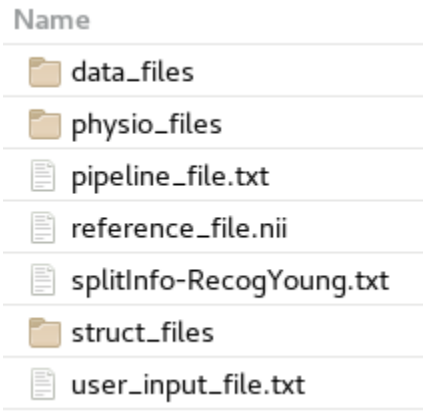


Adjusting OPPNI Files

You will have to make 3 small adjustments to a normal oppni dataset in order to get running on CBRAIN.

1. Adjust the input file to contain arbitrary paths. As described earlier in the manual:
    a. Make sure all of you paths have a common directory
        i. Ex. IN=/**mydirectory**/input_session/subject1_data.nii
        ii. Ex. OUT=/**mydirectory**/outputs/subject1_data.nii … and so on
    b. Replace your common directory with /CBRAIN_PROCESSING_LOCATION.
        i. Ex. IN=**/CBRAIN_PROCESSING_LOCATION**/input_session/subject1_data.nii
        ii. Ex. OUT=**/CBRAIN_PROCESSING_LOCATION**/outputs/subject1_data.nii

    The oppni CBRAIN task will adjust these paths for you depending on the HPC system and location it selected to run you data on.
2. Rename your file to be *"**user_input_file.txt**"*, this is a special name that cbrain will be able to find, adjust and rename to "user_input_file_cbrain.txt".
3. Make sure that you have actually moved all of these references to the common directory! You file folder should look something like:



Example Text for File#1:

For 1 run from each of 5 different subject datasets, you would produce a text file with the 5 lines below. Note that "CUSTOMREG" field is omitted, meaning that this pipeline step will be fixed OFF. This file will not run normally since /CBRAIN_PROCESSING_LOCATION does not exist, but this placeholder will be replaced by CBRAIN to a viable path when the task is run.

```
IN=/CBRAIN_PROCESSING_LOCATION/subj1_task.nii OUT=/CBRAIN_PROCESSING_LOCATION/subj1 DROP=[2,1] TASK=taskInfo_subj1.txt
        PHYSIO=/CBRAIN_PROCESSING_LOCATION/physio/subj1 STRUCT=/CBRAIN_PROCESSING_LOCATION/subj1_T1.nii
IN=/CBRAIN_PROCESSING_LOCATION/subj2_task.nii OUT=/CBRAIN_PROCESSING_LOCATION/subj2 DROP=[2,1] TASK=taskInfo_subj2.txt
        PHYSIO=/CBRAIN_PROCESSING_LOCATION/physio/subj2 STRUCT=/CBRAIN_PROCESSING_LOCATION/subj2_T1.nii
IN=/CBRAIN_PROCESSING_LOCATION/subj3_task.nii OUT=/CBRAIN_PROCESSING_LOCATION/subj3 DROP=[2,1] TASK=taskInfo_subj3.txt
        PHYSIO=/CBRAIN_PROCESSING_LOCATION/physio/subj3 STRUCT=/CBRAIN_PROCESSING_LOCATION/subj3_T1.nii
IN=/CBRAIN_PROCESSING_LOCATION/subj4_task.nii OUT=/CBRAIN_PROCESSING_LOCATION/subj4 DROP=[2,1] TASK=taskInfo_subj4.txt
        PHYSIO=/CBRAIN_PROCESSING_LOCATION/physio/subj4 STRUCT=/CBRAIN_PROCESSING_LOCATION/subj4_T1.nii
IN=/CBRAIN_PROCESSING_LOCATION/subj5_task.nii OUT=/CBRAIN_PROCESSING_LOCATION/subj5 DROP=[2,1] TASK=taskInfo_subj5.txt
        PHYSIO=/CBRAIN_PROCESSING_LOCATION/physio/subj5 STRUCT=/CBRAIN_PROCESSING_LOCATION/subj5_T1.nii
```

Uploading Data

Oppni uses a common large folder filled with many kinds of files. You may be able to zip your project and drag and drop it into CBRAIN, but it has a limit of 500MB. Instead what you will most likely have to do is use sftp to move the data to a data provider, and then register the files onto the CBRAIN mainservers from there. The necessary information is contained here: https://portal.cbrain.mcgill.ca/doc/manual/uploading.html

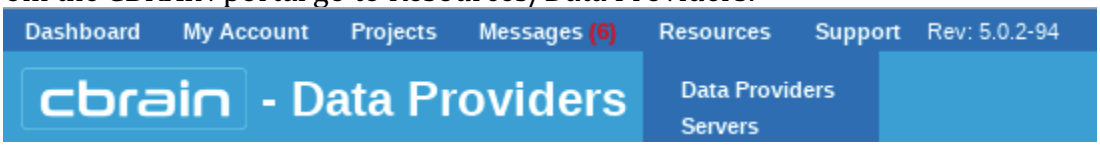
Using your CBRAIN username and password:

```
unix% sftp -o port=7500 myusername@ace-cbrain-1.cbrain.mcgill.ca
mkdir project_name
put /full/path/to/local/project_name
```

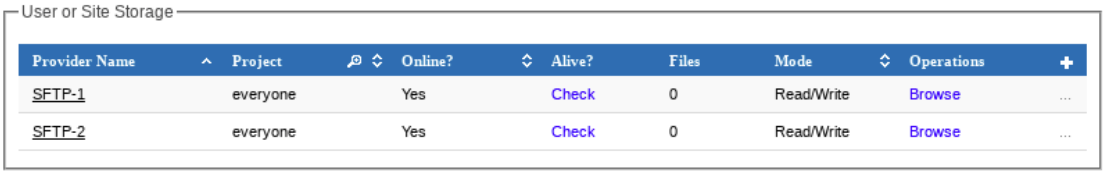
Make sure the **project_name** is the same for both.

Register the Data

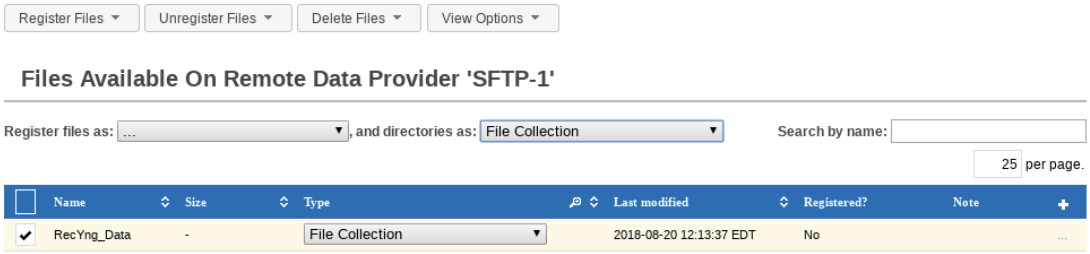
1. From the CBRAIN portal go to Resources/Data Providers.



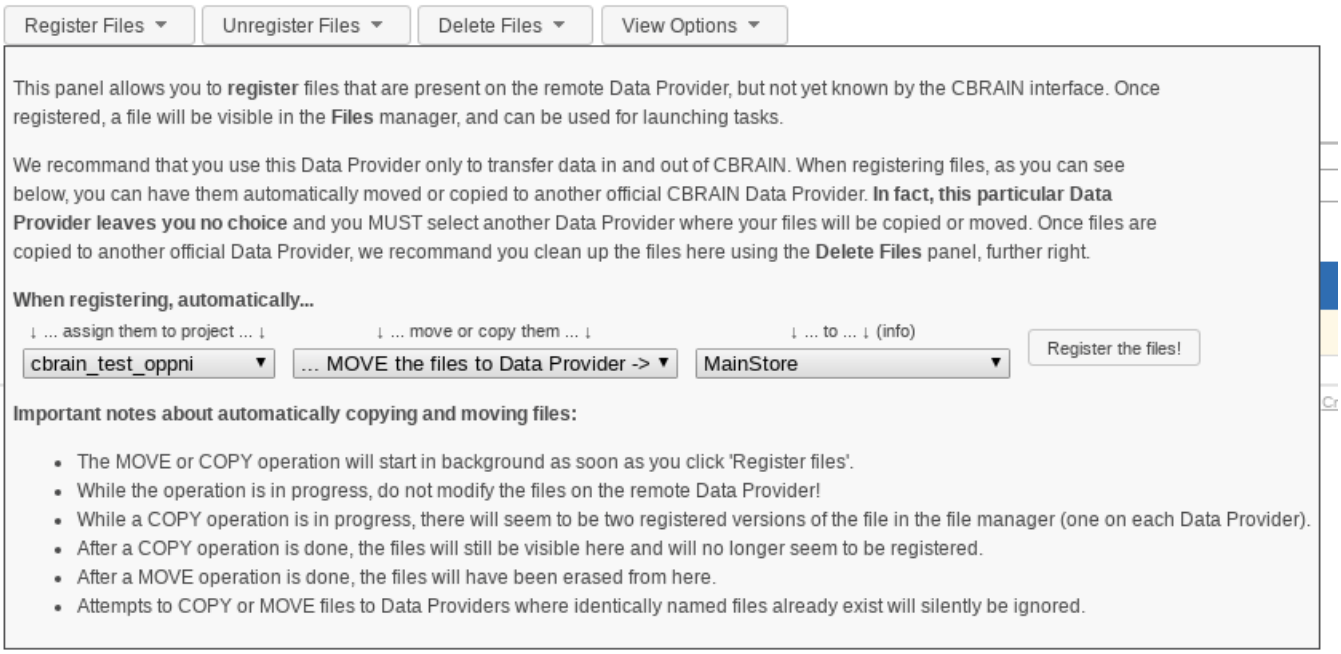
2. Click on “Browse” under the SFTP-1 Provider.



3. Select the checkbox beside your data folder, and make sure “directories as”: is set to “File Collection”. You can leave “Files as” set to … Then click “Register files”.



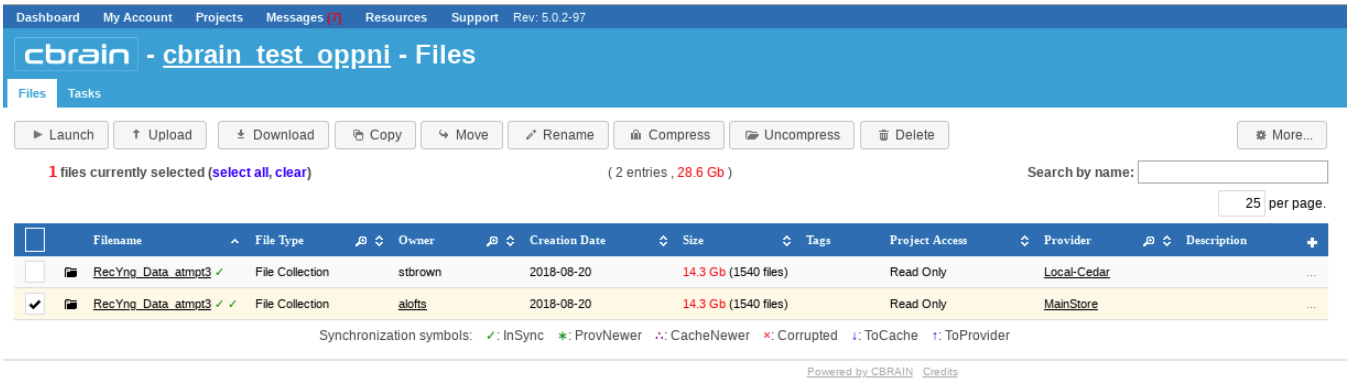
4. Select the project where you would like the data to go, and specify MOVE or COPY. You most likely will want to store the data in the default MainStore.



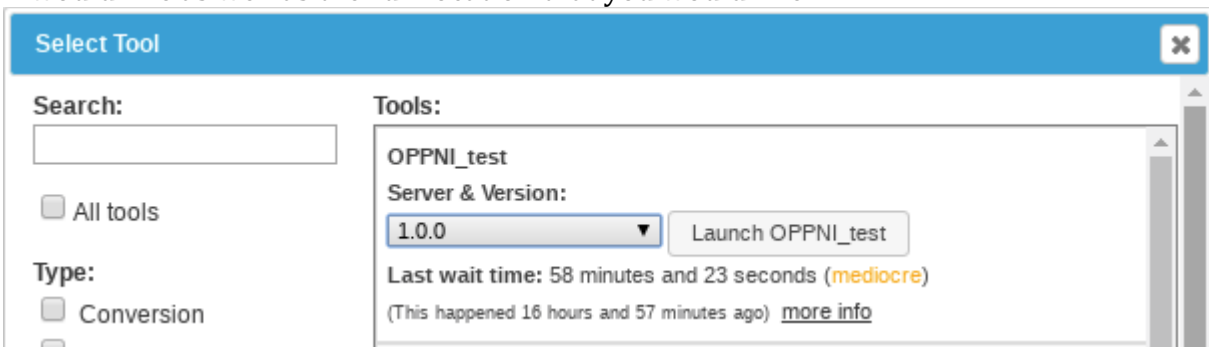
5. After selecting “Register the Files”, return to your projects to see the data folder.

Launching Oppni

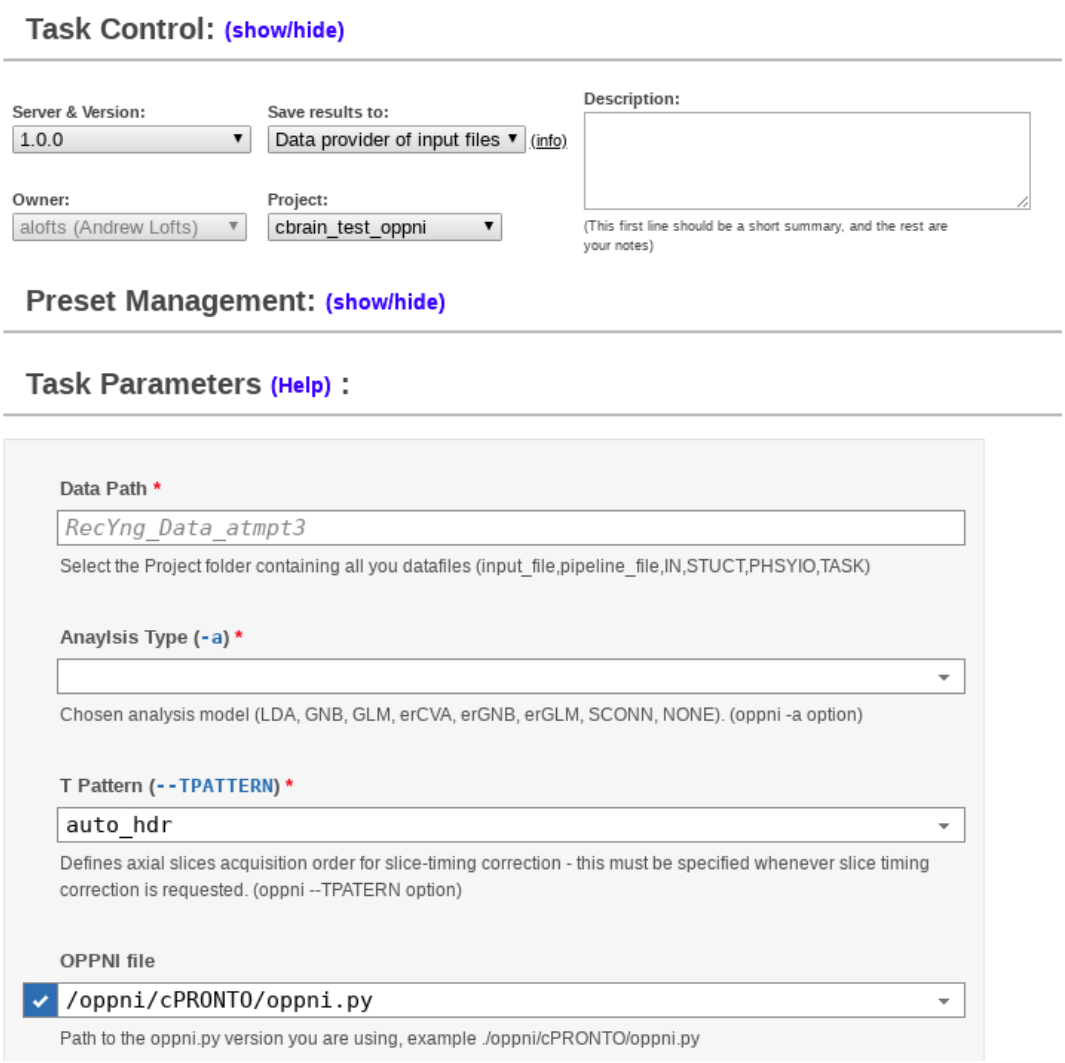
1. To start the tool, select your file collection and click the launch button.



2. Select OPPNI from the list of tools. From the drop down selection, pick the version that you would like as well as the run location that you would like.



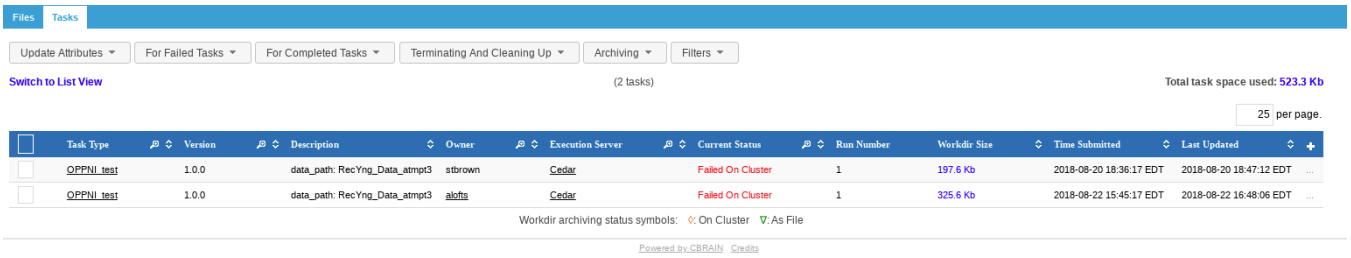
3. Fill out the GUI with the information you would be used to crete the oppni call. Note that many options are preselected to the defaults, and that red stars (*)indicate mandatory fields.



4. These selections will create the bash call needed to run Oppni. From the bottom of the GUI menu you can review your selections.

OPPNI file	/oppni/cPRONTO/oppni.py
Anaylsis Type	
T Pattern	auto_hdr
Start at Point	0
Convolve Value	0
Metric	dPR
Contrast Combination	None
Vascular Mask	1
Keep Mean	0
Blur to FWHM	
Control Motion Artifact	yes
Control WM Bias	no
Output the nii also	0

5. Once you have your selections completed click the Start OPPNI button at the bottom of the page.
6. At this point the job will be prepared and submitted to the HPC scheduler. You can review the task's progress by finding it under the "task" tab.



Retrieving the Data

If the job is successful you will see an indication….