clpipe: A MRI Processing Pipeline for HPCs Documentation

Release .1

Cohen Lab at UNC-CH

Sep 18, 2023

DOCUMENTATION

1 Installation 3

2 Overview 5

3 Project Setup 9

4 DICOM to BIDs Conversion 11

5 BIDS Validation 17

6 Preprocessing with fMRIprep 19

7 Postprocessing 23

8 GLM Analysis 37

9 ROI Extraction 45

10 Flywheel Sync 49

11 Change Log 53

12 Project Setup 59

13 BIDS Conversion 63

14 BIDS Validation 71

Index 75

clpipe was developed to streamline the processing of MRI data using the high performance cluster at University of North Carolina at Chapel Hill. It uses fmriprep for preprocessing fMRI data and implements a variety of additional processing steps important for functional connectivity analyses such as nuisance regression and filtering. Also included in clpipe are a variety of console commands for validation and checking the results of preprocessing. Please report any bugs, issues or feature requests on our Github page.

CHAPTER

ONE

INSTALLATION

Please note - if you're a UNC-CH user, clpipe is already installed and accessible with the module system - please see the section below, "For UNC-CH Users"

1.1 Python Environment Setup

clpipe requires Python v3.7. If you have the priviledges to add python packages to your system, you can install the most recent version of clpipe with:

```
pip3 install --upgrade git+https://github.com/cohenlabUNC/clpipe.git
```

If you don't have access to the global library (perhaps you are just a user of an HPC), you can install a local copy by adding the `--user` flag:

```
pip3 install --user --upgrade git+https://github.com/cohenlabUNC/clpipe.git
```

Pip will automatically install all required Python package dependencies.

1.2 External Dependencies

1.2.1 Singularity & Images

clpipe uses Singularity to run certain dependencies as images. clpipe has been tested against:

- Singularity == v3.2.1

If you are a UNC-CH Longleaf user, Singularity is made available by default when launching jobs, so you do not need to explicitly add this dependency.

The following programs are required as images:

- fMRIPrep >= v20
- BIDS-validator >= v0.0.0

If you don't already have a Singularity image of fMRIprep, head over to their site and follow the directions. You will have to change the fMRIprep image path in your configuration file.

Similarly, if you do not have a copy of the BIDS-validator Singularity image, you'll need to obtain this image as well:

1.2.2 Other Dependencies

Additionally, clpipe requires the following tools to be installed in order to run its postprocessing and analysis steps (UNC-CH Users - this is handled by the clpipe module):

- FSL >= v6.0.0
- AFNI >= v20.0.00
- R >= v4.0.0

1.3 For UNC-CH Users

If you are a Longleaf user and a member of the rc_hng_psx group, clpipe has already been installed for you via the module system.

clpipe is not currently available as part of Longleaf's default module collection. Instead, it is provided through the HNG's module directory, which you must setup manually.

First, make the HNG modules available:

```
module use /proj/hng/software/module
```

Now save this module source to your default collection:

```
module save
```

You can then use the following to access the latest version of clpipe at any time:

```
module add clpipe
```

You also already have access to the latest singularity images for both fmriprep and the bids validator at `/proj/hng/singularity_imgs`, so there is no need to construct your own, unless you want a older version.

1.4 Batch Languages

clpipe was originally designed for use on the University of North Carolina at Chapel Hill's HPC, Longleaf, which uses the SLURM task management system. The way clpipe handles what batch language to use is through a set of batch configuration files. These files are not directly exposed to users, and modification of these directly is ill advised. For other institutions that use task management systems other than SLURM, get in touch with the package maintainers, and we would be happy to help setup a configuration file for your system. In coming versions of clpipe, functionality will be added to allow users to change the batch management system settings.

CHAPTER

TWO

OVERVIEW

2.1 Configuration Files

clpipe is driven by configuration files, and most commands in clpipe require a configuration file path via their '-config_file' option. These configuration files are JSONs that contain all aspects of the preprocessing and postprocessing streams that you want applied to your dataset. clpipe provides you with a default configuration file after using the *project_setup* command. To create addition configuration files for your dataset, use the following command:

This command will create a default configuration file with whatever name you specified. The top of the default configuration file looks like this:

```
{
    "ProjectTitle": "A Neuroimaging Project",
    "Authors/Contributors": "SET AUTHOR",
    "ProjectDirectory": "",
    "EmailAddress": "SET EMAIL ADDRESS",
    "SourceOptions": {
        "SourceURL": "fw://",
        "DropoffDirectory": "",
        "TempDirectory": "",
        "CommandLineOpts": "-y",
        "TimeUsage": "1:0:0",
        "MemUsage": "10G",
        "CoreUsage": "1"
    },
    "DICOMToBIDSOptions": {
        "DICOMDirectory": "",
        "BIDSDirectory": "",
        "ConversionConfig": "",
        "DICOMFormatString": "",
        "TimeUsage": "1:0:0",
```

The configuration file consists of some project-level metadata, such as "ProjectTitle", and a set of Option blocks that contain their own sub-settings. Each Option block corresponds to a clpipe command, and controls the input parameters for that step. For example, "DICOMtoBIDSOptions" corresponds to the convert2bids command. You can find explanations for each specific Option block on the documenation page for its corresponding command.

All of these fields have what the designers of clpipe consider to be reasonable defaults for processing. Additionally, users at UNC-CH on the Longleaf cluster with access to the HNG group should be able to use the default options with no change. Other users will have to modify several fields.

Described here are the project-level meta fields of the configuration file:

class clpipe.config.options.**ProjectOptions**

Contains metadata for your project and option blocks for each command.

project_title: str = 'A Neuroimaging Project'

The title of your project.

contributors: str = 'SET CONTRIBUTORS'

Members of the project team.

project_directory: str = ''

The root directory of your clpipe project.

email_address: str = 'SET EMAIL ADDRESS'

Email address used for delivering batch job updates.

source: *clpipe.config.options.SourceOptions*

Options for the flywheel_sync command.

convert2bids: *clpipe.config.options.Convert2BIDSOptions*

Options for the convert2bids command.

bids_validation: *clpipe.config.options.BIDSValidatorOptions*

Options for the bids_validation command.

fmriprep: *clpipe.config.options.FMRIPrepOptions*

Options for the preprocessing command.

postprocessing: clpipe.config.options.PostProcessingOptions

Options for the postprocessing command.

processing_streams: List[clpipe.config.options.ProcessingStream]

Stores a list of postprocessing streams to be selected by name when using the postprocessing command.

get_logs_dir() → str

Get the project's top level log directory.

populate_project_paths(*project_dir: os.PathLike*, *source_data: os.PathLike*)

Sets all project paths relative to a given project directory.

Args: project_dir (os.PathLike): Root directory of the project. source_data (os.PathLike): Directory pointing to the source DICOM data.

classmethod transform_dict(*config_dict: dict*) → dict

Modify the inherited ClpipeData load() to transform the file-loaded dictionary in the case of an old config file.

2.2 The Command Line Interface

clpipe provides a unifed command line interface (CLI) under the `clpipe` command.

Running this command on its own will show you a subset of clpipe's processing commands:

```
> clpipe

Usage: clpipe [OPTIONS] COMMAND [ARGS]...

```

(continues on next page)

(continued from previous page)

```
Welcome to clpipe.

Please choose one of the commands below for more information.

If you're not sure where to begin, please see the documentation at:
https://clpipe.readthedocs.io/en/latest/index.html

Options:
    -version, -v   Display clpipe's version.
    -help          Show this message and exit.

Commands:
    project_setup       Initialize a clpipe project.
    convert2bids        Convert DICOM files to BIDS format.
    bids_validate       Validate if a directory BIDS standard.
    templateflow_setup  Installs the templates for preprocessing listed in...
    preprocess          Submit BIDS-formatted images to fMRIPrep.
    postprocess         Additional processing for GLM or connectivity...
    glm                 General Linear Model (GLM) Commands.
    roi                 Region of Interest (ROI) Commands.
    flywheel_sync       Sync your DICOM data with Flywheel.
    reports             Generate reports for your project.
    config              Configuration-related commands.
```

clpipe commands can be called with the following format: `clpipe <command>` To see the documentation for a particular command, include the `-help` option:

```
> clpipe setup -help

Usage: clpipe project_setup [OPTIONS]

Initialize a clpipe project.

Options:
    -project_title TEXT     Choose a title for your project.  [required]
    -project_dir DIRECTORY  Where the project will be located.  [required]
    -source_data DIRECTORY  Where the raw data (usually DICOMs) are located.
    -move_source_data       Move source data into project/data_DICOMs folder.
                            USE WITH CAUTION.
    -symlink_source_data    Symlink the source data into project/data_dicoms.
                            Usually safe to do.
    -debug                  Flag to enable detailed error messages and
                            traceback.
    -help                   Show this message and exit.
```

Here is an example of the same command being used:

```
> clpipe setup -project_title "My Project" -project_dir . -debug
```

Some of the clpipe commands, like `glm` and `roi`, contain their own nested sub-commands, which can be accessed like this: `clpipe <command> <sub-command>`

These commands contain their own help dialogue as well:

```
> clpipe glm launch -help

Usage: clpipe glm launch [OPTIONS] LEVEL MODEL

Launch all prepared .fsf files for L1 or L2 GLM analysis.

LEVEL is the level of anlaysis, L1 or L2

MODEL must be a a corresponding L1 or L2 model from your GLM configuration
file.

Options:
    -glm_config_file, -g FILE  The path to your clpipe configuration file.
                                [required]
    -test_one                  Only submit one job for testing purposes.
    -submit, -s                Flag to submit commands to the HPC.
    -debug, -d                 Flag to enable detailed error messages and
                                traceback.
    -help                      Show this message and exit.
```

CHAPTER

THREE

PROJECT SETUP

3.1 Overview

clpipe contains a convenience command for setting up the directories and configuration files for a given neuroimaging project, in a way that makes it simple to change configuration option.

This command will create the necessary directory structures, as well as create a default configuration file with many directory fields already filled out. For example,

```
|-- analyses
|-- clpipe_config.json
|-- conversion_config.json
|-- data_BIDS
|   |-- CHANGES
|   |-- code
|   |-- dataset_description.json
|   |-- derivatives
|   |-- participants.json
|   |-- participants.tsv
|   |-- README
|   |-- sourcedata
|-- data_DICOMs -> /symlink/to/your/dicoms
|-- data_fmriprep
|-- data_postproc
|   |-- betaseries_default
|   |-- betaseries_noGSR
|   |-- betaseries_noScrub
|   |-- postproc_default
|   |-- postproc_noGSR
|   |-- postproc_noScrub
|-- data_ROI_ts
|   |-- postproc_default
|-- logs
|   |-- betaseries_logs
|   |-- DCM2BIDS_logs
|   |-- postproc_logs
|   |-- ROI_extraction_logs
|-- scripts
```

3.2 Command

3.2.1 clpipe project_setup

```
clpipe project_setup [OPTIONS]
```

Options

-project_title <project_title>

Choose a title for your project.

-project_dir <project_dir>

Required Where the project will be located.

-source_data <source_data>

Where the raw data (usually DICOMs) are located.

-move_source_data

Move source data into project/data_DICOMs folder. USE WITH CAUTION.

-symlink_source_data

Symlink the source data into project/data_dicoms. Usually safe to do.

-debug

Flag to enable detailed error messages and traceback.

CHAPTER

FOUR

DICOM TO BIDS CONVERSION

4.1 Overview

clpipe's *convert2bids* commands facilitates the conversion of DICOM files into BIDS format, using either dcm2bids or heudiconv as your underlying converter.

4.2 Configuration

Definitions

class clpipe.config.options.**Convert2BIDSOptions**

Options for converting DICOM files to BIDS format.

dicom_directory: str = ''

Path to your source DICOM directory to be converted.

bids_directory: str = ''

Output directory where your BIDS data will be saved.

conversion_config: str = ''

The path to your conversion configuration file - either a conversion_config.json file for dcm2bids, or heuristic.py for heudiconv.

dicom_format_string: str = ''

Used to tell clpipe where to find subject and session level folders in you DICOM directory.

4.2.1 The DICOM format string

One important thing to note about using the main command is the need for a specifically formatted *dicom_dir_format* option. This is to appropriately map your dicom directories to subject/sessions. All subject session folders should be named the same way. A dicom_dir_format must contain at least {session} and can contain a {subject} formatting option. Two examples of a dicom_dir_format option are *{subject}_{session}/*, which corresponds to the following structure:

```
dicom_datadata/
    S01_pre/
        scan1/
        scan2/
        scan3
```

(continues on next page)

(continued from previous page)

```
    S01-post/
        scan1/
        scan2/
        scan3/
```

Alternatively, you can use *{subject}/{session}/*

```
data/
    S01/
        pre/
            scan1/
        post/
            scan1/
    S02/
        pre/
            scan1/
        post/
            scan1/
```

You can include other text in the formatting option, so that the program ignores that text. For example, *Subject-{subject}/* used on a dataset with *Subject-01* as a folder will determine the subject id to be *01* not *Subject-01*. Note that in all examples, there is a trailing forward slash.

4.2.2 dcm2bids configuration

dcm2bids is a JSON-driven tool for converting DICOM files. While not as flexible as heudiconv, dcm2bids is easier to learn and has a conversion configuration that is simpler to setup and modify for users less familiar with programming.

This documentation contains a tutorial for setting up a dcm2bids conversion on the *Tutorials/BIDS Conversion* page. You can also refer to dcm2bids' tutorial for futher help.

The Conversion Config File

dcm2bids is driven by a JSON configuration file. clpipe helps you out with this by generating a starter file for you when you use the project_setup command:

The starter demonstrates one anatomical and one functional image conversion configuration, as well one field map, which references the functional image.

The dcm2niix options are included to allow dcm2bids' search depth to be expanded enough to work with Flywheel's default file structure when syncing. You can ignore it, but it serves as a useful place to configure any other dcm2niix options you may need to specify. dcm2niix is the tool used by dcm2bids to perform DICOM to nifti conversion.

dcm2bids_helper

To obtain the information from the header, dcm2bids has a handy helper function:

```
usage: dcm2bids_helper [-h] -d DICOM_DIR [DICOM_DIR ...] [-o OUTPUT_DIR]

optional arguments:
    -h, --help            show this help message and exit
    -d DICOM_DIR [DICOM_DIR ...], --dicom_dir DICOM_DIR [DICOM_DIR ...] DICOM files␣
↪directory
    -o OUTPUT_DIR, --output_dir OUTPUT_DIR
                    Output BIDS directory, Default: current directory

        Documentation at https://github.com/cbedetti/Dcm2Bids
```

This command will create convert an entire folder's data, create a temporary directory containing all the converted files, and, more importantly, the sidecar JSONs. These JSONs contain the information needed to update the conversion configuration file.

4.2.3 heudiconv configuration

heudiconv is another tool for converting DICOM files to BIDS format. This tool is a bit trickier to use than dcm2bids, because its configuration is driven by a python file instead of json. However, it allows for more control over conversion than what dcm2bids can provide, allowing you to handle datasets with more variation or difficult edge cases.

See one of these various walkthroughs for instructions on setting up and using heudiconv.

clpipe does not currently provide a default heuristic file - run heudiconv on your dataset with the converter set to "None" to generate a *.heuristic* folder, and copy a heuristic file from one of the subject folders to the root of your clpipe directory. clpipe comes with heudionv installed as a Python library, so you should be able to use the *heudiconv* command directly on the command line if you're in the same virtual environment where you installed clpipe.

To use heudiconv, provide a heuristic file as your conversion config file:

```
"DICOMToBIDSOptions": {
        "DICOMDirectory": "...<clpipe_project>/data_DICOMs",
        "BIDSDirectory": "...<clpipe_project>/clpipe/data_BIDS",
        "ConversionConfig": "...<clpipe_project>/heuristic.py",
        "DICOMFormatString": "{subject}",
```

And when running convert2bids, include the *-heudiconv* flag:

```
clpipe convert2bids -heudiconv -c config_file.json
```

4.3 Command

4.3.1 clpipe convert2bids

Convert DICOM files to BIDS format.

Providing no SUBJECTS will default to all subjects. List subject IDs in SUBJECTS to process specific subjects:

> clpipe convert2bids 123 124 125 ...

Available subject IDs are determined by the dicom_dir_format string.

```
clpipe convert2bids [OPTIONS] [SUBJECTS]...
```

Options

-config_file, -c <config_file>

Required The path to your clpipe configuration file.

-conv_config_file <conv_config_file>

A conversion definition file, either a dcm2bids conversion config .json file or a heudiconv heuristic .py file.

-dicom_dir, -i <dicom_dir>

The folder where subject dicoms are located.

-dicom_dir_format <dicom_dir_format>

Format string which specifies how subjects/sessions are organized within the dicom_dir. Example: {subject}_{session}. See https://clpipe.readthedocs.io/en/latest/bids_convert.html for more details.

-BIDS_dir, -o <bids_dir>

The dicom info output file name.

-overwrite

Overwrite existing BIDS data?

-log_dir <log_dir>

Where to put your HPC output files (such as SLURM output files).

-subject <subject>

DEPRECATED: specify one subject to process - can give an arbitrary number of subjects as arguments now.

-session <session>

Specify the session to convert. Available sessions determined by the {session} placeholder given by dicom_dir_format.

-longitudinal

Convert all subjects/sessions into individual pseudo-subjects. Use if you do not want T1w averaged across sessions during FMRIprep

-submit, -s

Flag to submit commands to the HPC.

-debug, -d

Flag to enable detailed error messages and traceback.

-dcm2bids, -heudiconv

Specify which converter to use.

Arguments

SUBJECTS

Optional argument(s)

CHAPTER

FIVE

BIDS VALIDATION

5.1 Overview

clpipe contains a convenience function to validate your datasets directly on the HPC. This function uses a Singularity image of the BIDs Validator.

The output of this command will appear in your *logs/bids_validation_logs* folder by default.

Notably, fMRIPrep will refuse to run non-valid BIDS datasets, unless you turn the option off. The same bids-validator outputs can be viewed in fMRIPrep's logs, but you may find this stand-alone command more convenient.

5.2 Configuration

Definitions

class clpipe.config.options.**BIDSValidatorOptions**

Options for validating your BIDS directory.

bids_validator_image: str = '/proj/hng/singularity_imgs/validator.simg'

Path to your BIDS validator image.

5.3 Command

5.3.1 clpipe bids_validate

Validate if a directory BIDS standard.

Validates the directory at BIDS_DIR, or at the BIDS directory in your config file's DICOMToBIDSOptions if -config_file is given.

Results are viewable in logs/bids_validation_logs unless -interactive is used.

```
clpipe bids_validate [OPTIONS] [BIDS_DIR]
```

Options

-config_file, -c <config_file>

Required The path to your clpipe configuration file.

-log_dir <log_dir>

Where to put your HPC output files (such as SLURM output files).

-verbose, -v

Creates verbose validator output. Use if you want to see ALL files with errors/warnings.

-submit, -s

Flag to submit commands to the HPC.

-interactive

Run in interactive mode. Only use in an interactive compute session.

-debug, -d

Flag to enable detailed error messages and traceback.

Arguments

BIDS_DIR

Optional argument

CHAPTER

SIX

PREPROCESSING WITH FMRIPREP

6.1 Overview

clpipe uses fMRIprep via the `clpipe preprocess` command to perform minimal preprocessing on functional MRI data.

`clpipe preprocess` creates one batch job per subject. If you find that you are running out of memory, increase the *[FMRIPrepOptions][FMRIPrepMemoryUsage]* option in the configuration file.

To submit your dataset for preprocessing, use the following command:

6.2 Configuration

class clpipe.config.options.**FMRIPrepOptions**

Options for configuring fMRIPrep.

bids_directory: str = ''

Your BIDs formatted raw data directory.

working_directory: str = 'SET WORKING DIRECTORY'

Storage location for intermediary fMRIPrep files. Takes up a large amount of space - Longleaf users should use their /work folder.

output_directory: str = ''

Where to save your preprocessed images.

fmriprep_path: str = '/proj/hng/singularity_imgs/fmriprep_22.1.1.sif'

Path to your fMRIPrep Singularity image.

freesurfer_license_path: str = '/proj/hng/singularity_imgs/license.txt'

Path to your Freesurfer license .txt file.

use_aroma: bool = False

Set True to generate AROMA artifacts. Significantly increases run time and memory usage.

commandline_opts: str = ''

Any additional arguments to pass to FMRIprep.

templateflow_toggle: bool = True

Set True to activate to use templateflow, which allows you to generate multiple template variantions for the same outputs.

templateflow_path: str = '/proj/hng/singularity_imgs/template_flow'

The path to your templateflow directory.

templateflow_templates: list

Which templates (standard spaces) should clpipe download for use in templateflow?

fmap_roi_cleanup: int = 3

How many timepoints should the fmap_cleanup function extract from blip-up/blip-down field maps, set to -1 to disable.

fmriprep_memory_usage: str = '50G'

How much memory in RAM each subject's preprocessing will use.

fmriprep_time_usage: str = '16:0:0'

How much time on the cluster fMRIPrep is allowed to use.

n_threads: str = '12'

How many threads to use in each job.

log_directory: str = ''

Path to your logging directory for fMRIPrep outputs. Not generally changed from default.

docker_toggle: bool = False

Set True to use a Docker image.

docker_fmriprep_version: str = ''

Path to your fMRIPrep Docker image.

6.3 Command

6.3.1 clpipe preprocess

Submit BIDS-formatted images to fMRIPrep.

Providing no SUBJECTS will default to all subjects. List subject IDs in SUBJECTS to process specific subjects:

> clpipe preprocess 123 124 125 ...

```
clpipe preprocess [OPTIONS] [SUBJECTS]...
```

Options

-config_file, -c <config_file>

Required The path to your clpipe configuration file.

-bids_dir, -i <bids_dir>

The dicom info output file name.

-working_dir <working_dir>

Where to generate the working directory.

-output_dir, -o <output_dir>

Where to put the preprocessed data. If a configuration file is provided with a output directory, this argument is not necessary.

-log_dir <log_dir>
: Where to put your HPC output files (such as SLURM output files).

-submit, -s
: Flag to submit commands to the HPC.

-debug, -d
: Flag to enable detailed error messages and traceback.

Arguments

SUBJECTS
: Optional argument(s)

6.4 Quality Control Reports

fMRIprep produces detailed html reports for each subject, allowing users to visually inspect registration, normalization and confound plots. However, these reports do not have the images directly embedded in them, which means that directly downloading them from the HPC will not result in a usable report. There are two options:

1. Open the html reports directly on the HPC, using some sort of interactive web browser.
2. Download the reports and the images in the correct directory structure.

clpipe has a convenience function to organize and prepare a zip archive containing the reports and images, for quick download onto a personal computer.

This command uses the working directory previously specified to copy the reports and images to, then creates a zip archive containing them. This command is not a batch

: command, and can take a little time creating the archive, so be aware.

Once the archive is created, it can be downloaded and unzipped to a personal computer. The reports should correctly load images once opened.

6.4.1 clpipe reports fmriprep

Create a .zip directory of all fMRIPrep reports.

```
clpipe reports fmriprep [OPTIONS]
```

Options

-config_file, -c <config_file>
: **Required** The configuration file for the current data processing setup.

-output_name, -o <output_name>
: Path and name of the output archive. Defaults to current working directory and "fMRIPrep_Reports.zip"

-clear_temp, -keep_temp
: Keep or clear the built temporary directory. Defaults to clear_temp.

-debug, -d
: Print traceback on errors.

CHAPTER

SEVEN

POSTPROCESSING

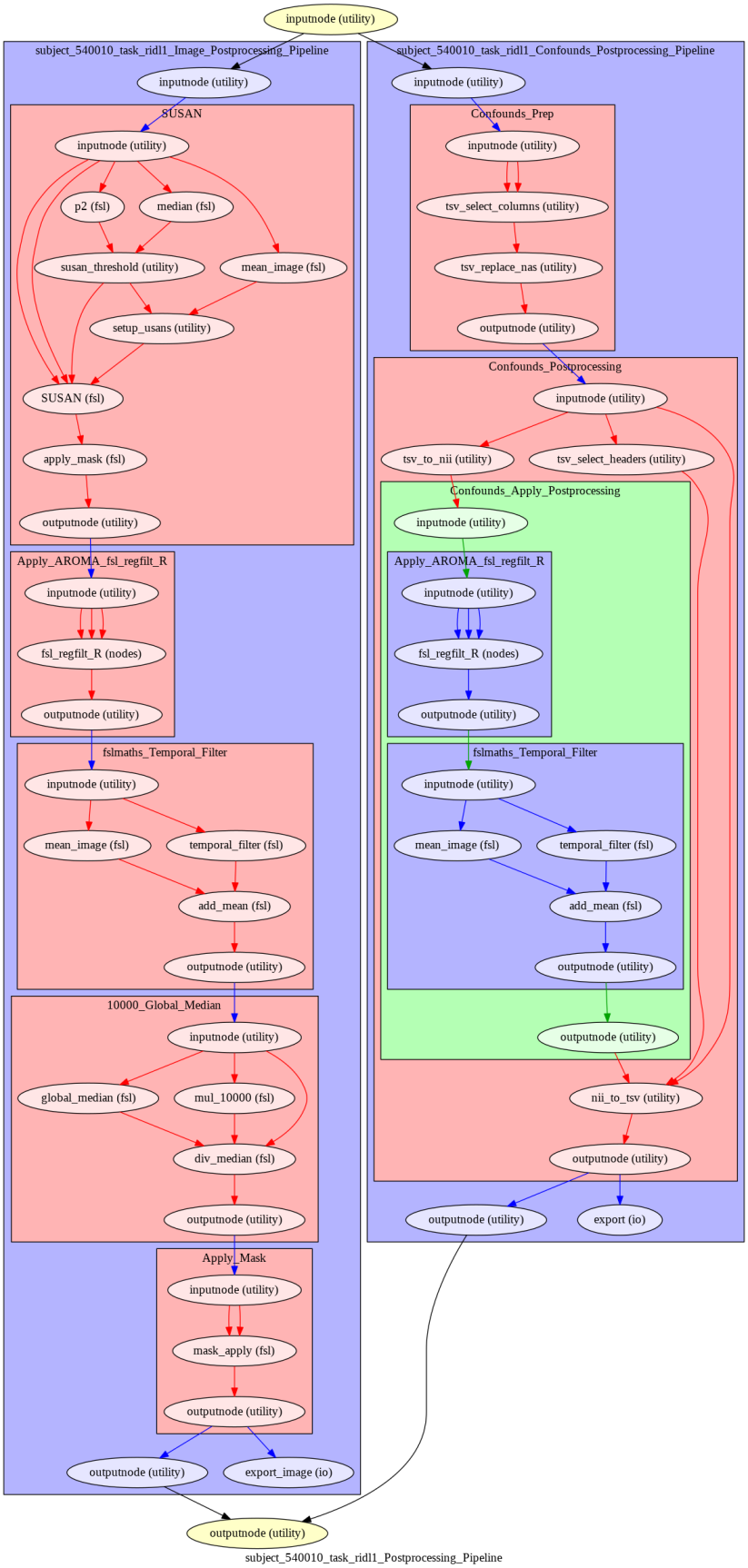
7.1 Overview

The `clpipe postprocess` command combines the functionality of the retired `fmri_postprocess` and `glm_setup` commands into a unified postprocessing stream.

This command allows for flexible creation of processing streams. The order of processing steps and their specific implementations can be modified in the configuration file. Any temporally-relevant processing steps can also be applied to each image's corresponding confounds file. `postprocess` caches its processing intermediaries in a working directory, which allows quick re-runs of pipelines with new parameters.

This command will also output a detailed processing graph for each processing stream.

Example Pipeline


inputnode (utility)
subject_540010_task_ridl1_Image_Postprocessing_Pipeline
inputnode (utility)
SUSAN
inputnode (utility)
p2 (fsl)
median (fsl)
susan_threshold (utility)
mean_image (fsl)
setup_usans (utility)
SUSAN (fsl)
apply_mask (fsl)
outputnode (utility)
Apply_AROMA_fsl_regfilt_R
inputnode (utility)
fsl_regfilt_R (nodes)
outputnode (utility)
fslmaths_Temporal_Filter
inputnode (utility)
mean_image (fsl)
temporal_filter (fsl)
add_mean (fsl)
outputnode (utility)
10000_Global_Median
inputnode (utility)
global_median (fsl)
mul_10000 (fsl)
div_median (fsl)
outputnode (utility)
Apply_Mask
inputnode (utility)
mask_apply (fsl)
outputnode (utility)
outputnode (utility)
export_image (io)
subject_540010_task_ridl1_Confounds_Postprocessing_Pipeline
inputnode (utility)
Confounds_Prep
inputnode (utility)
tsv_select_columns (utility)
tsv_replace_nas (utility)
outputnode (utility)
Confounds_Postprocessing
inputnode (utility)
tsv_to_nii (utility)
tsv_select_headers (utility)
Confounds_Apply_Postprocessing
inputnode (utility)
Apply_AROMA_fsl_regfilt_R
inputnode (utility)
fsl_regfilt_R (nodes)
outputnode (utility)
fslmaths_Temporal_Filter
inputnode (utility)
mean_image (fsl)
temporal_filter (fsl)
add_mean (fsl)
outputnode (utility)
outputnode (utility)
nii_to_tsv (utility)
outputnode (utility)
outputnode (utility)
export (io)
outputnode (utility)
subject_540010_task_ridl1_Postprocessing_Pipeline

7.2 Configuration

7.2.1 Overview

The top level of the postprocessing configuration section contains general options like the path to your working directory, which tasks to target, etc.

Following this are the `ProcessingSteps`, which define the steps used for postprocessing. Postprocessing will occur in the order of the list.

`ProcessStepOptions` displays all of the processing steps with configurable options, allowing them to be configured to suit the needs of your project. See the Processing Step Options section for more information about configuring this section.

`ConfoundOptions` contains settings specific to each image's confounds file, and `BatchOptions` contains settings for job submission.

Option Block

```
"PostProcessingOptions2": {
        "WorkingDirectory": "",
        "WriteProcessGraph": true,
        "TargetImageSpace": "MNI152NLin2009cAsym",
        "TargetTasks": [],
        "TargetAcquisitions": [],
        "ProcessingSteps": [
                "SpatialSmoothing",
                "TemporalFiltering",
                "IntensityNormalization",
                "ApplyMask"
        ],
        "ProcessingStepOptions": {
                "TemporalFiltering": {
                        "Implementation":"fslmaths",
                        "FilteringHighPass": 0.008,
                        "FilteringLowPass": -1,
                        "FilteringOrder": 2
                },
                ...additional processing step options
        },
        "ConfoundOptions": {
                "Columns": [
                        "csf", "csf_derivative1", "white_matter", "white_matter_
↪derivative1"
                ],
                "MotionOutliers": {
                        "Include": true,
                        "ScrubVar": "framewise_displacement",
                        "Threshold": 0.9,
                        "ScrubAhead": 0,
                        "ScrubBehind": 0,
                        "ScrubContiguous": 0
                }
        },
```

(continues on next page)

(continued from previous page)

```
        "BatchOptions": {
                "MemoryUsage": "20000",
                "TimeUsage": "2:0:0",
                "NThreads": "1"
}
}
```

Top-Level Definitions .. autoclass:: clpipe.config.options.PostProcessingOptions

members

7.2.2 Processing Step Options

Temporal Filtering

This step removes signals from an image's timeseries based on cutoff thresholds. This transformation is also applied to your confounds.

ProcessingStepOptions Block:

```
"TemporalFiltering": {
        "Implementation":"fslmaths",
        "FilteringHighPass": 0.008,
        "FilteringLowPass": -1,
        "FilteringOrder": 2
}
```

Definitions:

class clpipe.config.options.**TemporalFiltering**

This step removes signals from an image's timeseries based on cutoff thresholds. Also applied to confounds.

implementation: str = 'fslmaths'

Available implementations: fslmaths, 3dTProject

filtering_high_pass: float = 0.008

Values below this threshold are filtered. Defaults to .08 Hz. Set to -1 to disable.

filtering_low_pass: int = -1

Values above this threshold are filtered. Disabled by default (-1).

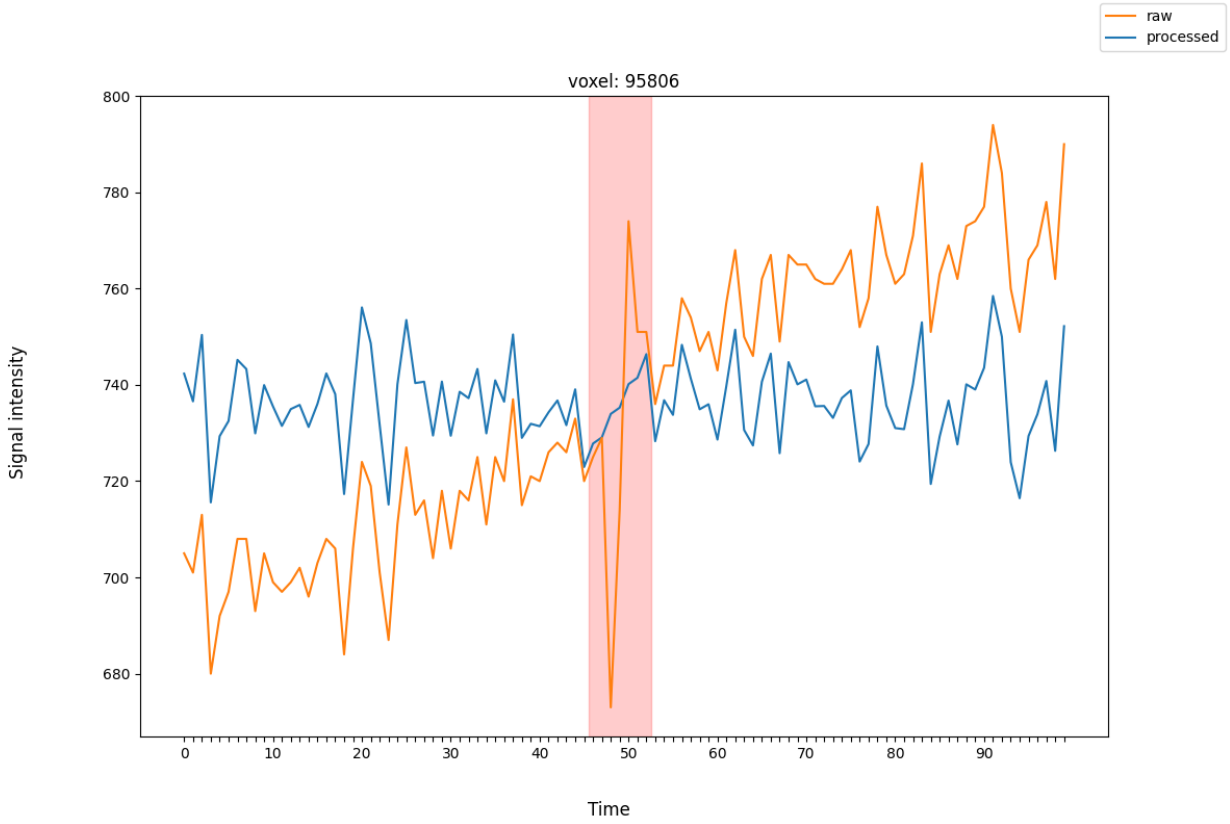
filtering_order: int = 2

Order of the filter. Defaults to 2.

Special Case: Filtering with Scrubbed Timepoints

When the scrubbing step is active at the same time as temporal filtering (see `ScrubTimepoints`), filtering is handled with a special workflow. This for two reasons: first, temporal filtering must be done before scrubbing, because this step cannot tolerate NAs or non-continuous gaps in the timeseries. Second, filtering can distribute the impact of a disruptive motion artifact throughout a timeseries, despite scrubbing the offending timepoints aftwards. The solution to this is to interpolate over the timepoints to be scrubbed when temporal filtering.

The following diagram shows a timeseries with a large motion artifact (blue), with the points to be scrubbed highlighted in red:



The processed timeseries (orange), after filtering, shows how the scrubbed points were interpolated to improve the performance of the filter.

Warning: To achieve interpolation, this special case always uses the 3dTproject implementation, regardless of the implementation requested.

Intensity Normalization

This step normalizes the central tendency of the data to a standard scale. As data acquired from different subjects can vary in relative intensity values, this step is important for accurate group-level statistics.

ProcessingStepOptions Block

```
"IntensityNormalization": {
        "Implementation": "10000_GlobalMedian"
}
```

Definitions

class clpipe.config.options.**IntensityNormalization**

Normalize the intensity of the image data.

implementation: str = '10000_GlobalMedian'

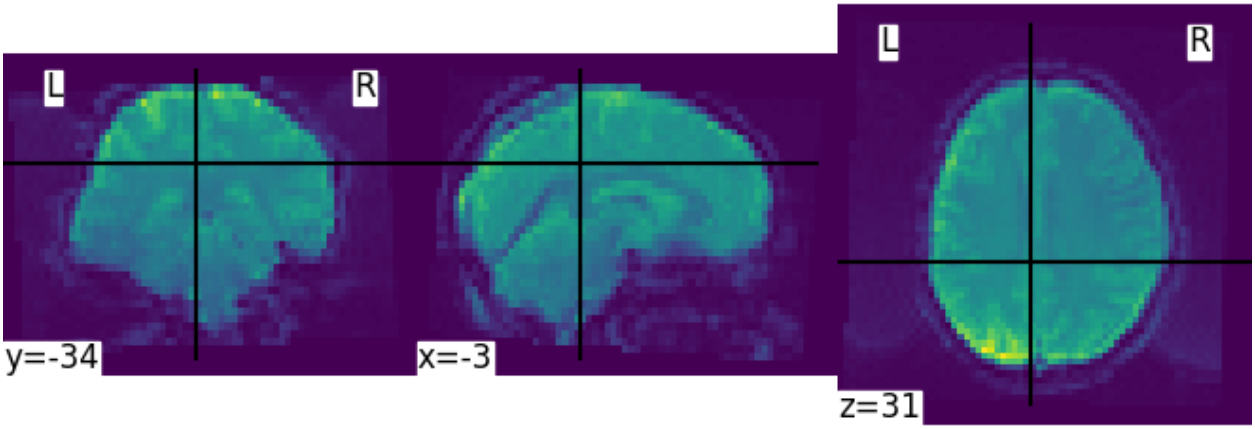
Currently limited to '10000_GlobalMedian'

Spatial Smoothing

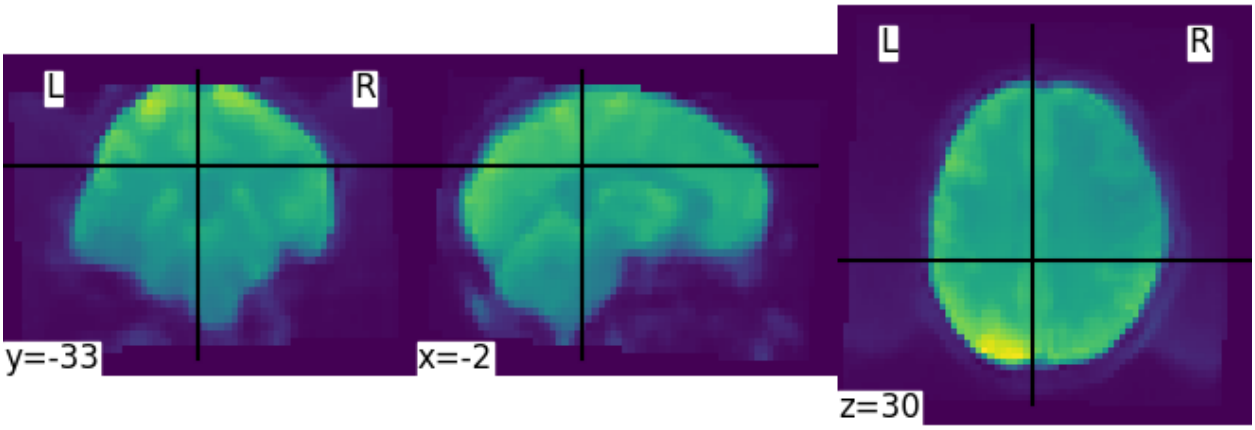
This step blurs the image data across adjacent voxels. This helps improve the validity of statistical testing by smoothing over random noise in the data, and enchancing underlying brain signal.

To achieve the smoothing, a 3D Gaussian filter is applied to the data. This filter takes as input a kernel radius, which is analogous to the size of the blur tool in a photo editing tool.

Unsmoothed Raw Image



Smoothed with 6mm Kernel



ProcessingStepOptions Block

```
"SpatialSmoothing": {
        "Implementation": "SUSAN",
        "FWHM": 6
}
```

Definitions

`class clpipe.config.options.SpatialSmoothing`

Apply spatial smoothing to the image data.

implementation: str = 'SUSAN'

Currently limited to 'SUSAN'

fwhm: int = 6

The size of the smoothing kernel. Specifically the full width half max of the Gaussian kernel. Scaled in millimeters.

AROMA Regression

This step removes AROMA-identified noise artifacts from the data with non-aggressive regression.

AROMA regression relies on the presence of AROMA output artifacts in your fMRIPrep directory - they are the files with `desc-MELODIC_mixing.tsv` and `AROMAnoiseICs.csv` as suffixes. Thus, you must have the UseAROMA option enabled in your preprocessing options to use this step.

Also applies to confounds.

ProcessingStepOptions Block

```
"AROMARegression": {
        "Implementation": "fsl_regfilt"
}
```

Definitions

class clpipe.config.options.**AROMARegression**

Regress out automatically classified noise artifacts from the image data using AROMA. Also applied to confounds.

Confound Regression

This step regresses the contents of the postprocessesed confounds file out of your data. Confounds are processed before their respective image, so regressed confounds will have any selected processing steps applied to them (such as `TemporalFiltering`) before this regression occurs. The columns used are those defined in the `ConfoundOptions` configuration block.

Confound regression is typically used for network analysis - GLM analysis removes these confounds through there inclusion in the model as nuisance regressors.

ProcessingStepOptions Block

```
"ConfoundRegression": {
        "Implementation": "afni_3dTproject"
}
```

Definitions

class clpipe.config.options.**ConfoundRegression**

Regress out the confound file values from your image. If any other processing steps are relevant to the confounds, they will be applied first.

implementation: str = 'afni_3dTproject'

Currently limited to "afni_3dTproject

Scrub Timepoints

The `ScrubTimepoints` step can be used to remove timepoints from the image timeseries based on a target variable from that image's confounds file. Timepoints scrubbed from an image's timeseries are also removed its respective confound file.

ProcessingStepOptions Block

```
"ScrubTimepoints": {
    "InsertNA": true,
    "Columns": [
        {
            "TargetVariable": "cosine*",
            "Threshold": 100,
            "ScrubAhead": 0,
            "ScrubBehind": 0,
            "ScrubContiguous": 0
        },
        {
            "TargetVariable": "framewise_displacement",
            "Threshold": 0.2,
            "ScrubAhead": 1,
            "ScrubBehind": 0,
            "ScrubContiguous": 0
        }
    ]
}
```

Definitions

class clpipe.config.options.**ScrubTimepoints**

This step can be used to remove timepoints from the image timeseries based on a target variable from that image's confounds file. Timepoints scrubbed from an image's timeseries are also removed its respective confound file.

insert_na: bool = True

Set true to replace scrubbed timepoints with NA. False removes the timepoints completely.

scrub_columns: List[*clpipe.config.options.ScrubColumn*]

A list of columns to be scrubbed.

class clpipe.config.options.**ScrubColumn**

A definition for a single column to be scrubbed.

target_variable: str = 'framewise_displacement'

Which confound variable to use as a reference for scrubbing. May use wildcard (*) to select multiple similar columns.

threshold: float = 0.9

Any timepoint of the target variable exceeding this value will be scrubbed

scrub_ahead: int = 0

Set the number of timepoints to scrub ahead of target timepoints

scrub_behind: int = 0

Set the number of timepoints to scrub behind target timepoints

scrub_contiguous: int = 0

Scrub everything between scrub targets up to this far apart

Resample

This step will resample your image into the same resolution as the given `ReferenceImage`. Exercise caution with this step - make sure you are not unintentionally resampling to an image with a lower resolution.

ProcessingStepOptions Block

```
"Resample": {
        "ReferenceImage": "SET REFERENCE IMAGE"
}
```

Definitions

class clpipe.config.options.**Resample**

Resample your image to a new space.

reference_image: str = 'SET REFERENCE IMAGE'

Path to an image against which to resample - often a template

Trim Timepoints

This step performs simple trimming of timepoints from the beginning and/or end of your timeseries with no other logic. Also applies to your confounds.

ProcessingStepOptions Block

```
"TrimTimepoints": {
        "FromEnd": 0,
        "FromBeginning": 0
}
```

Definitions

class clpipe.config.options.**TrimTimepoints**

Trim timepoints from the beginning or end of an image. Also applied to confounds.

from_end: int = 0

Number of timepoints to trim from the end of each image.

from_beginning: int = 0

Number of timepoints to trim from the beginning of each image.

Apply Mask

This step will apply the image's fMRIPrep mask.

Note - There is currently nothing to configure for this step, so it is simply added to the `ProcessingSteps` list as "ApplyMask" and does not have a section in `ProcessingStepOptions`

```
"ProcessingSteps": [
        "SpatialSmoothing",
        "TemporalFiltering",
        "IntensityNormalization",
        "ApplyMask"
]
```

7.2.3 Confounds Options

This option block defines your settings for processing the confounds file accompanying each image. A subset of the columns provided by your base fMRIPrep confounds file is chosen with the `Columns` list.

The `MotionOutliers` section is used to add spike regressors based on (usually) framewise displacement for inclusion in a GLM model. Note that this section is independent from the scrubbing step - the scrubbing step removes timepoints from both the image and the confounds, while this step adds a variable number of columns to your confounds.

Definitions

class clpipe.config.options.**ConfoundOptions**

The default options to apply to the confounds files.

columns: list

A list containing a subset of confound file columns to use from each image's confound file. You may use the wildcard '*' operator to select groups of columns, such as 'csf*'

motion_outliers: *clpipe.config.options.MotionOutliers*

Options specific to motion outliers.

class clpipe.config.options.**MotionOutliers**

These options control the construction of spike regressor columns based on a particular confound column (usually framewise_displacement) and a threshold. For each timepoint of the chosen variable that exceeds the threshold, a new column of all 0s and a single '1' at that timepoint is added to the end of the confounds file to serve as a spike regressor for GLM analysis.

include: bool = False

Set 'true' to add motion outlier spike regressors to each confound file.

scrub_var: str = 'framewise_displacement'

Which variable in the confounds file should be used to calculate motion outliers.

threshold: float = 0.9

Threshold at which to flag a timepoint as a motion outlier.

scrub_ahead: int = 0

How many time points ahead of a flagged time point should be flagged also.

scrub_behind: int = 0

If a timepoint is scrubbed, how many points before to remove.

scrub_contiguous: int = 0

How many good contiguous timepoints need to exist.

Resample

Trim Timepoints

7.2.4 Batch Options

These options specify the cluster compute options used when submitting jobs. The default values are usually sufficient to process the data.

Definitions

class clpipe.config.options.**BatchOptions**

The batch settings for postprocessing.

memory_usage: str = '20G'

How much memory to allocate per job.

time_usage: str = '2:0:0'

How much time to allocate per job.

n_threads: str = '1'

How many threads to allocate per job.

7.2.5 Processing Streams Setup

By default, the output from running fmri_postprocess will appear in your clpipe folder at `data_postproc/default`, reflecting the defaults from PostProcessingOptions.

However, you can utilize the power of processing streams to deploy multiple postprocessing streams. Options for processing streams are found in a separate section of your configuration file, `ProcessingStreams`. Each processing stream you define your config file's `ProcessingStreams` block will create a new output folder named after the stream setting.

Within each processing stream, you can override any of the settings in the main `PostProcessingOptions` section. For example, in the follow json snippet, the first processing stream will only pick "rest" tasks and defines its own set of processing steps. The second stream does the same thing, but specifies a filtering high pass by overriding the default value of -1 with .009.

Option Block

```
...
"ProcessingStreams": [
        ...
        {
                "ProcessingStream": "smooth_aroma-regress_filter-butterworth_normalize",
                "PostProcessingOptions": {
                        "TargetTasks": [
                                "rest"
                        ],
                        "ProcessingSteps": [
                                "SpatialSmoothing",
                                "AROMARegression",
                                "TemporalFiltering",
                                "IntensityNormalization",
                                "ApplyMask"
```

(continues on next page)

(continued from previous page)

```
                        ]
                }
        },
        {
                "ProcessingStream": "smooth_aroma-regress_filter-high-only_normalize",
                "PostProcessingOptions": {
                        "TargetTasks": [
                                "rest"
                        ],
                        "ProcessingSteps": [
                                "SpatialSmoothing",
                                "AROMARegression",
                                "TemporalFiltering",
                                "IntensityNormalization",
                                "ApplyMask"
                        ],
                        "ProcessingStepOptions": {
                                "TemporalFiltering": {
                                        "FilteringHighPass": .009
                                }
                        }
                }
        },
...
```

7.3 Command

7.3.1 CLI Options

clpipe postprocess

Additional processing for GLM or connectivity analysis.

Providing no SUBJECTS will default to all subjects. List subject IDs in SUBJECTS to process specific subjects:

> clpipe postprocess2 123 124 125 ...

```
clpipe postprocess [OPTIONS] [SUBJECTS]...
```

Options

-config_file, -c <config_file>

Required The path to your clpipe configuration file.

-fmriprep_dir, -i <fmriprep_dir>

Which fmriprep directory to process. If a configuration file is provided with a BIDS directory, this argument is not necessary. Note, must point to the `fmriprep` directory, not its parent directory.

-output_dir, -o <output_dir>

Where to put the postprocessed data. If a configuration file is provided with a output directory, this argument is not necessary.

-processing_stream, -p <processing_stream>
: Specify a processing stream to use defined in your configuration file.

-log_dir <log_dir>
: Where to put your HPC output files (such as SLURM output files).

-index_dir <index_dir>
: Give the path to an existing pybids index database.

-refresh_index, -r
: Refresh the pybids index database to reflect new fmriprep artifacts.

-batch, -no-batch
: Flag to create batch jobs without prompting.

-cache, -no-cache

-submit, -s
: Flag to submit commands to the HPC.

-debug, -d
: Flag to enable detailed error messages and traceback.

Arguments

SUBJECTS
: Optional argument(s)

7.3.2 Examples

Display jobs to be run without submitting.

```
clpipe postprocess -c clpipe_config.json
```

Submit jobs.

```
clpipe postprocess -c clpipe_config.json -submit
```

Submit jobs for specific subjects.

```
clpipe postprocess 123 124 125 -c clpipe_config.json -submit
```

To run a specific stream, give the `-processing_stream` (`-p` for short) option the name of the stream:

```
clpipe postprocess -c clpipe_config.json -p smooth_aroma-regress_filter-butterworth_
↪normalize -submit
```

CHAPTER

EIGHT

GLM ANALYSIS

8.1 Overview

clpipe includes functions to help you set up and run general linear models (GLM) on your neuroimaging data. It uses FSL's implementation of GLM (FEAT), and the functions provided in clpipe serve to help setup and manage the necessary files and folders.

Currently, clpipe includes the following commands:

1. **`fsl_onset_extract`** This function will extract task trial onset, duration and an optional parametric variable from the BIDS formatted events files, and constructs a set of FSL 3-column EV files.
2. **`glm_l1_preparefsfs`** This function takes an fsf file that you have generated as a prototype, and copies/modifies it for use with the images in your dataset. Multiple different L1 fsfs can be used, and images can be excluded or explicitly included in a given L1. See below for details on the glm configuration file.
3. **`glm_l1_launch`** Use this command to launch the fsf files generated by the `glm_l1_preparefsfs` command.
4. **`glm_l2_preparefsfs`** This function takes an fsf file, and a csv sheet that contains information about which image goes with which subject, and copies/modifies the fsf file for use with your dataset. Similarly to L1, multiple L2 fsf files can be specified. See below for details on how to do this.
5. **`glm_l2_launch`** Use this command to launch the fsf files generated by the `glm_l2_preparefsfs` command.

8.2 Configuration

clipe's GLM commands are all driven by a GLM configuration that is seperate from the main configuration file of a project. A GLM configuration file describes the L1 and L2 setup for a single task. It is structured as follows:

8.2.1 Preamble

- `GLMName`: Descriptive Name
- `Authors`: Author List
- `DateCreated`: Date
- `GLMSetupOptions`: Options for the initial resampling of data
  - `ParentClpipeConfig`: File path to project's clpipe config json,
  - `TargetDirectory`: Directory containing the image files, defaults to fmriprep directory,
  - `TargetSuffix`: Suffix that designates the files to be resampled, defaults to `space-MNI152NLin2009cAsym_desc-preproc_bold.nii.gz`,

- `WorkingDirectory`: Working directory, must be user specified,
- `TaskName`: Name of the task that the GLM is run on, must be the same as in the `task-*`, descriptors
- `ReferenceImage`: Reference image, used to determine dimensions and resolution. Should be the reference image for the space the data is normalized to, e.g. MNI152NLin2009cAsym ,
- `DummyScans`: Number of timepoints at the beginning of the scan to drop. Specify as an integer
- `ApplyFMRIPREPMask`: Apply brain mask, defaults to `true`
- `MaskFolderRoot`: Root of the directory tree that contains the masks, defaults to fmriprep directory,
- `MaskSuffix`: Suffix of the mask file, defaults to `space-MNI152NLin2009cAsym_desc-brain_mask.nii.gz`,
- `SUSANSmoothing`: Perform Susan Smoothing? Defaults to `false`
- **SUSANOptions: Options for SUSAN spatial smoothing**
    - `BrightnessThreshold`: Brightness threshold for SUSAN
    - `FWHM`: Full Width Half Max (Same as in FEAT GUI, not the same as in SUSAN GUI)
- `PreppedDataDirectory`: Output directory for resampled images, project setup defaults this to `main project directory/data_glm`,
- `PreppedSuffix`: Suffix for resampled data, defaults to `resampled.nii.gz`,
- `PrepareConfounds`: Boolean flag to prepare confound tables, defaults to true,
- `ConfoundSuffix`: Suffix for files containing confound information, defaults to `desc-confounds_regressors.tsv`,
- `Confounds`: A list of confound names. Note, the use of .* specifies a wildcard (i.e. `a_comp_cor.*` will extract all confounds that begin with `a_comp_cor`). Defaults to 6 motion parameters, CSF, WM and Global Signal.
- `ConfoundsQuad`: Which confounds to calculate quadratic expansions. Defaults to previous confounds list.
- `ConfoundsDerive`: Which confounds to calculate first derivatives? Defaults to previous confounds list.
- `ConfoundsQuadDerive`: Which confounds to calculate quadratic expansions of first derivatives.
- `MotionOutliers`: Calculate motion outliers. If `true` then dummy codes for motion outliers will be included in the confounds file. Defaults to true
- `ScrubVar`: Which variable in the confounds file should be used to calculate motion outliers, defaults to framewise displacement
- `Threshold`: Threshold at which to flag a timepoint as a motion outlier, defaults to .2
- `ScrubAhead`: How many time points ahead of a flagged time point should be flagged also, defaults to 0
- `ScrubBehind`: How many time points behind of a flagged time point should be flagged also, defaults to 0
- `ScrubContiguous`: How many "good" contiguous timepoints are needed, defaults to 5
- `MemoryUsage`: Memory usage, defaults to 5 GB
- `TimeUsage`: Time usage, defaults to 1 hour
- `NThreads`: Number of processing threads, defaults to 1
- `LogDirectory`: Log directory, defaults to `glm_setup_logs` in the project logs directory.

8.2.2 Level 1 Onsets

The entry `Level1Onsets` contains the specification for extracting the onset timing files and transforming them into FSL three column format EV files.

- `EventFileSuffix`: Suffix for the BIDS format event file. Unless your data is not in BIDS format, this likely shouldn't be changed.
- `TrialTypeVariable`: The name of the variable that contains information as to the trial type. Defaults to trial_type, which is a BIDS standard header for the events files, but can be changed to use any variable.
- `TrialTypeToExtract`: The values of the trial type variable to extract. A warning will be thrown if there are no trials with a given trial type (which might indicate a misspelling or a mistake in this field)
- `TimeConversionFactor`: The factor the onset/duration values need to be divided by to put them into units of seconds. For example, if your onsets are in milliseconds, this factor would be 1000. If in seconds, the factor is 1.
- `ParametricResponseVariable`: The name of the variable in the events file that corresponds to the third column of the FSL 3 column format EV file. If left empty (""), this defaults to 1
- `EVDirectory`: What directory to output the EV files to.

8.2.3 Level 1 Setups

The entry `Level1Setups` contains a list of Level 1 specifications of the following form:

- `ModelName`: Name of this L1 setup. Will be used when you use the `glm_l1_preparefsfs` function
- `TargetDirectory`: Target directory containing the files to be analyzed, defaults to resampled data directory from GLM setup
- `TargetSuffix`: File suffix that specifies which files are to be used, defaults to `resampled.nii.gz`,
- `FSFPrototype`: A .fsf file that acts as the prototype for this setup,
- `ImageIncludeList`: A list of which images should be included in this setup (MUTUALLY EXCLUSIVE WITH `ImageExcludeList`)
- `ImageExcludeList`: A list of which images should NOT be included in this setup (MUTUALLY EXCLUSIVE WITH `ImageIncludeList`)
- `FSFDir`: The directory that the generated .fsf files are created in, defaults to `l1_fsfs`,
- `EVDirectory`: The directory that contains the onset files for each image. These files must be in FSL 3 column format. The filenames have specific structuring as well (see below),
- `ConfoundDirectory`: Directory that contains the confound files, defaults to the directory containing the resampled data,
- `EVFileSuffices`: A list of file suffices that specify which event file to use. NOTE: This list is ordered, so the first suffix corresponds with EV 1, the second with EV 2, etc.
- `ConfoundSuffix`: Suffix that specifies which files are the confound files.
- `OutputDir`: Where the resulting FEAT directories will be created.

8.2.4 Filenames for EV Onset Files

Event Onset files must be in the FSL 3 column format. Additionally, the file names for the onset files must be of the following form: filename of image - target suffix + EV file suffix. For example. If the image filename was "sub-1001_ses-01_task-gng_run-01_bold.nii.gz", the target suffix was "_bold.nii.gz" and a EV suffix was "_hit.txt", then the EV file should be named: "sub-1001_ses-01_task-gng_run-01_hit.txt``.

8.2.5 Level 2 Setups

The entry `Level2Setups` contains a list of Level 2 specifications of the following form:

- `ModelName`: The model name, used in the `glm_l2_preparefsfs` function.
- `FSFPrototype`: A .fsf prototype used in this setup.
- `SubjectFile`: A .csv file containing information as to which images go into which L2 model. See below for details.
- `FSFDir`: The directory in which the fsfs will be generated.
- `OutputDir`: Which folder will the L2 gfeat folders be generated

8.2.6 Subject File Formatting

The L2 subject file maps each image onto a specific L2 model setup entry and onto a specific L2 model (i.e. assigns a subject's images to that subject.) This is a three column csv file, with the headers: `fsf_name`, `feat_folders`, `L2_name`. The `fsf_name` column contains the desired name of a L2 fsf file, the `feat_folders` column contains the paths to the feat folders that are used in the L2 FSF files (in order), and the `L2_name` column contains which `ModelName` corresponds to a given image. For an example, see the `l2_sublist.csv` file generated when you run the `project_setup` function.

8.3 Commands

8.3.1 clpipe glm fsl_onset_extract

Convert onset files to FSL's 3 column format.

```
clpipe glm fsl_onset_extract [OPTIONS]
```

Options

-config_file, -c <config_file>

Required Use a given configuration file.

-glm_config_file, -g <glm_config_file>

Required Use a given GLM configuration file.

-debug, -d

Print detailed processing information and traceback for errors.

8.3.2 clpipe glm fsl_onset_extract

Convert onset files to FSL's 3 column format.

```
clpipe glm fsl_onset_extract [OPTIONS]
```

Options

-config_file, -c <config_file>
: **Required** Use a given configuration file.

-glm_config_file, -g <glm_config_file>
: **Required** Use a given GLM configuration file.

-debug, -d
: Print detailed processing information and traceback for errors.

8.3.3 clpipe glm fsl_onset_extract

Convert onset files to FSL's 3 column format.

```
clpipe glm fsl_onset_extract [OPTIONS]
```

Options

-config_file, -c <config_file>
: **Required** Use a given configuration file.

-glm_config_file, -g <glm_config_file>
: **Required** Use a given GLM configuration file.

-debug, -d
: Print detailed processing information and traceback for errors.

8.3.4 clpipe glm fsl_onset_extract

Convert onset files to FSL's 3 column format.

```
clpipe glm fsl_onset_extract [OPTIONS]
```

Options

-config_file, -c <config_file>
: **Required** Use a given configuration file.

-glm_config_file, -g <glm_config_file>
: **Required** Use a given GLM configuration file.

-debug, -d
: Print detailed processing information and traceback for errors.

8.3.5 Legacy Commands

glm_l1_preparefsf

Propagate an .fsf file template for L1 GLM analysis.

You must create a template .fsf file in FSL's FEAT GUI first.

```
glm_l1_preparefsf [OPTIONS]
```

Options

-glm_config_file, -g <glm_config_file>
: **Required** Your GLM configuration file.

-l1_name <l1_name>
: **Required** Name for a given L1 model as defined in your GLM configuration file.

-debug, -d
: Flag to enable detailed error messages and traceback

glm_l1_launch

Launch all prepared .fsf files for L1 GLM analysis.

```
glm_l1_launch [OPTIONS]
```

Options

-glm_config_file, -g <glm_config_file>
: **Required** The path to your clpipe configuration file.

-l1_name <l1_name>
: **Required** Name of your L1 model

-test_one
: Only submit one job for testing purposes.

-submit, -s
: Flag to submit commands to the HPC.

-debug, -d
: Flag to enable detailed error messages and traceback.

glm_l2_preparefsf

Propagate an .fsf file template for L2 GLM analysis.

You must create a group-level template .fsf file in FSL's FEAT GUI first.

```
glm_l2_preparefsf [OPTIONS]
```

Options

-glm_config_file, -g <glm_config_file>

Required Your GLM configuration file.

-l2_name <l2_name>

Required Name for a given L2 model

-debug, -d

Flag to enable detailed error messages and traceback

glm_l2_launch

Launch all prepared .fsf files for L2 GLM analysis.

```
glm_l2_launch [OPTIONS]
```

Options

-glm_config_file, -g <glm_config_file>

Required The path to your clpipe configuration file.

-l2_name <l2_name>

Required Name of your L2 model

-test_one

Only submit one job for testing purposes.

-submit, -s

Flag to submit commands to the HPC.

-debug, -d

Flag to enable detailed error messages and traceback.

CHAPTER

NINE

ROI EXTRACTION

9.1 Overview

clpipe comes with a variety of functional and anatomical atlases, which can be used to extract ROI time series data from functional scans.

By default, ROIs are calculated with respect an image's fMRIPrep brain mask. ROIs with a with a percentage of voxels outside of this mask exceeding "PropVoxels" will be set to "nan". If any ROI has no voxels in the brain mask, then all ROIs will be extracted without respect to the brain mask, and then ROIs with fewer than "PropVoxels" voxels will be set to "nan". This is a workaround for the limitations on Nilearn's ROI extractor functions.

To view the available built-in atlases, you can use the `roi atlases` command.

9.2 Configuration

class clpipe.config.options.**ROIExtractOptions**

Options for ROI extraction.

target_directory: str = ''

Target folder for processing - usually an fMRIPrep output directory.

target_suffix: str = 'desc-postproc_bold.nii.gz'

Narrow down the images to use by specifying the path's suffix. Use 'desc-preproc_bold.nii.gz' if targeting the fMRIPrep dir.

output_directory: str = 'data_ROI_ts'

Location of this command's output. Defaults to data_ROI_ts.

atlases: list

List of atlases to use. Use 'clpipe roi atlases' to show available atlases.

require_mask: bool = True

Choose whether or not an accompanying mask for each image is required in the target directory.

prop_voxels: float = 0.5

ROIs with less than this proportion of voxels within the mask area are set to nan.

overlap_ok: bool = False

Are overlapping ROIs allowed?

9.3 Commands

9.3.1 clpipe roi extract

Extract ROIs with a given atlas.

```
clpipe roi extract [OPTIONS] [SUBJECTS]...
```

Options

-config_file, -c <config_file>

Use a given configuration file. If left blank, uses the default config file, requiring definition of BIDS, working and output directories. This will extract all ROI sets specified in the configuration file.

-target_dir, -i <target_dir>

Which postprocessed directory to process. If a configuration file is provided with a target directory, this argument is not necessary.

-target_suffix <target_suffix>

Which target suffix to process. If a configuration file is provided with a target suffix, this argument is not necessary.

-output_dir, -o <output_dir>

Where to put the ROI extracted data. If a configuration file is provided with a output directory, this argument is not necessary.

-task <task>

Which task to process. If none, then all tasks are processed.

-atlas_name <atlas_name>

What atlas to use. Use the command 'clpipe roi atlases' to see which are available. When specified for a custom atlas, this is what the output files will be named.

-custom_atlas <custom_atlas>

A custom atlas image, in .nii or .nii.gz for label or maps, or a .txt tab delimited set of ROI coordinates if for a sphere atlas. Not needed if specified in config.

-custom_label <custom_label>

A custom atlas label file. Not needed if specified in config.

-custom_type <custom_type>

What type of atlas? (label, maps, or spheres). Not needed if specified in config.

-sphere_radius <sphere_radius>

Sphere radius in mm. Only applies to sphere atlases.

-overlap_ok

Are overlapping ROIs allowed?

-overwrite

Overwrite existing ROI timeseries?

-log_output_dir <log_output_dir>

Where to put HPC output files (such as SLURM output files). If not specified, defaults to <outputDir>/batchOutput.

-submit, -s

Flag to submit commands to the HPC

-single

Flag to directly run command. Used internally.

-debug, -d

Flag to enable detailed error messages and traceback

Arguments

SUBJECTS

Optional argument(s)

9.3.2 clpipe roi atlases

Display all available atlases.

```
clpipe roi atlases [OPTIONS]
```

CHAPTER

TEN

FLYWHEEL SYNC

10.1 Overview

clpipe now provides an avenue for syncing DICOM data with a remote source through the `clpipe flywheel_sync` command.

10.2 Setup

First, the Flywheel CLI must be installed to make use of this command. For UNC-CH users, Flywheel should be automatically loaded as a module when the clpipe module is loaded. If you need to install Flywheel, you can find a link to the installer in your profile on the Flywheel web app.

You will also need to login to Flywheel via the Flywheel CLI to use this command. Navigate to the Flywheel web app. In the upper right, click on your profile drop down menu, select 'profile'. Scroll down and copy the command under 'Getting Started With the CLI.' It should look like: `login <FLYWHEEL URL>::<TOKEN>`. Run this command to login.

10.2.1 Using with convert2bids

Flywheel creates a DICOM folder structure that is too deep for the default depth setting of dcm2niix, which both dcm2bids and heudiconv use to discover DICOM files in your source directory. However, dcm2niix can be configured to search deeper with the `-d` option:

dcm2bids (clpipe default)

dcm2bids provides a method of passing options through to dcm2niix by adding a *dcm2niixOptions* item to your conversion conversion_config.json file, like this:

```
{
"dcm2niixOptions": "-b y -ba y -z y -f '%3s_%f_%p_%t' -d 9",
"descriptions": [
        {
                "dataType": "anat",
                "modalityLabel": "T1w",
                "criteria": {
                        "SeriesDescription": "ADNI3_t1_mprag_sag_p2_iso"
                }
        },
        {
```

(continues on next page)

(continued from previous page)

```
                "criteria": {
                        "SeriesDescription": "RIDL1"
                },
...
```

You must include all options shown, because this argument overwrites the dcm2niixOptions, as opposed to just appending to them.

The options above add the `-d 9` option, setting dcm2niix's search depth to the maximum value.

heudiconv

By default, heudiconv sets the search depth of dcm2niix high enough to find DICOM files within Flywheel's output structure, so no changes are required if you use this converter.

10.2.2 Additional Notes

This command creates its own log folder at `<project>/logs/sync_logs`

One quirk of Flywheel's sync command is that it creates a strangely named temporary directory at the currently working directory, which is empty after the sync is finished. clpipe removes this folder automatically.

10.3 Configuration

Configuration Block

```
"SourceOptions": {
        "SourceURL": "fw://<LAB>/<STUDY>/",
        "DropoffDirectory": "/path/to/your/dicom_folder",
        "TempDirectory": "/path/to/a/temp_folder",
        "CommandLineOpts": "-y",
        "TimeUsage": "1:0:0",
        "MemUsage": "10000",
        "CoreUsage": "1"
},
```

Definitions

class clpipe.config.options.**SourceOptions**

Options for configuring sources of DICOM data.

source_url: str = 'fw://'

The URL to your source data - for Flywheel this should start with fw: and point to a project. You can use fw ls to browse your fw project space to find the right path.

dropoff_directory: str = ''

A destination for your synced data - usually this will be data_DICOMs

temp_directory: str = ''

A location for Flywheel to store its temporary files - necessary on shared compute, because Flywheel will use system level tmp space by default, which can cause issues.

commandline_opts: str = '-y'

Any additional options you may need to include - you can browse Flywheel syncs other options with fw sync -help

10.4 Command

10.4.1 clpipe flywheel_sync

Sync your DICOM data with Flywheel.

You must first login to Flywheel with 'fw login' to sync. See the clpipe documentation on flywheel_sync for further help.

```
clpipe flywheel_sync [OPTIONS]
```

Options

-config_file, -c <config_file>

The path to your clpipe configuration file.

-source_url <source_url>

The path to your project in Flywheel. Starts with fw://. You can browse your available projects with "fw ls"

-dropoff_dir <dropoff_dir>

Where to sync your files.

-submit, -s

Flag to submit commands to the HPC.

-debug, -d

Flag to enable detailed error messages and traceback.

CHAPTER

ELEVEN

CHANGE LOG

11.1 1.9.0 (Sep 15, 2023)

11.1.1 Enhancements

- `postprocess` - postprocess2 is now named postprocess, replacing the original postprocessing functionality
- `postprocess` - Wildcards can now be used in the `target_variables` of the scrub configuration to select multiple columns with similar names, such as non_steady_state_outliers*
- `postprocess` - BIDS index now saves to the user's working directory
- `postprocess` - Logs now saved in folders according to stream, like the output and working directories
- `postprocess` - Distributor-level slurm jobs removed, simplifying job structure and removing the distributor log folder
- `postprocess` - Individual images now get their own rolling log files
- `postprocess` - Slurm output files are now saved to a separate `slurm_out` folder
- `postprocess` - There are now a default processing streams setup and named specifically for the GLM and functional connectivity analyses
- `glm` - GLM setup command now completely removed, in favor of using the GLM postprocessing stream
- `documentation` - Expanded documentation for Postprocessing section
- `documentation` - Sections reorganized and made more consistent with each other.
- `project_setup` - Now prompts for name of project if not given.
- `config update` - Converts config to new format and offers to backup old version

11.1.2 Bug Fixes

- `postprocess` - Fixed an issue where postprocess took excessively long to index large datasets due to a bug in pybids
- `postprocess` - Issue where streams did not properly update postprocessing config fixed

11.1.3 Deprecations & Removals

- `postprocess` - Removed original postprocessing command
- `postprocess` - Removed original susan command; now a step of postprocess
- `postprocess` - Disabled fmri-process-check report for now, pending rework
- `postprocess` - The stream file "processing_description.json" has been moved to the stream working directory and is now called "run_config.json"

11.1.4 Development

- `configuration` - Dataclass-based configuration has been applied to all major clpipe commands
- `configuration` - Configuration is now serialized/deserialized with Marshmallow, which allows both JSON and YAML file types to be used
- `postprocess` - Global workflow now constructs image and confounds workflows on its own

11.2 1.8.1 (Jun 28, 2023)

11.2.1 Enhancements

- `postproc2` - Added new step ScrubTimepoints, allowing timepoints to be removed from the image and confounds where a set threshold is exceeded for a chosen confound variable.
- `postproc2` - Wildcard statements such as `t_comp_cor*` can now be used in the `Columns` section of `ConfoundsOptions` to select multiple columns at once.
- `postproc2` - Added special case to Temporal Filtering, which will interpolate over any values removed in the ScrubTimepoints step. See the documentation at Postprocessing/postprocess2/Processing Step Options/Temporal Filtering

11.2.2 Development

- `clpipe` - Contribution guide is now in its own markdown file separate from the setup guide. More details added to guide
- `tests` - New helper added for plotting timeseries data
- `tests` - Longer 100 timepoint sample image added for steps like Temporal Filtering that need more data for effective testing

11.3 1.8.0 (Apr 05, 2023)

11.3.1 GLM Setup Command Deprecation

- `glm setup`: command deprecated, to be replaced by postprocess2
- `glm setup`: config file no longer contains GLMSetupOptions. TaskName, ReferenceImage, and ParentClpipeConfig options previously in GLMSetupOptions have been moved to the top level of the configuration file, as they are still used by Level 1 & 2 setups.

- `glm setup`: Will still run "classic" glm setup pipeline when using a clpipe < 1.8 style glm config file, but prints a warning. Will print deprecation error and exit if using new-style glm setup config file
- `project_setup`: The default TargetDirectory and ConfoundDirectory of the default glm config file now point to `postproc2/default`
- `project_setup`: The default folder `data_GLMPrep` is no longer created
- `project_setup`: Log folders `logs/glm_logs/L1_launch` and `logs/glm_logs/L2_launch` are created instead of `glm_setup_logs`, and these path are auto-populated in the LogDir fields of the default glm config file

11.3.2 Enhancements

- `preprocess`: `/work` added to the list of UNC bind paths, as `/pine` is being deprecated by Longleaf
- `preprocess`: when templateflow toggle is on, this step now automatically creates a .cache/templateflow folder for you in your home directory if it doesn't already exist
- `glm prepare`: Changed some message exclusive to the `-debug` flag to be viewable without the flag to improve verbosity of the command. Command also now gives a message when completed, won't run if no images are found, and won't run if the EV target folder doesn't exist
- `clpipe`: The `bids_validate` command was moved out of the `bids` sub-command and moved to the top level to make it easier to find/use and more closely resemble the original clpipe command. The `bids` sub-command is still useable, but has been hidden, and will be removed in a future update
- `clpipe`: Similar to above, the `dicom` command has been hidden, and its sub-commands `convert2bids` and `flywheel_sync` have been moved to the top level of the clpipe menu. The dicom command is still accessible, but hidden.
- `clpipe`: The `setup` command has been renamed `project_setup` for consistency with the original command, and to avoid a conflict where a general setup command for clpipe might be necessary (as opposed to setting up a project)
- `clpipe`: The double-dash form `--help` and `--version` options have been removed for consistency with the other commands. The short hand of help, `-h`, is removed for simplicity. The forms `-help` and `-version, -v` remain.
- `project_setup`: Several fields of config and glm_config files that need to be set by user, but appear blank in the starter config files, are now populated like "SET THIS FIELD" to make it clearer that they must be set
- `clpipe`: New documentation page "Overview" added to house top-level cli command info and future location for clpipe overview diagrams

11.3.3 Bug Fixes

- `clpipe`: Fixed issue where username for clpipe.log file was not obtainable from a desktop node, raising an exception
- `glm report_outliers`: Fixed issue where outputs were doubled
- `project_setup`: Fixed bug where the L2 fsfs dir was not auto-populated in the default glm_config file

11.3.4 Development

- `setup`: Decoupled creation of files at project_setup from the path setup of user's config file, paving the way to update the configuration system and making it easier to control when directory setup occurs.
- `tests`: Added fixture to allow testing for backwards compatibility with fmriprep < v21 style directory structure
- `CI/CD`: Added a modulefile generation script to support deploying modules automatically
- `CI/CD`: Updated build and deploy scripts to support automatic deployment of modules

11.4 1.7.3 (Feb 22, 2023)

11.4.1 Enhancements

- `setup`: the "SourceOptions" block for `dicom flywheel_sync` is now included in the default configuration file
- `setup`: more modality examples added to conversion_config.json starter file (T1w and fmap)
- `setup`: conversion_config.json starter file now includes dcm2niix customization line which sets its search depth to its maximum value, allowing dcm2bids to work with flywheel's DICOM sync directory
- `setup`: the default `.bidsignore` file now includes `scans.json`, a file generated by heudiconv which would cause validation to fail otherwise
- `bids validate`: In `clpipe_config.json`, Moved `BIDSValidatorImage` from `PostprocessingOptions` block to `BIDSValidationOptions` block. The command will still look for the image in its old location if it can't find it in BIDSValidationOptions, maintaining backwards compatibility with < 1.7.3 config files
- `glm prepare`: improved logging messages
- `glm prepare` : changed how file copying works so only file contents are copied, not their permissions, making the command easier to run in a shared environment
- `clpipe`: `-config_file/-c` is now a required argument in most commands to prevent unhelpful error output if no config file given, and to steer clpipe more towards being configuration file driven
- `clpipe`: ignore writing to clpipe.log if the user does not have permission to do so
- `clpipe`: clpipe.log file now includes usernames
- `clpipe`: clpipe.log file is written with group write permission by default

11.4.2 Bug Fixes

- `setup`: generated glm_config.json file's `[GLMSetupOptions][LogDirectory]` field is now automatically filled out. Previously it was left blank, despite the appropriate log folder being automatically created, and would throw an error in the batch manager if not filled out manually.
- `reports fmriprep`: fixed issue where the main .html report files were not being bundled into the output zip
- `glm prepare`: fixed issue where FileNotFoundErrors were not caught correctly, causing the program to exit earlier than intended

11.5 1.7.2 (Jan 31, 2023)

11.5.1 Flywheel Sync

clpipe can now be used to sync DICOM data from Flywheel. Using Flywheel through clpipe allows the user to store their project's remote Flywheel path and download destination in their clpipe configuration file, and to automatically submit Flywheel sync commands to a cluster as batch jobs. clpipe also cleans up an oddly-named, empty temporary directory that is left behind when running Flywheel's sync command.

This feature is accessed with the command `clpipe dicom flywheel_sync`. The clpipe subcommand `clpipe dicom` was added to give this command a "home" and to house future dicom-specific commands.

```
clpipe dicom flywheel_sync -c /path/to/your/clpipe_config.json -submit
```

11.5.2 GLM

- combined `l1_prepare_fsf` and `l2_prepare_fsf` into the `clpipe glm prepare` command to match how the launch command works:

```
> clpipe glm prepare
Usage: clpipe glm prepare [OPTIONS] LEVEL MODEL

  Propagate an .fsf file template for L1 or L2 GLM analysis.

  LEVEL is the level of anlaysis, L1 or L2

  MODEL must be a a corresponding L1 or L2 model from your GLM configuration
  file.

Options:
  -glm_config_file FILE  The path to your clpipe configuration file.
                         [required]
  -debug                 Flag to enable detailed error messages and traceback.
  -help, -h, --help      Show this message and exit.
```

- `l1_prepare_fsf` no longer leaves an underscore at the end of the .fsf file names or the feat output directories
- L1 and L2 config files now support LogDirectory options for specifying where to send the output of the launch command. The output folder is used by default if no LogDirectory is specified.
- Improved error handling for `clpipe glm prepare` command

11.5.3 clpipe bids convert

- Command has been moved from `clpipe bids` to the new `clpipe dicom`, which better reflects the BIDS conversion operation (a command acting on DICOM data to convert it to BIDS format). It has also been renamed from from `convert` back to its original name, `convert2bids`, to improve the clarity of this command's name.

Usage now looks like this:

```
clpipe dicom convert2bids -c path/to/my/clpipe_config.json 256 -submit
```

11.5.4 postproc2

- now supports either fmriprep v20 or v21's directory structure
- no longer deletes "processing_graph.dot" (source of "processing_graph.png" to avoid a race condition issues sometimes causing jobs to fail

11.5.5 get_reports

Command `get_reports`, used for combining fMRIPrep QC report outputs into a ZIP archive:

- has been added to the main clpipe menu as `clpipe reports fmriprep`
- now prints more log messages
- creates an archive with a shorter directory path
- archive is now named "fMRIPrep_Archive" by default

11.5.6 Other Updates

- Default version of fMRIPrep referenced in config updated to v21.0.2
- Shorthand commands (e.g. -c for config_file, -s for submit) have been made consistent across all commands under the `clpipe` command
- `clpipe postproc` no longer fails silently when no subjects are found - error is now raised

CHAPTER

TWELVE

PROJECT SETUP

12.1 Installation and Folder Setup

First, install clpipe using pip and Github

```
pip3 install --upgrade git+https://github.com/cohenlabUNC/clpipe.git
```

Create a new folder for your project

```
mkdir clpipe_tutorial_project
cd clpipe_tutorial_project
```

To run the project setup, you need to have a source data directory prepared. For now, please create an empty one.

```
mkdir data_DICOMs
```

Now you are ready to run the project_setup command

12.2 Running project_setup

```
project_setup -project_title clpipe_tutorial_project -project_dir . -source_data data_
↪DICOMs
```

If successful, your folder will now contain the following structure:

```
.
├── analyses
├── clpipe_config.json
├── conversion_config.json
├── data_BIDS
├── data_DICOMs
├── data_fmriprep
├── data_GLMPrep
├── data_onsets
├── data_postproc
├── data_ROI_ts
├── glm_config.json
├── l1_feat_folders
├── l1_fsfs
```

(continues on next page)

(continued from previous page)

```
├── l2_fsfs
├── l2_gfeat_folders
├── l2_sublist.csv
├── logs
└── scripts
```

clpipe automatically creates many of the directories we will need in the future. For now, let's just familiarize ourselves with the most important file, `clpipe_config.json`, which allows you to configure clpipe's core functionalities. Open `clpipe_config.json` with the editor of your choice.

12.3 Understanding the clpipe_config.json File

There is quite a bit going on in this file, because it controls most of clpipe's processing options. As a `.json` file, this configuration is organized as a collection of key/value pairs, such as:

```
"ProjectTitle": "A Neuroimaging Project"
```

The key here is "ProjectTitle", an attribute corresponding to the project's name, and the value is "A Neuroimaging Project", the name of the project.

Examine the first few lines of the file, which contain metadata about your project:

```
"ProjectTitle": "clpipe_tutorial_project",
"Authors/Contributors": "",
"ProjectDirectory": "<your system's path>/clpipe_tutorial_project",
"EmailAddress": "",
"TempDirectory": "",
```

Notice that the project directory and title have already been filled in by clpipe.

Let's make our first configuration change by setting your name as the author, and providing your email address -

```
"ProjectTitle": "clpipe_tutorial_project",
"Authors/Contributors": "Your Name Here",
"ProjectDirectory": "/nas/longleaf/home/willasc/data/clpipe/clpipe_tutorial_project",
"EmailAddress": "myemail@domain.com",
"TempDirectory": "",
```

Values in a key/value pair are not just limited to text - we can also have a list of more key/value pairs, which allows for hierarchial structures:

```
"top-level-key": {
        "key1":"value",
        "key2":"value",
        "key3":"value"
}
```

The options for clpipe's various processing steps, such as "DICOMToBIDSOptions", follow this structure:

```
"DICOMToBIDSOptions": {
```

(continues on next page)

(continued from previous page)

```
"DICOMToBIDSOptions": {
 "DICOMDirectory": "<your system's path>/clpipe_tutorial_project/data_DICOMs",
 "BIDSDirectory": "<your system's path>/clpipe_tutorial_project/data_BIDS",
 "ConversionConfig": "<your system's path>/clpipe_tutorial_project/conversion_config.json
↪",
 "DICOMFormatString": "",
 "TimeUsage": "1:0:0",
 "MemUsage": "5000",
 "CoreUsage": "2",
 "LogDirectory": "<your system's path>/clpipe_tutorial_project/logs/DCM2BIDS_logs"
 }
}
```

We will go over these processing step options in the following tutorial

CHAPTER

THIRTEEN

BIDS CONVERSION

The primary objective of this processing step is to transform DICOM format images into BIDS format. If your data is already in BIDS format, you can proceed directly to the BIDS Validation step.

Converting DICOM to BIDS involves manual labeling and can be one of the more challenging aspects of setting up clpipe. However, don't be disheartened by this initial step.

> Note: This tutorial is a clpipe implmentation of the dcm2bids tutorial, which clpipe uses for dcm to BIDS conversion.

13.1 Obtaining Sample Raw Data

To obtain the raw DICOM data necessary for this tutorial, run the following commands:

```
cd data_DICOMs
git clone git@github.com:neurolabusc/dcm_qa_nih.git
cd ..
```

Let's examine this data:

```
dcm_qa_nih/In/
├── 20180918GE
│   ├── mr_0004
│   ├── mr_0005
│   ├── mr_0006
│   ├── mr_0007
│   └── README-Study.txt
└── 20180918Si
    ├── mr_0003
    ├── mr_0004
    ├── mr_0005
    ├── mr_0006
    └── README-Study.txt
```

This dataset contains two sets of data, one from a GE scanner, containing functional images, and another from a Siemens, containing field map images. The labels in the form `mr_000x` are subject ids, which will be important for setting up our bids conversion.

> Note: The BIDS data generated in this step will also be used in the BIDS Validation tutorial, but tutorials starting from fMRIprep and on we will use a different BIDS dataset

13.2 clpipe_config.json Setup

Open clpipe_config.json and navigate to the "DICOMToBIDSOptions":

```
"DICOMToBIDSOptions": {
        "DICOMDirectory": "<your system's path>/clpipe_tutorial_project/data_DICOMs",
        "BIDSDirectory": "<your system's path>/clpipe_tutorial_project/data_BIDS",
        "ConversionConfig": "<your system's path>/clpipe_tutorial_project/conversion_
↪config.json",
        "DICOMFormatString": "",
        "TimeUsage": "1:0:0",
        "MemUsage": "5000",
        "CoreUsage": "2",
        "LogDirectory": "<your system's path>/clpipe_tutorial_project/logs/DCM2BIDS_logs"
}
```

This section tells clpipe how to run your BIDS conversion. Note that clpipe has been automatically configured to point to your DICOM directory, "DICOMDirectory", which will serve as the input to the dcm2bids command. The output folder, "BIDSDirectory", is also already set. These can be modified to point to new locations if necessary - for example, you may want to create more than one BIDS directory for testing purposes.

The option "DICOMFormatString" must be set to run your bids conversion. This configuration tells clpipe how to identify subjects and (if relevant) sessions within `data_DICOMs`. To pick up on the subject ids in our example dataset, the placeholder `{subject}` should be given in place of a specific subject's directory. The subject ID cannot contain underscores, so we will include the `mr_` portion of the subject folder name before the `{subject}` placeholder to exclude it from the subject's generated id:

```
"DICOMFormatString": "dcm_qa_nih/In/20180918GE/mr_{subject}"
```

If your data contained an addtional folder layer corresponding to session ids, you would similarily mark this with a `{session}` placeholder

The "ConversionConfig" command gives a path to your automatically generated conversion_config.json file. Let's open that file now:

```
{
        "descriptions": [
                {
                        "dataType": "func",
                        "modalityLabel": "bold",
                        "customLabels": "task-srt",
                        "criteria": {
                                "SeriesDescription": "*_srt",
                                "ImageType": [
                                        "ORIG*",
                                        "PRIMARY",
                                        "M",
                                        "ND",
                                        "MOSAIC"
                                ]
                        }
                }
        ]
}
```

The conversion file contains a list of descriptions, each of which attempts to map raw DICOM images to a given critera. clpipe has prepopulated the conversion config file with an example description.

The "critera" item gives a list of critera by which to match DICOM images. The other tags specify the format of the output NIfTI image that match this critera. "dataType" and "modalityLabel" configure the name of your output

More information on setting up clpipe for BIDS conversion can be found in the clpipe documentation.

!!!!!!!!!!!!!!!!!!!!!! !!!!!!!!!!!!!!!!! !!!!!!!!!!!! !!!!!!

#TODO THIS LINK DOESN'T WORK RIGHT NOW

!!!!!! !!!!!!!!!!! !!!!!!!!!!!!!!!!! !!!!!!!!!!!!!!!!!!!!!!!!

13.3 Setting up the Conversion File

From here, follow the Dcm2Bids tutorial and stop before the "Running dcm2bids" section - clpipe will handling running dcm2bids for you. The helper command `dcm2bids_helper` will be available to you via the clpipe installation, and should be used as indicated in the tutorial to help you get started. You should also skip the "Building the configuration file" step because, as shown above, clpipe has already created this file.

You can find a supplentary explanation and another example of a conversion_config.json file in the clpipe documentation

13.4 Running the Conversion

Now you are ready to launch the conversion with clpipe, which will convert your raw DICOMs into NIfTI format, then rename and sort them into the BIDS standard.

If everything has been set up correctly, running the conversion only takes a simple call to the CLI application with the configuration file as an argument:

```
convert2bids -config_file clpipe_config.json
```

clpipe will then print out a "plan" for executing your jobs:

```
dcm_qa_nih/In/20180918GE/*
<your system's path>/clpipe_tutorial_project/data_DICOMs/dcm_qa_nih/In/20180918GE/
↪{subject}/
sbatch --no-requeue -n 1 --mem=5000 --time=1:0:0 --cpus-per-task=2 --job-name="convert_
↪sub-0004" --output=<your system's path>/clpipe_tutorial_project/logs/DCM2BIDS_logs/
↪Output-convert_sub-0004-jobid-%j.out --wrap="dcm2bids -d <your system's path>/clpipe_
↪tutorial_project/data_DICOMs/dcm_qa_nih/In/20180918GE/0004/ -o <your system's path>/
↪clpipe/clpipe_tutorial_project/data_BIDS -p 0004 -c <your system's path>/clpipe_
↪tutorial_project/conversion_config.json"
sbatch --no-requeue -n 1 --mem=5000 --time=1:0:0 --cpus-per-task=2 --job-name="convert_
↪sub-0005" --output=<your system's path>/clpipe_tutorial_project/logs/DCM2BIDS_logs/
↪Output-convert_sub-0005-jobid-%j.out --wrap="dcm2bids -d <your system's path>clpipe_
↪tutorial_project/data_DICOMs/dcm_qa_nih/In/20180918GE/0005/ -o <your system's path>/
↪clpipe_tutorial_project/data_BIDS -p 0005 -c <your system's path>/clpipe_tutorial_
↪project/conversion_config.json"
...
```

Each `sbatch` command here will submit a separate job to the cluster.

Check that your output looks correct, especially the subject ids, then run again with the `-submit` flag to run your conversions in parallel:

```
convert2bids -config_file clpipe_config.json -submit
```

clpipe should then report that you have submitted 4 jobs to the cluster:

```
dcm_qa_nih/In/20180918GE/*
<your system's path>/clpipe_tutorial_project/data_DICOMs/dcm_qa_nih/In/20180918GE/mr_
↪{subject}/
Submitted batch job 38210854
Submitted batch job 38210855
Submitted batch job 38210856
Submitted batch job 38210857
```

Now, your BIDS directory should look something like this:

```
├── CHANGES
├── code
├── dataset_description.json
├── derivatives
├── participants.json
├── participants.tsv
├── README
├── sourcedata
├── sub-0004
│   └── func
│       ├── sub-0004_task-rest_bold.json
│       └── sub-0004_task-rest_bold.nii.gz
└── tmp_dcm2bids
```

But wait a second! You were expecting four subjects to be in your BIDS directory, but only `sub-0004` is present. The next section will guide you through how to tune your `conversion_config.json` file to pick up all of the images you need.

13.5 Iterating on Your Conversion

Inevitably, you will probably not get the conversion completely correct on the first try, and some files may have been missed.

The folder `tmp_dcm2bids` contains all of the images that were not matched to any of the patterns described in your `conversion_config.json` file, as well as helpful log files:

```
└── tmp_dcm2bids
    ├── log
    │   ├── sub-0004_2022-02-17T103129.039268.log
    │   ├── sub-0005_2022-02-17T103129.116426.log
    │   ├── sub-0006_2022-02-17T103129.082788.log
    │   └── sub-0007_2022-02-17T103129.191004.log
    ├── sub-0004
    ├── sub-0005
    │   ├── 005_0005_Axial_EPI-FMRI_(Sequential_I_to_S)_20180918114023.json
    │   └── 005_0005_Axial_EPI-FMRI_(Sequential_I_to_S)_20180918114023.nii.gz
```

(continues on next page)

(continued from previous page)

```
    ├── sub-0006
    │   ├── 006_0006_Axial_EPI-FMRI_(Interleaved_S_to_I)_20180918114023.json
    │   └── 006_0006_Axial_EPI-FMRI_(Interleaved_S_to_I)_20180918114023.nii.gz
    └── sub-0007
        ├── 007_0007_Axial_EPI-FMRI_(Sequential_S_to_I)_20180918114023.json
        └── 007_0007_Axial_EPI-FMRI_(Sequential_S_to_I)_20180918114023.nii.gz
```

As the raw data folder we have pointed clpipe toward, `20180918GE`, contains functional data, we will look at the `"func"` list item in our `conversion_config.json` to adjust:

```
{
        "dataType": "func",
        "modalityLabel": "bold",
        "customLabels": "task-rest",
        "criteria": {
                "SeriesDescription": "Axial_EPI-FMRI*",
                "SidecarFilename": "*Interleaved_I_to_S*"
        }
}
```

Sidecars are .json files that have the same name as the `.nii.gz` main image file, and contain additional metadata for the image; they are part of the BIDS standard.

This configuration is trying to match on a side car with the pattern `"*Interleaved_I_to_S*"`, but we can see in the `tmp_dcm2bids` folder that none of the subjects here match this pattern. If we want to pick up the remaining subjects, we can relax this critera by removing it from `conversion_config.json` :

```
{
        "dataType": "func",
        "modalityLabel": "bold",
        "customLabels": "task-rest",
        "criteria": {
                "SeriesDescription": "Axial_EPI-FMRI*"
        }
}
```

> Note: Take special care to remove the comma following the “SeriesDescription” list item - using commas in a single-item list will result in invalid JSON that will cause an error

And rerunning the conversion:

```
convert2bids -config_file clpipe_config.json -submit
```

Now, all subjects should be present in your BIDS directory along with their resting state images:

```
...
├── sub-0004
│   └── func
│       ├── sub-0004_task-rest_bold.json
│       └── sub-0004_task-rest_bold.nii.gz
├── sub-0005
│   └── func
│       ├── sub-0005_task-rest_bold.json
│       └── sub-0005_task-rest_bold.nii.gz
```

(continues on next page)

(continued from previous page)

```
├── sub-0006
│   └── func
│       ├── sub-0006_task-rest_bold.json
│       └── sub-0006_task-rest_bold.nii.gz
├── sub-0007
│   └── func
│       ├── sub-0007_task-rest_bold.json
│       └── sub-0007_task-rest_bold.nii.gz
└── tmp_dcm2bids
```

13.6 Adding an Additional Data Source

Our raw data folder at `data_DICOMs/dcm_qa_nih/In/20180918Si` contains additonal images for our sample study. These are field maps that correspond to our functional resting state images.

Due to the location of these field maps being in a separate folder from the functional data, they are a distinct source of data from the point of view of clpipe. Although we could combine the functional images and field maps into one folder, under their respective subjects, often we only want to read from source data.

We can point clpipe to this additional data by modifying the `clpipe_config.json` to point to its path in `DICOMToBIDSOptions`, under `DICOMFormatString`:

```
"DICOMToBIDSOptions": {
        "DICOMDirectory": "<your system's path>/clpipe_tutorial_project/data_DICOMs",
        "BIDSDirectory": "<your system's path>/clpipe_tutorial_project/data_BIDS",
        "ConversionConfig": "<your system's path>/clpipe_tutorial_project/conversion_
↪config.json",
        "DICOMFormatString": "dcm_qa_nih/In/20180918Si/mr_{subject}",
        "TimeUsage": "1:0:0",
        "MemUsage": "5000",
        "CoreUsage": "2",
        "LogDirectory": "<your system's path>/clpipe_tutorial_project/logs/DCM2BIDS_logs"
}
```

We pick up a new subject this way:

```
├── sub-0003
│   └── fmap
│       ├── sub-0003_dir-AP_epi.json
│       └── sub-0003_dir-AP_epi.nii.gz
```

However, our tmp_dcm2bids now contains more images that were not picked up. The images with "EPI" in the title are the field maps that our critera did not match:

```
├── tmp_dcm2bids
│       ├── sub-0004
│   │   ├── 004_0004_Axial_EPI-FMRI_(Interleaved_I_to_S)_20180918114023.json
│   │   └── 004_0004_Axial_EPI-FMRI_(Interleaved_I_to_S)_20180918114023.nii.gz
│   ├── sub-0005
│   │   ├── 005_0005_EPI_PE=RL_20180918121230.json
│   │   └── 005_0005_EPI_PE=RL_20180918121230.nii.gz
```

(continues on next page)

(continued from previous page)

```
    ├── sub-0006
    │   ├── 006_0006_EPI_PE=LR_20180918121230.json
    │   └── 006_0006_EPI_PE=LR_20180918121230.nii.gz
```

Like our last iteration, we can adjust the conversion_config.json file to pick up these images. However, the two fmap critera we have already are properly picking up on the AP/PA field maps. Create two new critera to match the field maps found in `sub-0005` and `sub-0006`, one from LR and another for RL:

```
{
        "dataType": "fmap",
        "modalityLabel": "epi",
        "customLabels": "dir-RL",
        "criteria": {
                "SidecarFilename": "*EPI_PE=RL*"
        },
        "intendedFor": 0
},
{
        "dataType": "fmap",
        "modalityLabel": "epi",
        "customLabels": "dir-LR",
        "criteria": {
                "SidecarFilename": "*EPI_PE=LR*"
        },
        "intendedFor": 0
}
```

> Note: The "intendedFor" field points the fieldmap critera to the index of its corresponding functional image critera. In this case, we only have one functional image critera, for the resting state image, which is listed first (index 0). Therefore, it is important that the resting image critera stays first in the list; otherwise, these indexes would need to be updated.

After running the conversion again, you should now see that `sub-0005` and `sub-0006` have fieldmap images in addition to their functional scans:

```
├── sub-0005
│   ├── fmap
│   │   ├── sub-0005_dir-RL_epi.json
│   │   └── sub-0005_dir-RL_epi.nii.gz
│   └── func
│       ├── sub-0005_task-rest_bold.json
│       └── sub-0005_task-rest_bold.nii.gz
├── sub-0006
│   ├── fmap
│   │   ├── sub-0006_dir-LR_epi.json
│   │   └── sub-0006_dir-LR_epi.nii.gz
│   └── func
│       ├── sub-0006_task-rest_bold.json
│       └── sub-0006_task-rest_bold.nii.gz
```

Great! However, this BIDS directory is not finished quite yet - next we will use the BIDS validator tool to detect the problems with our BIDS directory.

CHAPTER

FOURTEEN

BIDS VALIDATION

14.1 Running the bids_validate command

Use the `bids_validate` command without the `-submit` flag to check your execution plan:

```
bids_validate -config_file clpipe_config.json
```

```
sbatch --no-requeue -n 1 --mem=3000 --time=1:0:0 --cpus-per-task=1 --job-name=
↪"BIDSValidator" --output=<your system's path>/clpipe_tutorial_project/Output-
↪BIDSValidator-jobid-%j.out --wrap="singularity run --cleanenv -B /proj,/pine,/nas02,/
↪nas <your validation image path>/validator.simg <your system's path>/clpipe_tutorial_
↪project/data_BIDS"
```

If you are happy with the submission plan, add the submit flag:

```
bids_validate -config_file clpipe_config.json -submit
```

14.2 Interpreting & Fixing the Validation Results

You should see the output of your validation at the root of your project directory, like this (it really should end up in the logs folder - we're working on it!):

```
├── analyses
├── clpipe_config.json
├── conversion_config.json
├── data_BIDS
├── data_DICOMs
├── data_fmriprep
├── data_GLMPrep
├── data_onsets
├── data_postproc
├── data_ROI_ts
├── glm_config.json
├── l1_feat_folders
├── l1_fsfs
├── l2_fsfs
├── l2_gfeat_folders
├── l2_sublist.csv
```

(continues on next page)

(continued from previous page)

```
├── logs
├── Output-BIDSValidator-jobid-41362508.out
```

Now open this file - there will be two types of issues, `[ERR]` for errors and `[WARN]` for warnings. The errors must be resolved for the dataset to be considered a valid BIDS dataset. Warnings are important to review for insight into further potential problems in the data, but do not invalidate your dataset.

> Note: fMRIPrep runs BIDS validation again before it starts processing, and will not start if the dataset contains any errors!

14.2.1 Error #1

Let's start with the first error:

```
        [31m1: [ERR] Files with such naming scheme are not part of BIDS specification...
                ./tmp_dcm2bids/log/sub-0003_2022-03-23T155433.420971.log
                        Evidence: sub-0003_2022-03-23T155433.420971.log
                ./tmp_dcm2bids/log/sub-0004_2022-03-23T153854.035740.log
```

If you look closely, the BIDS validator is complaining about the tmp_dcm2bids files, which are not intended to be part of the dataset! In order to ask for this folder to not be considered part of the BIDS dataset, we need to specify this in a `.bidsignore` file.

Create a `.bidsignore` file in your BIDS directory:

```
touch data_BIDS/.bidsignore
```

Now, open this file and add the folder you'd like to ignore:

```
tmp_dcm2bids
```

Next, rerun the validation command and open your new validation results (make sure you aren't looking at the old results file again!). You should see that the error message about the tmp_dcm2bids folder is gone.

> Note: We plan to have clpipe automatically create this file soon

14.2.2 Error #2

The next error should look like this:

```
[31m1: [ERR] 'IntendedFor' field needs to point to an existing file. (code: 37 -␣
↪INTENDED_FOR)[39m
                ./sub-0003/fmap/sub-0003_dir-AP_epi.nii.gz
                        Evidence: func/sub-0003_task-rest_bold.nii.gz
```

It looks like sub-0003's 'IntendedFor' filed points to a non existent file. Let's verify this by opening the subject's .json sidecar, located at `data_BIDS/sub-0003/fmap/sub-0003_dir-AP_epi.json`

At the bottom of the file, we can see that sub-0003's IntendedFor field is pointing to a function image, but this subject has no functional images!

```
...
    "InPlanePhaseEncodingDirectionDICOM": "COL",
    "ConversionSoftware": "dcm2niix",
    "ConversionSoftwareVersion": "v1.0.20190902",
    "Dcm2bidsVersion": "2.1.6",
    "IntendedFor": "func/sub-0003_task-rest_bold.nii.gz"
}
```

Let's erase this field (don't forget to remove the comma on the line before it, too):

```
...
    "InPlanePhaseEncodingDirectionDICOM": "COL",
    "ConversionSoftware": "dcm2niix",
    "ConversionSoftwareVersion": "v1.0.20190902",
    "Dcm2bidsVersion": "2.1.6"
}
```

And try again. Now, the error message for this subject should be gone.

14.2.3 Error #3

The final error is asking for the 'TaskName' on our rest data:

```
[31m1: [ERR] You have to define 'TaskName' for this file. (code: 50 - TASK_NAME_MUST_
↪DEFINE)[39m
                ./sub-0004/func/sub-0004_task-rest_bold.nii.gz
                ./sub-0005/func/sub-0005_task-rest_bold.nii.gz
                ./sub-0006/func/sub-0006_task-rest_bold.nii.gz
                ./sub-0007/func/sub-0007_task-rest_bold.nii.gz
```

This error is asking us to include a "TaskName" field in our .json sidecar files. Luckily, we can ask dcm2bids to specify this in the `conversion_config.json` file. Open up `conversion_config.json` and add the `sidecarChanges` field to specify a task name to automatically add to our generated sidecar files:

```
{
    "descriptions": [
        {
        "dataType": "func",
        "modalityLabel": "bold",
        "customLabels": "task-rest",
        "sidecarChanges":{
            "TaskName": "rest"
        },
        "criteria": {
            "SeriesDescription": "Axial_EPI-FMRI*"
            }
        },
...
```

For this change, we will need to rerun the BIDS conversion. However, because these rest images were already successfully sorted into BIDS format, we will need to add the `-overwrite` flag to our `convert2bids` coomand (which calls dcm2bid's `--forceDcm2niix` and `--clobber` options under the hood)

Now we will have a clean slate when rerunning convert2bids, and we can see that the rest image sidecars now contain the `TaskName` field:

```
...
    "InPlanePhaseEncodingDirectionDICOM": "COL",
    "ConversionSoftware": "dcm2niix",
    "ConversionSoftwareVersion": "v1.0.20190902",
    "Dcm2bidsVersion": "2.1.6",
    "TaskName": "rest"
}
```

Finally, because we used the `-overwrite` flag, sub-0003's IntendedFor field will be re-inserted (Error #2). Repeat the fix for this problem by removing the IntendedFor field from sub-0003's sidecar .json.

Now, rerun bids_validate - you should be completely free of errors!

INDEX

Symbols

-BIDS_dir
    clpipe-convert2bids command line option, 14
-atlas_name
    clpipe-roi-extract command line option, 46
-batch
    clpipe-postprocess command line option, 35
-bids_dir
    clpipe-preprocess command line option, 20
-c
    clpipe-bids_validate command line option, 18
    clpipe-convert2bids command line option, 14
    clpipe-flywheel_sync command line option, 51
    clpipe-glm-fsl_onset_extract command line option, 40, 41
    clpipe-postprocess command line option, 34
    clpipe-preprocess command line option, 20
    clpipe-reports-fmriprep command line option, 21
    clpipe-roi-extract command line option, 46
-cache
    clpipe-postprocess command line option, 35
-clear_temp
    clpipe-reports-fmriprep command line option, 21
-config_file
    clpipe-bids_validate command line option, 18
    clpipe-convert2bids command line option, 14
    clpipe-flywheel_sync command line option, 51
    clpipe-glm-fsl_onset_extract command line option, 40, 41
    clpipe-postprocess command line option, 34
    clpipe-preprocess command line option, 20
    clpipe-reports-fmriprep command line option, 21
    clpipe-roi-extract command line option, 46
-conv_config_file
    clpipe-convert2bids command line option, 14
-custom_atlas
    clpipe-roi-extract command line option, 46
-custom_label
    clpipe-roi-extract command line option, 46
-custom_type
    clpipe-roi-extract command line option, 46
-d
    clpipe-bids_validate command line option, 18
    clpipe-convert2bids command line option, 14
    clpipe-flywheel_sync command line option, 51
    clpipe-glm-fsl_onset_extract command line option, 40, 41
    clpipe-postprocess command line option, 35
    clpipe-preprocess command line option, 21
    clpipe-reports-fmriprep command line option, 21
    clpipe-roi-extract command line option, 47
    glm_l1_launch command line option, 42
    glm_l1_preparefsf command line option, 42
    glm_l2_launch command line option, 43
    glm_l2_preparefsf command line option, 43
-dcm2bids
    clpipe-convert2bids command line option,

14
-debug
clpipe-bids_validate command line option, 18
clpipe-convert2bids command line option, 14
clpipe-flywheel_sync command line option, 51
clpipe-glm-fsl_onset_extract command line option, 40, 41
clpipe-postprocess command line option, 35
clpipe-preprocess command line option, 21
clpipe-project_setup command line option, 10
clpipe-reports-fmriprep command line option, 21
clpipe-roi-extract command line option, 47
glm_l1_launch command line option, 42
glm_l1_preparefsf command line option, 42
glm_l2_launch command line option, 43
glm_l2_preparefsf command line option, 43
-dicom_dir
clpipe-convert2bids command line option, 14
-dicom_dir_format
clpipe-convert2bids command line option, 14
-dropoff_dir
clpipe-flywheel_sync command line option, 51
-fmriprep_dir
clpipe-postprocess command line option, 34
-g
clpipe-glm-fsl_onset_extract command line option, 40, 41
glm_l1_launch command line option, 42
glm_l1_preparefsf command line option, 42
glm_l2_launch command line option, 43
glm_l2_preparefsf command line option, 43
-glm_config_file
clpipe-glm-fsl_onset_extract command line option, 40, 41
glm_l1_launch command line option, 42
glm_l1_preparefsf command line option, 42
glm_l2_launch command line option, 43
glm_l2_preparefsf command line option, 43
-heudiconv
clpipe-convert2bids command line option, 14
-i
clpipe-convert2bids command line option, 14
clpipe-postprocess command line option, 34
clpipe-preprocess command line option, 20
clpipe-roi-extract command line option, 46
-index_dir
clpipe-postprocess command line option, 35
-interactive
clpipe-bids_validate command line option, 18
-keep_temp
clpipe-reports-fmriprep command line option, 21
-l1_name
glm_l1_launch command line option, 42
glm_l1_preparefsf command line option, 42
-l2_name
glm_l2_launch command line option, 43
glm_l2_preparefsf command line option, 43
-log_dir
clpipe-bids_validate command line option, 18
clpipe-convert2bids command line option, 14
clpipe-postprocess command line option, 35
clpipe-preprocess command line option, 20
-log_output_dir
clpipe-roi-extract command line option, 46
-longitudinal
clpipe-convert2bids command line option, 14
-move_source_data
clpipe-project_setup command line option, 10
-no-batch
clpipe-postprocess command line option, 35
-no-cache
clpipe-postprocess command line option, 35
-o
clpipe-convert2bids command line option, 14
clpipe-postprocess command line option, 34
clpipe-preprocess command line option, 20
clpipe-reports-fmriprep command line option, 21
clpipe-roi-extract command line option, 46

-output_dir
    clpipe-postprocess command line option, 34
    clpipe-preprocess command line option, 20
    clpipe-roi-extract command line option, 46
-output_name
    clpipe-reports-fmriprep command line option, 21
-overlap_ok
    clpipe-roi-extract command line option, 46
-overwrite
    clpipe-convert2bids command line option, 14
    clpipe-roi-extract command line option, 46
-p
    clpipe-postprocess command line option, 35
-processing_stream
    clpipe-postprocess command line option, 35
-project_dir
    clpipe-project_setup command line option, 10
-project_title
    clpipe-project_setup command line option, 10
-r
    clpipe-postprocess command line option, 35
-refresh_index
    clpipe-postprocess command line option, 35
-s
    clpipe-bids_validate command line option, 18
    clpipe-convert2bids command line option, 14
    clpipe-flywheel_sync command line option, 51
    clpipe-postprocess command line option, 35
    clpipe-preprocess command line option, 21
    clpipe-roi-extract command line option, 46
    glm_l1_launch command line option, 42
    glm_l2_launch command line option, 43
-session
    clpipe-convert2bids command line option, 14
-single
    clpipe-roi-extract command line option, 47
-source_data
    clpipe-project_setup command line option, 10
-source_url
    clpipe-flywheel_sync command line option, 51
-sphere_radius
    clpipe-roi-extract command line option, 46
-subject
    clpipe-convert2bids command line option, 14
-submit
    clpipe-bids_validate command line option, 18
    clpipe-convert2bids command line option, 14
    clpipe-flywheel_sync command line option, 51
    clpipe-postprocess command line option, 35
    clpipe-preprocess command line option, 21
    clpipe-roi-extract command line option, 46
    glm_l1_launch command line option, 42
    glm_l2_launch command line option, 43
-symlink_source_data
    clpipe-project_setup command line option, 10
-target_dir
    clpipe-roi-extract command line option, 46
-target_suffix
    clpipe-roi-extract command line option, 46
-task
    clpipe-roi-extract command line option, 46
-test_one
    glm_l1_launch command line option, 42
    glm_l2_launch command line option, 43
-v
    clpipe-bids_validate command line option, 18
-verbose
    clpipe-bids_validate command line option, 18
-working_dir
    clpipe-preprocess command line option, 20

A

AROMARegression (*class in clpipe.config.options*), 29

atlases (*clpipe.config.options.ROIExtractOptions attribute*), 45

B

BatchOptions (*class in clpipe.config.options*), 33
BIDS_DIR
    clpipe-bids_validate command line option, 18
bids_directory (*clpipe.config.options.Convert2BIDSOptions attribute*), 11
bids_directory (*clpipe.config.options.FMRIPrepOptions attribute*), 19
bids_validation (*clpipe.config.options.ProjectOptions attribute*), 6
bids_validator_image (*clpipe.config.options.BIDSValidatorOptions attribute*), 17
BIDSValidatorOptions (*class in clpipe.config.options*), 17

C

clpipe-bids_validate command line option
    -c, 18
    -config_file, 18
    -d, 18
    -debug, 18
    -interactive, 18
    -log_dir, 18
    -s, 18
    -submit, 18
    -v, 18
    -verbose, 18
    BIDS_DIR, 18
clpipe-convert2bids command line option
    -BIDS_dir, 14
    -c, 14
    -config_file, 14
    -conv_config_file, 14
    -d, 14
    -dcm2bids, 14
    -debug, 14
    -dicom_dir, 14
    -dicom_dir_format, 14
    -heudiconv, 14
    -i, 14
    -log_dir, 14
    -longitudinal, 14
    -o, 14
    -overwrite, 14
    -s, 14
    -session, 14
    -subject, 14
    -submit, 14
    SUBJECTS, 15
clpipe-flywheel_sync command line option
    -c, 51
    -config_file, 51
    -d, 51
    -debug, 51
    -dropoff_dir, 51
    -s, 51
    -source_url, 51
    -submit, 51
clpipe-glm-fsl_onset_extract command line option
    -c, 40, 41
    -config_file, 40, 41
    -d, 40, 41
    -debug, 40, 41
    -g, 40, 41
    -glm_config_file, 40, 41
clpipe-postprocess command line option
    -batch, 35
    -c, 34
    -cache, 35
    -config_file, 34
    -d, 35
    -debug, 35
    -fmriprep_dir, 34
    -i, 34
    -index_dir, 35
    -log_dir, 35
    -no-batch, 35
    -no-cache, 35
    -o, 34
    -output_dir, 34
    -p, 35
    -processing_stream, 35
    -r, 35
    -refresh_index, 35
    -s, 35
    -submit, 35
    SUBJECTS, 35
clpipe-preprocess command line option
    -bids_dir, 20
    -c, 20
    -config_file, 20
    -d, 21
    -debug, 21
    -i, 20
    -log_dir, 20
    -o, 20
    -output_dir, 20
    -s, 21
    -submit, 21
    -working_dir, 20
    SUBJECTS, 21
clpipe-project_setup command line option

-debug, 10
-move_source_data, 10
-project_dir, 10
-project_title, 10
-source_data, 10
-symlink_source_data, 10
clpipe-reports-fmriprep command line option
-c, 21
-clear_temp, 21
-config_file, 21
-d, 21
-debug, 21
-keep_temp, 21
-o, 21
-output_name, 21
clpipe-roi-extract command line option
-atlas_name, 46
-c, 46
-config_file, 46
-custom_atlas, 46
-custom_label, 46
-custom_type, 46
-d, 47
-debug, 47
-i, 46
-log_output_dir, 46
-o, 46
-output_dir, 46
-overlap_ok, 46
-overwrite, 46
-s, 46
-single, 47
-sphere_radius, 46
-submit, 46
-target_dir, 46
-target_suffix, 46
-task, 46
SUBJECTS, 47
columns (*clpipe.config.options.ConfoundOptions attribute*), 32
commandline_opts (*clpipe.config.options.FMRIPrepOptions attribute*), 19
commandline_opts (*clpipe.config.options.SourceOptions attribute*), 50
ConfoundOptions (*class in clpipe.config.options*), 32
ConfoundRegression (*class in clpipe.config.options*), 29
contributors (*clpipe.config.options.ProjectOptions attribute*), 6
conversion_config (*clpipe.config.options.Convert2BIDSOptions attribute*), 11
convert2bids (*clpipe.config.options.ProjectOptions attribute*), 6
Convert2BIDSOptions (*class in clpipe.config.options*), 11

D

dicom_directory (*clpipe.config.options.Convert2BIDSOptions attribute*), 11
dicom_format_string (*clpipe.config.options.Convert2BIDSOptions attribute*), 11
docker_fmriprep_version (*clpipe.config.options.FMRIPrepOptions attribute*), 20
docker_toggle (*clpipe.config.options.FMRIPrepOptions attribute*), 20
dropoff_directory (*clpipe.config.options.SourceOptions attribute*), 50

E

email_address (*clpipe.config.options.ProjectOptions attribute*), 6

F

filtering_high_pass (*clpipe.config.options.TemporalFiltering attribute*), 26
filtering_low_pass (*clpipe.config.options.TemporalFiltering attribute*), 26
filtering_order (*clpipe.config.options.TemporalFiltering attribute*), 26
fmap_roi_cleanup (*clpipe.config.options.FMRIPrepOptions attribute*), 20
fmriprep (*clpipe.config.options.ProjectOptions attribute*), 6
fmriprep_memory_usage (*clpipe.config.options.FMRIPrepOptions attribute*), 20
fmriprep_path (*clpipe.config.options.FMRIPrepOptions attribute*), 19
fmriprep_time_usage (*clpipe.config.options.FMRIPrepOptions attribute*), 20
FMRIPrepOptions (*class in clpipe.config.options*), 19
freesurfer_license_path (*clpipe.config.options.FMRIPrepOptions attribute*), 19
from_beginning (*clpipe.config.options.TrimTimepoints attribute*), 31
from_end (*clpipe.config.options.TrimTimepoints attribute*), 31
fwhm (*clpipe.config.options.SpatialSmoothing attribute*), 29

G

get_logs_dir() (*clpipe.config.options.ProjectOptions*

method), 6
glm_l1_launch command line option
-d, 42
-debug, 42
-g, 42
-glm_config_file, 42
-l1_name, 42
-s, 42
-submit, 42
-test_one, 42
glm_l1_preparefsf command line option
-d, 42
-debug, 42
-g, 42
-glm_config_file, 42
-l1_name, 42
glm_l2_launch command line option
-d, 43
-debug, 43
-g, 43
-glm_config_file, 43
-l2_name, 43
-s, 43
-submit, 43
-test_one, 43
glm_l2_preparefsf command line option
-d, 43
-debug, 43
-g, 43
-glm_config_file, 43
-l2_name, 43

I

implementation (*clpipe.config.options.ConfoundRegression attribute*), 29
implementation (*clpipe.config.options.IntensityNormalization attribute*), 27
implementation (*clpipe.config.options.SpatialSmoothing attribute*), 28
implementation (*clpipe.config.options.TemporalFiltering attribute*), 26
include (*clpipe.config.options.MotionOutliers attribute*), 32
insert_na (*clpipe.config.options.ScrubTimepoints attribute*), 30
IntensityNormalization (*class in clpipe.config.options*), 27

L

log_directory (*clpipe.config.options.FMRIPrepOptions attribute*), 20

M

memory_usage (*clpipe.config.options.BatchOptions attribute*), 33
motion_outliers (*clpipe.config.options.ConfoundOptions attribute*), 32
MotionOutliers (*class in clpipe.config.options*), 32

N

n_threads (*clpipe.config.options.BatchOptions attribute*), 33
n_threads (*clpipe.config.options.FMRIPrepOptions attribute*), 20

O

output_directory (*clpipe.config.options.FMRIPrepOptions attribute*), 19
output_directory (*clpipe.config.options.ROIExtractOptions attribute*), 45
overlap_ok (*clpipe.config.options.ROIExtractOptions attribute*), 45

P

populate_project_paths() (*clpipe.config.options.ProjectOptions method*), 6
postprocessing (*clpipe.config.options.ProjectOptions attribute*), 6
processing_streams (*clpipe.config.options.ProjectOptions attribute*), 6
project_directory (*clpipe.config.options.ProjectOptions attribute*), 6
project_title (*clpipe.config.options.ProjectOptions attribute*), 6
ProjectOptions (*class in clpipe.config.options*), 5
prop_voxels (*clpipe.config.options.ROIExtractOptions attribute*), 45

R

reference_image (*clpipe.config.options.Resample attribute*), 31
require_mask (*clpipe.config.options.ROIExtractOptions attribute*), 45
Resample (*class in clpipe.config.options*), 31
ROIExtractOptions (*class in clpipe.config.options*), 45

S

scrub_ahead (*clpipe.config.options.MotionOutliers attribute*), 32
scrub_ahead (*clpipe.config.options.ScrubColumn attribute*), 30
scrub_behind (*clpipe.config.options.MotionOutliers attribute*), 32
scrub_behind (*clpipe.config.options.ScrubColumn attribute*), 30

scrub_columns (*clpipe.config.options.ScrubTimepoints attribute*), 30
scrub_contiguous (*clpipe.config.options.MotionOutliers attribute*), 32
scrub_contiguous (*clpipe.config.options.ScrubColumn attribute*), 30
scrub_var (*clpipe.config.options.MotionOutliers attribute*), 32
ScrubColumn (*class in clpipe.config.options*), 30
ScrubTimepoints (*class in clpipe.config.options*), 30
source (*clpipe.config.options.ProjectOptions attribute*), 6
source_url (*clpipe.config.options.SourceOptions attribute*), 50
SourceOptions (*class in clpipe.config.options*), 50
SpatialSmoothing (*class in clpipe.config.options*), 28
SUBJECTS
    clpipe-convert2bids command line option, 15
    clpipe-postprocess command line option, 35
    clpipe-preprocess command line option, 21
    clpipe-roi-extract command line option, 47

T

target_directory (*clpipe.config.options.ROIExtractOptions attribute*), 45
target_suffix (*clpipe.config.options.ROIExtractOptions attribute*), 45
target_variable (*clpipe.config.options.ScrubColumn attribute*), 30
temp_directory (*clpipe.config.options.SourceOptions attribute*), 50
templateflow_path (*clpipe.config.options.FMRIPrepOptions attribute*), 19
templateflow_templates (*clpipe.config.options.FMRIPrepOptions attribute*), 20
templateflow_toggle (*clpipe.config.options.FMRIPrepOptions attribute*), 19
TemporalFiltering (*class in clpipe.config.options*), 26
threshold (*clpipe.config.options.MotionOutliers attribute*), 32
threshold (*clpipe.config.options.ScrubColumn attribute*), 30
time_usage (*clpipe.config.options.BatchOptions attribute*), 33
transform_dict() (*clpipe.config.options.ProjectOptions class method*), 6
TrimTimepoints (*class in clpipe.config.options*), 31

U

use_aroma (*clpipe.config.options.FMRIPrepOptions attribute*), 19

W

working_directory (*clpipe.config.options.FMRIPrepOptions attribute*), 19