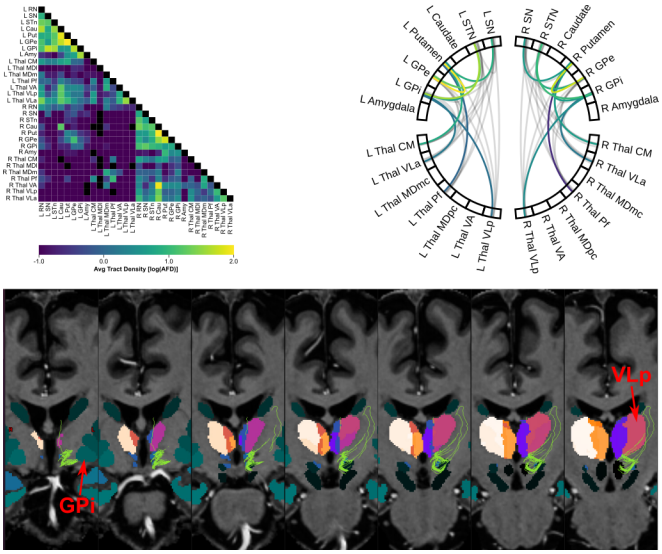
SCATTR

Jason Kai

Sep 25, 2023

GETTING STARTED

SCATTR is a BIDS App that performs a tractography processing workflow to identify connections between targetted structures of interest (extracted from various atlases) in the brain, using some common neuroimaging tools like `prepdwi`, `Freesurfer`, and `Mrtrix3`.



This is useful for:

- Analyzing structural connections of known brain circuits
- Examining specific connections of interest
- Comparison across different groups (e.g. controls vs patients)

CHAPTER

ONE

EXAMPLE USE OF WORKFLOW

This workflow was used to process and analyze the data from `hcp_subcortical_repo` (see Kai et al., 2022).

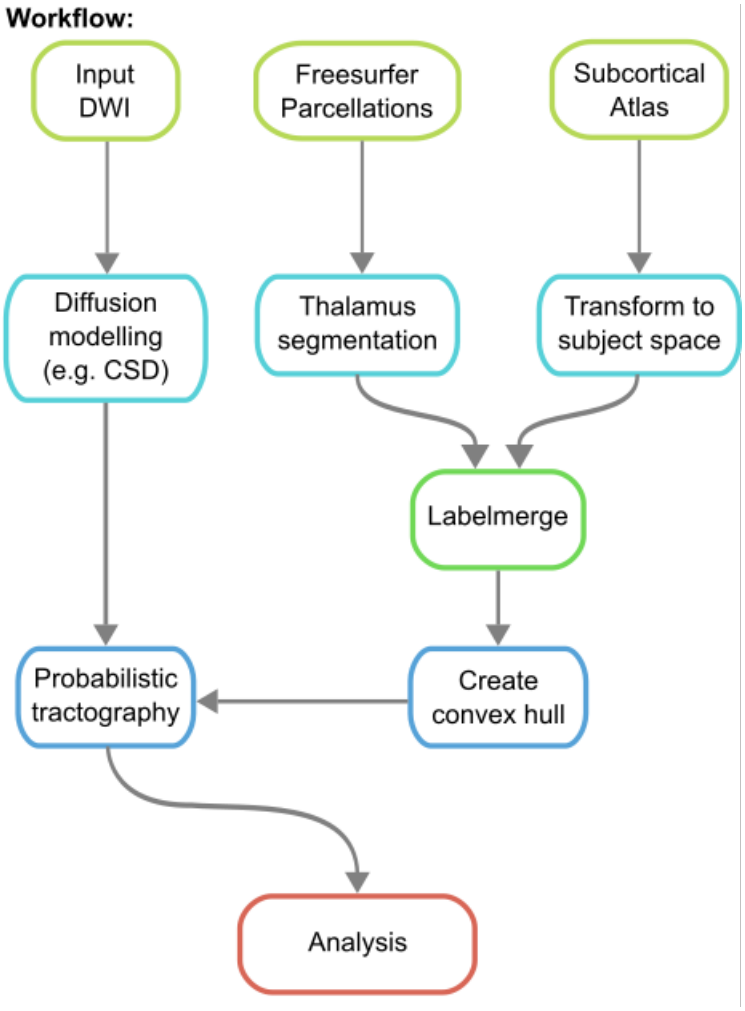
CHAPTER

TWO

WORKFLOW

A brief summary of the workflow can be found below (see documentation for a detailed summary):

Note: The workflow assumes Freesurfer has already been run on the dataset, as well as diffusion preprocessing (e.g. distortion correction).



1. Merge the segmentations of structures in a standard template space from various sources (if necessary) via labelmerge into a combined atlas, which is used downstream to identify targeted connections.
2. Estimate and apply transformations from standard template space to subject-specific space.

3. Further process the pre-processed diffusion data to enable tractography (e.g. compute response functions, fibre orientation distribution, normalization).
4. Perform whole-brain tractography from computed files, applying filtering (e.g. SIFT2) to computed tractogram. Once filtered, connections between the merged segmentations (from step 1) are identified, generating a connectome map.
5. Analysis can then be performed on the map to examine and explore the connectome of interest.

2.1 Full documentation: here

CHAPTER

THREE

RELEVANT PAPERS

- Kai, J., Khan, A.R., Haast, R.A.M., Lau, J.C. (2022). Mapping the subcortical connectome using in vivo diffusion MRI: feasibility and reliability. Terra incognita: diving into the human subcortex, special issue of NeuroImage. doi: 10.1016/j.neuroimage.2022.119553.

3.1 Installation

BIDS App for Structural Connectivity Applied To Targeted Regions (SCATTR)

3.1.1 Requirements

- Docker (Intel Mac/Windows/Linux) or Singularity (Linux)
- For those wishing to contribute or modify the code, `pip install` or `poetry install` are also available (Linux), but will still require Singularity to handle some dependencies. See Contributing to SCATTR.
- *Note: Apple ARM-based chips (e.g. M1, M2, etc.) are **not currently supported**. We do not have a Docker arm64 container yet.*

Notes

- Inputs to SCATTR should be a BIDS dataset, including processed Freesurfer and DWI derivatives (which can be stored separately).

3.1.2 Docker on Windows / Mac (Intel) / Linux

The SCATTR BIDS App is available on DuckerHub as versioned releases. Instructions can be found in the Docker documentation page.

Pros

- Compatible with non-Linux systems
- All dependencies are in a single container

Cons

- Typically not possible on shared machines
- Cannot use Snakemake cluster execution profiles
- Cannot edit code

3.1.3 Singularity Container

The same Docker container can also be used with Singularity (now Apptainer). Instructions can be found in the Singularity / Apptainer documentation page.

Pros

- All dependencies are in a single container, stored as a single file (.sif)
- Compatible on shared systems with Singularity installed

Cons

- Cannot use Snakemake cluster execution profiles
- Cannot edit code

3.1.4 Python Environment with Singularity Dependencies

Instructions can be found in the Contributing documentation page.

Pros

- Flexibility to modify code

Cons

- Only compatible on systems with Singularity for external dependencies

3.2 Running SCATTR with Docker on Windows

Note, these instructions you have Docker installed already on a Windows system. Docker can also be run on Linux or MacOS with similar commands, but here, we will assume the default Windows CLI is being used.

3.2.1 First time setup

Open your Windows Command Prompt by clicking the `Windows` button and typing `cmd` and pressing the `Enter` on your keyboard. This is where you will enter your SCATTR commands. Feel free to make a new directory with `mkdir` or move to a directory you would like to work out of with `cd`. For this example, we will work from:

```
cd c:\Users\username\Downloads
```

Pull the container (this will take some time and storage space, but like an installation, it only needs to be done once and can be then be run on many datasets). The example below pulls the latest versioned container (replace `latest` with `vX.X.X` for a specific version).

```
docker pull khanlab/scattr:latest
```

Run SCATTR without any arguments to print the short help:

```
docker run -it --rm khanlab/scattr:latest
```

Use the `-h` option to get a detailed help listing:

```
docker run -it --rm khanlab_scattr_latest.sif -h
```

_Note that all the Snakemake command-line options are also available in SCATTR, and can be listed with `--help-snakemake`:

```
docker run -it --rm khanlab_scattr_latest.sif --help-snakemake
```

3.2.2 Running an example

We will use the `test` folder found from the Github repository via `git clone` to the previously mentioned folder to demonstrate an example of how to run SCATTR

```
docker run -it --rm -v c:\Users\username\Downloads\scattr\test:\test khanlab_scattr_
↪latest.sif /test/data/bids /test/data/derivatives participant --fs-license /test/fs_
↪license --force-output -n
```

Explanation

Everything prior to the container (`khanlab_scattr_latest.sif`) are arguments to Docker and after are to SCATTR itself. The first three arguments to Docker are to enable interactive mode (`-it`), run and subsequently remove the Docker container upon completion (`--rm`) and mount the the directory (`-v c:\Users\username\Downloads\scattr\test`) to a directory within the container named `\test`. These are not specific to SCATTR, but are general ways to use Docker. You may want to familiarize yourself with Docker options.

The first three arguments to SCATTR (as with any BIDS App) are the input folder (`test/data/bids`), the output folder (`test/data/derivatives`), and the analysis level (`participant`). The `participant` analysis level is used in SCATTR to perform further participant-level processing of Freesurfer (thalamus segmentation), external atlases

(combining segmentations) and diffusion derived data (estimation of fibre orientation distributions). This includes estimating an average response function from input data. The `--fs-license` argument allows for specification of the location of the required Freesurfer license file if not already specified in the `FS_LICENSE` environment variable. Note, that this is required to perform any Freesurfer-related processing. The `--force-output` flag is a Snakemake argument that is invoked to allow for writing of output file to already existing folders - in this case, for thalamus segmentations via Freesurfer. We also used the `--dry-run/-n` option to print out what would run, without actually running the workflow.

When you run the above command, a long listing will print out, describing all the rules that will be run. We can also have a shell command used for each rule printed to screen using the `-p` Snakemake option

```
docker run -it --rm -v c:\Users\username\Downloads\scattr\test:\test  khanlab_scattr_
↪latest.sif /test/data/bids /test/data/derivatives participant --fs-license /test/fs_
↪license --force-output -np
```

Now to actually run the workflow, we need to specify how many cores to use and leave out the dry-run option. The Snakemake `--cores` option tells SCATTR how many cores to use. Using `--cores 8` means that SCATTR will only make use of 8 cores at most. Generally speaking, you should use `--cores all`, so it can make maximal use of all available CPU cores it has access to on your system. This is especially useful if you are running multiple subjects.

Running the follow command (SCATTR on a single subject) may take up to ~36 hours with 8 cores and default parameters, but could be much longer (several days) if you only have a single core.

```
docker run -it --rm -v c:\Users\username\Downloads\scattr\test:\test /test/data/bids /
↪test/data/derivatives participant --fs-license /test/fs_license --force-output -p --
↪cores all
```

After this completes, you have additional folders in your output folder, `c:\Users\username\Downloads\scattr\test\data\derivatives`, for the one subject.

Exploring different options

If you alternatively want to run SCATTR using a pre-defined average response function, you can use the `--responsemean_dir` flag to specify the location to where the average response function is located.

```
docker run -it --rm -v c:\Users\username\Downloads\scattr\test:\test /test/data/bids /
↪test/data/derivatives/ participant --responsemean_dir /test/data/derivatives/mrtrix/
↪avg --fs-license /test/.fs_license -np --force-output
```

Other parameters exist, which may help to improve processing times at the expense of sensitivity / specificity (e.g. reducing the number of streamlines generated).

3.3 Running SCATTR with Singularity

3.3.1 Pre-requisites

1. Singularity / Apptainer is is installed on your system. For more info, see the detailed Apptainer install instructions.
2. The following command-line tools are installed:
   - wget
3. Sufficient disk-space (rough estimate)

- in your `/tmp` folder (>30GB) to build the container
- in your working folder to store the container (~20GB)
- for SCATTR outputs (~40GB per subject using default parameters)

4. Sufficient CPU and memory - the more you have, the faster it will run and the more streamlines that can be estimated. We recommand at least 8 CPU cores and 64GB memory if using default parameters.

3.3.2 First time setup

Pull the container. This can be done from DockerHub, but requires a large amount of disk space in your `/tmp` folder, since it has to convert from a Docker container to a Singularity/Apptainer container. The example below pulls the latest versioned container (replace `latest` with `vX.X.X` for a specific version).

```
singularity pull docker://khanlab/scattr:latest
```

Note: If you encounter any errors pulling the container from DockerHub, it may be because you are running out of disk space in your cache folders. You can change these locations by setting environment variables, however, using a network file system for the folders may result in poor performance:

```
export SINGULARITY_CACHEDIR=/YOURDIR/.cache/singularity
```

Run SCATTR without any arguments to print the short help:

```
singularity run -e khanlab_scattr_latest.sif
```

Use the `-h` option to get a detailed help listing:

```
singularity run -e khanlab_scattr_latest.sif -h
```

Note that all the Snakemake command-line options are also available in SCATTR, and can be listed with `--help-snakemake`:

```
singularity run -e khanlab_scattr_latest.sif --help-snakemake
```

3.3.3 Running an example

We will use the `test` folder found from the Github repository to demonstrate an example of how to run SCATTR:

```
singularity run -e khanlab_scattr_latest.sif test/data/bids test/data/derivatives␣
↪participant --fs-license test/fs_license --force-output -n
```

Explanation

Everything prior to the container (`khanlab_scattr_latest.sif`) are arguments to Singluarity / Apptainer, and after are to SCATTR itself. The first three arguments to SCATTR (as with any BIDS App) are the input folder (`test/data/bids`), the output folder (`test/data/derivatives`), and the analysis level (`participant`). The `participant` analysis level is used in SCATTR to perform further participant-level processing of Freesurfer (thalamus segmentation), external atlases (combining segmentations) and diffusion derived data (estimation of fibre orientation distributions). This includes estimating an average response function from input data. The `--fs-license` argument allows for specification of the location of the required Freesurfer license file if not already specified in the `FS_LICENSE` environment variable. Note, that this is required to perform any Freesurfer-related processing. The `--force-output` flag is a

Snakemake argument that is invoked to allow for writing of output file to already existing folders - in this case, for thalamus segmentations via Freesurfer. We also used the `--dry-run/-n` option to print out what would run, without actually running the workflow.

When you run the above command, a long listing will print out, describing all the rules that will be run. We can also have a shell command used for each rule printed to screen using the `-p` Snakemake option

```
singularity run -e khanlab_scattr_latest.sif test/data/bids test/data/derivatives␣
↪participant --fs-license test/fs_license --force-output -np
```

Now to actually run the workflow, we need to specify how many cores to use and leave out the dry-run option. The Snakemake `--cores` option tells SCATTR how many cores to use. Using `--cores 8` means that SCATTR will only make use of 8 cores at most. Generally speaking, you should use `--cores all`, so it can make maximal use of all available CPU cores it has access to on your system. This is especially useful if you are running multiple subjects.

Running the follow command (SCATTR on a single subject) may take up to ~36 hours with 8 cores and default parameters, but could be much longer (several days) if you only have a single core.

```
singularity run -e khanlab_scattr_latest.sif test/data/bids test/data/derivatives␣
↪participant --fs-license test/fs_license --force-output -p --cores all
```

Note that you may need to adjust your Singularity / Apptainer options to ensure the container can read and write to your input and output directories, respectively. You can bind paths easily by setting an environment variable, e.g. if you have a `/project` folder that contains your data, you can add it to the `SINGULARITY_BINDPATH` so it is available when you are running a container:

```
export SINGULARITY_BINDPATH=/data:/data
```

After this completes, you have additional folders in your output folder, `test/data/derivatives`, for the one subject.

Exploring different options

If you alternatively want to run SCATTR using a pre-defined average response function, you can use the `--responsemean_dir` flag to specify the location to where the average response function is located.

```
singularity run khanlab_scattr_latest.sif test/data/bids test/data/derivatives/␣
↪participant --responsemean_dir test/data/derivatives/mrtrix/avg --fs-license test/.fs_
↪license -np --force-output
```

Other parameters exist, which may help to improve processing times at the expense of sensitivity / specificity (e.g. reducing the number of streamlines generated).

3.4 Command Line Interface (CLI)

3.4.1 Scattr CLI

The following can also be seen by entering `scattr -h` into your terminal.

These are all the required and optional arguments SCATTR accepts in order to run flexibly on many different input data types and with many options. In most cases, only the required arguments are needed.

Snakebids helps build BIDS Apps with Snakemake

```
usage: scattr [-h] [--pybidsdb-dir PYBIDSDB_DIR] [--pybidsdb-reset]
              [--force-output] [--help-snakemake]
              [--participant_label PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]]
              [--exclude_participant_label EXCLUDE_PARTICIPANT_LABEL [EXCLUDE_
↪PARTICIPANT_LABEL ...]]
              [--slurm_tmpdir] [--freesurfer-dir [FREESURFER_DIR]]
              [--dwi_dir [DWI_DIR]] [--pybidsdb-dwi-dir [PYBIDSDB_DWI_DIR]]
              [--responsemean-dir [RESPONSEMEAN_DIR]]
              [--responsemean_ses [RESPONSEMEAN_SES]]
              [--fs-license [FS_LICENSE]]
              [--labelmerge-base-dir [LABELMERGE_BASE_DIR]]
              [--labelmerge-overlay-dir [LABELMERGE_OVERLAY_DIR]]
              [--labelmerge_base_desc [LABELMERGE_BASE_DESC]]
              [--labelmerge_overlay_desc [LABELMERGE_OVERLAY_DESC]]
              [--labelmerge-base-drops [LABELMERGE_BASE_DROPS ...]]
              [--labelmerge-overlay-drops [LABELMERGE_OVERLAY_DROPS ...]]
              [--labelmerge-base-exceptions [LABELMERGE_BASE_EXCEPTIONS ...]]
              [--labelmerge-overlay-exceptions [LABELMERGE_OVERLAY_EXCEPTIONS ...]]
              [--skip-labelmerge] [--skip-brainstem] [--skip-thal-seg]
              [--bzero_thresh [BZERO_THRESH]] [--shells [SHELLS ...]]
              [--lmax [LMAX ...]] [--step STEP] [--sl-count SL_COUNT]
              [--radial-search RADIAL_SEARCH]
              [--filter-T1w FILTER_T1W [FILTER_T1W ...]]
              [--wildcards-T1w WILDCARDS_T1W [WILDCARDS_T1W ...]]
              [--path-T1w PATH_T1W]
             bids_dir output_dir {participant}
```

STANDARD

Standard options for all snakebids apps

--pybidsdb-dir, --pybidsdb_dir Optional path to directory of SQLite databasefile for PyBIDS. If directory is passed and folder exists, indexing is skipped. If pybidsdb_reset is called, indexing will persist

--pybidsdb-reset, --pybidsdb_reset Reindex existing PyBIDS SQLite database

Default: False

--force-output, --force_output Force output in a new directory that already has contents

Default: False

--help-snakemake, --help_snakemake Options to Snakemake can also be passed directly at the command-line, use this to print Snakemake usage

SNAKEBIDS

Options for snakebids app

bids_dir The directory with the input dataset formatted according to the BIDS standard.

output_dir The directory where the output files should be stored. If you are running group level analysis, this folder should be prepopulated with the results of the participant level analysis.

analysis_level Possible choices: participant

Level of the analysis that will be performed

--participant_label, --participant-label The label(s) of the participant(s) that should be analyzed. The label corresponds to sub-<participant_label> from the BIDS spec (so it does not include "sub-"). If this parameter is not provided, all subjects will be analyzed. Multiple participants can be specified with a space seperated list.

--exclude_participant_label, --exclude-participant-label The label(s) of the participant(s) that should be excluded. The label corresponds to sub-<participant_label> from the BIDS spec (so it does not include "sub-"). If this parameter is not provided, all subjects will be analyzed. Multiple participants can be specified with a space sepearated list.

--slurm_tmpdir, --slurm-tmpdir Flag to indicate use of SLURM temporary directory. A temporary directory is used to improve write speeds of output files on a networked system. If not used, the workflow will default to system /tmp directory.

Default: False

--freesurfer-dir, --freesurfer_dir The path to the freesurfer directory. If not provided, workflow assumes the directory exists at <output_dir>/freesurfer.

--dwi_dir, --dwi-dir The path to the directory containing pre-processed dwi data transformed to subject T1w space. If not provided, workflow assumes this data exists in <bids_dir>/<subject>/dwi.

--pybidsdb-dwi-dir, --pybidsdb_dwi_dir The path to the pybids database associated with provided dwi_dir

--responsemean-dir, --responsemean_dir The path to the directory containing average response functions. If not provided, one will be computed from the subjects in the input directory.

--responsemean_ses, --responsemean-ses The session used to compute the average response function. If multiple are available in the dataset, it is HIGHLY RECOMMENDED to set this to a specific session. If not provided, the average response function will be computed with all possible sessions.

--fs-license, --fs_license Path to Freesurfer license file. If not provided, workflow will check FS_LICENSE environment variable for one.

--labelmerge-base-dir, --labelmerge_base_dir BIDS directory containing base input labelmaps.

--labelmerge-overlay-dir, --labelmerge_overlay_dir BIDS directory containing overlay input labelmaps.

--labelmerge_base_desc, --labelmerge-base-desc Description entity for base labelmaps used in labelmerge. By default, uses ZonaBB entity as intended in the subcortical connectome workflow (default: ZonaBB).

Default: "ZonaBB"

--labelmerge_overlay_desc, --labelmerge-overlay-desc Description entity for overlay labelmaps used in labelmerge. By default, uses FreesurferThal value as intended in the subcortical connectome workflow (default: FreesurferThal).

Default: "FreesurferThal"

--labelmerge-base-drops, --labelmerge_base_drops Space-separated list of indices to drop from the base image used in labelmerge. By default, will drop the thalamus labels for intended use in the subcortical connectome workflow (default: 15 16).

Default: "15 16"

--labelmerge-overlay-drops, --labelmerge_overlay_drops Space-separated list of indices to drop from the overlay image used in labelmerge.

--labelmerge-base-exceptions, --labelmerge_base_exceptions Space-separated list of integer labels from the base labelmap to keep over overlay labels at corresponding voxels.

--labelmerge-overlay-exceptions, --labelmerge_overlay_exceptions Space-separated list of integer labels from the overlay labelmap to be overwritten by base labels at corresponding voxels.

--skip-labelmerge, --skip_labelmerge Skip the merging step, using only the base labelmap to identify structures of interest to target.

Default: False

--skip-brainstem, --skip_brainstem By default, the worklow adds the brainstem to the combined output of labelmerge. Use of this flag will disable the addition of the brainstem label.

Default: False

--skip-thal-seg, --skip_thal_seg By default, the workflow performs segmentation of the thalamus on (previously processed) Freesurfer outputs. Use of this flag will skip the thalamus segmentation step.

Default: False

--bzero_thresh, --bzero-thresh Set the threshold for a shell to be considered b=0. By default, this value is set to 10

Default: 10

--shells (Mrtrix3) specify one or more b-values to use during processing, as a space-separated list of the desired approximate b-values (b-values are clustered to allow for small deviations). Note that some commands are incompatible with multiple b-values, and will report an error if more than one b-value is provided. WARNING: note that, even though b=0 volumes are never referred to as shells in the literature, they still have to be explicitly included in the list of b-values as provided to the -shell option! Several algorithms which include the b=0 volumes in their computations may otherwise return an undesired reuslt.

--lmax (Mrtrix3) the maximum spherical harmonic order for the output FOD(s). For algorithms with multiple outputs, this should be provided as a space-sperated list of integers, one for each output image; for single-output algorithms, only a single integer should be provided. If omitted, commands will use the lmax of the corresponding response function (i.e. based on its number of coefficients), up to a maximum of 8.

--step (Mrtrix3) set the step size of the algorithm in mm. Should set to 4 steps per voxel (e.g. 1/4 * voxel size) in order to sample more frequently in compact region. (default: 0.35mm)

Default: 0.35

--sl-count, --sl_count (Mrtrix3) set the desired number of streamlines to be selected, after all selection criteria has been applied (i.e. inclusion/exclusion ROIS, min/max length, etc. Streamlines will be seeded until this number of streamlines have been selected, or the maximum allowed number of seeds has been exceeded. Set to zero to disable, which will result in streamlines being seeded until the maximum allowed number of seeds has been reached. (default: 20,000,000 streamlines)

Default: 20000000

--radial-search, --radial_search (Mrtrix3) perform a radial search from each streamline endpoint to locate the nearest node. Argument is the maximum radius in mm; if no node is found within this radius, the streamline endpoint is not assigned to any node. (default: 1.5mm)

Default: 1.5

BIDS FILTERS

Filters to customize PyBIDS get() as key=value pairs, or as key:{REQUIRED|OPTIONAL|NONE} (case-insensitive), to enforce the presence or absence of values for that key.

--filter-T1w, --filter_T1w (default: suffix=T1w extension=.nii.gz datatype=anat desc=brain part=['mag', False])

INPUT WILDCARDS

File path entities to use as wildcards in snakemake

--wildcards-T1w, --wildcards_T1w (default: subject session run)

PATH OVERRIDE

Options for overriding BIDS by specifying absolute paths that include wildcards, e.g.: /path/to/my_data/{subject}/t1.nii.gz

--path-T1w, --path_T1w

3.4.2 Snakemake CLI

In addition to the above command line arguments, Snakemake arguments can also be passed at the SCATTR command line.

The most critical of these is the `--cores / -c` and `--force-output` arguments, which are **required** arguments for SCATTR.

The complete list of Snakemake arguments are below, and most act to determine your environment and app behaviours. They will likely only need to be used for running in cloud environments or troubleshooting. These can be listed from the command line with `scattr --help-snakemake`.

Snakemake is a Python based language and execution environment for GNU Make-like workflows.

```
usage: snakemake [-h] [--dry-run] [--profile PROFILE]
                 [--workflow-profile WORKFLOW_PROFILE] [--cache [RULE ...]]
                 [--snakefile FILE] [--cores [N]] [--jobs [N]]
                 [--local-cores N] [--resources [NAME=INT ...]]
                 [--set-threads RULE=THREADS [RULE=THREADS ...]]
                 [--max-threads MAX_THREADS]
                 [--set-resources RULE:RESOURCE=VALUE [RULE:RESOURCE=VALUE ...]]
                 [--set-scatter NAME=SCATTERITEMS [NAME=SCATTERITEMS ...]]
                 [--set-resource-scopes RESOURCE=[global|local]
                 [RESOURCE=[global|local] ...]]
                 [--default-resources [NAME=INT ...]]
                 [--preemption-default PREEMPTION_DEFAULT]
                 [--preemptible-rules PREEMPTIBLE_RULES [PREEMPTIBLE_RULES ...]]
                 [--config [KEY=VALUE ...]] [--configfile FILE [FILE ...]]
                 [--envvars VARNAME [VARNAME ...]] [--directory DIR] [--touch]
                 [--keep-going]
                 [--rerun-triggers {mtime,params,input,software-env,code} [{mtime,params,
↪input,software-env,code} ...]]
                 [--force] [--forceall] [--forcerun [TARGET ...]]
                 [--prioritize TARGET [TARGET ...]]
                 [--batch RULE=BATCH/BATCHES] [--until TARGET [TARGET ...]]
                 [--omit-from TARGET [TARGET ...]] [--rerun-incomplete]
                 [--shadow-prefix DIR] [--scheduler [{ilp,greedy}]]
                 [--wms-monitor [WMS_MONITOR]]
                 [--wms-monitor-arg [NAME=VALUE ...]]
                 [--scheduler-ilp-solver {PULP_CBC_CMD}]
                 [--scheduler-solver-path SCHEDULER_SOLVER_PATH]
                 [--conda-base-path CONDA_BASE_PATH] [--no-subworkflows]
                 [--groups GROUPS [GROUPS ...]]
                 [--group-components GROUP_COMPONENTS [GROUP_COMPONENTS ...]]
                 [--report [FILE]] [--report-stylesheet CSSFILE]
                 [--draft-notebook TARGET] [--edit-notebook TARGET]
                 [--notebook-listen IP:PORT] [--lint [{text,json}]]
                 [--generate-unit-tests [TESTPATH]] [--containerize]
                 [--export-cwl FILE] [--list] [--list-target-rules] [--dag]
                 [--rulegraph] [--filegraph] [--d3dag] [--summary]
                 [--detailed-summary] [--archive FILE]
                 [--cleanup-metadata FILE [FILE ...]] [--cleanup-shadow]
                 [--skip-script-cleanup] [--unlock] [--list-version-changes]
                 [--list-code-changes] [--list-input-changes]
                 [--list-params-changes] [--list-untracked]
                 [--delete-all-output] [--delete-temp-output]
                 [--bash-completion] [--keep-incomplete] [--drop-metadata]
                 [--version] [--reason] [--gui [PORT]] [--printshellcmds]
                 [--debug-dag] [--stats FILE] [--nocolor]
                 [--quiet [{progress,rules,all} ...]] [--print-compilation]
                 [--verbose] [--force-use-threads] [--allow-ambiguity]
                 [--nolock] [--ignore-incomplete]
                 [--max-inventory-time SECONDS] [--latency-wait SECONDS]
                 [--wait-for-files [FILE ...]] [--wait-for-files-file FILE]
                 [--notemp] [--all-temp] [--keep-remote] [--keep-target-files]
                 [--allowed-rules ALLOWED_RULES [ALLOWED_RULES ...]]
                 [--target-jobs TARGET_JOBS [TARGET_JOBS ...]]
```

(continues on next page)

(continued from previous page)

```
                [--local-groupid LOCAL_GROUPID]
                [--max-jobs-per-second MAX_JOBS_PER_SECOND]
                [--max-status-checks-per-second MAX_STATUS_CHECKS_PER_SECOND]
                [-T RETRIES] [--attempt ATTEMPT]
                [--wrapper-prefix WRAPPER_PREFIX]
                [--default-remote-provider {S3,GS,FTP,SFTP,S3Mocked,gfal,gridftp,iRODS,
↪AzBlob,XRootD}]
                [--default-remote-prefix DEFAULT_REMOTE_PREFIX]
                [--no-shared-fs] [--greediness GREEDINESS] [--no-hooks]
                [--overwrite-shellcmd OVERWRITE_SHELLCMD] [--debug]
                [--runtime-profile FILE] [--mode {0,1,2}]
                [--show-failed-logs] [--log-handler-script FILE]
                [--log-service {none,slack,wms}] [--slurm]
                [--cluster CMD | --cluster-sync CMD | --drmaa [ARGS]]
                [--cluster-config FILE] [--immediate-submit]
                [--jobscript SCRIPT] [--jobname NAME]
                [--cluster-status CLUSTER_STATUS]
                [--cluster-cancel CLUSTER_CANCEL]
                [--cluster-cancel-nargs CLUSTER_CANCEL_NARGS]
                [--cluster-sidecar CLUSTER_SIDECAR] [--drmaa-log-dir DIR]
                [--kubernetes [NAMESPACE]] [--container-image IMAGE]
                [--k8s-cpu-scalar FLOAT]
                [--k8s-service-account-name SERVICEACCOUNTNAME] [--tibanna]
                [--tibanna-sfn TIBANNA_SFN] [--precommand PRECOMMAND]
                [--tibanna-config TIBANNA_CONFIG [TIBANNA_CONFIG ...]]
                [--google-lifesciences]
                [--google-lifesciences-regions GOOGLE_LIFESCIENCES_REGIONS [GOOGLE_
↪LIFESCIENCES_REGIONS ...]]
                [--google-lifesciences-location GOOGLE_LIFESCIENCES_LOCATION]
                [--google-lifesciences-keep-cache]
                [--google-lifesciences-service-account-email GOOGLE_LIFESCIENCES_
↪SERVICE_ACCOUNT_EMAIL]
                [--google-lifesciences-network GOOGLE_LIFESCIENCES_NETWORK]
                [--google-lifesciences-subnetwork GOOGLE_LIFESCIENCES_SUBNETWORK]
                [--az-batch] [--az-batch-enable-autoscale]
                [--az-batch-account-url [AZ_BATCH_ACCOUNT_URL]] [--flux]
                [--tes URL] [--use-conda]
                [--conda-not-block-search-path-envvars] [--list-conda-envs]
                [--conda-prefix DIR] [--conda-cleanup-envs]
                [--conda-cleanup-pkgs [{tarballs,cache}]]
                [--conda-create-envs-only] [--conda-frontend {conda,mamba}]
                [--use-singularity] [--singularity-prefix DIR]
                [--singularity-args ARGS] [--cleanup-containers]
                [--use-envmodules]
                [target ...]
```

EXECUTION

target Targets to build. May be rules or files.

--dry-run, --dryrun, -n Do not execute anything, and display what would be done. If you have a very large workflow, use –dry-run –quiet to just print a summary of the DAG of jobs.

Default: False

--profile Name of profile to use for configuring Snakemake. Snakemake will search for a corresponding folder in /etc/xdg/snakemake and /home/docs/.config/snakemake. Alternatively, this can be an absolute or relative path. The profile folder has to contain a file 'config.yaml'. This file can be used to set default values for command line options in YAML format. For example, '–cluster qsub' becomes 'cluster: qsub' in the YAML file. Profiles can be obtained from https://github.com/snakemake-profiles. The profile can also be set via the environment variable $SNAKEMAKE_PROFILE. To override this variable and use no profile at all, provide the value 'none' to this argument.

--workflow-profile Path (relative to current directory) to workflow specific profile folder to use for configuring Snakemake with parameters specific for this workflow (like resources). If this flag is not used, Snakemake will by default use 'profiles/default' if present (searched both relative to current directory and relative to Snakefile, in this order). For skipping any workflow specific profile provide the special value 'none'. Settings made in the workflow profile will override settings made in the general profile (see –profile). The profile folder has to contain a file 'config.yaml'. This file can be used to set default values for command line options in YAML format. For example, '–cluster qsub' becomes 'cluster: qsub' in the YAML file. It is advisable to use the workflow profile to set or overwrite e.g. workflow specific resources like the amount of threads of a particular rule or the amount of memory needed. Note that in such cases, the arguments may be given as nested YAML mappings in the profile, e.g. 'set-threads: myrule: 4' instead of 'set-threads: myrule=4'.

--cache Store output files of given rules in a central cache given by the environment variable $SNAKEMAKE_OUTPUT_CACHE. Likewise, retrieve output files of the given rules from this cache if they have been created before (by anybody writing to the same cache), instead of actually executing the rules. Output files are identified by hashing all steps, parameters and software stack (conda envs or containers) needed to create them.

--snakefile, -s The workflow definition in form of a snakefile. Usually, you should not need to specify this. By default, Snakemake will search for 'Snakefile', 'snakefile', 'workflow/Snakefile', 'workflow/snakefile' beneath the current working directory, in this order. Only if you definitely want a different layout, you need to use this parameter.

--cores, -c Use at most N CPU cores/jobs in parallel. If N is omitted or 'all', the limit is set to the number of available CPU cores. In case of cluster/cloud execution, this argument sets the maximum number of cores requested from the cluster or cloud scheduler. (See https://snakemake.readthedocs.io/en/stable/snakefiles/rules.html#resources-remote-execution for more info)This number is available to rules via workflow.cores.

--jobs, -j Use at most N CPU cluster/cloud jobs in parallel. For local execution this is an alias for –cores. Note: Set to 'unlimited' in case, this does not play a role.

--local-cores In cluster/cloud mode, use at most N cores of the host machine in parallel (default: number of CPU cores of the host). The cores are used to execute local rules. This option is ignored when not in cluster/cloud mode.

Default: 2

--resources, --res Define additional resources that shall constrain the scheduling analogously to –cores (see above). A resource is defined as a name and an integer value. E.g. –resources mem_mb=1000. Rules can use resources by defining the resource keyword, e.g. resources: mem_mb=600. If now two rules require 600 of the resource 'mem_mb' they won't be run in parallel by the scheduler. In cluster/cloud mode, this argument will also constrain the amount of resources requested from the server. (See https://snakemake.readthedocs.io/en/stable/snakefiles/rules.html#resources-remote-execution for more info)

--set-threads Overwrite thread usage of rules. This allows to fine-tune workflow parallelization. In particular, this is helpful to target certain cluster nodes by e.g. shifting a rule to use more, or less threads than defined in the workflow. Thereby, THREADS has to be a positive integer, and RULE has to be the name of the rule.

--max-threads Define a global maximum number of threads available to any rule. Rules requesting more threads (via the threads keyword) will have their values reduced to the maximum. This can be useful when you want to restrict the maximum number of threads without modifying the workflow definition or overwriting rules individually with –set-threads.

--set-resources Overwrite resource usage of rules. This allows to fine-tune workflow resources. In particular, this is helpful to target certain cluster nodes by e.g. defining a certain partition for a rule, or overriding a temporary directory. Thereby, VALUE has to be a positive integer or a string, RULE has to be the name of the rule, and RESOURCE has to be the name of the resource.

--set-scatter Overwrite number of scatter items of scattergather processes. This allows to fine-tune workflow parallelization. Thereby, SCATTERITEMS has to be a positive integer, and NAME has to be the name of the scattergather process defined via a scattergather directive in the workflow.

--set-resource-scopes Overwrite resource scopes. A scope determines how a constraint is reckoned in cluster execution. With RESOURCE=local, a constraint applied to RESOURCE using –resources will be considered the limit for each group submission. With RESOURCE=global, the constraint will apply across all groups cumulatively. By default, only *mem_mb* and *disk_mb* are considered local, all other resources are global. This may be modified in the snakefile using the *resource_scopes:* directive. Note that number of threads, specified via –cores, is always considered local. (See https://snakemake.readthedocs.io/en/stable/snakefiles/rules.html#resources-remote-execution for more info)

--default-resources, --default-res Define default values of resources for rules that do not define their own values. In addition to plain integers, python expressions over inputsize are allowed (e.g. '2*input.size_mb'). The inputsize is the sum of the sizes of all input files of a rule. By default, Snakemake assumes a default for mem_mb, disk_mb, and tmpdir (see below). This option allows to add further defaults (e.g. account and partition for slurm) or to overwrite these default values. The defaults are 'mem_mb=max(2*input.size_mb, 1000)', 'disk_mb=max(2*input.size_mb, 1000)' (i.e., default disk and mem usage is twice the input file size but at least

1GB), and the system temporary directory (as given by $TMPDIR, $TEMP, or $TMP) is used for the tmpdir resource. The tmpdir resource is automatically used by shell commands, scripts and wrappers to store temporary data (as it is mirrored into $TMPDIR, $TEMP, and $TMP for the executed subprocesses). If this argument is not specified at all, Snakemake just uses the tmpdir resource as outlined above.

--preemption-default A preemptible instance can be requested when using the Google Life Sciences API. If you set a –preemption-default, all rules will be subject to the default. Specifically, this integer is the number of restart attempts that will be made given that the instance is killed unexpectedly. Note that preemptible instances have a maximum running time of 24 hours. If you want to set preemptible instances for only a subset of rules, use –preemptible-rules instead.

--preemptible-rules A preemptible instance can be requested when using the Google Life Sciences API. If you want to use these instances for a subset of your rules, you can use –preemptible-rules and then specify a list of rule and integer pairs, where each integer indicates the number of restarts to use for the rule's instance in the case that the instance is terminated unexpectedly. –preemptible-rules can be used in combination with –preemption-default, and will take priority. Note that preemptible instances have a maximum running time of 24. If you want to apply a consistent number of retries across all your rules, use –preemption-default instead. Example: snakemake –preemption-default 10 –preemptible-rules map_reads=3 call_variants=0

--config, -C Set or overwrite values in the workflow config object. The workflow config object is accessible as variable config inside the workflow. Default values can be set by providing a JSON file (see Documentation).

--configfile, --configfiles Specify or overwrite the config file of the workflow (see the docs). Values specified in JSON or YAML format are available in the global config dictionary inside the workflow. Multiple files overwrite each other in the given order. Thereby missing keys in previous config files are extended by following configfiles. Note that this order also includes a config file defined in the workflow definition itself (which will come first).

--envvars Environment variables to pass to cloud jobs.

--directory, -d Specify working directory (relative paths in the snakefile will use this as their origin).

--touch, -t Touch output files (mark them up to date without really changing them) instead of running their commands. This is used to pretend that the rules were executed, in order to fool future invocations of snakemake. Fails if a file does not yet exist. Note that this will only touch files that would otherwise be recreated by Snakemake (e.g. because their input files are newer). For enforcing a touch, combine this with –force, –forceall, or –forcerun. Note however that you lose the provenance information when the files have been created in reality. Hence, this should be used only as a last resort.

Default: False

--keep-going, -k Go on with independent jobs if a job fails.

Default: False

--rerun-triggers Possible choices: mtime, params, input, software-env, code

Define what triggers the rerunning of a job. By default, all triggers are used, which guarantees that results are consistent with the workflow code and configuration.

If you rather prefer the traditional way of just considering file modification dates, use '–rerun-trigger mtime'.

Default: ['mtime', 'params', 'input', 'software-env', 'code']

--force, -f Force the execution of the selected target or the first rule regardless of already created output.

Default: False

--forceall, -F Force the execution of the selected (or the first) rule and all rules it is dependent on regardless of already created output.

Default: False

--forcerun, -R Force the re-execution or creation of the given rules or files. Use this option if you changed a rule and want to have all its output in your workflow updated.

--prioritize, -P Tell the scheduler to assign creation of given targets (and all their dependencies) highest priority. (EXPERIMENTAL)

--batch Only create the given BATCH of the input files of the given RULE. This can be used to iteratively run parts of very large workflows. Only the execution plan of the relevant part of the workflow has to be calculated, thereby speeding up DAG computation. It is recommended to provide the most suitable rule for batching when documenting a workflow. It should be some aggregating rule that would be executed only once, and has a large number of input files. For example, it can be a rule that aggregates over samples.

--until, -U Runs the pipeline until it reaches the specified rules or files. Only runs jobs that are dependencies of the specified rule or files, does not run sibling DAGs.

--omit-from, -O Prevent the execution or creation of the given rules or files as well as any rules or files that are downstream of these targets in the DAG. Also runs jobs in sibling DAGs that are independent of the rules or files specified here.

--rerun-incomplete, --ri Re-run all jobs the output of which is recognized as incomplete.

Default: False

--shadow-prefix Specify a directory in which the 'shadow' directory is created. If not supplied, the value is set to the '.snakemake' directory relative to the working directory.

--scheduler Possible choices: ilp, greedy

Specifies if jobs are selected by a greedy algorithm or by solving an ilp. The ilp scheduler aims to reduce runtime and hdd usage by best possible use of resources.

Default: "greedy"

--wms-monitor IP and port of workflow management system to monitor the execution of snakemake (e.g. http://127.0.0.1:5000) Note that if your service requires an authorization token, you must export WMS_MONITOR_TOKEN in the environment.

--wms-monitor-arg If the workflow management service accepts extra arguments, provide. them in key value pairs with –wms-monitor-arg. For example, to run an existing workflow using a wms monitor, you can provide the pair id=12345 and the arguments will be provided to the endpoint to first interact with the workflow

--scheduler-ilp-solver Possible choices: PULP_CBC_CMD

Specifies solver to be utilized when selecting ilp-scheduler.

Default: "COIN_CMD"

--scheduler-solver-path Set the PATH to search for scheduler solver binaries (internal use only).

--conda-base-path Path of conda base installation (home of conda, mamba, activate) (internal use only).

--no-subworkflows, --nosw Do not evaluate or execute subworkflows.

Default: False

GROUPING

--groups Assign rules to groups (this overwrites any group definitions from the workflow).

--group-components Set the number of connected components a group is allowed to span. By default, this is 1, but this flag allows to extend this. This can be used to run e.g. 3 jobs of the same rule in the same group, although they are not connected. It can be helpful for putting together many small jobs or benefitting of shared memory setups.

REPORTS

--report Create an HTML report with results and statistics. This can be either a .html file or a .zip file. In the former case, all results are embedded into the .html (this only works for small data). In the latter case, results are stored along with a file report.html in the zip archive. If no filename is given, an embedded report.html is the default.

--report-stylesheet Custom stylesheet to use for report. In particular, this can be used for branding the report with e.g. a custom logo, see docs.

NOTEBOOKS

--draft-notebook Draft a skeleton notebook for the rule used to generate the given target file. This notebook can then be opened in a jupyter server, executed and implemented until ready. After saving, it will automatically be reused in non-interactive mode by Snakemake for subsequent jobs.

--edit-notebook Interactively edit the notebook associated with the rule used to generate the given target file. This will start a local jupyter notebook server. Any changes to the notebook should be saved, and the server has to be stopped by closing the notebook and hitting the 'Quit' button on the jupyter dashboard. Afterwards, the updated notebook will be automatically stored in the path defined in the rule. If the notebook is not yet present, this will create an empty draft.

--notebook-listen The IP address and PORT the notebook server used for editing the notebook (–edit-notebook) will listen on.

Default: "localhost:8888"

UTILITIES

--lint Possible choices: text, json

Perform linting on the given workflow. This will print snakemake specific suggestions to improve code quality (work in progress, more lints to be added in the future). If no argument is provided, plain text output is used.

--generate-unit-tests Automatically generate unit tests for each workflow rule. This assumes that all input files of each job are already present. Rules without a job with present input files will be skipped (a warning will be issued). For each rule, one test case will be created in the specified test folder (.tests/unit by default). After successful execution, tests can be run with 'pytest TESTPATH'.

--containerize Print a Dockerfile that provides an execution environment for the workflow, including all conda environments.

Default: False

--export-cwl Compile workflow to CWL and store it in given FILE.

--list, -l Show available rules in given Snakefile.

Default: False

--list-target-rules, --lt Show available target rules in given Snakefile.

Default: False

--dag Do not execute anything and print the directed acyclic graph of jobs in the dot language. Recommended use on Unix systems: snakemake –dag | dot | display. Note print statements in your Snakefile may interfere with visualization.

Default: False

--rulegraph Do not execute anything and print the dependency graph of rules in the dot language. This will be less crowded than above DAG of jobs, but also show less information. Note that each rule is displayed once, hence the displayed graph will be cyclic if a rule appears in several steps of the workflow. Use this if above option leads to a DAG that is too large. Recommended use on Unix systems: snakemake –rulegraph | dot | display. Note print statements in your Snakefile may interfere with visualization.

Default: False

--filegraph Do not execute anything and print the dependency graph of rules with their input and output files in the dot language. This is an intermediate solution between above DAG of jobs and the rule graph. Note that each rule is displayed once, hence the displayed graph will be cyclic if a rule appears in several steps of the workflow. Use this if above option leads to a DAG that is too large. Recommended use on Unix systems: snakemake –filegraph | dot | display. Note print statements in your Snakefile may interfere with visualization.

Default: False

--d3dag Print the DAG in D3.js compatible JSON format.

Default: False

--summary, -S Print a summary of all files created by the workflow. The has the following columns: filename, modification time, rule version, status, plan. Thereby rule version contains the versionthe file was created with (see the version keyword of rules), and status denotes whether the file is missing, its input files are newer or

if version or implementation of the rule changed since file creation. Finally the last column denotes whether the file will be updated or created during the next workflow execution.

Default: False

--detailed-summary, -D Print a summary of all files created by the workflow. The has the following columns: filename, modification time, rule version, input file(s), shell command, status, plan. Thereby rule version contains the version the file was created with (see the version keyword of rules), and status denotes whether the file is missing, its input files are newer or if version or implementation of the rule changed since file creation. The input file and shell command columns are self explanatory. Finally the last column denotes whether the file will be updated or created during the next workflow execution.

Default: False

--archive Archive the workflow into the given tar archive FILE. The archive will be created such that the workflow can be re-executed on a vanilla system. The function needs conda and git to be installed. It will archive every file that is under git version control. Note that it is best practice to have the Snakefile, config files, and scripts under version control. Hence, they will be included in the archive. Further, it will add input files that are not generated by by the workflow itself and conda environments. Note that symlinks are dereferenced. Supported formats are .tar, .tar.gz, .tar.bz2 and .tar.xz.

--cleanup-metadata, --cm Cleanup the metadata of given files. That means that snakemake removes any tracked version info, and any marks that files are incomplete.

--cleanup-shadow Cleanup old shadow directories which have not been deleted due to failures or power loss.

Default: False

--skip-script-cleanup Don't delete wrapper scripts used for execution

Default: False

--unlock Remove a lock on the working directory.

Default: False

--list-version-changes, --lv List all output files that have been created with a different version (as determined by the version keyword).

Default: False

--list-code-changes, --lc List all output files for which the rule body (run or shell) have changed in the Snakefile.

Default: False

--list-input-changes, --li List all output files for which the defined input files have changed in the Snakefile (e.g. new input files were added in the rule definition or files were renamed). For listing input file modification in the filesystem, use –summary.

Default: False

--list-params-changes, --lp List all output files for which the defined params have changed in the Snakefile.

Default: False

--list-untracked, --lu List all files in the working directory that are not used in the workflow. This can be used e.g. for identifying leftover files. Hidden files and directories are ignored.

Default: False

--delete-all-output Remove all files generated by the workflow. Use together with –dry-run to list files without actually deleting anything. Note that this will not recurse into subworkflows. Write-protected files are not removed. Nevertheless, use with care!

Default: False

--delete-temp-output Remove all temporary files generated by the workflow. Use together with –dry-run to list files without actually deleting anything. Note that this will not recurse into subworkflows.

Default: False

--bash-completion Output code to register bash completion for snakemake. Put the following in your .bashrc (including the accents): *snakemake –bash-completion* or issue it in an open terminal session.

Default: False

--keep-incomplete Do not remove incomplete output files by failed jobs.

Default: False

--drop-metadata Drop metadata file tracking information after job finishes. Provenance-information based reports (e.g. –report and the –list_x_changes functions) will be empty or incomplete.

Default: False

--version, -v show program's version number and exit

OUTPUT

--reason, -r Print the reason for each executed rule (deprecated, always true now).

Default: False

--gui Serve an HTML based user interface to the given network and port e.g. 168.129.10.15:8000. By default Snakemake is only available in the local network (default port: 8000). To make Snakemake listen to all ip addresses add the special host address 0.0.0.0 to the url (0.0.0.0:8000). This is important if Snakemake is used in a virtualised environment like Docker. If possible, a browser window is opened.

--printshellcmds, -p Print out the shell commands that will be executed.

Default: False

--debug-dag Print candidate and selected jobs (including their wildcards) while inferring DAG. This can help to debug unexpected DAG topology or errors.

Default: False

--stats Write stats about Snakefile execution in JSON format to the given file.

--nocolor Do not use a colored output.

Default: False

--quiet, -q Possible choices: progress, rules, all

Do not output certain information. If used without arguments, do not output any progress or rule information. Defining 'all' results in no information being printed at all.

--print-compilation Print the python representation of the workflow.

Default: False

--verbose Print debugging output.

Default: False

BEHAVIOR

--force-use-threads Force threads rather than processes. Helpful if shared memory (/dev/shm) is full or unavailable.

Default: False

--allow-ambiguity, -a Don't check for ambiguous rules and simply use the first if several can produce the same file. This allows the user to prioritize rules by their order in the snakefile.

Default: False

--nolock Do not lock the working directory

Default: False

--ignore-incomplete, --ii Do not check for incomplete output files.

Default: False

--max-inventory-time Spend at most SECONDS seconds to create a file inventory for the working directory. The inventory vastly speeds up file modification and existence checks when computing which jobs need to be executed. However, creating the inventory itself can be slow, e.g. on network file systems. Hence, we do not spend more than a given amount of time and fall back to individual checks for the rest.

Default: 20

--latency-wait, --output-wait, -w Wait given seconds if an output file of a job is not present after the job finished. This helps if your filesystem suffers from latency (default 5).

Default: 5

--wait-for-files Wait –latency-wait seconds for these files to be present before executing the workflow. This option is used internally to handle filesystem latency in cluster environments.

--wait-for-files-file Same behaviour as –wait-for-files, but file list is stored in file instead of being passed on the commandline. This is useful when the list of files is too long to be passed on the commandline.

--notemp, --nt Ignore temp() declarations. This is useful when running only a part of the workflow, since temp() would lead to deletion of probably needed files by other parts of the workflow.

Default: False

--all-temp Mark all output files as temp files. This can be useful for CI testing, in order to save space.

Default: False

--keep-remote Keep local copies of remote input files.

Default: False

--keep-target-files Do not adjust the paths of given target files relative to the working directory.

Default: False

--allowed-rules Only consider given rules. If omitted, all rules in Snakefile are used. Note that this is intended primarily for internal use and may lead to unexpected results otherwise.

--target-jobs Target particular jobs by RULE:WILDCARD1=VALUE,WILDCARD2=VALUE,... This is meant for internal use by Snakemake itself only.

--local-groupid Name for local groupid, meant for internal use only.

Default: "local"

--max-jobs-per-second Maximal number of cluster/drmaa jobs per second, default is 10, fractions allowed.

Default: 10

--max-status-checks-per-second Maximal number of job status checks per second, default is 10, fractions allowed.

Default: 10

-T, --retries, --restart-times Number of times to restart failing jobs (defaults to 0).

Default: 0

--attempt Internal use only: define the initial value of the attempt parameter (default: 1).

Default: 1

--wrapper-prefix Prefix for URL created from wrapper directive (default: https://github.com/snakemake/snakemake-wrappers/raw/). Set this to a different URL to use your fork or a local clone of the repository, e.g., use a git URL like 'git+file://path/to/your/local/clone@'.

Default: "https://github.com/snakemake/snakemake-wrappers/raw/"

--default-remote-provider Possible choices: S3, GS, FTP, SFTP, S3Mocked, gfal, gridftp, iRODS, AzBlob, XRootD

Specify default remote provider to be used for all input and output files that don't yet specify one.

--default-remote-prefix Specify prefix for default remote provider. E.g. a bucket name.

Default: ""

--no-shared-fs Do not assume that jobs share a common file system. When this flag is activated, Snakemake will assume that the filesystem on a cluster node is not shared with other nodes. For example, this will lead to downloading remote files on each cluster node separately. Further, it won't take special measures to deal with filesystem latency issues. This option will in most cases only make sense in combination with –default-remote-provider. Further, when using –cluster you will have to also provide –cluster-status. Only activate this if you know what you are doing.

Default: False

--greediness Set the greediness of scheduling. This value between 0 and 1 determines how careful jobs are selected for execution. The default value (1.0) provides the best speed and still acceptable scheduling quality.

--no-hooks Do not invoke onstart, onsuccess or onerror hooks after execution.

Default: False

--overwrite-shellcmd Provide a shell command that shall be executed instead of those given in the workflow. This is for debugging purposes only.

--debug Allow to debug rules with e.g. PDB. This flag allows to set breakpoints in run blocks.

Default: False

--runtime-profile Profile Snakemake and write the output to FILE. This requires yappi to be installed.

--mode Possible choices: 0, 1, 2

Set execution mode of Snakemake (internal use only).

Default: 0

--show-failed-logs Automatically display logs of failed jobs.

Default: False

--log-handler-script Provide a custom script containing a function 'def log_handler(msg):'. Snakemake will call this function for every logging output (given as a dictionary msg) allowing to e.g. send notifications in the form of e.g. slack messages or emails.

--log-service Possible choices: none, slack, wms

Set a specific messaging service for logging output. Snakemake will notify the service on errors and completed execution. Currently slack and workflow management system (wms) are supported.

SLURM

--slurm Execute snakemake rules as SLURM batch jobs according to their 'resources' definition. SLURM resources as 'partition', 'ntasks', 'cpus', etc. need to be defined per rule within the 'resources' definition. Note, that memory can only be defined as 'mem_mb' or 'mem_mb_per_cpu' as analogous to the SLURM 'mem' and 'mem-per-cpu' flags to sbatch, respectively. Here, the unit is always 'MiB'. In addition '–default_resources' should contain the SLURM account.

Default: False

CLUSTER

--cluster Execute snakemake rules with the given submit command, e.g. qsub. Snakemake compiles jobs into scripts that are submitted to the cluster with the given command, once all input files for a particular job are present. The submit command can be decorated to make it aware of certain job properties (name, rulename, input, output, params, wildcards, log, threads and dependencies (see the argument below)), e.g.: $ snakemake –cluster 'qsub -pe threaded {threads}'.

--cluster-sync cluster submission command will block, returning the remote exit status upon remote termination (for example, this should be used if the cluster command is 'qsub -sync y' (SGE)

--drmaa Execute snakemake on a cluster accessed via DRMAA, Snakemake compiles jobs into scripts that are submitted to the cluster with the given command, once all input files for a particular job are present. ARGS can be used to specify options of the underlying cluster system, thereby using the job properties name, rulename, input, output, params, wildcards, log, threads and dependencies, e.g.: –drmaa ' -pe threaded {threads}'. Note that ARGS must be given in quotes and with a leading whitespace.

--cluster-config, -u A JSON or YAML file that defines the wildcards used in 'cluster' for specific rules, instead of having them specified in the Snakefile. For example, for rule 'job' you may define: { 'job' : { 'time' : '24:00:00' } } to specify the time for rule 'job'. You can specify more than one file. The configuration files are merged with later values overriding earlier ones. This option is deprecated in favor of using –profile, see docs.

Default: []

--immediate-submit, --is Immediately submit all jobs to the cluster instead of waiting for present input files. This will fail, unless you make the cluster aware of job dependencies, e.g. via: $ snakemake –cluster 'sbatch –dependency {dependencies}. Assuming that your submit script (here sbatch) outputs the generated job id to the first stdout line, {dependencies} will be filled with space separated job ids this job depends on. Does not work for workflows that contain checkpoint rules.

Default: False

--jobscript, --js Provide a custom job script for submission to the cluster. The default script resides as 'jobscript.sh' in the installation directory.

--jobname, --jn Provide a custom name for the jobscript that is submitted to the cluster (see –cluster). NAME is "snakejob.{name}.{jobid}.sh" per default. The wildcard {jobid} has to be present in the name.

Default: "snakejob.{name}.{jobid}.sh"

--cluster-status Status command for cluster execution. This is only considered in combination with the –cluster flag. If provided, Snakemake will use the status command to determine if a job has finished successfully or failed. For this it is necessary that the submit command provided to –cluster returns the cluster job id. Then, the status command will be invoked with the job id. Snakemake expects it to return 'success' if the job was successful, 'failed' if the job failed and 'running' if the job still runs.

--cluster-cancel Specify a command that allows to stop currently running jobs. The command will be passed a single argument, the job id.

--cluster-cancel-nargs Specify maximal number of job ids to pass to –cluster-cancel command, defaults to 1000.

Default: 1000

--cluster-sidecar Optional command to start a sidecar process during cluster execution. Only active when –cluster is given as well.

--drmaa-log-dir Specify a directory in which stdout and stderr files of DRMAA jobs will be written. The value may be given as a relative path, in which case Snakemake will use the current invocation directory as the origin. If given, this will override any given

'-o' and/or '-e' native specification. If not given, all DRMAA stdout and stderr files are written to the current working directory.

FLUX

--flux Execute your workflow on a flux cluster. Flux can work with both a shared network filesystem (like NFS) or without. If you don't have a shared filesystem, additionally specify –no-shared-fs.

Default: False

GOOGLE_LIFE_SCIENCE

--google-lifesciences Execute workflow on Google Cloud cloud using the Google Life. Science API. This requires default application credentials (json) to be created and export to the environment to use Google Cloud Storage, Compute Engine, and Life Sciences. The credential file should be exported as GOOGLE_APPLICATION_CREDENTIALS for snakemake to discover. Also, –use-conda, –use-singularity, –config, –configfile are supported and will be carried over.

Default: False

--google-lifesciences-regions Specify one or more valid instance regions (defaults to US)

Default: ['us-east1', 'us-west1', 'us-central1']

--google-lifesciences-location The Life Sciences API service used to schedule the jobs. E.g., us-centra1 (Iowa) and europe-west2 (London) Watch the terminal output to see all options found to be available. If not specified, defaults to the first found with a matching prefix from regions specified with –google-lifesciences-regions.

--google-lifesciences-keep-cache Cache workflows in your Google Cloud Storage Bucket specified by –default-remote-prefix/{source}/{cache}. Each workflow working directory is compressed to a .tar.gz, named by the hash of the contents, and kept in Google Cloud Storage. By default, the caches are deleted at the shutdown step of the workflow.

Default: False

--google-lifesciences-service-account-email Specify a service account email address

--google-lifesciences-network Specify a network for a Google Compute Engine VM instance

--google-lifesciences-subnetwork Specify a subnetwork for a Google Compute Engine VM instance

KUBERNETES

--kubernetes Execute workflow in a kubernetes cluster (in the cloud). NAMESPACE is the namespace you want to use for your job (if nothing specified: 'default'). Usually, this requires –default-remote-provider and –default-remote-prefix to be set to a S3 or GS bucket where your . data shall be stored. It is further advisable to activate conda integration via –use-conda.

--container-image Docker image to use, e.g., when submitting jobs to kubernetes Defaults to 'https://hub.docker.com/r/snakemake/snakemake', tagged with the same version as the currently running Snakemake instance. Note that overwriting this value is up to

your responsibility. Any used image has to contain a working snakemake installation that is compatible with (or ideally the same as) the currently running version.

--k8s-cpu-scalar K8s reserves some proportion of available CPUs for its own use. So, where an underlying node may have 8 CPUs, only e.g. 7600 milliCPUs are allocatable to k8s pods (i.e. snakemake jobs). As 8 > 7.6, k8s can't find a node with enough CPU resource to run such jobs. This argument acts as a global scalar on each job's CPU request, so that e.g. a job whose rule definition asks for 8 CPUs will request 7600m CPUs from k8s, allowing it to utilise one entire node. N.B: the job itself would still see the original value, i.e. as the value substituted in {threads}.

Default: 0.95

--k8s-service-account-name This argument allows the use of customer service accounts for kubernetes pods. If specified serviceAccountName will be added to the pod specs. This is needed when using workload identity which is enforced when using Google Cloud GKE Autopilot.

TES

--tes Send workflow tasks to GA4GH TES server specified by url.

TIBANNA

--tibanna Execute workflow on AWS cloud using Tibanna. This requires –default-remote-prefix to be set to S3 bucket name and prefix (e.g. 'bucketname/subdirectory') where input is already stored and output will be sent to. Using –tibanna implies –default-resources is set as default. Optionally, use –precommand to specify any preparation command to run before snakemake command on the cloud (inside snakemake container on Tibanna VM). Also, –use-conda, –use-singularity, –config, –configfile are supported and will be carried over.

Default: False

--tibanna-sfn Name of Tibanna Unicorn step function (e.g. tibanna_unicorn_monty). This works as serverless scheduler/resource allocator and must be deployed first using tibanna cli. (e.g. tibanna deploy_unicorn –usergroup=monty –buckets=bucketname)

--precommand Any command to execute before snakemake command on AWS cloud such as wget, git clone, unzip, etc. This is used with –tibanna.Do not include input/output download/upload commands - file transfer between S3 bucket and the run environment (container) is automatically handled by Tibanna.

--tibanna-config Additional tibanna config e.g. –tibanna-config spot_instance=true subnet=<subnet_id> security group=<security_group_id>

AZURE_BATCH

--az-batch Execute workflow on azure batch

Default: False

--az-batch-enable-autoscale Enable autoscaling of the azure batch pool nodes, this option will set the initial dedicated node count to zero, and requires five minutes to resize the cluster, so is only recommended for longer running jobs.

Default: False

--az-batch-account-url Azure batch account url, requires AZ_BATCH_ACCOUNT_KEY environment variable to be set.

CONDA

--use-conda If defined in the rule, run job in a conda environment. If this flag is not set, the conda directive is ignored.

Default: False

--conda-not-block-search-path-envvars Do not block environment variables that modify the search path (R_LIBS, PYTHONPATH, PERL5LIB, PERLLIB) when using conda environments.

Default: False

--list-conda-envs List all conda environments and their location on disk.

Default: False

--conda-prefix Specify a directory in which the 'conda' and 'conda-archive' directories are created. These are used to store conda environments and their archives, respectively. If not supplied, the value is set to the '.snakemake' directory relative to the invocation directory. If supplied, the *–use-conda* flag must also be set. The value may be given as a relative path, which will be extrapolated to the invocation directory, or as an absolute path. The value can also be provided via the environment variable $SNAKEMAKE_CONDA_PREFIX.

--conda-cleanup-envs Cleanup unused conda environments.

Default: False

--conda-cleanup-pkgs Possible choices: tarballs, cache

Cleanup conda packages after creating environments. In case of 'tarballs' mode, will clean up all downloaded package tarballs. In case of 'cache' mode, will additionally clean up unused package caches. If mode is omitted, will default to only cleaning up the tarballs.

--conda-create-envs-only If specified, only creates the job-specific conda environments then exits. The *–use-conda* flag must also be set.

Default: False

--conda-frontend Possible choices: conda, mamba

Choose the conda frontend for installing environments. Mamba is much faster and highly recommended.

Default: "mamba"

SINGULARITY

--use-singularity If defined in the rule, run job within a singularity container. If this flag is not set, the singularity directive is ignored.

Default: False

--singularity-prefix Specify a directory in which singularity images will be stored. If not supplied, the value is set to the '.snakemake' directory relative to the invocation directory. If supplied, the *–use-singularity* flag must also be set. The value may be given as a relative path, which will be extrapolated to the invocation directory, or as an absolute path.

--singularity-args Pass additional args to singularity.

Default: ""

--cleanup-containers Remove unused (singularity) containers

Default: False

ENVIRONMENT MODULES

--use-envmodules If defined in the rule, run job within the given environment modules, loaded in the given order. This can be combined with –use-conda and –use-singularity, which will then be only used as a fallback for rules which don't define environment modules.

Default: False

3.5 Running SCATTR on your data

This section goes over the command line options you will find most useful when running SCATTR on your dataset, along with describing some issues you may face.

Note: Please first refer to the simple example in the Installation section, which goes over running SCATTR on a test dataset and the essential required options.

3.5.1 Freesurfer License

To perform Freesurfer-related processing (e.g. thalamus segmentation), Freesurfer is directly invoked. As such, a Freesurfer license is required to perform such steps in the workflow. By default, SCATTR attempts to use the Freesurfer license saved in the environment variable `FS_LICENSE`. Alternatively, the path to the Freesurfer license may be passed along by invoking the `--fs-license` parameter:

```
--fs-license /path/to/fs_license
```

3.5.2 Including / excluding subjects to process

By default, SCATTR will run on **all** subjects in the dataset. If you wish to run on only a subset of subjects, you can use the `--participant-label` flag:

```
--participant-label 001
```

which would only run on `sub-001`. You can add additional subjects by passing a space-separated list to this option:

```
--participant-label 001 002
```

which would run for `sub-001` and `sub-002`.

Similarly, subjects can be excluded from processing using the `--exclude-participant-label` flag.

3.5.3 Alternate Freesurfer / derived-diffusion data locations

By default, SCATTR attempts to locate Freesurfer and derived diffusion data locations. Users can overwrite these options, by passing along the actual location of each using either `--freesurfer_dir` or `--dwi_dir`, respectively.

```
--freesurfer_dir /path/to/fs_dir --dwi_dir /path/to/dwi_dir
```

3.5.4 Pre-generated average response function

In some cases, an average response function may have already been separately generated, which is then used for downstream processing (e.g. generated from controls and applied to a patient population). To use a pre-generated average resposne function, the location of the directory containing the associated files can be passed along using `--responsemean_dir`:

```
--responsemean_dir /path/to/average_response_dir
```

3.5.5 Tractography on network storage

Performing tractography can require millions of reads and writes to the storage system in order to update the file on-the-fly. This process can be extremely slow on network storages. To help with this, SCATTR always reads and writes the tractography to a temporary location (e.g. `/tmp`) before copying the output to the final output, significantly improving the time it takes for tractography to be generated. On a network system, you may be unable to write to `/tmp`. An alternative on systems with `SLURM` workload managers is to invoke `--slurm_tmpdir`, which requests that the workflow write to the local temporary storage system (e.g. `/localscratch`) instead of the network temporary storage.

3.5.6 BIDS Parsing limitations

SCATTR uses Snakebids, which makes use of pybids to parse a BIDS-compliant dataset. However, because of the way Snakebids and Snakemake operate, one limitation is that the input files in your BIDS dataset needs to be consistent in terms of what optional BIDS entities exist in them. We can use the acqusition (`acq`) entity as an example. SCATTR should have no problem parsing the following dataset:

```
PATH_TO_BIDS_DIR/
└── dataset_description.json
└── sub-001/
    └── anat/
        └── sub-001_acq-mprage_T1w.nii.gz
└── sub-002/
    └── anat/
        ├── sub-002_acq-spgr_T1w.nii.gz
...
```

as the path (with wildcards) will be interpreted as `sub-{subject}_acq-{acq}_T1w.nii.gz`.

However, the following dataset will raise an error:

```
PATH_TO_BIDS_DIR/
└── dataset_description.json
└── sub-001/
    └── anat/
        ├── sub-001_acq-mprage_T1w.nii.gz
└── sub-002/
    └── anat/
        ├── sub-002_T1w.nii.gz
...
```

because two distinct paths (with wildcards) would be found for T1w images: `sub-{subject}_acq-{acq}_T1w.nii.gz` and `sub-{subject}_T1w.nii.gz`.

Similarly, you could not have some subjects with the `ses` identifier, and some subjects without it. There will soon be added functionality in Snakebids to filter out extra files, but for now, if your dataset has these issues, you will need to rename or remove extraneous files.

More example of possible BIDS-compliant datasets can be found in `scattr/test/data`.

3.6 Frequently Asked Questions (FAQs)

3.6.1 1. Why do I get the error `No input images found for <modality>`?

The workflow was unable to find the required input files from the input BIDS directory. This can happen if:

- Singularity or Docker cannot access your input directory. For Singularity, ensure your Singularity options are appropriate, in particular `SINGULARITY_BINDPATH`. For Docker, ensure you are mounting the correct directory with the `-v` flag described in the Getting Started section.
- SCATTR does not recognize your BIDS-formatted input images. This can occur if, for example, T1w images are labelled with the suffix `t1w.nii.gz` instead of `T1w.nii.gz` as per BIDS specifications. SCATTR makes use of pybids to parse the dataset, so we suggest using the BIDS Validator to ensure your dataset has no errors.

If you passed Freesurfer or diffusion derivative locations (`--freesurfer-dir` or `--dwi-dir`), you may get get this message as the files are a different location than in the input directory provided.

3.6.2 2. What if I want to use merge two different atlases?

SCATTR uses labelmerge to combine labels from different atlases. To that end, we have exposed the `labelmerge` command-line arguments, enabling substitution of different atlases. These arguments are pre-pended with `--labelmerge_<labelmerge_arg>`. We encourage you to check out the `labelmerge` documention (linked above), as well as SCATTR's help command to see all available options.

NOTE: If using a different atlas, you will need to ensure that the associated metadata is available (according to the BIDS specification) for each subject part of the input dataset and in the same space as the T1w image.

3.6.3 3. I only want to use labels from a single atlas.

If you only want to use a labels from a single atlas (e.g. skip labelmerge), you can invoke the `--skip_labelmerge` argument. If you are using this flag, you can combine this with the `--labelmerge_base_dir` and `--labelmerge_base_desc` arguments to provide your own atlas. Similar to using custom atlases (from *2*), it is expected that the imaging and associated metadata is available for all subjects in the dataset and that all files follow the BIDS specification.

3.7 Workflow Details

This section describes the SCATTR workflow (i.e. steps taken to produce intermediate and final files). SCATTR is a Snakemake workflow, and thus a directed acyclic graph (DAG) that is automatically configured based on a set of rules.

3.7.1 Overall workflow

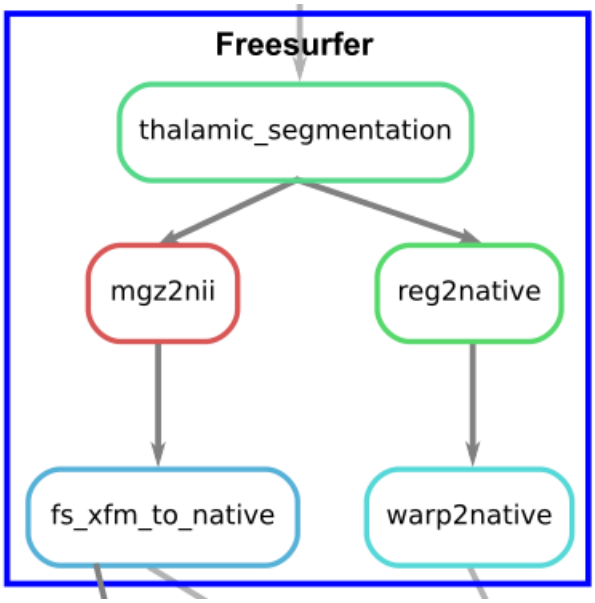
Below is an image exhibiting the workflow DAG. Each rounded rectangle in the DAG represents a rule (i.e. some code or script that produces an output), with arrows representing the connections (i.e. inputs / outputs) to these rules.

Although it may look complex, it is also organized into groups of rules, each representing the different phases of the workflow. Each grouped set of rules also exist in separate rule files, which can be found under the rules sub-directories in the workflow source. For example, the `freesurfer.smk` file contains rules associated with further processing using Freesurfer (e.g. thalamus segmentation), and these are grouped together in the above diagram by a blue rectangle labeled `freesurfer`.

The main phases of the workflow are described in the sections below, zooming in on the rules within each blue rectangle.

Freesurfer

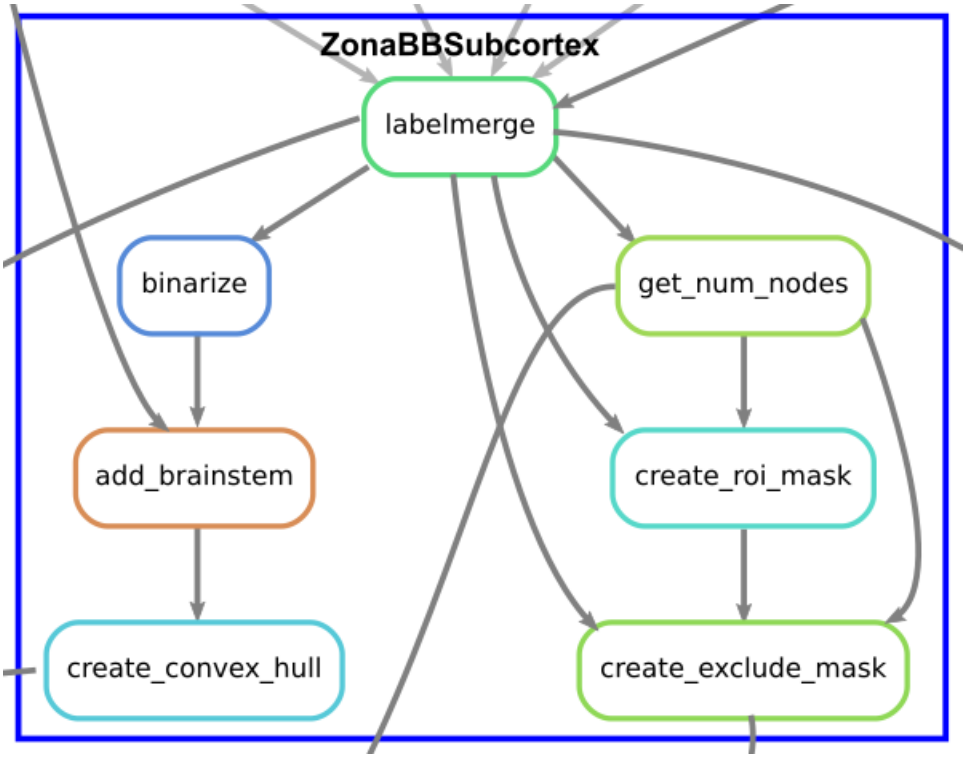
Segmentation of the thalamus into its subnuclei is performed via the Freesurfer script. As the output from this segmentation is in the Freesurfer file format (.mgz), a conversion to nifti (.nii.gz) is performed. Once converted, a transformation is computed to align the segmentations from the Freesurfer space to the subject's native space. The computed transformation is applied to the segmentations and used for further downstream processing.



Zona BB Subcortex

SCATTR also uses labelmerge to combine segmentations from varying sources. This allows for the previously segmented thalamic nuclei to be combined with an atlas of other subcortical structures. This newly combined atlas is used for two separate purposes:

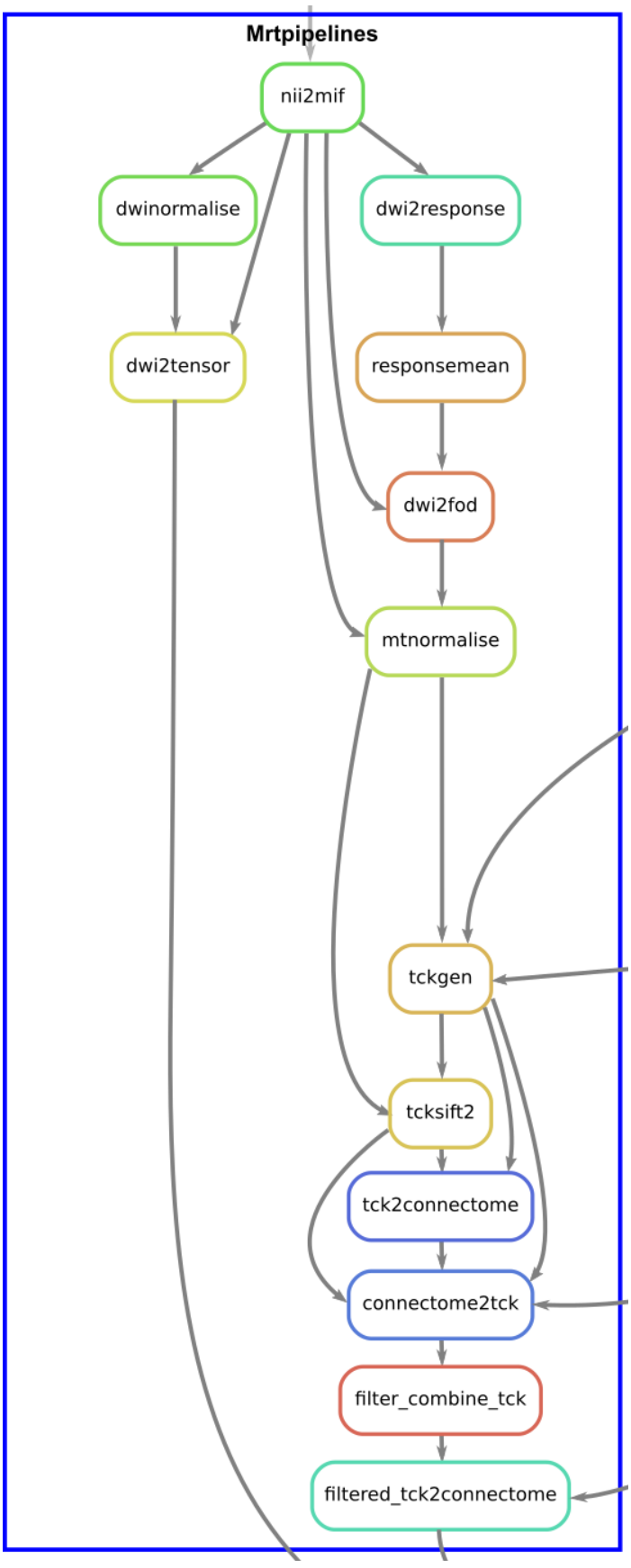
1. To individual binarized masks for the structures of interest. This is used to identify connections terminating within these structures.
2. To create a convex hull (including the brainstem) allow for identification of only those connections within the larger region of interest (e.g. subcortical region around the subcortical structures)



Mrtpipelines

MRtrix3 is used in the workflow for tractography-related processing. These rules first convert the necessary nifti images to the MRtrix file format, `.mif`. Similar to the subcortex workflow, two different processes are performed:

1. Diffusion tensor imaging is performed to calculate quantitative maps of fractional anisotropy, mean diffusivity, radial diffusvity and axial diffusivity.
2. Individual subject response functions for the three tissue types, white matter, grey matter, and corticospinal fluid, are first estimated, and an average response function is then computed. Alternatively, a pre-computed response function can be supplied to be used in the downstream rules. From the average response functions, fibre orientation distribution (FOD) maps are computed for individual subjects. FOD maps are normalized to correct for the effects of (residual) intensity inhomogeneties. Once normalized, whole-brain tractogaphy is computed from the FOD maps, before spherical-deconvolutional filtering of tractograms (`tcksift2`) is performed to weight the tractogram to match the underlying diffusion signal. Connections between specific structures of interest are identified, and further filtered to retain only those passing through the white matter (`tck2connectome`, `connectome2tck`, `filter_combine_tck`, and `filtered_tck2connectome`).


Mrtpipelines
nii2mif
dwinormalise
dwi2response
dwi2tensor
responsemean
dwi2fod
mtnormalise
tckgen
tcksift2
tck2connectome
connectome2tck
filter_combine_tck
filtered_tck2connectome

3.8 Output Files

After running the workflow, the `/path/to/output/dir` folder will contain a hidden `.logs` folder for troubleshooting, as well as additional folders associated with the tools that were used to generate files (e.g. `freesurfer`, `labelmerge`), but for most purposes, all the primary outputs of interest will be in the `mrtrix` directory with the following structure:

```
mrtrix/
└── sub-{subject}
    ├── dti
    ├── dwi
    ├── response
    └── tractography
```

Briefly, `dti` contains processed diffusion tensor images along with quantitative maps (e.g. `FA`, `MD`, etc.), `dwi` contains images that were converted from Nifti (`.nii.gz`) to the MRtrix imaging format (`.mif`), `response` contains subject-specific response functions, and `tractography` contains tractogram files.

3.8.1 dti

This folder contains processed diffusion tensor images (DTI) that were used to compute quantitative maps (stored in the MRtrix imaging format - `.mif`), also found within the folder:

```
sub-{subject}
└── dti
    ├── sub-{subject}_model-dti_ad.mif
    ├── sub-{subject}_model-dti_fa.mif
    ├── sub-{subject}_model-dti_md.mif
    ├── sub-{subject}_model-dti_rd.mif
    └── sub-{subject}_model-dti_tensor.mif
```

As per the BIDS extension proposal, `model-dti` denotes to the DTI model used to process the data, and the suffix (`ad`, `fa`, `md`, `rd`, `tensor`) describes the imaging data.

3.8.2 dwi

This folder contains the input diffusion weighted image (dwi) and brain mask converted from Nifti (`.nii.gz`) to the MRtrix imaging format (`.mif`) used in downstream processing. It also contains an intensity normalized dwi image used for DTI:

```
sub-{subject}
└── dwi
    ├── sub-{subject}_desc-brain_mask.mif
    ├── sub-{subject}_desc-normalized_dwi.mif
    └── sub-{subject}_dwi.mif
```

3.8.3 response

This folder contains the subject-specific response functions that were estimated from the input diffusion data, as well as both the normalized and unnormalized versions of the estimated fibre orientation distribution (FOD) maps. Both response functions and FOD maps are estimated for the three different tissue types: white matter (wm), grey matter (gm), and corticospinal fluid (csf):

```
sub-{subject}
└── response
    ├── sub-{subject}_desc-csf_model-csd_fod.mif
    ├── sub-{subject}_desc-csf_model-csd_fodNormalized.mif
    ├── sub-{subject}_desc-csf_response.txt
    ├── sub-{subject}_desc-gm_model-csd_fod.mif
    ├── sub-{subject}_desc-gm_model-csd_fodNormalized.mif
    ├── sub-{subject}_desc-gm_response.txt
    ├── sub-{subject}_desc-wm_model-csd_fod.mif
    ├── sub-{subject}_desc-wm_model-csd_fodNormalized.mif
    └── sub-{subject}_desc-wm_response.txt
```

3.8.4 tractography

This folder contains generated tractogram data and associated files describing how streamlines connect different subcortical structures:

```
sub-{subject}
└── tractography
    ├── sub-{subject}_desc-filteredsubcortical_nodeAssignment.txt
    ├── sub-{subject}_desc-filteredsubcortical_nodeWeights.csv
    ├── sub-{subject}_desc-filteredsubcortical_tractography.tck
    ├── sub-{subject}_desc-iFOD2_muCoefficient
    ├── sub-{subject}_desc-iFOD2_tckWeights.txt
    ├── sub-{subject}_desc-iFOD2_tractography.tck
    ├── sub-{subject}_desc-subcortical_nodeAssignment.txt
    ├── sub-{subject}_desc-subcortical_nodeWeights.csv
```

The `desc-filteredsubcortical` entity pair describes associated tractogram files that have been filtered to only pass through WM and connect two GM structures, while the `desc-subcortical` entity pair denotes files associated with the unfiltered subcortical connectome. Whole-brain tractogram associated files are denoted by `desc-iFOD2`, which describes the tractography algorithm used to perform tractograpy.

Note: As the BIDS extension proposal has not been finalized, the naming of such files may be subject to change

3.8.5 Additional Files

The top-level `/path/to/output/dir` contains additional files / folders:

```
/path/to/output/dir
├── ...
├── config
├── .logs
├── .snakebids
└── .snakemake
```

The `config` folder, along with the hidden `.snakebids` and `.snakemake` folders contain a record of the code and parameters used, and paths to the inputs.

Workflow steps that write logs to file are stored in the hidden `.logs` subfolder, with the file names based on the tools used (e.g. `mrtrix`) and rule wildcards (e.g. `subject`).

If the app is run in workflow mode (`--workflow-mode` / `-W`), which enables direct use of the Snakemake CLI to run scattr, output folders (e.g. `work`) will be placed in a `results` folder.

3.9 Visualization

3.9.1 MRview

MRview is a powerful viewer distributed with MRtrix that works well with both volume (e.g. `.nii`, `.nii.gz`) and tractography (e.g. `.tck`) data. It may take a while to load and navigate large tractography files.

Tips and tricks:

- Consider extracting a particular connection passing through two anatomical structures to view instead of the full tractogram. This can be done with the tractogram data, and the structures of interest:

```
tckedit -include structure1_roi.nii.gz -include structure2_roi.nii.gz in_tractogram.tck␣
↪out_tractogram.tck
```

- Extracting a subset number of streamlines from tracts (e.g. 10K):

```
tckedit in_tractogram.tck out_tractogram.tck -number 1000
```

Note: MRtrix has a number of different options for manipulating tractograms which can aid the ability to visualize such tracts. Take a look through the `tckedit` documentation as well as both the `tck2connectome` and `connectome2tck` documentation.

3.9.2 ITK-SNAP

ITK-SNAP is a lightweight tool that is able to quickly open volumes and is ideal for viewing segmentation images (`_dseg.nii.gz`). Segmentations can be loaded as overlays and ITK-SNAP can create a 3D rendering of the contours of each label.

3.10 Contributing to SCATTR

SCATTR python package dependencies are managed with Poetry (`v1.2.0`+), which you will need installed on your machine. You can find installation instructions on the Poetry website.

SCATTR also has a few dependencies outside of python, including popular neuroimaging packages like `ANTs`, `Freesurfer`, `MRtrix3`, and others. We **strongly** recommend using SCATTR with the `--use-singularity` flag, which will pull and use the required containers, unless you are comfortable using installing and using all of these tools yourself.

Note: These instructions are only recommended if you are making changes to the SCATTR codebase and committing these back to the repository, or if you are using Snakemake's cluster execution profiles. If not, it is easier to run SCATTR using the packaged singularity container (e.g. `docker://khanlab/scattr:latest`).

3.10.1 Setup the development environment

Clone the repository and install all dependencies (including `dev`) with poetry:

```
git clone https://github.com/khanlab/scattr.git
cd scattr
poetry install --with dev
```

Poetry will automatically create a virtual environment. To customize where these virtual environments are stored, see the poetry docs here

Then, you can run SCATTR with:

```
poetry run scattr
```

or you can activate a virtual environment shell and run SCATTR directly:

```
poetry shell
scattr
```

You can exit the poetry shell with `exit`

3.10.2 Running and fixing code format quality

SCATTR uses poethepoet as a task runner. You can see what commands are available by running:

```
poetry run poe
```

We use a a few tools, including `black`, `flake8`, `isort`, `snakefmt`, and `yamlfix` to ensure formatting and style of our codebase is consistent. There are two task runners you can use to check and fix your code, which can be invoked with:

```
poetry run poe quality-check
poetry run poe quality
```

Note: If you are in a poetry shell, you do not need to prepend `poetry run` to the command.

3.10.3 Dry-run / testing your workflow

Using Snakemake's dry-run option (`--dry-run`/`-n`) is an easy way to verify any changes made to the workflow are working direcctly. The `test/data` folder contains a *fake* BIDS dataset (i.e. dataset with zero-sized files) that is useful for verifying different aspects of the workflow. These dry-run tests are part of the automated Github actions that are run for every commit.

You can invoke the pre-configured task via poethepoet to perform a dry-run:

```
poetry run poe test
```

This performs a number of tests, involving different scenarios in which a user may use SCATTR.