clpipe: A MRI Processing Pipeline for HPCs Documentation

Release .1

Cohen Lab at UNC-CH

Nov 15, 2022

DOCUMENTATION

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

Installation of clpipe is fairly simple. If you have priviledges to add python packages to a 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
```

The installation will also install any additional packaged needed.

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, go ahead and construct one.

Once these images are available, clpipe is ready to go.

1.1 A Note for UNC-CH Users

If you are a Longleaf user, and a member of the hng posix group, you already have access to the latest singularity images for both fmriprep and the bids validator, so there is no need to construct your own, unless you want a older version.

1.2 Batch Languages

clpipe was originally designed for use on 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

PROJECT SETUP

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 options:

```
Usage: project_setup [OPTIONS]

Options:
    -project_title TEXT     [required]
    -project_dir PATH       Where the project will be located.  [required]
    -source_data DIRECTORY  Where the raw data (usually DICOMs) are located.
    -symlink_source_data    symlink the source data into project/data_dicoms.
                            Usually safe to do.
    -submit                 Flag to submit commands to the HPC
    -debug                  Flag to enable detailed error messages and
                            traceback.
    --help                  Show this message and exit.
```

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
```

(continues on next page)

(continued from previous page)

```
|   |-- postproc_default
|-- logs
|   |-- betaseries_logs
|   |-- DCM2BIDS_logs
|   |-- postproc_logs
|   |-- ROI_extraction_logs
|-- scripts
```

CHAPTER

THREE

CONFIGURATION FILES

Most command line functions in clpipe can take a configuration file in the '-configFile' option. These configuration files are JSONs that contain all aspects of the preprocessing and postprocessing streams that you want applied to your dataset. To create a configuration file for your dataset use the following command

```
get_config_file -outputFile <adirectory/yourfilename>.json
```

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

```
{
    "ProjectTitle": "A Neuroimaging Project",
    "Authors/Contributors": "",
    "ProjectDirectory": "",
    "EmailAddress": "",
    "DICOMToBIDSOptions": {
            "DICOMDirectory": "",
            "BIDSDirectory": "",
            "ConversionConfig": "",
            "DICOMFormatString": "",
            "TimeUsage": "1:0:0",
            "MemUsage": "5000",
    "CoreUsage": "2",
    "LogDirectory": ""
    },
    "FMRIPrepOptions": {
            "BIDSDirectory": "",
            "WorkingDirectory": "",
            "OutputDirectory": "",
            "FMRIPrepPath": "/proj/hng/singularity_imgs/fmriprep_1.5.3.sif",
            "FreesurferLicensePath": "/proj/hng/singularity_imgs/license.txt",
            "CommandLineOpts": "",
            "TemplateFlowToggle": true,
            "TemplateFlowPath": "",
            "TemplateFlowTemplates": ["MNI152NLin2009cAsym", "MNI152NLin6Asym",
↪"OASIS30ANTs", "MNIPediatricAsym", "MNIInfant"],
            "FMapCleanupROIs": 3,
            "FMRIPrepMemoryUsage": "20000",
            "FMRIPrepTimeUsage": "16:0:0",
            "NThreads": "8",
    "LogDirectory": ""
    },
```

(continues on next page)

(continued from previous page)

```
    "PostProcessingOptions": {
            "TargetDirectory": "",
            "TargetSuffix": "space-MNI152NLin2009cAsym_desc-preproc_bold.nii.gz",
            "OutputDirectory": "",
            "OutputSuffix": "",
            "ConfoundSuffix": "desc-confounds_regressors.tsv",
            "DropCSV": "",
            "Regress": true,
        "Confounds": ["trans_x", "trans_y", "trans_z", "rot_x", "rot_y", "rot_z",
             "csf", "white_matter", "global_signal", "a_comp_cor.*"],
            "ConfoundsQuad": ["trans_x", "trans_y", "trans_z", "rot_x", "rot_y", "rot_z
↪",
     "csf", "white_matter", "global_signal"],
            "ConfoundsDerive": ["trans_x", "trans_y", "trans_z", "rot_x", "rot_y", "rot_
↪z",
     "csf", "white_matter", "global_signal"],
            "ConfoundsQuadDerive": ["trans_x", "trans_y", "trans_z", "rot_x", "rot_y",
↪"rot_z",
     "csf", "white_matter", "global_signal"],
            "FilteringHighPass": 0.008,
            "FilteringLowPass": -1,
            "FilteringOrder": 2,
            "OversamplingFreq": 4,
            "PercentFreqSample": 1,
            "Scrubbing": true,
            "ScrubVar": "framewise_displacement",
            "ScrubFDThreshold": 0.3,
            "ScrubAhead": 1,
            "ScrubBehind": 1,
            "ScrubContig": 2,
            "RespNotchFilter": true,
            "MotionVars": ["trans_x", "trans_y", "trans_z", "rot_x", "rot_y", "rot_z"],
            "RespNotchFilterBand":[0.31,0.43],
            "PostProcessingMemoryUsage": "20000",
            "PostProcessingTimeUsage": "8:0:0",
            "NThreads": "1",
            "SpectralInterpolationBinSize": 5000,
            "BIDSValidatorImage": "/proj/hng/singularity_imgs/validator.simg",
    "LogDirectory": ""
    },
    "BetaSeriesOptions": {
            "TargetDirectory": "",
            "TargetSuffix": "preproc_bold.nii.gz",
            "OutputDirectory": "",
            "OutputSuffix": "betaseries.nii.gz",
            "ConfoundSuffix": "desc-confounds_regressors.tsv",
            "Regress": true,
            "NuisanceRegression": "QuadLagged",
            "WhiteMatter": true,
            "CSF": true,
            "GlobalSignalRegression": true,
            "FilteringHighPass": 0.008,
```

(continues on next page)

(continued from previous page)

```
            "FilteringLowPass": -1,
            "FilteringOrder": 2,
            "TaskSpecificOptions": [
                    {
                            "Task": "",
                            "ExcludeColumnInfo": "trial_type",
                            "ExcludeTrialTypes": ["block"]
                    }
            ],
  "LogDirectory": ""
},
    "SUSANOptions": {
            "TargetDirectory": "",
            "TargetSuffix": "preproc_bold.nii.gz",
            "OutputDirectory": "",
            "OutputSuffix": "preproc_susan.nii.gz",
            "BrightnessThreshold": 500,
            "FWHM": 0,
            "MemoryUsage": "5000",
            "TimeUsage": "2:0:0",
            "NThreads": "4",
    "LogDirectory": ""
    },
    "ProcessingStreams": [
            {
                    "ProcessingStream": "noGSR",
                    "PostProcessingOptions": {
                            "GlobalSignalRegression": false,
                            "OutputDirectory": "",
                            "OutputSuffix": ""
                    },
                    "BetaSeriesOptions": {
                            "GlobalSignalRegression": false,
                            "OutputDirectory": "",
                            "OutputSuffix": ""
                    },
                    "SUSANOptions": {
                            "OutputSuffix": "preproc_susan250.nii.gz",
                            "BrightnessThreshold": 250
                    }
            },
            {
                    "ProcessingStream": "noScrub",
                    "PostProcessingOptions": {
                            "Scrubbing": false,
                            "OutputDirectory": "",
                            "OutputSuffix": ""
                    },
                    "BetaSeriesOptions":{

                    },
                    "SUSANOptions": {
```

(continues on next page)

(continued from previous page)

```
                    }
            }

    ],
    "ROIExtractionOptions": {
            "TargetDirectory": "",
            "TargetSuffix": "",
            "OutputDirectory": "",
            "Atlases": ["power"],
            "RequireMask": true,
            "PropVoxels": 0.5,
    "MemoryUsage":"3000",
    "TimeUsage": "2:0:0",
    "NThreads": "1",
    "LogDirectory": ""
    },
    "RunLog": [],
    "BatchConfig": "slurmUNCConfig.json"
```

}

All of these fields are required and 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. We describe the various sections of the config now.

3.1 Header

- `ProjectTitle:` The title of your project. Not used in processing.
- `Authors/Contributors` Members of the project team. Not used in processing.
- `ProjectDirectory` Where the project is. Not used in processing.

3.2 FMRIPrep Options

- `FMRIPrepOptions:` Options regarding fMRIprep.
  - `BIDSDirectory:` Your BIDs formatted raw data directory. Use absolute paths if possible.
  - `OutputDirectory:` Where you want your preprocessed files to go. Use absolute paths
  - `WorkingDirectory` Where you want your working files to go. Use absolute paths. For Longleaf users, use /pine/scr/<o>/<n>/<onyen>, where <onyen> is your onyen, and <o> <n> are the first and second letters of your onyen respectively.
  - `FMRIPrepPath:` Where the fMRIprep Singularity image is.
  - `FreesurferLicensePath:` Where your Freesurfer license .txt file is.
  - `TemplateFlowToggle:` This flag activates the use of templateflow, which is used in later versions of FMRIPREP,
  - `TemplateFlowPath`: Where the templateflow template folder is located,

- `TemplateFlowTemplates`: Which templates (standard spaces) should clpipe download for use in templateflow?
- `FMapCleanupROIs`: How many timepoints should the fmap_cleanup function extract from blip-up/blip-down field maps, set to -1 to disable.
- `CommandLineOpts:` Additional arguments to pass to FMRIprep
- `FMRIPrepMemoryUsage:` How much memory in RAM each subject's preprocessing will use, in Mbs. Default is 20000Mb or 20Gb.
- `FMRIPrepTimeUsage:` How much time on the cluster FMRIPrep is allowed to use. Defaults to 16 hours.
- `LogDirectory:` Where cluster output files are stored.

3.3 Postprocessing Options

These are the processing options for function connectivity postprocessing only. Beta Series or GLM are separate option blocks. Note: These are the master options, and changes in `` `ProcessingStreams` `` are changes from the master options.

- `PostProcessingOptions:` Options for various postprocessing steps.
  - `TargetDirectory:` What directory holds your fMRIPrep preprocessed data.
  - `TargetSuffix:` What suffix do your preprocessed fMRI NiFTi files have? Default is preproc_bold.nii.gz.
  - `OutputDirectory:` Where you want your postprocessed files to go.
  - `OutputSuffix:` What suffix do you want appended to your postprocessed files? Make sure to end it with .nii.gz.
  - `ConfoundSuffix:` What suffix does the confound regressors file have. Default is confound_regressor.txt.
  - `Regress:` True/False. Do you want to perform nuisance regression on the data. Default True. For more info see Postprocessing/Nuisance Regression.
  - `RegressionParameters:` These are the headers for the various regression parameters in the fmriprep confound file. The defaults are for the latest fmriprep version. Change only if you are using a much earlier version of fmriprep.
  - `NuisanceRegression:` What type of nuisance regression do you want to perform. Default to QuadLagged (33 Parameter Regression). For more information see Postprocessing/Nuisance Regression.
  - `WhiteMatter:` True/False. Include mean whitematter signal into nuisance regression. Defaults to True.
  - `CSF:` True/False. Include mean cerebral spinal fluid signal into nuisance regression. Defaults to True.
  - `GlobalSignalRegression:` True/False. Include global signal into nuisance regression. Defaults to True.
  - `FilteringHighPass:` High pass frequency for filtering. Defaults to .08 Hz. For more information on filtering see Postprocessing/Frequency Filtering. Set to -1 to remove high pass filtering.
  - `FilteringLowPass:` Low pass frequency for filtering. Defaults to no filter (-1). For more information on filtering see Postprocessing/Frequency Filtering. Set to -1 to remove low pass filtering.
  - `FilteringOrder:` Order of filter. Defaults to 2. For more information on filtering see Postprocessing/Frequency Filtering.
  - `OversamplingFreq`: The oversampling frequency for the spectral interpolation. Defaults to 4. For more information on spectral interpolation see Postprocessing/Spectral Interpolation.

- `PercentFrequencySample:` Proportion (0 to 1, 1 being 100%) of spectrum to use in spectral interpolation. Defaults to 1. For more information on spectral interpolation see Postprocessing/Spectral Interpolation.
- `Scrubbing:` True/False. Use scrubbing. Defaults to true. For more information on scrubbing see Postprocessing/Scrubbing.
- `ScrubFDThreshold:` At what framewise displacement to scrub. Defaults to .3.
- `ScrubAhead:` If a timepoint is scrubbed, how many points after to remove. Defaults to 2.
- `ScrubBehind:` If a timepoint is scrubbed, how many points before to remove. Defaults to 2.
- `ScrubContig:` How many good contiguous timepoints need to exist. Defaults to 4.
- `PostProcessingMemoryUsage:` How much memory (RAM) per subject to request, in Mbs. Defaults to 20000Mb or 20Gb.
- `PostProcessingMemoryUsage:` How much time per subject to request. Format is Hours:Mins:Seconds. Defaults to 8 hours.
- `NThreads:` How many CPUs to request. Defaults to 1. Do not modify lightly.
- `SpectralInterpolationBinSize:` How many voxels per bin to work on in spectral interpolation. Increasing this reduces time but increases memory usage. Defaults to 5000.
- `BIDSValidatorImage:` Where the BIDS validator Singularity image is.
- `LogDirectory:` Where cluster output files are stored.

3.4 Beta Series Options

These options are for the beta series calculations. This is a complex method, please see DOCUMENTATION NOT COMPLETE, for implementation details.

- `BetaSeriesOptions` Options for various postprocessing steps.
  - `TargetDirectory:` What directory holds your fMRIPrep preprocessed data.
  - `TargetSuffix:` What suffix do your preprocessed fMRI NiFTi files have? Default is preproc_bold.nii.gz.
  - `OutputDirectory:` Where you want your postprocessed files to go.
  - `OutputSuffix:` What suffix do you want appended to your postprocessed files? Make sure to end it with .nii.gz.
  - `ConfoundSuffix:` What suffix does the confound regressors file have. Default is confound_regressor.txt.
  - `Regress:` True/False. Do you want to perform nuisance regression on the data. Default True. For more info see Postprocessing/Nuisance Regression.
  - `RegressionParameters:` These are the headers for the various regression parameters in the fmriprep confound file. The defaults are for the latest fmriprep version. Change only if you are using a much earlier version of fmriprep.
  - `NuisanceRegression:` What type of nuisance regression do you want to perform. Default to QuadLagged (33 Parameter Regression). For more information see Postprocessing/Nuisance Regression.
  - `WhiteMatter:` True/False. Include mean whitematter signal into nuisance regression. Defaults to True.
  - `CSF:` True/False. Include mean cerebral spinal fluid signal into nuisance regression. Defaults to True.
  - `GlobalSignalRegression:` True/False. Include global signal into nuisance regression. Defaults to True.

- `FilteringHighPass:` High pass frequency for filtering. Defaults to .08 Hz. For more information on filtering see Postprocessing/Frequency Filtering. Set to -1 to remove high pass filtering.
- `FilteringLowPass:` Low pass frequency for filtering. Defaults to no filter (-1). For more information on filtering see Postprocessing/Frequency Filtering. Set to -1 to remove low pass filtering.
- `FilteringOrder:` Order of filter. Defaults to 2. For more information on filtering see Postprocessing/Frequency Filtering.
- **`TaskSpecificOptions:` A list of option blocks, one for each task you are interested in using beta series with.**
    - `Task:` Task name, must match BIDS task- signifier.
    - `ExcludeColumnInfo` The name of the column in the BIDS formatted events files that contain the information about the trials needed to be excluded from the beta series analysis. (for example, if you have events nested within blocks, then you would want to exclude the block "events")
    - `ExcludeTrialType:` A list of trial types to exclude.
- `LogDirectory:` Where cluster output files are stored.

3.5 SUSAN Smoothing

- **`SUSANOptions` Options for FSL's SUSAN smoothing procedure**
    - `BrightnessThreshold`: The voxel intensity threshold used to distinguish where to smooth. It should be above background level, but below the contrast between edges.
    - `FWHM`: The size of the smoothing kernel. Specifically the full width half max of the Gaussian kernel. Scaled in millimeters. 0 uses a 3x3x3 voxel smoother.

3.6 Processing Streams

- `ProcessingStreams:` A list of processing streams, consisting of the following:

    ***ProcessingStream:: The name of the processing stream** *`PostProcessingOptions::` A list of options to overwrite. *`BetaSeriesOptions::` A list of options to overwrite.

These options are for specific processing streams, and allow the user to overwrite the defaults.

3.7 ROI Extraction Options

***`ROIExtractionOptions:` Options for ROI extraction**

- `TargetDirectory:` What directory holds your fMRIPrep preprocessed data.
- `TargetSuffix:` What suffix do your preprocessed fMRI NiFTi files have? Default is preproc_bold.nii.gz.
- `OutputDirectory:` Where you want your postprocessed files to go.
- `Atlases`: A list of atlas names. Please refer to the ROI extraction documentation for a full list of included atlases.

3.8 Other Options

- `RunLog:` This list contains a record of how a given configuration file is used.
- `BatchConfig:` What batch configuration file to use. For more information see For Advanced Users/Batch Configuration.

CHAPTER

FOUR

DICOM TO BIDS CONVERSION

clpipe has several commands to facilitate the conversion of DICOM files into NiFTI files all in BIDS format. We use dcm2bids, which is a flexible system for converting DICOM files. It does require the creation of a configuration file, a JSON that directs the conversion. Below we have an example of what this might look like.

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
    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.

Finally, instead of using the command line option, this DICOM format string can be specified in the clpipe configuration JSON.

The other important ingredient in converting your DICOMs to BIDS format is the conversion configuration. An example file is generated when you use the project_setup command. Below is a brief example:

```
{
    "descriptions": [
            {
                    "dataType": "anat",
                    "modalityLabel": "T1w",
                    "customLabels": "",
                    "criteria": {
                            "SeriesDescription": "T1w_MPRAGE",
                            "ImageType": ["ORIGINAL", "PRIMARY", "M", "ND", "NORM"]
                    }
            },
            {
                    "dataType": "anat",
                    "modalityLabel": "T2w",
                    "customLabels": "",
                    "criteria": {
                            "SeriesDescription": "T2_AX_struct"
                    }
            },
            {
                    "dataType": "func",
                    "modalityLabel": "bold",
                    "customLabels": "task-rest",
        "sidecarChanges": {
            "TaskName": "Resting State"
        },
                    "criteria": {
                            "SeriesDescription": "*_resting_AP_*"
                    }
            },
            {
                    "dataType": "func",
                    "modalityLabel": "bold",
                    "customLabels": "task-gngreg",
                    "criteria": {
                            "SeriesDescription": "*_GNGregular*"
                    }
            },
            {
                    "dataType": "func",
                    "modalityLabel": "bold",
                    "customLabels": "task-gngrew",
                    "criteria": {
                            "SeriesDescription": "*_GNGreward*"
                    }
            },
            {
                    "dataType": "dwi",
                    "modalityLabel": "dwi",
                    "customLabels": "acq-APref",
                    "criteria": {
                            "SeriesDescription": "*p2_AP_TRACEW*"
                    }
```

(continues on next page)

(continued from previous page)

```
        },
        {
                "dataType": "dwi",
                "modalityLabel": "dwi",
                "customLabels": "acq-PAref",
                "criteria": {
                        "SeriesDescription": "*p2_PA"
                }
        },
        {
                "dataType": "dwi",
                "modalityLabel": "dwi",
                "customLabels": "acq-AP",
                "criteria": {
                        "SeriesDescription": "*p2_AP"
                }
        },
        {
                "dataType": "fmap",
                "modalityLabel": "epi",
                "criteria":{
                        "SeriesDescription": "*_resting_PA*"
        },
                "intendedFor": [2,3]
        }

        ]
}
```

This configuration file looks for all scans that have "_srt" anywhere in the SeriesDescription field of the header, converts them into NIFTI, labels them in the BIDS standards, and adds the custom label of *task-srt*. It does the same for anatomical scans with "MPRAGE" contained in the series description. Any header field in the dicoms can be used as criteria. If multiple scans meet the criteria, then they will be labeled *run-1, run-2, ...* in order of acquisition.

Note that for fieldmaps, one can use the "intendedFor" option to specify which BOLD images a fieldmap should be used for. There are two important points here. The first is that the "intendedFor" field is 0-indexed, in that 0 corresponds to the first entry in the converstion config, 1 corresponds to the second entry, etc, etc. In the example above, the fieldmap is intended for the resting state scan and the GNG regular scan. Additionally, the intended for field is not sensitive to multiple runs. For example, if there are 2 resting state scans, and therefore the file names look like "sub-9999_task-rest_run-01_bold.nii.gz" and "sub-9999_task-rest_run-02_bold.nii.gz" after conversion, the IntendedFor field in the fieldmap's JSON will list "sub-9999_task-rest_bold.nii.gz" This is due to an issue with the dcm2bids package, and will result in the fieldmaps not being used. The workaround is to list each run explicitly in your conversion configuration, or to modify each fieldmap JSON after it is generated.

Finally, there are several varieties of fieldmaps allowable in the BIDS format, each needing a different set of conversion config entries. For a detailed look at these types, please see the BIDS Specification.

4.1 Conversion Commands

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, and 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.

Once you have updated your conversion configuration file, you can convert your entire dataset with:

```
convert2bids [OPTIONS]

    Options:
        -config_file PATH       The configuration file for the study, use if you
                                have a custom batch configuration.
    -conv_config_file PATH      The configuration file for the study, use if you
                                have a custom batch configuration.
    -dicom_dir TEXT             The folder where subject dicoms are located.
    -dicom_dir_format TEXT      Format string for how subjects/sessions are
                                organized within the dicom_dir.
    -BIDS_dir TEXT              The dicom info output file name.
    -overwrite                  Overwrite existing BIDS data?
    -log_dir TEXT               Where to put the log files. Defaults to Batch_Output
                                in the current working directory.
    -subject TEXT               A subject  to convert using the supplied configuration
                                file.  Use to convert single subjects, else leave empty.
    -session TEXT               A session  to convert using the supplied configuration
                                file.  Use in combination with -subject to convert single
                                subject/sessions, else leave empty.
    --help                      Show this message and exit.
```

CHAPTER

FIVE

BIDS VALIDATION

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

```
Usage: bids_validate [OPTIONS] [BIDS_DIR]

  Runs the BIDS-Validator program on a dataset. If a configuration file has
  a BIDSDirectory specified, you do not need to provide a BIDS directory in
  the command.

Options:
  -config_file FILE       Uses a given configuration file
  -verbose               Creates verbose validator output. Use if you want to
                         see ALL files with errors/warnings.
  -submit                Submit command to HPC.
  -interactive           Run in an interactive session. Only use in an
                         interactive compute session.
  -debug                 Produce detailed debug and traceback.
  --help                 Show this message and exit.
```

`bids_validate` produces an output file titled *Output-BIDSValidator.out* at your current working directory that contains the output of the BIDS validator.

CHAPTER

SIX

PREPROCESSING WITH FMRIPREP

clpipe uses fMRIprep to perform minimal preprocessing on functional MRI data. To submit your dataset for preprocessing, use the following command:

```
Usage: fmriprep_process [OPTIONS] [SUBJECTS]...

  This command runs a BIDS formatted dataset through fMRIprep. Specify
  subject IDs to run specific subjects. If left blank, runs all subjects.

Options:
  -config_file FILE          Use a given configuration file. If left blank, uses
                             the default config file, requiring definition of
                             BIDS, working and output directories.
  -bids_dir DIRECTORY        Which BIDS directory to process. If a configuration
                             file is provided with a BIDS directory, this
                             argument is not necessary.
  -working_dir DIRECTORY      Where to generate the working directory. If a
                             configuration file is provided with a working
                             directory, this argument is not necessary.
  -output_dir DIRECTORY       Where to put the preprocessed data. If a
                             configuration file is provided with a output
                             directory, this argument is not necessary.
  -log_output_dir DIRECTORY  Where to put HPC output files (such as SLURM output
                             files). If not specified, defaults to
                             <outputDir>/batchOutput.
  -submit                    Flag to submit commands to the HPC
  -debug                     Flag to enable detailed error messages and
                             traceback
  --help                     Show this message and exit.
```

`fmriprep_process` 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.

6.1 Getting 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.

```
Usage: get_reports [OPTIONS]

  This command creates a zip archive of fmriprep QC reports for download.

Options:
  -config_file FILE    The configuration file for the current data processing
                       setup.  [required]
  -output_name TEXT    Path and name of the output archive. Defaults to current
                       working directory and "Report_Archive.zip"
  -debug               Print traceback on errors.
  --help               Show this message and exit.
```

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.

CHAPTER

SEVEN

POSTPROCESSING FMRI

When performing functional connectivity analysis, there are several additional processing steps that need to be taken after the minimal preprocessing of fMRIPrep. clpipe implements these steps in Python, and a fMRIprep preprocessed dataset can be postprocessed using the following command:

```
 Usage: fmri_postprocess [OPTIONS] [SUBJECTS]...

This command runs an fMRIprep'ed dataset through additional processing, as
defined in the configuration file. To run specific subjects, specify their
IDs. If no IDs are specified, all subjects are ran.

  Options:
  -config_file PATH        Use a given configuration file. If left blank, uses
                           the default config file, requiring definition of
                           BIDS, working and output directories.
  -target_dir DIRECTORY    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.
  -target_suffix TEXT      Which file suffix to use. If a configuration file
                           is provided with a target suffix, this argument is
                           not necessary. Defaults to "preproc_bold.nii.gz"
  -output_dir DIRECTORY    Where to put the postprocessed data. If a
                           configuration file is provided with a output
                           directory, this argument is not necessary.
  -output_suffix TEXT      What suffix to append to the postprocessed files.
                           If a configuration file is provided with a output
                           suffix, this argument is not necessary.
  -task TEXT               Which task to postprocess. If left blank, defaults
                           to all tasks.
  -TR TEXT                 The TR of the scans. If a cofig file is not
                           provided, this option is required. If a config file
                           is provided, this information is found from the
                           sidecar jsons.
  -processing_stream TEXT  Optional processing stream selector.
  -log_dir DIRECTORY       Where to put HPC output files. If not specified,
                           defaults to <outputDir>/batchOutput.
  -beta_series             Flag to activate beta-series correlation
                           correlation. ADVANCED METHOD, refer to the
                           documentation.
```

(continues on next page)

(continued from previous page)

```
  -submit                  Flag to submit commands to the HPC.
  -batch / -single         Submit to batch, or run in current session. Mainly
                           used internally.
  -debug                   Print detailed processing information and traceback
                           for errors.
  --help                   Show this message and exit.
```

7.1 Processing Checker

clpipe has a convenient function for determining which scans successfully made it through both preprocessing using fMRIprep and postprocessing.

```
Usage: fmri_process_check [OPTIONS]

  This command checks a BIDS dataset, an fMRIprep'ed dataset and a
  postprocessed dataset, and creates a CSV file that lists all scans across
  all three datasets. Use to find which subjects/scans failed processing.

Options:
  -config_file FILE  The configuration file for the current data processing
                     setup.  [required]
  -output_file TEXT  Path and name of the output CSV. Defaults to config
                     file directory and "Checker-Output.csv"
  -debug             Print traceback and detailed processing messages.
  --help             Show this message and exit.
```

This command will create a csv file listing all scans found in the BIDS dataset, and corresponding scans in the fMRIprep dataset and the postprocessed dataset.

For a description of the various postprocessing steps, along with references, please see the following documentation:

1. Nuisance Regression
2. Frequency Filtering
3. Scrubbing
4. Spectral Interpolation

7.2 fmri_postprocess2

New to clpipe v1.5, the command fmri_postprocess2 combines the functionality of fmri_postprocess and glm_setup into a unified postprocessing stream.

```
Usage: fmri_postprocess2 [OPTIONS] [SUBJECTS]...

Options:
  -config_file FILE        Use a given configuration file.  [required]
  -fmriprep_dir DIRECTORY  Which fmriprep directory to process.  If a
                           configuration file is provided with a BIDS
                           directory, this argument is not necessary.  Note,
```

(continues on next page)

(continued from previous page)

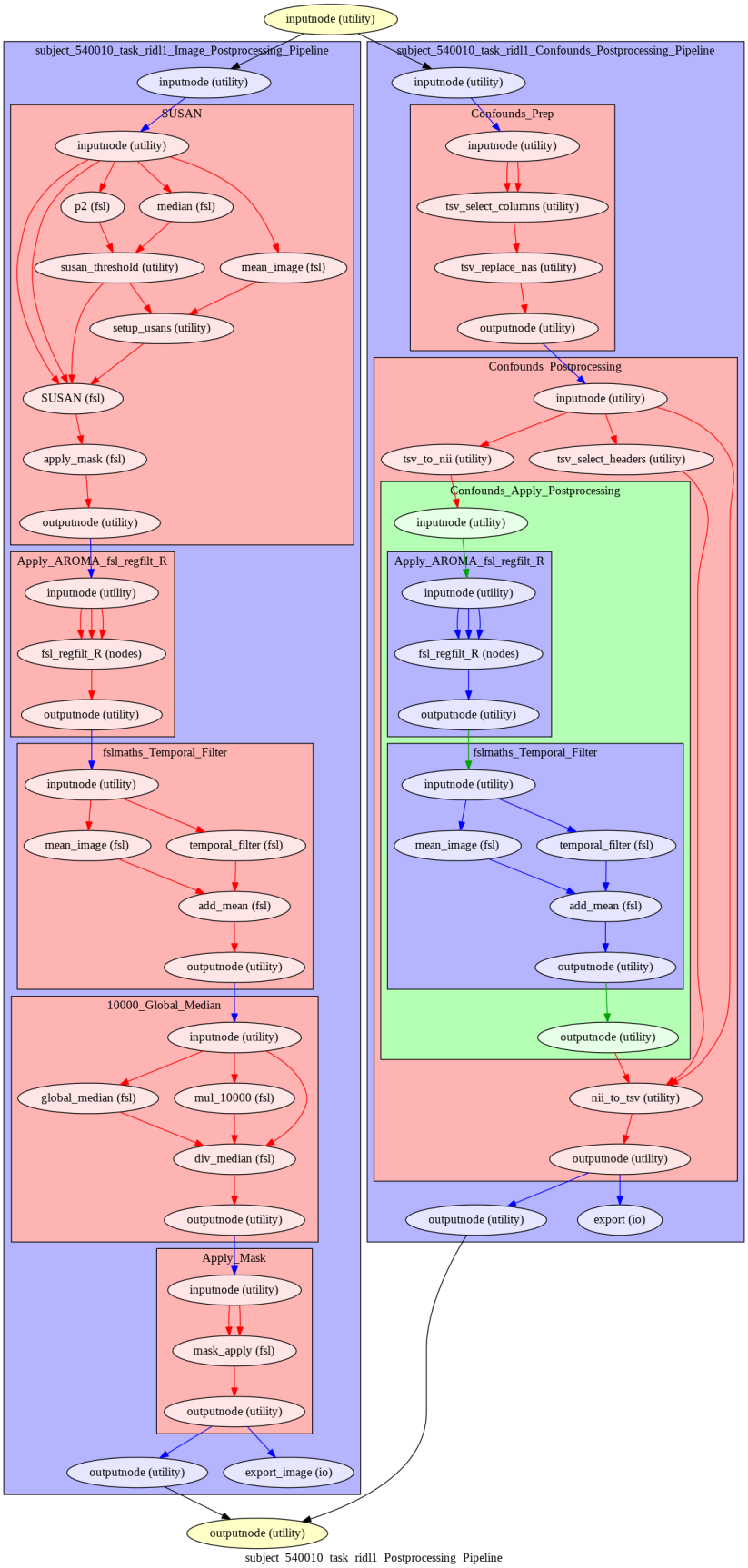
```
                        must point to the ``fmriprep`` directory, not its
                        parent directory.
  -output_dir DIRECTORY    Where to put the postprocessed data.  If a
                        configuration file is provided with a output
                        directory, this argument is not necessary.
  -processing_stream TEXT  Specify a processing stream to use defined in your
                        configuration file.
  -log_dir DIRECTORY       Path to the logging directory.
  -index_dir DIRECTORY     Give the path to an existing pybids index database.
  -refresh_index           Refresh the pybids index database to reflect new
                        fmriprep artifacts.
  -batch / -no-batch       Flag to create batch jobs without prompt.
  -submit                  Flag to submit commands to the HPC without prompt.
  -debug                   Print detailed processing information and traceback
                        for errors.
  --help                   Show this message and exit.
```

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. fmri_postprocess2 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.

Available processing steps:

- Temporal Filtering
- Intensity Normalization
- Spatial Smoothing
- AROMA Regression
- Confound Regression
- Apply Mask
- Resample
- Trim Timepoints


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

7.2.1 Configuration Setup

This command requires a new configuration block - if you using an existing clpipe project, you will have to insert this json into your configuration file. Otherwise, this block will be included when running "project setup."

```
"PostProcessingOptions2": {
        "WorkingDirectory": "",
        "WriteProcessGraph": true,
        "TargetImageSpace": "MNI152NLin2009cAsym",
        "TargetTasks": [],
        "TargetAcquisitions": [],
        "ProcessingSteps": [
                "SpatialSmoothing",
                "TemporalFiltering",
                "IntensityNormalization",
                "ApplyMask"
        ],
        "ProcessingStepOptions": {
                "TemporalFiltering": {
                        "Implementation":"fslmaths",
                        "FilteringHighPass": 0.008,
                        "FilteringLowPass": -1,
                        "FilteringOrder": 2
                },
                "IntensityNormalization": {
                        "Implementation": "10000_GlobalMedian"
                },
                "SpatialSmoothing": {
                        "Implementation": "SUSAN",
                        "FWHM": 6
                },
                "AROMARegression":{
                        "Implementation": "fsl_regfilt_R"
                },
                "Resample":{
                        "ReferenceImage": "SET REFERENCE IMAGE"
                },
                "TrimTimepoints": {
                        "FromEnd": 0,
                        "FromBeginning": 0
                },
                "ConfoundRegression": {
                        "Implementation": "afni_3dTproject"
                }
        },
        "ConfoundOptions": {
                "Columns": [
                        "csf", "csf_derivative1", "white_matter", "white_matter_
↪derivative1"
                ],
                "MotionOutliers": {
                        "Include": true,
                        "ScrubVar": "framewise_displacement",
```

(continues on next page)

(continued from previous page)

```
                        "Threshold": 0.9,
                        "ScrubAhead": 0,
                        "ScrubBehind": 0,
                        "ScrubContiguous": 0
                }
        },
        "BatchOptions": {
                "MemoryUsage": "20000",
                "TimeUsage": "2:0:0",
                "NThreads": "1"
}
}
```

- `PostProcessingOptions:` Options for configuring post-fmriprep processing steps.
  - `WorkingDirectory:` Directory for caching intermediary processing files.
  - `WriteProcessGraph:` Set 'true' to write a processing graph alongside your output.
  - `TargetImageSpace:` Which space to use from your fmriprep output. This is the value that follows "space-" in the image file names.
  - `TargetTasks:` Which tasks to use from your fmriprep output. This is the value that follows "task-" in the image file names. Leave blank to target all tasks.
  - `TargetAcquisitions:` Which acquisitions to use from your fmriprep output. This is the value that follows "acq-" in the image file names. Leave blank to target all acquisitions.
  - `ProcessingSteps:` The default list of processing steps to use. Processing will follow the order of this list.
  - `ProcessingStepOptions:` The default processing options for each step.
    - `TemporalFiltering:` Apply temporal filtering to the image data. Also be applied to confounds.
      - `Implementation:` Currently limited to "fslmaths"
      - `FilteringHighPass:` High pass frequency for filtering. Defaults to .08 Hz. Set to -1 to remove high pass filtering.
      - `FilteringLowPass:` Low pass frequency for filtering. Defaults to no filter (-1). Set to -1 to remove low pass filtering.
      - `FilteringOrder:` Order of filter. Defaults to 2.
    - `IntensityNormalization:` Apply intensity normalization to the image data.
      - `Implementation:` Currently limited to "10000_GlobalMedian"
    - `SpatialSmoothing:` Apply spatial smoothing to the image data.
      - `Implementation:` Currently limited to "SUSAN"
      - `FWHM:` The size of the smoothing kernel. Specifically the full width half max of the Gaussian kernel. Scaled in millimeters.
    - `AROMARegression:` Regress out AROMA artifacts from the image data. Also be applied to confounds.
      - `Implementation:` Currently limited to "fsl_regfilt_R"
    - `Resample:` Resample the image into a new space.

    * `TrimTimepoints:` Trim timepoints from the beginning or end of an image. Also be applied to confounds.
        - `FromEnd:` Number of timepoints to trim from the end of each image.
        - `FromBeginning:` Number of timepoints to trim from the beginning of each image.
    * `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:` Currently limited to "afni_3dTproject"
- `ConfoundOptions:` The default options to apply to the confounds files.
    * `Columns:` A list containing a subset of confound file columns to use from each image's confound file.
    * `MotionOutliers:` Options specific to motion outliers.
        - `Include:` Set 'true' to add motion outlier spike regressors to each confound file.
        - `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 .9
        - `ScrubAhead:` How many time points ahead of a flagged time point should be flagged also, defaults to 0.
        - `ScrubBehind:` If a timepoint is scrubbed, how many points before to remove. Defaults to 0.
        - `ScrubContiguous:` How many good contiguous timepoints need to exist. Defaults to 0.
- `BatchOptions:` The batch settings for postprocessing.
    * `MemoryUsage:` How much memory to allocate per job.
    * `TimeUsage:` How much time to allocate per job.
    * `NThreads:` How many threads to allocate per job.

7.2.2 Processing Streams Setup

By default, the output from running fmri_postprocess2 will appear in your clpipe folder at data_postproc2/smooth_filter_normalize, reflecting the defaults from PostProcessingOptions2.

However, you can utilize the power of processing streams to deploy multiple postprocessing streams. Each processing stream you define your config file's ProcessingStreams block will create a new output folder named after the ProcessingStream setting.

Within each processing stream, you can override any of the settings in the main PostProcessingOptions2 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.

```
...
"ProcessingStreams": [
        ...
        {
                "ProcessingStream": "smooth_aroma-regress_filter-butterworth_normalize",
                "PostProcessingOptions": {
                        "TargetTasks": [
                                "rest"
```

(continues on next page)

(continued from previous page)

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

To run a specific stream, give the -processing_stream stream option of fmri_postprocess2 the name of the stream:

```
fmri_postprocess2 -config_file clpipe_config.json -processing_stream smooth_aroma-
↪regress_filter-butterworth_normalize -submit
```

CHAPTER

EIGHT

SUSAN SPATIAL SMOOTHING

clpipe uses FSL's SUSAN smoothing to perform spatial smoothing. This step is usually done after postprocessing. Options for this are configurable on a processing stream basis, see config file for more details.

```
Usage: susan_smoothing [OPTIONS] [SUBJECTS]...

Options:
  -config_file PATH        Use a given configuration file. If left blank, uses
                           the default config file, requiring definition of
                           BIDS, working and output directories.
  -target_dir DIRECTORY    Which directory to process. If a configuration file
                           is provided.
  -target_suffix TEXT      Which file suffix to use. If a configuration file
                           is provided with a target suffix, this argument is
                           not necessary. Defaults to "preproc_bold.nii.gz"
  -output_dir DIRECTORY    Where to put the postprocessed data. If a
                           configuration file is provided with a output
                           directory, this argument is not necessary.
  -output_suffix TEXT      What suffix to append to the smoothed files. If a
                           configuration file is provided with a output
                           suffix, this argument is not necessary.
  -task TEXT               Which task to smooth. If left blank, defaults to
                           all tasks.
  -processing_stream TEXT  Optional processing stream selector.
  -log_dir DIRECTORY       Where to put HPC output files. If not specified,
                           defaults to <outputDir>/batchOutput.
  -submit                  Flag to submit commands to the HPC.
  -batch / -single         Submit to batch, or run in current session. Mainly
                           used internally.
  -debug                   Print detailed processing information and traceback
                           for errors.
  --help                   Show this message and exit.
```

CHAPTER

NINE

ROI EXTRACTION

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

```
Usage: fmri_roi_extraction [OPTIONS] [SUBJECTS]...

Options:
  -config_file PATH          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 DIRECTORY      Which postprocessed directory to process. If a
                             configuration file is provided with a target
                             directory, this argument is not necessary.
  -target_suffix TEXT        Which target suffix to process. If a
                             configuration file is provided with a target
                             suffix, this argument is not necessary.
  -output_dir DIRECTORY      Where to put the ROI extracted data. If a
                             configuration file is provided with a output
                             directory, this argument is not necessary.
  -task TEXT                 Which task to process. If none, then all tasks
                             are processed.
  -atlas_name TEXT           What atlas to use. Please refer to documentation,
                             or use the command get_available_atlases to see
                             which are available. When specified for a custom
                             atlas, this is what the output files will be
                             named.
  -custom_atlas TEXT         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 TEXT         A custom atlas label file. Not needed if
                             specified in config.
  -custom_type TEXT          What type of atlas? (label, maps, or spheres).
                             Not needed if specified in config.
  -radius TEXT               If a sphere atlas, what radius sphere, in mm. Not
                             needed if specified in config.
  -overlap_ok                Are overlapping ROIs allowed?
  -overwrite                 Overwrite existing files?
  -log_output_dir DIRECTORY  Where to put HPC output files (such as SLURM
```

(continues on next page)

(continued from previous page)

```
                        output files). If not specified, defaults to
                        <outputDir>/batchOutput.
  -submit               Flag to submit commands to the HPC
  -single               Flag to directly run command. Used internally.
  -debug                Flag to enable detailed error messages and
                        traceback
  --help                Show this message and exit.
```

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

By default, ROIs are calculated with respect to the brain mask, and ROIs with fewer than the "PropVoxels" option voxels 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.

CHAPTER

TEN

GLM SETUP, L1 MODELS AND L2 MODELS

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. `glm_setup` This function serves to resample your fmriprepped data into a standard resolution and image size, apply a brain mask (calculated by FMRIprep), optionally perform SUSAN smoothing, and drop timepoints from the beginning and ending of the images. Note: The resampling is not registration and this `glm_setup` function requires that you have already preprocessed your data using FMRIprep.
2. `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.
3. `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.
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.

CHAPTER

ELEVEN

GLM CONFIGURATION FILES

In clpipe, a GLM configuration file describes the L1 and L2 setup for a single task. It is structured as follows:

11.1 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.

11.2 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.

11.3 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.

11.4 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``.

11.5 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

11.6 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.

11.7 `glm_setup` Command:

```
Usage: glm_setup [OPTIONS] [SUBJECTS]...

Options:
  -config_file PATH      Use a given configuration file.  [required]
  -glm_config_file PATH  Use a given GLM configuration file.  [required]
  -drop_tps PATH         Drop timepoints csv sheet
  -submit                Flag to submit commands to the HPC.
  -batch / -single       Submit to batch, or run in current session. Mainly
                      used internally.
  -debug                 Print detailed processing information and traceback
                      for errors.
  --help                 Show this message and exit.
```

11.8 `fsl_onset_extract` Command:

```
fsl_onset_extract [OPTIONS]

Options:
  -config_file FILE      Use a given configuration file.  [required]
  -glm_config_file FILE  Use a given GLM configuration file.  [required]
  -debug                 Print detailed processing information and traceback
                         for errors.

  --help                 Show this message and exit.
```

11.9 `glm_l1_preparefsf` Command:

```
Usage: glm_l1_preparefsf [OPTIONS]

Options:
  -glm_config_file PATH  Use a given GLM configuration file.  [required]
  -l1_name TEXT          Name for a given L1 model  [required]
  -debug                 Flag to enable detailed error messages and traceback
  --help                 Show this message and exit.
```

11.10 glm_l2_preparefsf Command:

```
Usage: glm_l2_preparefsf [OPTIONS]

Options:
  -glm_config_file PATH  Use a given GLM configuration file.  [required]
  -l2_name TEXT          Name for a given L1 model  [required]
  -debug                 Flag to enable detailed error messages and traceback
  --help                 Show this message and exit.
```

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
```

Running project setup requires us to have a source data directory ready. Create an empty one for now

```
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 ourself with the most import 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 goal of this processing step is to convert "raw" DICOM format images into BIDS format. If your data is already in BIDS format, move on to the BIDS Validation step.

Due to the manual labeling necessary for DICOM to BIDS conversion, this is one of the most difficult parts of clpipe to setup - do not be discourged by this early 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.

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
    ├── 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
├── 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
```

(continues on next page)

(continued from previous page)

```
        └── 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
    ├── 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!