HippUnfold Documentation

HippUnfold Development Team

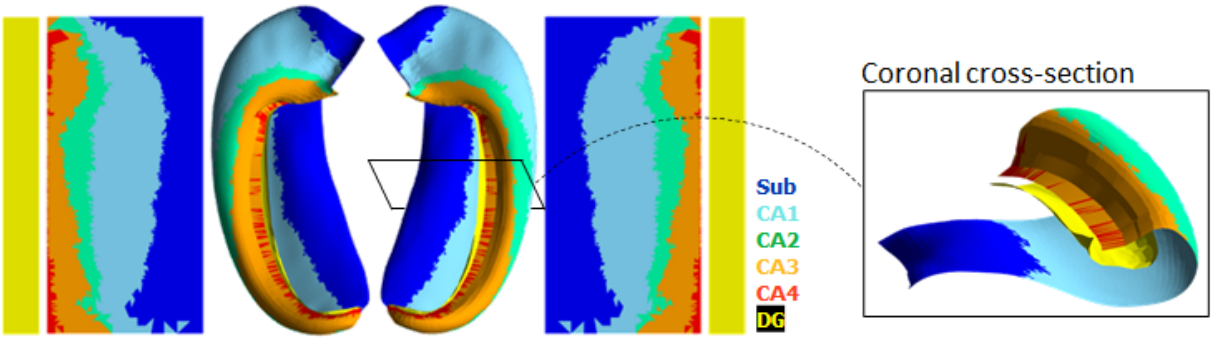
Apr 25, 2024

GETTING STARTED

Hipp Unfold

Full Documentation: here

This tool aims to automatically model the topological folding structure of the human hippocampus, and computationally unfold it.



This is especially useful for:

- Visualization
- Topologically-constrained intersubject registration
- Parcellation (ie. registration to an unfolded atlas)
- Morphometry (eg. thickness, surface area, curvature, and gyrification measures)
- Quantitative mapping (eg. map your qT1 MRI data to a midthickness surface; extract laminar profiles perpendicular to this surface)

CHAPTER

ONE

NEW: VERSION 1.3.X RELEASE

Major changes include the addition of unfolded space registration to a reference atlas harmonized across seven ground-truth histology samples. This method allows shifting in unfolded space, providing even better intersubject alignment.

Note: this replaces the default workflow, however you can revert to the legacy workflow, disabling unfolded space registration, by setting `--atlas bigbrain` *or* `--no-unfolded-reg`

Read more in our manuscript

Also the ability to specify a new **experimental** UNet model that is contrast-agnostic using synthseg and trained using more detailed segmentations. This generally produces more detailed results but has not been extensively tested yet.

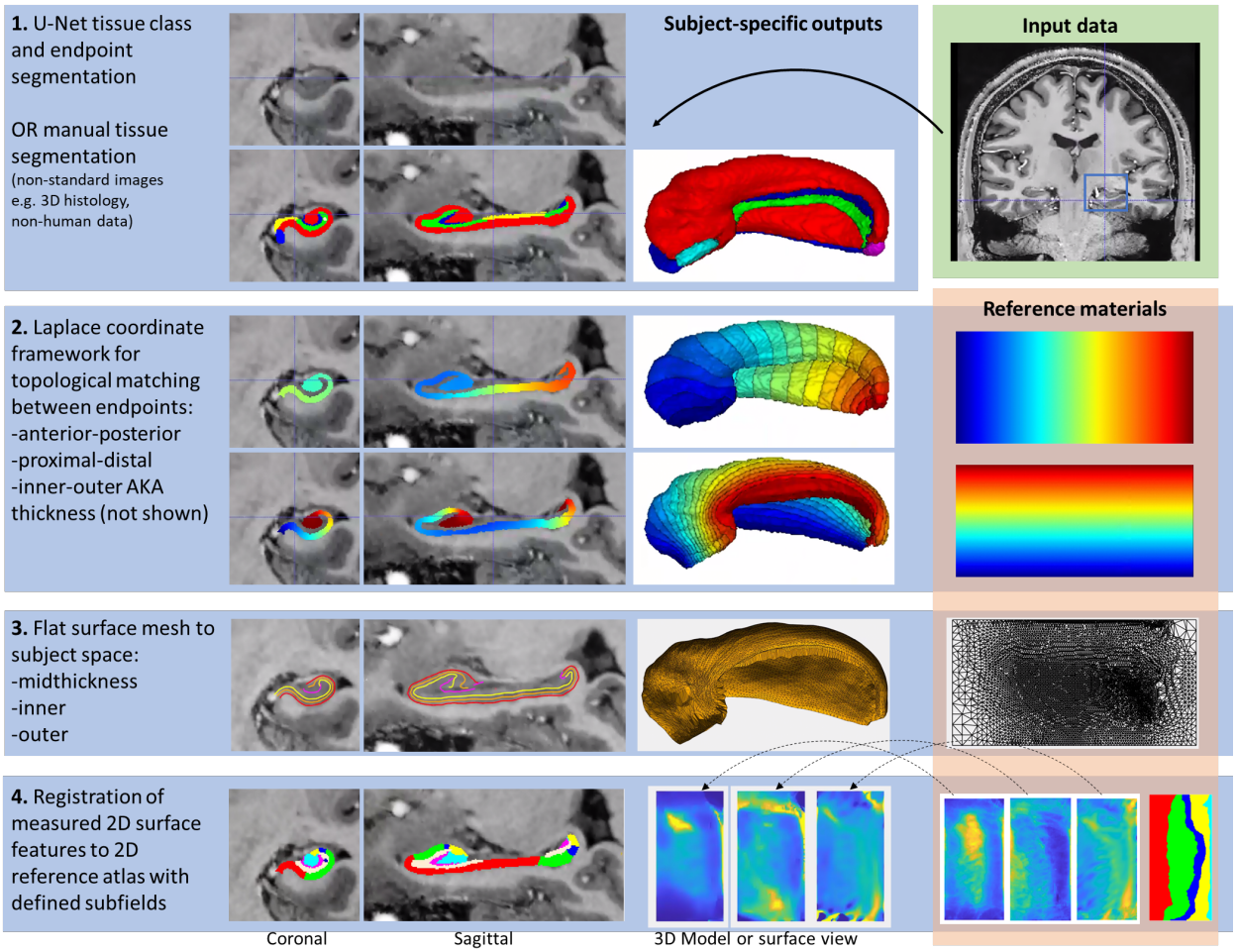
Note: Docker containers for version 1.3.x and above do not come pre-shipped with nnU-net models (and are accordingly more lightweight!) - models are downloaded automatically when running, but please see the FAQ for more information!

CHAPTER

TWO

WORKFLOW

The overall workflow can be summarized in the following steps:



For more information, see **Full Documentation:** here

CHAPTER

THREE

ADDITIONAL TOOLS

For plotting, mapping fMRI, DWI or other data, and manipulating surfaces, see here

For statistical testing (spin tests) in unfolded space, see here

CHAPTER

FOUR

PUBLICATIONS

4.1 HippUnfold methods paper

- DeKraker, J., Haast, R. A., Yousif, M. D., Karat, B., Lau, J. C., Köhler, S., & Khan, A. R. (2022). Automated hippocampal unfolding for morphometry and subfield segmentation with HippUnfold. Elife, 11, e77945. link
  - **Please cite this if you use any version of HippUnfold)**

4.2 Unfolded space registration and multihist7 atlas

- DeKraker Jordan, Palomero-Gallagher Nicola, Kedo Olga, Ladbon-Bernasconi Neda, Muenzing Sascha E.A., Axer Markus, Amunts Katrin, Khan Ali R., Bernhardt Boris, Evans Alan C. (2023) Evaluation of surface-based hippocampal registration using ground-truth subfield definitions eLife 12:RP88404 link
  - **Please cite this if you use HippUnfold version >= 1.3.0)**

4.3 Commentary on surface-based hippocampal segmentation

- DeKraker J, Köhler S, Khan AR. Surface-based hippocampal subfield segmentation. Trends Neurosci. 2021 Nov;44(11):856-863. doi: 10.1016/j.tins.2021.06.005. Epub 2021 Jul 22. PMID: 34304910. link

4.4 Related papers

- DeKraker J, Ferko KM, Lau JC, Köhler S, Khan AR. Unfolding the hippocampus: An intrinsic coordinate system for subfield segmentations and quantitative mapping. Neuroimage. 2018 Feb 15;167:408-418. doi: 10.1016/j.neuroimage.2017.11.054. Epub 2017 Nov 23. PMID: 29175494. link
- DeKraker J, Lau JC, Ferko KM, Khan AR, Köhler S. Hippocampal subfields revealed through unfolding and unsupervised clustering of laminar and morphological features in 3D BigBrain. Neuroimage. 2020 Feb 1;206:116328. doi: 10.1016/j.neuroimage.2019.116328. Epub 2019 Nov 1. PMID: 31682982. link
- Karat BG, DeKraker J, Hussain U, Köhler S, Khan AR. Mapping the macrostructure and microstructure of the in vivo human hippocampus using diffusion MRI. Hum Brain Mapp. 2023 Nov;44(16):5485-5503. Epub 2023 Aug 24. PMID: 37615057; PMCID: PMC10543110.link

4.4.1 Installation

BIDS App for Hippocampal AutoTop (automated hippocampal unfolding and subfield segmentation)

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 HippUnfold.
- GPU not required
- Note: Apple M1 is currently **not supported**. We don't have a Docker arm64 container yet, and hippunfold is unusably slow with the emulated amd64 container.

Notes:

- Inputs to Hippunfold should typically be a BIDS dataset including T1w images or T2w images. Higher-resolution data are preferred (<= 0.8mm) but the pipeline will still work with 1mm T1w images. See Tutorials.
- Other 3D imaging modalities (eg. ex-vivo MRI, 3D histology, etc.) can be used, but may require manual tissue segmentation as the current workflow relies on U-net segmentation trained only on common MRI modalities.

Comparison of methods for running HippUnfold

There are several different ways of running HippUnfold. In order of increasing complexity/flexibility, we have:

1. CBRAIN Web-based Platform
2. Singularity Container on Linux
3. Docker Container on Windows/Mac (Intel)/Linux
4. Python Environment with Singularity Dependencies

CBRAIN Web-based Platform

HippUnfold is available on the CBRAIN platform, a web-based platform for batch high-performance computing that is free for researchers.

Pros:

- No software installation required
- Fully point and click interface (no CLI)
- Can perform batch-processing

Cons:

- Must upload data for processing
- Limited command-line options exposed
- Cannot edit code

Docker on Windows/Mac (Intel)/Linux

The HippUnfold BIDS App is available on a DockerHub as versioned releases and development branches.

Pros:

- Compatible with non-Linux systems
- All dependencies+models (* See Note 1) in a single container

Cons:

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

Singularity Container

The same docker container can also be used with Singularity (now Apptainer). Instructions can be found below.

Pros:

- All dependencies+models (* See Note 1) in a single container
- Container stored as a single file (.sif)

Cons:

- Compatible on shared systems with Singularity installed
- Cannot use Snakemake cluster execution profiles
- Cannot edit code

Python Environment with Singularity Dependencies

Instructions for this can be found in the **Contributing** documentation page.

Pros:

- Complete flexibility to modify code
- External (python and non-python) dependencies as Singularity containers

Cons:

- Must use Python virtual environment
- Only compatible on Linux systems with Singularity for external dependencies

Note 1:

As of version 1.3.0 of HippUnfold, containers are no longer shipped with all the models, and the models are downloaded as part of the workflow. By default, models are placed in `~/.cache/hippunfold` unless you set the `HIPPUNFOLD_CACHE_DIR` environment variable. See Deep learning nnU-net model files for more information.

4.4.2 Running HippUnfold with Docker on Windows

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

First time setup

Open your Windows Command Prompt by clicking the bottom left `Windows` button and type `cmd` followed by `Enter`. This is where you will enter your HippUnfold commands. Feel free to make a new directory with `mkdir` or move to a directory you would like to work out of with `cd`, and for this example we will work from:

```
cd c:\Users\jordan\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 then be run on many datasets):

```
docker pull khanlab/hippunfold:latest
```

Run HippUnfold without any arguments to print the short help:

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

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

```
docker run -it --rm khanlab/hippunfold:latest -h
```

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

```
docker run -it --rm khanlab/hippunfold:latest --help-snakemake
```

Running an example

Download and extract a single-subject BIDS dataset for this test from hippunfold_test_data.tar. Here we will also assume you chose to save and extract to the directory `c:\Users\jordan\Downloads\`.

This contains a `ds002168/` directory with a single subject, that has a both T1w and T2w images.

```
ds002168/
├── dataset_description.json
├── README.md
└── sub-1425
    └── anat
        ├── sub-1425_T1w.json
        ├── sub-1425_T1w.nii.gz
        ├── sub-1425_T2w.json
        └── sub-1425_T2w.nii.gz

2 directories, 6 files
```

Now let's run HippUnfold on the test dataset. Docker will need read/write access to the input and output directories, respectively. This is achieved with the `-v` flag. This 'binds' or 'mounts' a directory to a new directory inside the container.

```
docker run -it --rm -v c:\Users\jordan\Downloads\ds002168:/bids -v c:\Users\jordan\
↪Downloads\ds002168_hippunfold:/output khanlab/hippunfold:latest /bids /output␣
↪participant --modality T1w -n
```

Explanation:

`-v c:\Users\jordan\Downloads\ds002168:/bids` tells Docker to mount the directory `c:\Users\jordan\Downloads\ds002168` into a new directory inside the container named `/bids`. We then do the same things for our output directory named `ds002168_hippunfold`, which we mount to `/output` inside the container. These arguments are not specific to HippUnfold but rather are general ways to use Docker. You may want to familiarize yourself with Docker options.

Everything after we specified the container (`khanlab/hippunfold:latest`) are arguments to HippUnfold itself. The first of these arguments (as with any BIDS App) are the input directory (`/bids`), the output directory (`/output`), and then the analysis level (`participant`). The `participant` analysis level is used in HippUnfold for performing the segmentation, unfolding, and any participant-level processing. The `group` analysis is used to combine subfield volumes across subjects into a single tsv file. The `--modality` flag is also required, and describes which image we use for segmentation. Here we used the T1w image. We also used the `--dry-run/-n` option to just print out what would run, without actually running anything.

When you run the above command, a long listing will print out, describing all the rules that will be run. 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 HippUnfold how many cores to use. Using `--cores 8` means that HippUnfold will only make use of 8 cores at most. Generally speaking you should use `--cores all`, so it can make maximal use of all the CPU cores it has access to on your system. This is especially useful if you are running multiple subjects.

Running the following command (hippunfold on a single subject) may take ~30 minutes if you have 8 cores, shorter if you have more cores, but could be much longer (several hours) if you only have a single core.

```
docker run -it --rm -v c:\Users\jordan\Downloads\ds002168:/bids -v c:\Users\jordan\
↪Downloads\ds002168_hippunfold:/output khanlab/hippunfold:latest /bids /output␣
↪participant --modality T1w -p --cores all
```

After this completes, you should have a `ds002168_hippunfold` directory with outputs for the one subject.

Exploring different options

If you alternatively want to run HippUnfold using a different modality, e.g. the high-resolution T2w image in the BIDS test dataset, you can use the `--modality T2w` option. In this case, since the T2w image in the test dataset has a limited FOV, we should also make use of the `--t1-reg-template` command-line option, which will make use of the T1w image for template registration, since a limited FOV T2w template does not exist.

```
docker run -it --rm -v c:\Users\jordan\Downloads\ds002168:/bids -v c:\Users\jordan\
↪Downloads\ds002168_hippunfold_t2w:/output khanlab/hippunfold:latest /bids /output␣
↪participant --modality T2w --t1-reg-template -p --cores all
```

Note that if you run with a different modality, you should use a separate output directory, since some of the files would be overwritten if not.

4.4.3 Running HippUnfold with Singularity

Pre-requisities:

1. Singularity or Apptainer is installed on your system. For more info, see the detailed apptainer install instructions.
2. The following command-line tools are installed:
   - wget
   - tar
3. Sufficient disk-space
   - in your `/tmp` folder (>30GB) to build the container (not needed for dropbox download)
   - in your working folder to store the container (~15GB)
   - for HippUnfold outputs (~4GB per subject)
4. Sufficient CPU and memory - the more you have, the faster it will run, but we recommend at least 8 CPU cores and 16GB memory.

First time setup

Pull the container:

```
singularity pull khanlab_hippunfold_latest.sif docker://khanlab/hippunfold:latest
```

Run HippUnfold without any arguments to print the short help:

```
singularity run -e khanlab_hippunfold_latest.sif
```

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

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

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

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

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. Note, you can change these locations by setting environment variables, however, using a network file system for the folders may result in poor performance and/or errors e.g.:

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

Running an example

Download and extract a single-subject BIDS dataset for this test:

```
wget https://www.dropbox.com/s/mdbmpmmq6fi8sk0/hippunfold_test_data.tar
tar -xvf hippunfold_test_data.tar
```

This will create a `ds002168/` folder with a single subject, that has a both T1w and T2w images:

```
ds002168/
├── dataset_description.json
├── README.md
└── sub-1425
    └── anat
        ├── sub-1425_T1w.json
        ├── sub-1425_T1w.nii.gz
        ├── sub-1425_T2w.json
        └── sub-1425_T2w.nii.gz

2 directories, 6 files
```

Now let's run HippUnfold.

```
singularity run -e khanlab_hippunfold_latest.sif ds002168 ds002168_hippunfold␣
↪participant -n --modality T1w
```

Explanation:

Everything prior to the container (`khanlab_hippunfold_latest.sif`) are arguments to singularity, and after are to HippUnfold itself. The first three arguments to HippUnfold (as with any BIDS App) are the input folder (`ds002168`), the output folder (`ds002168_hippunfold`), and then the analysis level (`participant`). The `participant` analysis level is used in HippUnfold for performing the segmentation, unfolding, and any participant-level processing. The `group` analysis is used to combine subfield volumes across subjects into a single tsv file. The `--modality` flag is a required argument, and describes what image we use for segmentation. Here we used the T1w image. We also used the `--dry-run/-n` option to just print out what would run, without actually running anything.

When you run the above command, a long listing will print out, describing all the rules that will be run. This is a long listing, and you can better appreciate it with the `less` tool. We can also have the shell command used for each rule printed to screen using the `-p` Snakemake option:

```
singularity run -e khanlab_hippunfold_latest.sif ds002168 ds002168_hippunfold␣
↪participant -np --modality T1w | less
```

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 HippUnfold how many cores to use. Using `--cores 8` means that HippUnfold will only make use of 8 cores at most. Generally speaking you should use `--cores all`, so it can make maximal use of all the CPU cores it has access to on your system. This is especially useful if you are running multiple subjects.

Running the following command (hippunfold on a single subject) may take ~30 minutes if you have 8 cores, shorter if you have more cores, but could be much longer (several hours) if you only have a single core.

```
singularity run -e khanlab_hippunfold_latest.sif ds002168 ds002168_hippunfold␣
↪participant -p --cores all --modality T1w
```

Note that you may need to adjust your Singularity options to ensure the container can read and write to yout 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 should have a `ds002168_hippunfold` folder with outputs for the one subject.

Exploring different options

If you alternatively want to run HippUnfold using a different modality, e.g. the high-resolution T2w image in the BIDS test dataset, you can use the `--modality T2w` option. In this case, since the T2w image in the test dataset has a limited FOV, we should also make use of the `--t1-reg-template` command-line option, which will make use of the T1w image for template registration, since a limited FOV T2w template does not exist.

```
singularity run -e khanlab_hippunfold_latest.sif ds002168 ds002168_hippunfold_t2w␣
↪participant --modality T2w --t1-reg-template -p --cores all
```

Note that if you run with a different modality, you should use a separate output folder, since some of the files would be overwritten if not.

4.4.4 Running HippUnfold with a Vagrant VM

This option uses Vagrant to create a virtual machine that has Linux and Singularity installed. This allows you to use Singularity to run HippUnfold from a clean environment, whether you are running Linux, Mac or Windows (since all three are supported by Vagrant). Note: VirtualBox does the actual virtualization in this example, but Vagrant provides an easy and reproducible way to create and connect to the VMs (as shown below).

Install VirtualBox and Vagrant

The example below uses Vagrant and VirtualBox installed on Ubuntu 20.04.

The Vagrant install instructions describe what you need to do to install on Mac, Windows or Linux.

Vagrant must use a **provider** for the actual virtualization. The instructions here assume you are using VirtualBox for this, since it is free and easy to use, but in principle should work with any virtualization provider. The VirtualBox downloads page can guide you through the process of installing it on your system (Mac, Windows, Linux supported).

Create a Vagrant Box

Once you have Vagrant and VirtualBox installed, the following screencast demonstrates how you can setup a Box with Singularity pre-loaded on it. The main steps are to 1) create a Vagrantfile, 2) start the box using `vagrant up`, and 3) connect to it using `vagrant ssh`.

Note: These screencasts are more than just videos, they are asciinema recordings – you can pause them and then copy-paste text directly from the asciinema cast!

This is the `Vagrantfile` used in the video, for quick reference:

```
Vagrant.configure("2") do |config|
  config.vm.box = "sylabs/singularity-3.7-ubuntu-bionic64"
  config.vm.provider "virtualbox" do |vb|
     vb.cpus = 8
     vb.memory = "8096"
   end
end
```

Download the test dataset

We are downloading the test dataset with the following command:

```
wget https://www.dropbox.com/s/mdbmpmmq6fi8sk0/hippunfold_test_data.tar
```

Download the HippUnfold container

We pull/build the container from DockerHub:

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

Run HippUnfold

This demonstrates the basic HippUnfold options, and how to perform a dry-run:

Finally, we can run HippUnfold using all the cores:

4.4.5 Command-line interface

HippUnfold Command-line interface

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

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

Snakebids helps build BIDS Apps with Snakemake

```
usage: hippunfold [-h]
                  [--participant_label PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]]
                  [--exclude_participant_label EXCLUDE_PARTICIPANT_LABEL [EXCLUDE_
↪PARTICIPANT_LABEL ...]]
                  [--version] --modality
                  {T1w,T2w,hippb500,segT1w,segT2w,cropseg}
                  [--template {CITI168,dHCP,MBMv2,MBMv3,CIVM}]
                  [--inject_template {upenn,MBMv2,MBMv3,CIVM}]
                  [--use-template-seg]
                  [--template_seg_smoothing_factor TEMPLATE_SEG_SMOOTHING_FACTOR]
                  [--derivatives DERIVATIVES] [--skip_preproc] [--skip_coreg]
                  [--skip_inject_template_labels]
                  [--inject-template-smoothing-factor INJECT_TEMPLATE_SMOOTHING_FACTOR]
                  [--rigid-reg-template] [--no-reg-template]
                  [--t1-reg-template] [--crop_native_res CROP_NATIVE_RES]
                  [--crop_native_box CROP_NATIVE_BOX]
                  [--atlas {bigbrain,magdeburg,freesurfer,multihist7} [{bigbrain,
↪magdeburg,freesurfer,multihist7} ...]]
                  [--no_unfolded_reg] [--generate_myelin_map] [--use-gpu]
                  [--nnunet_enable_tta]
                  [--output-spaces {native,T1w} [{native,T1w} ...]]
                  [--output-density {0p5mm,1mm,2mm,unfoldiso} [{0p5mm,1mm,2mm,unfoldiso}␣
↪...]]
                  [--hemi {L,R} [{L,R} ...]]
                  [--laminar-coords-method {laplace,equivolume}]
                  [--autotop-labels {hipp,dentate} [{hipp,dentate} ...]]
                  [--keep-work]
                  [--force_nnunet_model {T1w,T2w,T1T2w,b1000,trimodal,hippb500,
↪neonateT1w,synthseg_v0.1,synthseg_v0.2,neonateT1w_v2}]
                  [--filter-T2w ENTITY=VALUE [ENTITY=VALUE ...]]
                  [--filter-hippb500 ENTITY=VALUE [ENTITY=VALUE ...]]
                  [--filter-T1w ENTITY=VALUE [ENTITY=VALUE ...]]
                  [--filter-seg ENTITY=VALUE [ENTITY=VALUE ...]]
                  [--filter-cropseg ENTITY=VALUE [ENTITY=VALUE ...]]
                  [--wildcards-T2w WILDCARD [WILDCARD ...]]
                  [--wildcards-hippb500 WILDCARD [WILDCARD ...]]
                  [--wildcards-T1w WILDCARD [WILDCARD ...]]
                  [--wildcards-seg WILDCARD [WILDCARD ...]]
                  [--wildcards-cropseg WILDCARD [WILDCARD ...]]
                  [--path-T2w PATH] [--path-hippb500 PATH] [--path-T1w PATH]
                  [--path-seg PATH] [--path-cropseg PATH]
                  [--pybidsdb-dir PATH] [--pybidsdb-reset] [--help-snakemake]
```

(continues on next page)

(continued from previous page)

```
                [--force-output]
                bids_dir output_dir {participant,group}
```

Positional Arguments

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, group

Level of the analysis that will be performed.

Named Arguments

--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 should be analyzed. Multiple participants can be specified with a space separated 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 should be analyzed. Multiple participants can be specified with a space separated list.

--version Print the version of HippUnfold

--modality Possible choices: T1w, T2w, hippb500, segT1w, segT2w, cropseg

Type of image to run hippunfold on. Modality prefixed with seg will import an existing (manual) hippocampal tissue segmentation from that space, instead of running neural network (default: None)

--template Possible choices: CITI168, dHCP, MBMv2, MBMv3, CIVM

Set the template to use for registration to coronal oblique (and optionally for template-based segmentation if –use-template-seg is enabled). CITI168 is for adult human data, dHCP is for neonatal human data, MBMv2 is for ex vivo marmoset data, MBMv3 is for in vivo marmoset data, and CIVM is for in vivo macaque data. (default: "CITI168")

Default: "CITI168"

--inject_template, --inject-template Possible choices: upenn, MBMv2, MBMv3, CIVM

Set the template to use for shape injection. (default: "upenn")

Default: "upenn"

--use-template-seg, --use_template_seg Use template-based segmentation for hippocampal tissue *instead of* nnUnet and shape injection. This is only to be used if nnUnet models are not trained for the data you are using, e.g. for non-human primate data with the MBMv2 (ex vivo marmoset), MBMv3 (in vivo marmoset), or CIVM (ex vivo macaque) template. (default: False)

Default: False

--template_seg_smoothing_factor, --template-seg-smoothing-factor Scales the default smoothing sigma for gradient and warp in greedy registration for template-based segmentation. Using a value higher than 1 will use result in a smoother warp. (default: 2.0)

Default: 2.0

--derivatives Path to the derivatives folder (e.g. for finding manual segs) (default: False)

Default: False

--skip_preproc, --skip-preproc Set this flag if your inputs (e.g. T2w, dwi) are already pre-processed (default: False)

Default: False

--skip_coreg, --skip-coreg Set this flag if your inputs (e.g. T2w, dwi) are already registered to T1w space (default: False)

Default: False

--skip_inject_template_labels, --skip-inject-template-labels Set this flag to skip post-processing template injection into CNN segmentation. Note this will disable generation of DG surfaces. (default: False)

Default: False

--inject-template-smoothing-factor, --inject_template_smoothing_factor Scales the default smoothing sigma for gradient and warp in template shape injection. Using a value higher than 1 will use result in a smoother warp, and greater capacity to patch larger holes in segmentations. Try setting to 2 if nnunet segmentations have large holes. Note: the better solution is to re-train network on the data you are using (default: 1.0)

Default: 1.0

--rigid-reg-template, --rigid_reg_template Use rigid instead of affine for registration to template. Try this if your images are reduced FOV (default: False)

Default: False

--no-reg-template, --no_reg_template Use if input data is already in space-CITI168 (default: False)

Default: False

--t1-reg-template, --t1_reg_template Use T1w to register to template space, instead of the segmentation modality. Note: this was the default behavior prior to v1.0.0. (default: False)

Default: False

--crop_native_res, --crop-native-res Sets the bounding box resolution for the crop native (e.g. cropT1w space). Under the hood, hippUnfold operates at higher resolution than the native image, so this tries to preserve some of that detail. (default: "0.2x0.2x0.2mm")

Default: "0.2x0.2x0.2mm"

--crop_native_box, --crop-native-box Sets the bounding box size for the crop native (e.g. cropT1w space). Make this larger if your hippocampi in crop{T1w,T2w} space are getting cut-off. This must be in voxels (vox) not millimeters (mm). (default: "256x256x256vox")

Default: "256x256x256vox"

--atlas Possible choices: bigbrain, magdeburg, freesurfer, multihist7

Select the atlas (unfolded space) to use for subfield labels. (default: ['multihist7'])

Default: ['multihist7']

--no_unfolded_reg, --no-unfolded-reg Do not perform unfolded space (2D) registration based on thickness, curvature, and gyrification for closer alignment to the reference atlas. NOTE: only multihist7 has these features currently, so this unfolded_reg is automatically skipped if a different atlas is chosen. (default: False)

Default: False

--generate_myelin_map, --generate-myelin-map Generate myelin map using T1w divided by T2w, and map to surface with ribbon approach. Requires both T1w and T2w images to be present. (default: False)

Default: False

--use-gpu, --use_gpu Enable gpu for inference by setting resource gpus=1 in run_inference rule (default: False)

Default: False

--nnunet_enable_tta, --nnunet-enable-tta Enable test-time augmentation for nnU-net inference, slows down inference by 8x, but potentially increases accuracy (default: False)

Default: False

--output-spaces, --output_spaces Possible choices: native, T1w

Sets output spaces for results (default: ['native'])

Default: ['native']

--output-density, --output_density Possible choices: 0p5mm, 1mm, 2mm, unfoldiso

Sets the output vertex density for results. Options correspond to approximate vertex spacings of 0.5mm, 1.0mm, and 2.0mm, respectively, with the unfoldiso (32k hipp) vertices legacy option having unequal vertex spacing. (default: ['0p5mm'])

Default: ['0p5mm']

--hemi Possible choices: L, R

Hemisphere(s) to process (default: ['L', 'R'])

Default: ['L', 'R']

--laminar-coords-method, --laminar_coords_method Possible choices: laplace, equivolume

Method to use for laminar coordinates. Equivolume uses equivolumetric layering from Waehnert et al 2014 (Nighres implementation). (default: ['equivolume'])

Default: ['equivolume']

--autotop-labels, --autotop_labels Possible choices: hipp, dentate

Run hipp (CA + subiculum) alone or include dentate (default: ['hipp', 'dentate'])

Default: ['hipp', 'dentate']

--keep-work, --keep_work Keep work folder intact instead of archiving it for each subject (default: False)

Default: False

--force_nnunet_model, --force-nnunet-model Possible choices: T1w, T2w, T1T2w, b1000, trimodal, hippb500, neonateT1w, synthseg_v0.1, synthseg_v0.2, neonateT1w_v2

Force nnunet model to use (expert option). (default: False)

Default: False

--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

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

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

Default: False

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-T2w, --filter_T2w (default: suffix=T2w extension=.nii.gz datatype=anat invalid_filters=allow space=None)

--filter-hippb500, --filter_hippb500 (default: suffix=b500 extension=.nii.gz invalid_filters=allow datatype=dwi)

--filter-T1w, --filter_T1w (default: suffix=T1w extension=.nii.gz datatype=anat invalid_filters=allow space=None)

--filter-seg, --filter_seg (default: suffix=dseg extension=.nii.gz datatype=anat invalid_filters=allow)

--filter-cropseg, --filter_cropseg (default: suffix=dseg extension=.nii.gz datatype=anat invalid_filters=allow)

INPUT WILDCARDS

File path entities to use as wildcards in snakemake

--wildcards-T2w, --wildcards_T2w (default: subject session acquisition run)

--wildcards-hippb500, --wildcards_hippb500 (default: subject session)

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

--wildcards-seg, --wildcards_seg (default: subject session)

--wildcards-cropseg, --wildcards_cropseg (default: subject session hemi)

PATH OVERRIDE

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

--path-T2w, --path_T2w

--path-hippb500, --path_hippb500

--path-T1w, --path_T1w

--path-seg, --path_seg

--path-cropseg, --path_cropseg

Snakemake command-line interface

In addition to the above command-line arguments, Snakemake arguments are also be passed at the `hippunfold` command-line.

The most critical of these is the `--cores` or `-c` argument, which is a **required** argument for HippUnfold.

The complete list of Snakemake arguments are below, and mostly 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 `hippunfold --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 [RULE ...]]] [--snakefile FILE] [--cores [N]]
                 [--jobs [N]] [--local-cores N]
                 [--resources [NAME=INT [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 [NAME=INT ...]]]
                 [--preemption-default PREEMPTION_DEFAULT]
                 [--preemptible-rules PREEMPTIBLE_RULES [PREEMPTIBLE_RULES ...]]
                 [--config [KEY=VALUE [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 [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 [NAME=VALUE ...]]]
```

(continues on next page)

(continued from previous page)

```
                [--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} [{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 [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 ...]]
                [--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]
```

(continues on next page)

(continued from previous page)

```
                [--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 [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

4.4.6 Running HippUnfold on your data

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

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

Selecting the modality to use

The `--modality` option must be chosen when running HippUnfold, and it affects what U-net model will be used, and how the pre-processing will be performed on the images.

If you have sub-millimetric, isotropic, whole-brain T1w data, the `--modality T1w` option is recommended.

If you T2w data, you can use the `--modality T2w` option, however, you may need to also use the T1w data for template registration (`--t1-reg-template`), especially if you have a limited FOV. This is typically most robust as long as a full brain FOV T1w image is available. If this registration is still failing then it may be improved with the `--rigid-reg-template` flag.

For protocols employing high-resolution, b-value 500, hippocampal diffusion-weighted imaging, the `--modality hippb500` option can be used, and does not require registration to a template (providing your acquisition is axial and oblique to the hippocampus).

Specifying a manual segmentation (eg. `--modality segT1w`) expects to additionally find an input file with the suffix `_dseg` which should contain labels following the protocol outlined here. More details are provided on using manual segmentations on the following page.

Selecting and excluding subjects to process

By default, hippunfold will run on **all** the subjects in a dataset. If you want to run only on a subset of subjects, you can use the `--participant_label` flag, e.g. adding:

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

would run only on `sub-001`. You can add additional subjects by listing additional arguments to this option, e.g.:

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

runs for `sub-001` and `sub-001`.

Also, if you want to exclude a subject, you can use the `--exclude-participant-label` option.

Known limitations for BIDS parsing

HippUnfold 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 need to be consistent in terms of what optional BIDS entities exist in them. We can use the acquisition (`acq`) entity as an example. HippUnfold 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 examples of possible BIDS-compliant datasets can be found in hippunfold/test_data/.

Parsing Non-BIDS datasets with custom paths

Custom paths can be used to parse input datasets if the data are not in BIDS format, but still are uniquely identified by subject (or subject+session) identifiers. For example:

```
PATH_TO_nonBIDS_DIR/
└── s_001_T1w.nii.gz
└── s_001_T2SPACE.nii.gz
└── s_001_TSE.nii.gz
└── s_002_T1w.nii.gz
└── s_002_T2SPACE.nii.gz
└── s_002_TSE.nii.gz
...
```

This directory doesn't separate subjects into different folders or contain an `anat/` folder for structural images. However, we can still specify what subjects and images to use with subject `wildcards`. This is done by using the `--path-{modality}` options to specify the absolute location of the `nii.gz` files. Note that here, T2SPACE and TSE are both T2-weighted acquisitions, and can be captured by using the `--path-T2w` flag to specify exactly which of these file(s) to use as inputs. For example, the following command:

```
hippunfold - PATH_TO_OUTPUT_DIR participant \
--modality T2w \
--t1-reg-template \
--path_T1w PATH_TO_nonBIDS_DIR/s_{subject}_T1w.nii.gz \
--path_T2w PATH_TO_nonBIDS_DIR/s_{subject}_T2SPACE.nii.gz
```

will search for any files following the naming scheme and fill in `{subject}` IDs for any files it finds, using the T1w and T2SPACE images for `T1w` and `T2w` inputs

Prerequisities for using custom path parsing:

Not all non-BIDS datasets can be parsed, and may still need some reformatting or renaming.

Specifically:

- The subject (or subject/session) wildcard(s) can only contain letters or numbers, e.g. they cannot include underscores, hyphens, or spaces.
- The subject (or subject/session) wildcard(s) must be the only unique identifiers in the filenames.

For example, this datasets would be **ineligible**:

```
PATH_TO_nonBIDS_DIR/
└── s_2019-05-29_001_T1w.nii.gz
└── s_2019-05-29_001_T2SPACE.nii.gz
└── s_2019-05-29_001_TSE.nii.gz
└── s_2018-02-24_002_T1w.nii.gz
└── s_2018-02-24_002_T2SPACE.nii.gz
└── s_2018-02-24_002_TSE.nii.gz
...
```

You would need to rename/symlink your images to remove the additional unique date identifiers, or integrate it into thes subject wildcard, ensuring only letters and numbers appear in the wildcard, e.g.:

```
PATH_TO_nonBIDS_DIR/
└── s_20190529s001_T1w.nii.gz
└── s_20190529s001_T2SPACE.nii.gz
└── s_20190529s001_TSE.nii.gz
└── s_20180224s002_T1w.nii.gz
└── s_20180224s002_T2SPACE.nii.gz
└── s_20180224s002_TSE.nii.gz
...
```

4.4.7 Specialized scans

This tutorial will cover how HippUnfold can be applied to non-standard data including ex-vivo scans, super-high resolution data (eg. <0.3mm isotropic), non-MRI 3D imaging data, or scans where a corresponding whole-brain T1w image is not available.

We will show how the available flags can be adapted for these use-cases with several worked examples.

Case 1: super high resolution

In this example, we have only a limited field of view covering the hippocampus, and the resolution and contrast do not closely match the training data of HippUnfold (0.3-1.0mm isotropic T1w, T2w, or DWI data). This could be ex-vivo MRI data, or it could even be 3D microscopy data as in our recent 3D BigBrain publication. Thus we don't expect HippUnfold's inbuilt UNet to be successful in segmenting hippocampal tissue before unfolding, and we do not want to downsample our data to accommodate HippUnfold's usual UNet and unfolding workflow in `space-corobl` (which consists of 0.3mm isotropic resampling cropped coronally-oblique to the hippocampus).

This will require manual segmentation of hippocampal grey matter, SRLM, and neighbouring structures, though in the future we hope to include models trained with higher resolution data (and contrasts more common in ex-vivo scanning). This should be done according to the protocol outlined here or, more recently, the video example here. This manual segmentation file should have the `_dseg` suffix.

Here is an example of what the input directory might look like:

```
exvivo/
└── sub-001/
    ├── sub-001_hemi-R_desc-hippo_T2w.nii.gz
    └── sub-001_hemi-R_desc-hippo_dseg.nii.gz
```

This can be unfolded with the command:

```
hippunfold . PATH_TO_OUTPUT_DIR participant --modality cropseg \
--path_cropseg exvivo/sub-{subject}/sub-{subject}_hemi-{hemi}_desc-hippo_dseg.nii.gz \
--hemi R --skip_inject_template_labels
```

Explanation: `--modality cropseg` informs HippUnfold that the input manual segmentation should not be resampled and UNet does not need to be run. Because of a limitation in bids parsing for the `hemi` entity, we need to use the generic path input, `--path_cropseg` in this case, making sure we use the `{subject}` and `{hemi}` wildcards in the filename. Output files will be named with `space-corobl` because HippUnfold is coded to effectively treat all files as already being in this space. We need the `--hemi R` to prevent HippUnfold looking for both hemispheres. Finally, because this segmentation was performed manually on very high resolution data, we can optionally consider skipping the template shape injection step with `--skip_inject_template_labels`. Template shape injection can fix minor errors in segmentation from UNet or from an imperfect manual rater, at the cost of smoothing out some details of the hippocampus due to the fact that it uses deformable registration with inherent smoothness contraints.

Note that because we are not resampling to the CITI168 template or using UNet, the T2w image in this example is effectively not being used at all. Instead, the provided manual segmentation makes up the basis for unfolding.

Case 2: one ex-vivo hemisphere

In this example, we have a single hemisphere that was scanned ex-vivo at a nearly standard resolution and T2w contrast. Because the resolution and contrast are similar to the HippUnfold training data, we expect UNet will work and so we don't need to perform manual segemntation. However, due to gross deformations and the missing hemisphere, we don't expect this sample to register well to the standard CITI168 template. Thus, we will need to manually resample the image to the CITI168 template prior to running HippUnfold, focusing in particular on aligning the hippocampus. Once done, we may have a directory like this:

```
PATH_TO_EXVIVO_DIR/
└── sub-001/
    ├── sub-001_hemi-R_desc-exvivo_T2w.nii.gz
    ├── sub-001_hemi-R_affine-exvivo-to-CITI168_xfm.txt
    └── sub-001_hemi-R_desc-exvivo_space-CITI168_T2w.nii.gz
```

Note that only the last file is needed for unfolding:

```
hippunfold . PATH_TO_OUTPUT_DIR participant --output_spaces corobl --hemi R --no_reg_
↪template \
--path_T2w PATH_TO_EXVIVO_DIR/sub-001/sub-001_hemi-R_desc-exvivo_space-CITI168_T2w.nii.
↪gz \
--output_spaces corobl
```

Here we need to use `--path_T2w` to specify which input should be used, and `--no_reg_template` to specify that it is already in `space-CITI168`. In this case, we also specified `--output_spaces corobl`. This is not needed, but is useful when we are interested in only the hippocampus as `space-corobl` is higher resolution and cropped more nicely around the hippocampus then the original scan, making it a good space to perform subsequent analyses. Alternatively, outputs can be transformed back to the original space using the inverted transform `sub-001_hemi-R_affine-exvivo-to-CITI168_xfm.txt`.

This same usage could also be applied in a standard MRI case where no T1w image is available.

4.4.8 Frequently asked questions

1. *Why is the workflow stopping at the run_inference step?*
2. *Why do I get the error, No input images found for T1w, or No input images found for T2w*
3. *Why is the HippUnfold Docker/Singularity/Apptainer container so large?*
4. *Why do I end up with large files in ~/.cache/hippunfold after running HippUnfold?*

Why is the workflow stopping at the run_inference step?

If you are getting an error in the run_inference step, e.g. as follows:

```
[Thu Nov 10 02:11:20 2022]
Finished job 65.
18 of 193 steps (9%) done
Select jobs to execute...

[Thu Nov 10 02:11:20 2022]
rule run_inference:
    input: work/sub-1425/anat/sub-1425_hemi-R_space-corobl_desc-preproc_T1w.nii.gz, /opt/
↪hippunfold_cache/trained_model.3d_fullres.Task101_hcp1200_T1w.nnUNetTrainerV2.model_
↪best.tar
    output: work/sub-1425/anat/sub-1425_hemi-R_space-corobl_desc-nnunet_dseg.nii.gz
    log: logs/sub-1425/sub-1425_hemi-R_space-corobl_nnunet.txt
    jobid: 64
    reason: Missing output files: work/sub-1425/anat/sub-1425_hemi-R_space-corobl_desc-
↪nnunet_dseg.nii.gz; Input files updated by another job: work/sub-1425/anat/sub-1425_
↪hemi-R_space-corobl_desc-preproc_T1w.nii.gz
    wildcards: subject=1425, hemi=R
    resources: tmpdir=/tmp, gpus=0, mem_mb=16000, time=60

mkdir -p tempmodel tempimg templbl && cp work/sub-1425/anat/sub-1425_hemi-R_space-corobl_
↪desc-preproc_T1w.nii.gz tempimg/temp_0000.nii.gz && tar -xf /opt/hippunfold_cache/
↪trained_model.3d_fullres.Task101_hcp1200_T1w.nnUNetTrainerV2.model_best.tar -C␣
↪tempmodel && export RESULTS_FOLDER=tempmodel && export nnUNet_n_proc_DA=1 && nnUNet_
↪predict -i tempimg -o templbl -t Task101_hcp1200_T1w -chk model_best --disable_tta &>␣
↪logs/sub-1425/sub-1425_hemi-R_space-corobl_nnunet.txt && cp templbl/temp.nii.gz work/
↪sub-1425/anat/sub-1425_hemi-R_space-corobl_desc-nnunet_dseg.nii.gz
Shutting down, this might take some time.
Exiting because a job execution failed. Look above for error message
Complete log: .snakemake/log/2022-11-10T020645.651622.snakemake.log
```

it is likely that you do not have enough memory available on your system. You need to have at least 8GB of memory on your system. If you are running Docker on Windows/Mac or another virtual machine (e.g. VirtualBox) you will need to increase the amount of memory dedicated to the virtual machine.

Why do I get the error, `No input images found for T1w`, or `No input images found for T2w`

The workflow is unable to find any input files to run HippUnfold.

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.
- HippUnfold 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. HippUnfold makes use of PyBIDS to parse the dataset, so we suggest you use the BIDS Validator to ensure your dataset has no errors. Note: You can override BIDS parsing and use custom filenames with the `--path-*` option as described in the *Parsing Non-BIDS datasets with custom paths* section.

Why is the HippUnfold Docker/Singularity/Apptainer container so large?

In addition to some large software dependencies, the container has historically included U-net models for all the possible modalities we trained, each model taking up 2-4GB. We have addressed this issue in versions >= 1.3.0, by updating the workflow to download models on the fly (when they have not been previously downloaded), and not including any models in the container itself. This drops the container size significantly (<4GB compressed).

Why do I end up with large files in `~/.cache/hippunfold` after running HippUnfold?

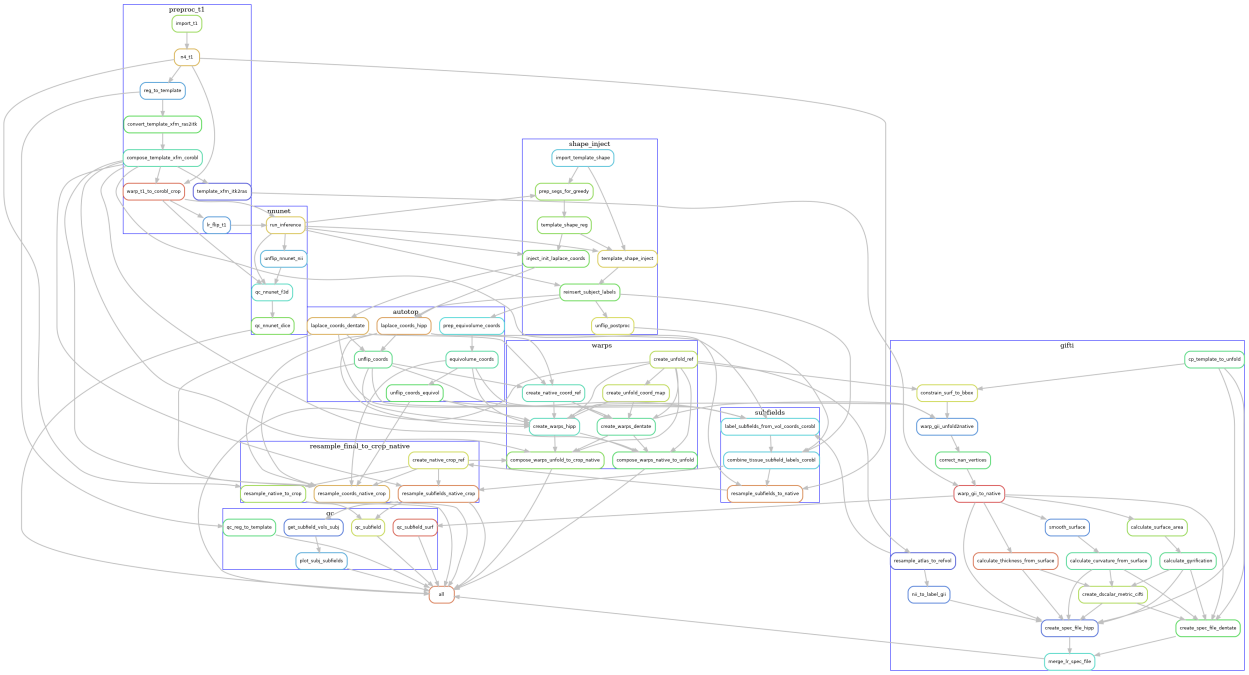
This folder is where the nnU-net model parameters are stored by default. You can override the location with the `HIPPUNFOLD_CACHE_DIR` environment variable. See *Deep learning nnU-net model files* for more details.

4.4.9 Pipeline Details

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

Overall workflow

Below is a example *simplified* visualization of the workflow DAG for the `--modality T1w` workflow. Each rounded rectangle in the DAG represents a *rule*, that is, some code or script that produces output file(s), and the arrows represent file inputs and outputs to these rules. It is *simplified* in that multiple instances of each rule are not shown, e.g. the `run_inference` rule runs on both left and right hemispheres (`hemi=L`, `hemi=R`), but only one `run_inference` box is shown here.



Although it may still look very complex (click on the image to enlarge), it is also organized into groups of rules, each representing the main phases of the workflow. Each grouped set of rules also exist in separate rule files, which can be found in the rules sub-folder

in the workflow source. For example, the preproc_t1 file contains the rules related to pre-processing the T1w images, and these are grouped together in the above diagram by a blue rectangle labelled `preproc_t1`.

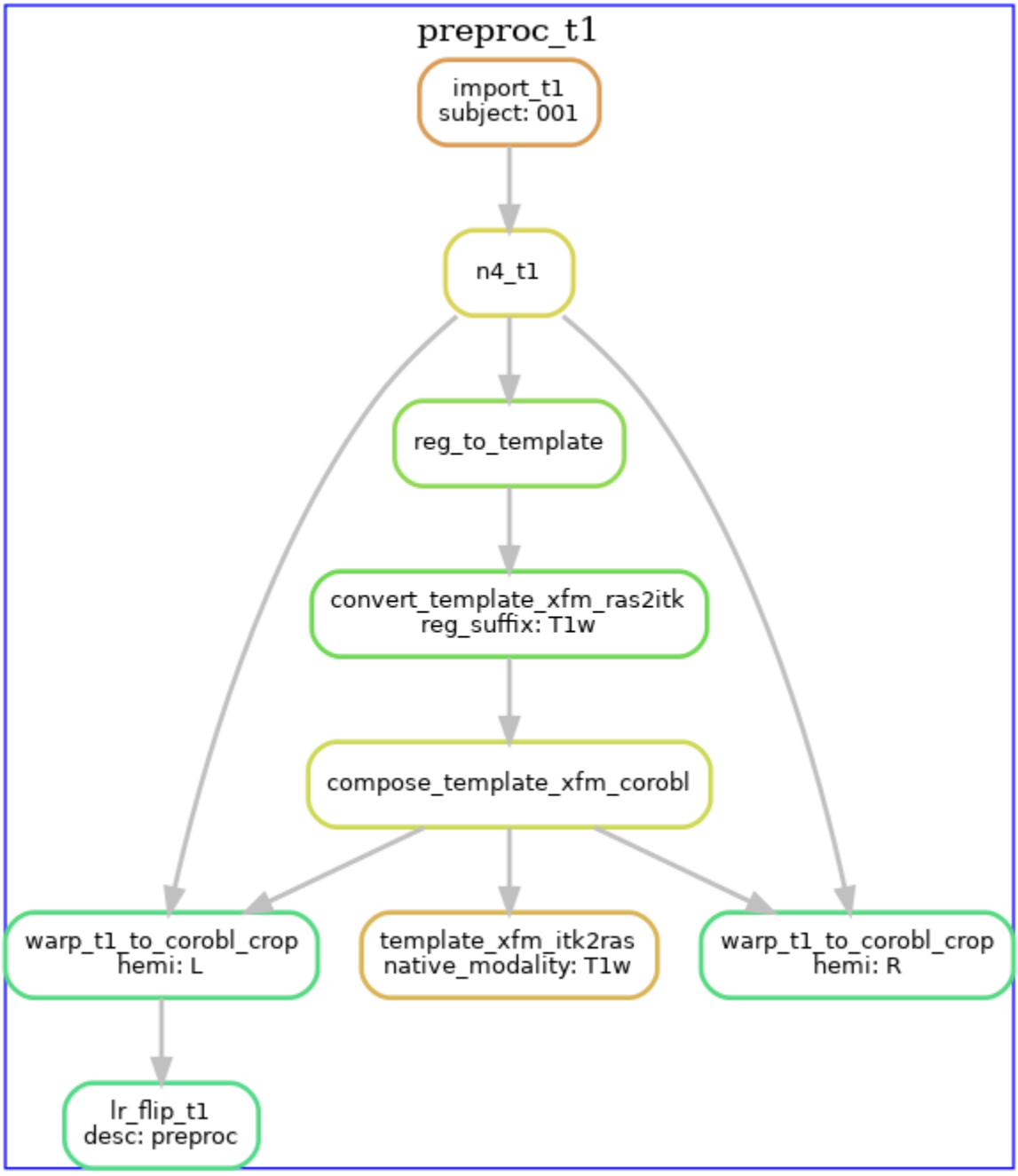
The main phases of the workflow are described in the sections below, zooming in on the rules used in each blue rectangle, one at a time.

Pre-processing

The pre-processing workflow for HippUnfold is generated based on the input data (e.g. whether there are multiple T2w images or a single T2w image), what modality is used (e.g. `--modality T1w` or `--modality T2w`), and what optional arguments are specified (e.g. `--t1-reg-template`).

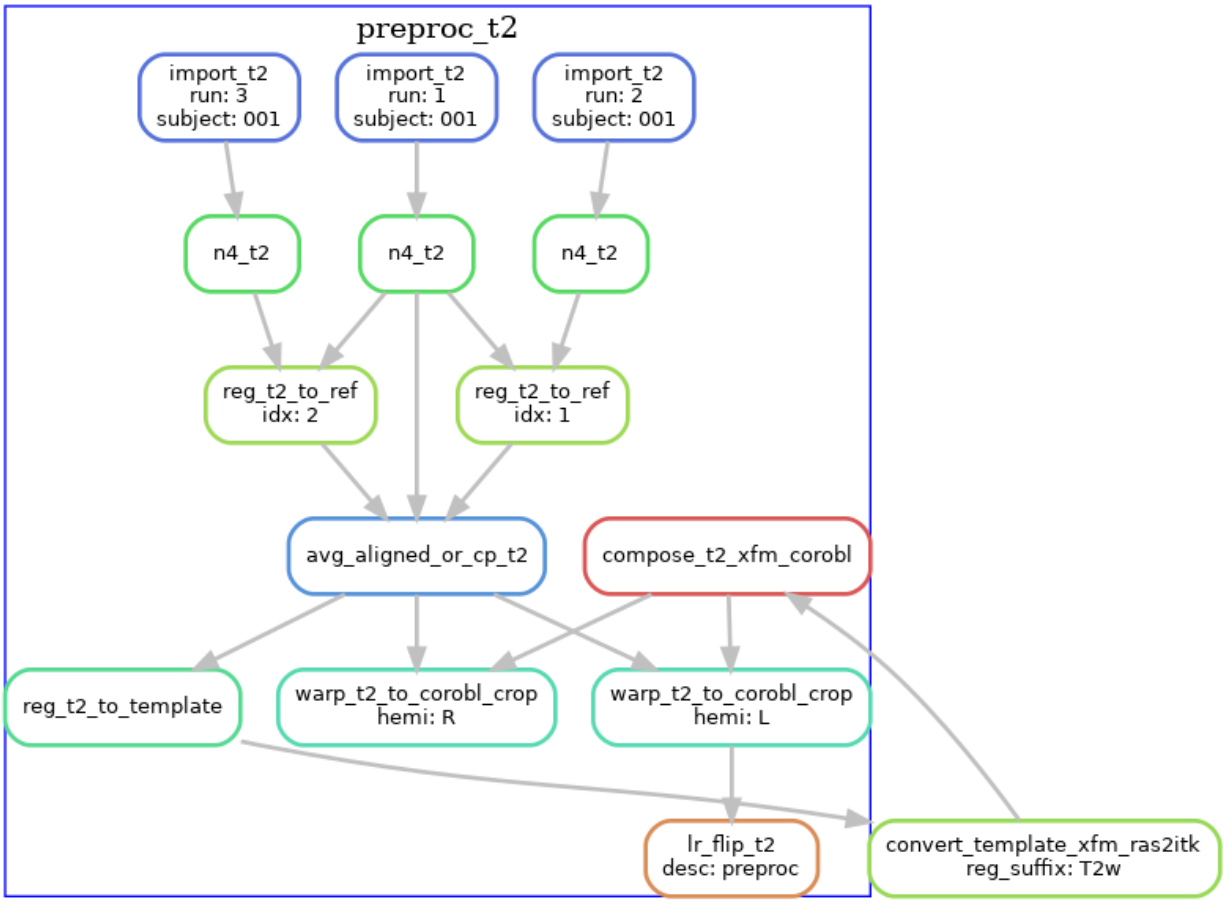
T1w pre-processing

T1w images are imported, intensity-corrected using N4, and linearly registered to the template image (default: CITI168 - an HCP T1w template). An existing transformation to align the images in a coronal oblique (`space-corobl`) orientation is concatenated, and this space is used to define the left and right hippocampus bounding boxes in 0.3mm isotropic space. The left hippocampus subvolume is left-right flipped at this stage too (subsequent steps in the `corobl` space operate on both the `hemi-R` and `hemi-Lflip` images).


preproc_t1
import_t1
subject: 001
n4_t1
reg_to_template
convert_template_xfm_ras2itk
reg_suffix: T1w
compose_template_xfm_corobl
warp_t1_to_corobl_crop
hemi: L
template_xfm_itk2ras
native_modality: T1w
warp_t1_to_corobl_crop
hemi: R
lr_flip_t1
desc: preproc

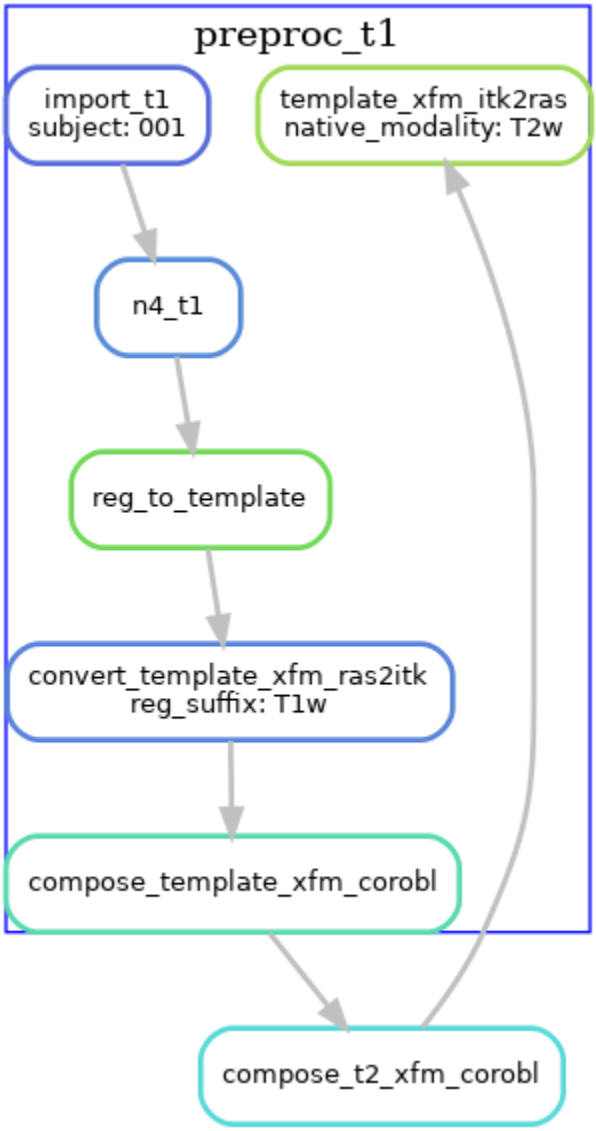
T2w pre-processing

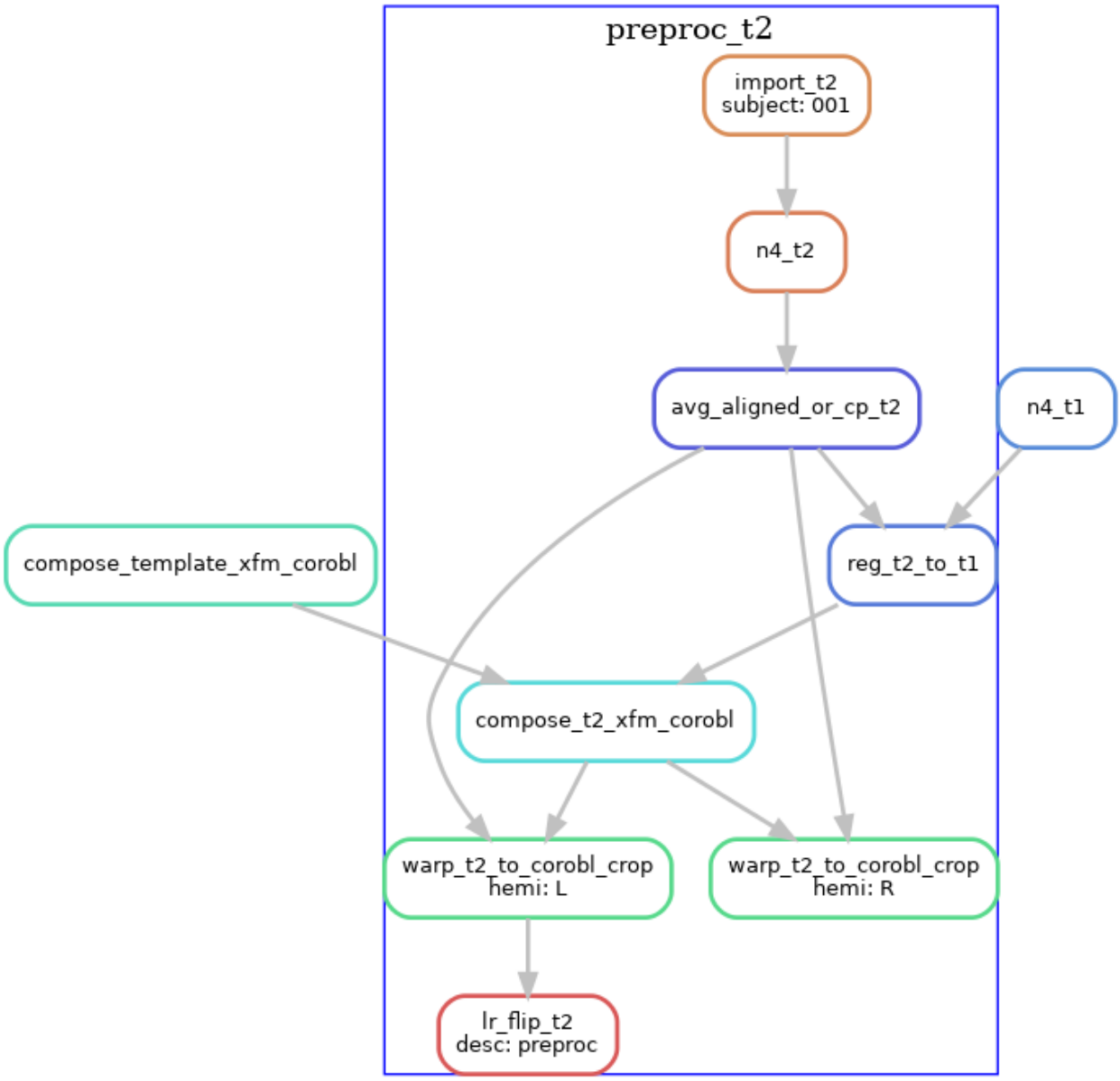
T2w images are processed similarly, except the T2w version of the template is used. If multiple T2w images exist, these are motion-corrected and averaged prior to N4 correction. The diagram below shows the T2w pre- processing workflow for a dataset with three T2w runs.



T2w with T1w template registration

For T2w images where template registration is failing (e.g. because the T2w images have a limited FOV), the `--t1-reg-template` option can be used, and will perform template registration with the T1w images, along with a within-subject registration of the T2w to the T1w, concatenating all the transforms. This is shown in the diagrams below (with a single T2w image in this case):

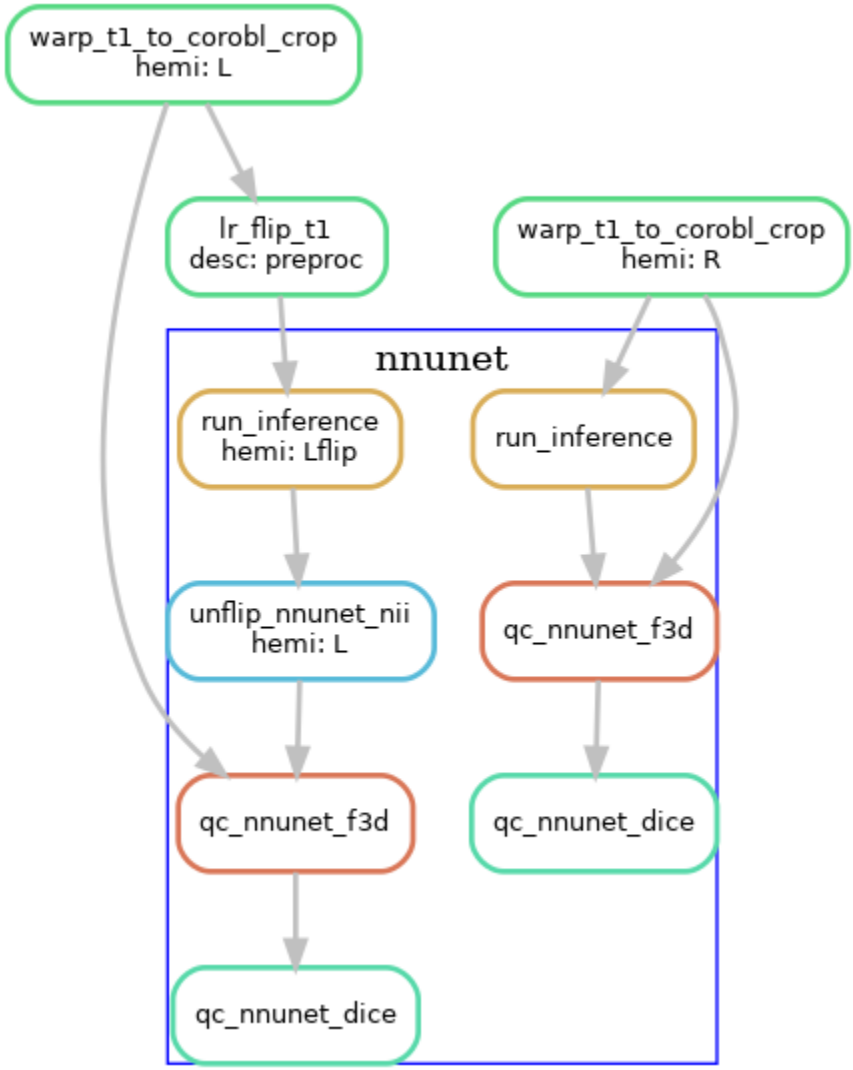

preproc_t1
import_t1
subject: 001
template_xfm_itk2ras
native_modality: T2w
n4_t1
reg_to_template
convert_template_xfm_ras2itk
reg_suffix: T1w
compose_template_xfm_corobl
compose_t2_xfm_corobl



Note that these are not the only workflow configurations possible, several other variants exist by using the command-line flags. For example, if you have T1w and T2w images that are already pre-processed and co-registered (e.g. HCP processed data), then you should use the `--skip-preproc` and `--skip-coreg` options to skip N4 and T1w/T2w co-registration.

U-net segmentation

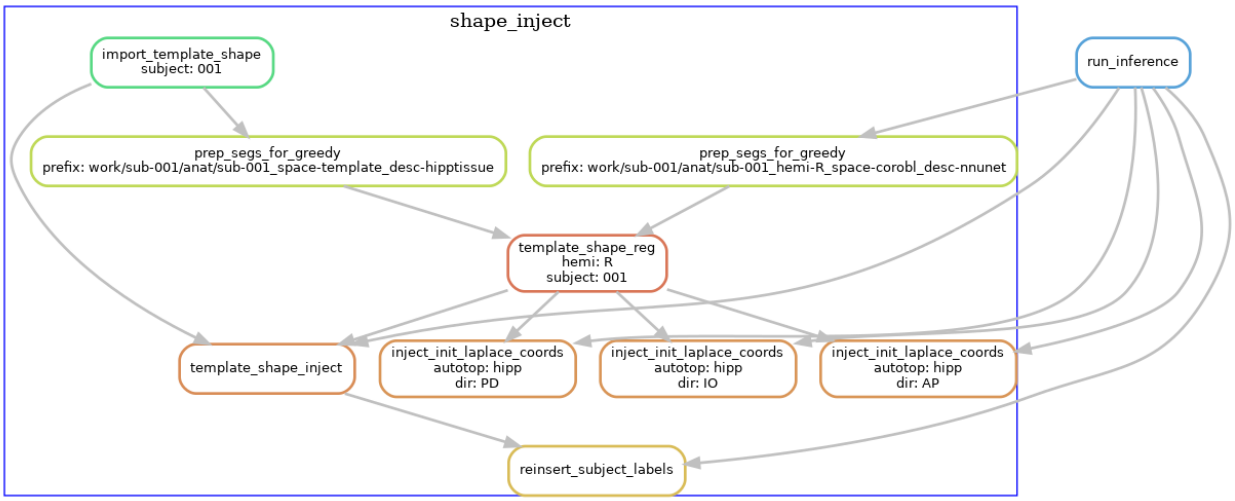
The U-net segmentation is performed on the cropped, `space-corobl` images, producing tissue segmentations (gray matter, SRLM, and anatomical landmarks for unfolding). This step is done in a single rule, which runs inference on the image using the corresponding nnU-net model based on the `modality` chosen. This is done on the `R` and `Lflip` hippocampus images, and the `Lflip` is subsequently unflipped.



Template-based shape injection

Since the nnU-net segmentation may possibly contain topological errors that can cause issues when the Laplace-based coordinates, we perform an additional registration-based correction step, a shape injection, where we perform non-linear registration of a template hippocampus segmentation and the U-net segmentation, to warp the template shape. Regularization from the registration ensures topology from the template is preserved while it is warped to match the subject hippocampus. In addition to the segmentation, we also propagate Laplace coordinates to serve as an initialization to the next step.

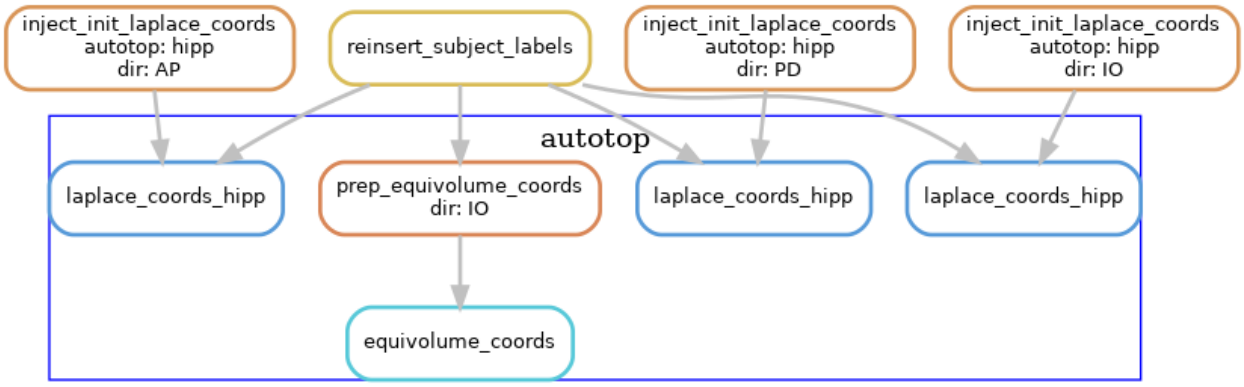
The following diagram shows the workflow, but simplified to contain one hemisphere (`--hemi R`), and excluding the dentate gyrus.



Laplace & equivolume coordinates

The basis of the hippocampal unfolding is the definition of the Laplace coordinates. Here, Laplace's equation is solved on the domain of the gray matter, using the anatomical landmarks to define boundary conditions. This provides the intrinsic set of anatomical coordinates (AP, PD, IO) for unfolding the hippocampus. For the IO (laminar) coordinates we make use of the equivolume solution instead of Laplace.

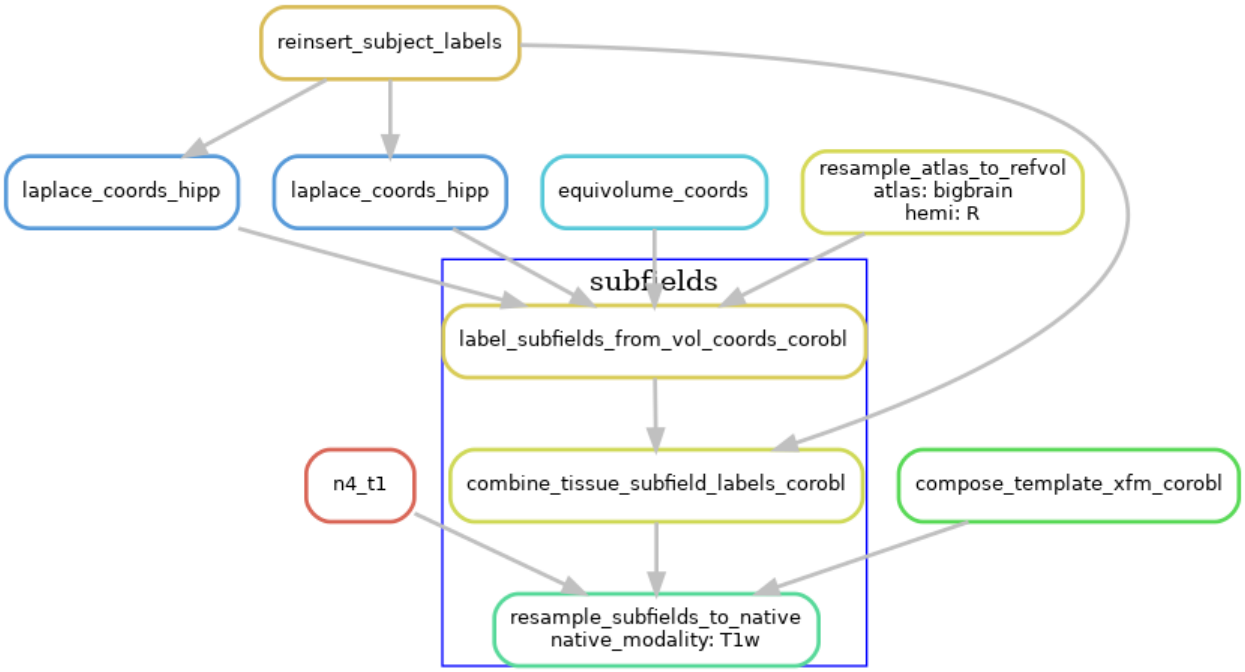
The following diagram shows the workflow, but simplified to contain one hemisphere (`--hemi R`), and excluding the dentate gyrus.



Subfields processing

The volumetric subfield segmentation is derived from the coord images from the last step, along with the atlas that defines how the coordinates map to subfields.

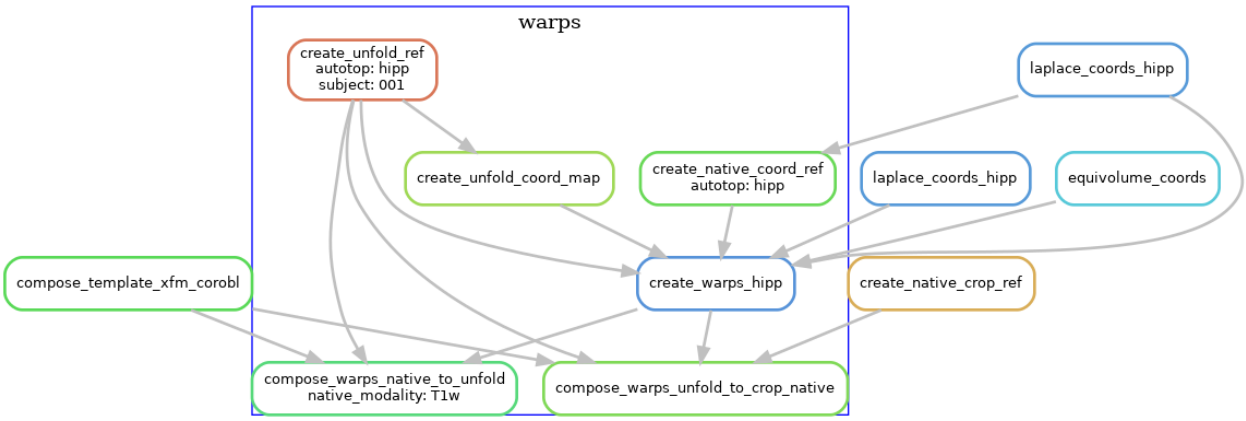
The following diagram shows the workflow, but simplified to contain one hemisphere (`--hemi R`), and excluding the dentate gyrus.



Generating warp files

To allow users to transform data between the different spaces, we generate warp files that can be applied to transform volumes of surfaces to and from the native and unfolded spaces.

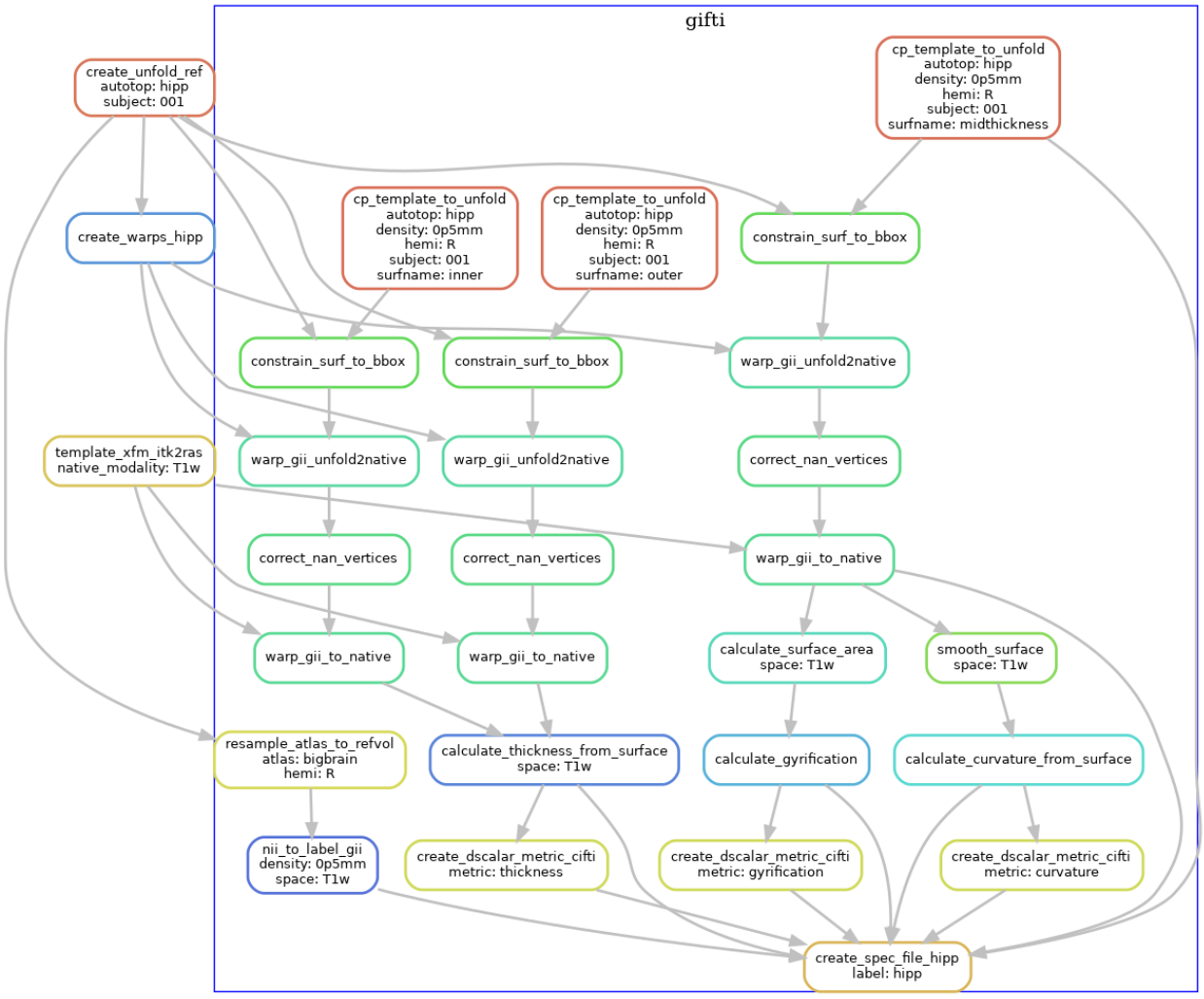
The following diagram shows the workflow, but simplified to contain one hemisphere (`--hemi R`), and excluding the dentate gyrus.



Surface processing

Using the warps, we transform standard template unfolded meshes to each subject hippocampus, in order to obtain surface meshes in the native space. These are stored in GIFTI format, and we also produce metric files to quantify surface morphometry (thickness, gyrification, curvature).

The following diagram shows the workflow, but simplified to contain one hemisphere (`--hemi R`), and excluding the dentate gyrus.



Additional steps

Resampling to output resolution, quality control snapshot generation, and archiving the work folder are steps that are also carried out by the workflow, but the DAGs are now shown here because of the many inputs/outputs, and generally have straightforward workflow structures.

4.4.10 Algorithmic details

Hippocampal unfolding

Our approach to unfolding the hippocampus involves constructing a coordinate system, defined using the solutions to partial differential equations to enforce smoothness, and to employ anatomically-derived boundary conditions. Each of the three coordinates (AP, PD, IO) are solved independently of each other, each using distinct boundary conditions defined by the hippocampus tissue segmentation. With the notation L_{ROI} to represent the labelled set of voxels in the hippocampus of a specific ROI, the domain of the solution, along with boundary conditions as source and sink, are defined as follows:

$$L_{domain} = \begin{cases} L_{GM} \cup L_{DG}, & \text{if coords} = AP \vee PD \vee IO \end{cases}$$

$$L_{source} = \begin{cases} L_{HATA}, & \text{if coords} = AP \\ L_{MTLC}, & \text{if coords} = PD \\ L_{SRLM} \cup L_{Pial} \cup L_{Cyst}, & \text{if coords} = IO \end{cases}$$

$$L_{sink} = \begin{cases} L_{IndGris}, & \text{if coords} = AP \\ L_{DG}, & \text{if coords} = PD \\ L_{background}, & \text{if coords} = IO \end{cases}$$

Template-based shape injection

We make use of a fluid diffeomorphic image registration, between a template hippocampus tissue segmentation, and the U-net tissue segmentation, in order to 1) help enforce the template topology, and 2) provide an initialization to the Laplace solution. By performing a fluid registration, driven by the segmentations instead of the MRI images, the warp is able to bring the template shape into close correspondence with the subject, but the regularization helps ensure that the topology present in the template is not broken. The template we use was built from 22 *ex vivo* images from the Penn Hippocampus Atlas.

The registration is performed using greedy, initialized using moment tensor matching (without reflections) to obtain an affine transformation, and a multi-channel sum of squared differences cost function for the fluid registration. The channels are made up of binary images, split from the multi-label tissue segmentations, which are then smoothed with a Gaussian kernel with standard deviation of 0.5*mm*. The Cyst label is replaced by the SRLM prior to this, since the locations of cysts are not readily mapped using a template shape. After warping the discrete template tissue labels to the subject, the subject's Cyst label is then re-combined with the transformed template labels.

The pre-computed Laplace solutions on the template image (analogous to method described below), $\psi^{template}_{A\to P}$, $\psi^{template}_{P\to D}$, $\psi^{template}_{I\to O}$, are then warped to the subject using the diffeomorphic registration to provide an initialization for the subject.

Fast marching initialization

As an alternative if template-based shape injection is not used, we employ a fast marching method to provide an initialization to the Laplace solution, to speed up convergence. We make use of the scikit-fmm Python package, that finds approximate solutions to the boundary value problems of the Eikonal equation,

$$F(\mathbf{x})\,|\nabla\phi(\mathbf{x})| = 1,$$

which describes the evolution of a closed curve as a function of time, ϕ, with speed $F(\mathbf{x}) > 0$ in the normal direction at a point $\mathbf{x}$ on the curve. The fast marching implementation provides a function (image) representing travel time to the zero contour of an input, ϕ.

We first perform fast marching from the source (forward direction), by initializing the zero contour with:

$$\phi_0(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} \in L_{source} \\ 1, & \mathbf{x} \notin L_{source} \end{cases}$$

and we make use of the NumPy masked arrays to avoid computations in voxels outside of L_{domain}. We use a constant speed function of 1, and perform fast marching to produce a travel-time image, $T_{forward}(\mathbf{x})$, that is normalized by $\max(T_{forward}(\mathbf{x}))$ to obtain an image from 0 to 1 (0 at the *source*). We perform the same process for the *sink* region, by setting ϕ based on L_{sink}, which produces a normalized $T_{backward}(\mathbf{x})$ image. We combine forward and backward images by averaging $T_{forward}$ and $1 - T_{backward}$ to produce the combined fast marching image, $T_{fastmarch}$.

Solving Laplace's equation

Laplace's equation is a second-order partial differential equation,

$$\nabla^2\psi(\mathbf{x}) = 0,$$

where ψ is a scalar field enclosed between the source and sink boundaries. A simple approach to solve Laplace's equation is with an iterative finite-differences approach (Jacobian method), where each voxel in the field is updated at each iteration as the average of the neighbouring grid points, e.g. for a 2-D field,

$$\psi_{i+1}(x,y) = \frac{1}{4}\left[\psi_i(x+\Delta x, y) + \psi_i(x-\Delta x, y) + \psi_i(x, y+\Delta y) + \psi_i(x, y-\Delta y)\right].$$

For our 3-D implementation, we use the nearest 18 neighbours, and perform the operation using convolution with a kernel size of $3 \times 3 \times 3$, or 27 voxels. We initialize the ψ field as follows:

$$\psi_{i=0}(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} \in L_{source} \\ 1, & \mathbf{x} \in L_{sink} \\ T_{fastmarch}(\mathbf{x}), & \mathbf{x} \in L_{domain} \\ NaN, & \text{otherwise.} \end{cases}$$

We used the convolve method from the AstroPy Python package instead of NumPy's convolve, because it avoids using NaN values (i.e. voxels outside the gray matter) in the convolution, replacing them with interpolated values using the convolution kernel. We iteratively update ψ until either the sum-of-squared-differences, $\sum\left[\psi_i(\mathbf{x}) - \psi_{i-1}(\mathbf{x})\right]^2$, is less than 1×10^{-5}, or a maximum iterations of 10,000 are reached. Note that more efficient approaches to solving Laplace's equation are possible (such as successive over-relaxation), however, we used this more conservative approach to avoid stability and convergence issues.

We use this approach to independently produce $\psi_{A\to P}$ and $\psi_{P\to D}$. Note that because we are solving these fields independent of one another, their gradient fields are not guaranteed to be perpendicular, however, we have not observed large deviations in practice. A solution for jointly solving $\psi_{A\to P}$ and $\psi_{P\to D}$ is left for future work.

Equivolumetric laminar coordinates

For the laminar, or inner-outer coordinates, , $\psi_{I\to O}$, it has been shown that an equivolumetric approach, that preserves the volume of cortical segments by altering laminar thickness based on the curvature, is more anatomically-realistic for the cerebral cortex. We implement this approach as the default for the IO coordinates, making use of the implementation in NighRes. Here, we set the inner level-set to be L_{source}, effectively the SRLM, and the outer level-set as the entire hippocampus. The continuous depth image returned by the volumetric layering function is then used directly as $\psi_{I\to O}$.

Warps for unfolding

We make use of the three coordinates, $\psi_{A\to P}$, $\psi_{P\to D}$, and $\psi_{I\to O}$, to create 3D warp fields that transform images and surfaces between the native domain $D_{native} \subset \mathbf{R}^3$, and the unfolded domain $D_{unfolded} \subset \mathbf{R}^3$.

Because solve Laplace's equation in voxels restricted to the gray matter, the native domain, D_{native} is made up of $\mathbf{x} = (x, y, z)$, where $\mathbf{x} \in L_{GM}$.

The unfolded domain, $D_{unfolded}$, is a distinct 3D space, indexed by $\mathbf{u} = (u, v, w)$, where $u = \psi_{A\to P}(x, y, z)$, $v = \psi_{P\to D}(x, y, z)$, and $w = \psi_{I\to O}(x, y, z)$. The ψ fields are initially normalized to $0 \to 1$, which would produce a rectangular prism between $(0, 0, 0)$ and $(1, 1, 1)$. However, we have re-scaled the aspect ratio and discretization to better approximate the true size of the hippocampus along each dimension, producing a volume of size $256x128x16$. To facilitate visualization, we set the origin to $(0, 200, 0)$ (in *mm*) so as not to overlap with our native space) and set a physical voxel spacing of $0.15625mm$ in each direction.

Forward warps

The transformation, or displacement warp field, that takes points, $\mathbf{x} \in \mathbf{R}^3$, (or surfaces) from native to unfolded space, is denoted as $T^{surf}_{\mathbf{x}\to\mathbf{u}} : (x, y, z) \to (u, v, w)$, and is simply defined as:

$$T^{surf}_{\mathbf{x}\to\mathbf{u}}(x, y, z) = (\psi_{A\to P}(x, y, z) - x, \psi_{P\to D}(x, y, z) - y, \psi_{I\to O}(x, y, z) - z)\,,$$

and is valid for any point, or surface vertex, within the native domain, D_{native}. Note that construction of this displacement field also involves rescaling for the physical voxel dimensions of the unfolded domain as described above, which is left out of the above equations.

Warps for surfaces and images

The warp field that transforms points/surfaces from native to unfolded also transforms images from the unfolded to the native domain,

$$T^{surf}_{\mathbf{x}\to\mathbf{u}} = T^{img}_{\mathbf{u}\to\mathbf{x}},$$

since images on a rectilinear grid must be warped with the inverse of the transformation that is required for points or surfaces. This is not particular to HippUnfold, and is true for any transformations. This is because instead of pushing forward from the moving image grid (which leads to off-grid locations), we start at the fixed grid-point (e.g. in native space), and pull-back with the inverse transformation to determine an (off-grid location) in unfolded space, to interpolate image intensities from neighbouring grid locations (e.g. in the unfolded space).

Inverse warps

To obtain, $T^{surf}_{\mathbf{u}\to\mathbf{x}} : (u, v, w) \to (x, y, z)$, or equivalently, $T^{img}_{\mathbf{u}\to\mathbf{x}} : D_{native} \to D_{unfolded}$, requires determining the inverse of the transformation that is provided by the ψ fields. We achieve this by first applying the forward transformation on all grid locations in the native domain, obtaining

$$T^{surf}_{\mathbf{x}\to\mathbf{u}}(x, y, z) = (Tx, Ty, Tz)\,, \qquad \forall (x, y, z) \in D_{native}.$$

The source native grid location, (x, y, z) for each the transformed points (Tx, Ty, Tz) is used to define the inverse transformation:

$$T^{scattered}_{\mathbf{u}\to\mathbf{x}}(Tx, Ty, Tz) = (x - Tx, y - Ty, z - Tz)$$

However, these points are only defined at scattered locations in the unfolded space, thus we need to use interpolation between these points to obtain $T^{surf}_{\mathbf{u}\to\mathbf{x}}$ defined at all grid locations in $D_{unfolded}$. We perform this operation using the *griddata* function from *SciPy*, which interpolates unstructured multi-variate data onto a grid, by triangulating the input data with Qhull, then performing piecewise linear barycentric interpolation on each triangle. Due to discretization in the ψ fields that produced the forward transformation, there are voxels outside the convex hull of the points that are not able to be linearly interpolated. To fill these values in, we make use of the *griddata* function with nearest neighbour interpolation instead. Note that this produces singularities in the warp (since points outside the convex hull have the same destination as the nearest convex hull point, but this is strictly limited to the edges of the hippocampus, and have little practical implications in our experience. After linear and nearest neighbour interpolation, the final warp field is produced:

$$T^{surf}_{\mathbf{u}\to\mathbf{x}} = T^{img}_{\mathbf{x}\to\mathbf{u}}.$$

Altogether, this provides transformations to warp either images or surfaces, in either direction (that is, native to unfolded, or unfolded to native). Image warps are defined using ITK format standards (Left-posterior-superior, or LPS coordinate system), and thus are compatible with existing tools (e.g. ANTS) to perform the transformation, or to concatenate transforms. The surface warps use a different coordinate system (Right-anterior-superior, or RAS coordinate system), for compatibility with the Connectome Workbench *surface-apply-warpfield* function, that operates on GIFTI files.

Standard surface meshes

Since the unfolding produces individual warps that can be used to transform surfaces from the unfolded domain to any individual native domain, we can produce a standardized mesh in the unfolded space (e.g. spanning a 2-D plane at a constant $w = C$ laminar level), and transform this to each hippocampus to generate a native-space hippocampal surface mesh, with 1-1 correspondence in vertices across hippocampi.

Our previous work made use of a spatially-uniform triangulated mesh in the unfolded space, now referred to as the *unfoldiso* mesh. Triangles in this mesh have equal size in the unfolded domain, however, when transformed to a subject's native space, distortions in triangle size are produced. To address this, we triangulated surfaces with an locally-adaptive number of points, where the spacing of the points was calculated to obtain approximately equal vertex spacing once transformed to the native space. The surface areas and vertex spacing were optimized on the Human Connectome Project Unrelated 100 subset, by transforming the *unfoldiso* surface, calculating the average area and spacing over all 100 subjects, then generating a range of triangular meshes with adaptive spacing. We selected meshes with mean vertex spacing close to 2mm, 1mm, and 0.5mm for our standard meshes.

Subfield segmentation

Subfield atlases in HippUnfold are now defined in the volumetric unfolded space, and are propagated to individual images or surfaces with distinct methods. For surface meshes, the subfield volumetric labels are sampled on a standard surface mesh using the Connectome Workbench function *volume-to-surface-mapping*, which creates a label GIFTI file for the surface vertices that is applicable to both unfolded and native surfaces. For volumetric images, it is possible to simply apply $T^{img}_{\mathbf{u}\rightarrow\mathbf{x}}$ to the subfield atlas. The current workflow applies an analogous approach, using $\psi_{A\rightarrow P}$ and $\psi_{P\rightarrow D}$ to interpolate, as our subfield atlas labels historically existed as surface labels, instead of the current volumetric labels, but both approaches should yield the same result.

The volumetric subfield labels are then modified to override the L_{SRLM}, L_{Cyst}, and L_{DG} labels from the tissue segmentation, since these labels are not included in the subfield atlas.

Dentate gyrus unfolding

Unfolding for the dentate gyrus conceptually identical to the hippocampus, however, the $\psi_{I\rightarrow O}$ and $\psi_{P\rightarrow D}$ fields are swapped, since the dentate gyrus tissue is topologically-perpendicular to the rest of the hippocampus.

Furthermore, because the dentate gyrus is a much smaller structure than the hippocampus, solving Laplace's equation for each individual hippocampus can be challenging if the spatial resolution is limited. Thus instead, we solely make use of the template shape injection, and use the pre-computed Laplace solution, $\psi^{template}$, to define the coordinates. Also, for the pre-computed solution, the $\psi_{A\rightarrow P}$ field is computed from the hippocampus (since this coordinate is naturally constrained to be identical for both structures).

4.4.11 Output Files

The `PATH_TO_OUTPUT_DIR` folder contains a `logs` and `work` folder for troubleshooting, but for most purposes all the outputs of interest will be in a subfolder called `hippunfold` with the following structure:

```
hippunfold/
└── sub-{subject}
    ├── anat
    ├── coords
    ├── qc
    ├── surf
    └── warps
```

Briefly, `anat` contains preprocessed volumetric input images and output segmentations in nifti format, `surf` contains surface data in gifti format, `coords` contain Laplace fields spanning the hippocampus, `warps` contains transformations between unfolded and native or 'unfolded' space, and `qc` contains snapshots and useful diagnostic information for quality control.

anat

This folder contains input anatomical images that have been non-uniformity corrected, motion-corrected, and, where appropriate, averaged and registered. In this example, a `T1w` image was used as a standard reference image, but a `T2w` was also registered and used in tissue segemntation:

```
sub-001
 └── anat
     ├── sub-001_desc-preproc_T1w.nii.gz
     ├── sub-001_space-T1w_desc-preproc_T2w.nii.gz
```

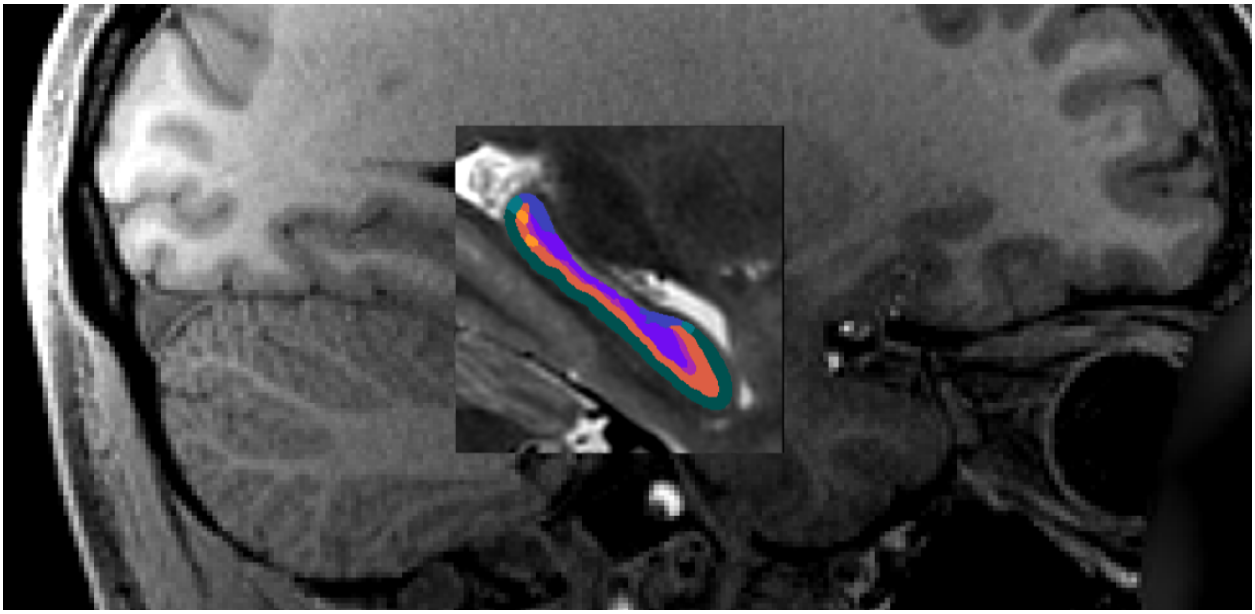
(continues on next page)

(continued from previous page)

```
    ├── sub-001_hemi-R_space-T1w_desc-subfields_atlas-bigbrain_dseg.nii.gz
    ├── sub-001_hemi-R_space-cropT1w_desc-preproc_T2w.nii.gz
    └── sub-001_hemi-R_space-cropT1w_desc-subfields_atlas-bigbrain_dseg.nii.gz
```

As per BIDS guidelines, `desc-preproc` refers to preprocessed input images, `space-T1w` refers to the volume to which the image is registered, `hemi` refers to the left or right hemisphere (only shown for the right in this example), and `dseg` (discrete-segmentation) images with `desc-subfields` contains subfield labels (coded as integers as described in the included `volumes.tsv` file). The subfield atlas used will also be included, by default as `atlas-bigbrain`. Note that HippUnfold does most intermediate processing in an unshown (available in the `work/` folder) `space-corobl` which is cropped, upsampled, and rotated. Downsampling to the original `T1w` space can thus degrade the results and so they are also provided in a higher resolution `space-cropT1w` space which is ideal for conducting volumetry or morphometry measures with high precision and detail.

For example, the following image shows a whole-brain `T1w` image, a `space-cropT1w` overlay of the upsampled T2w image (centre square), and a similarly upsampled output subfield segmentation (colour).



surf

surface meshes

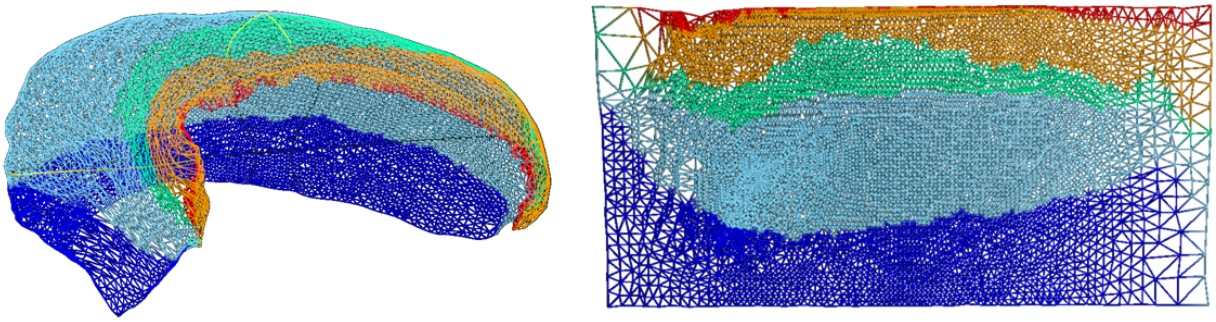
Surface meshes (geometry files) are in `.surf.gii` format, and are provided in both the native space (`space-T1w`) and the unfolded space (`space-unfolded`). In each space, there are `inner`, `midthickness`, and `outer` surfaces, which correspond to `white`, `midthickness`, and `pial` for cortical surfaces:

```
sub-{subject}
 └── surf
     └── sub-001_hemi-R_space-{T1w,unfolded}_den-0p5mm_{inner,midthickness,outer}.surf.
↪gii
```

The following shows surfaces `inner`, `midthickness`, and `outer` in yellow, orange, and red, respectively.

surface densities

Surfaces are provided in different density configurations, and are labelled based on the approximate vertex spacing in each. The default density is `0p5mm`, which has an approximate vertex spacing of 0.5mm. There are also `1mm` and `2mm` surfaces which have 1mm or 2mm spacing, respectively (suitable for lower-resolution BOLD data). Previous versions of hippunfold exclusively used a `unfoldiso` template surface, formed by a 254x126 grid in the unfolded space, however a downside of this template is that it results in very non-uniform vertex spacing when transformed to the native space. The newer `0p5mm`, `1mm` and `2mm` surfaces are designed to have closer to uniform vertex spacing in native space, though vertex spacing will not remain uniform when unfolded. This is illustrated in the the following `den-1mm` mesh in folded and unfolded space.



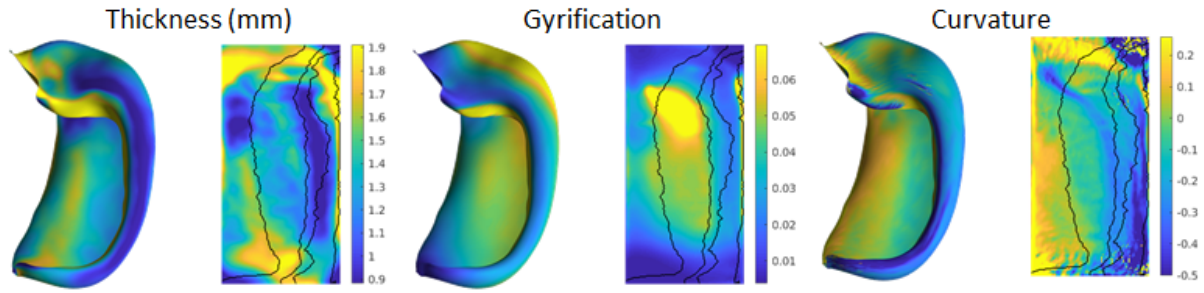
All surfaces of the same density (e.g. `1mm`), in both `space-T1w` and `space-unfolded`, share the same mesh topology and have corresponding vertices with each other. The vertex locations for unfolded surfaces are identical for all subjects as well (note that this of course is not the case for the `space-T1w` surfaces).

surface metrics

In addition to the geometry files, surface-based shape metrics are provided in `.shape.gii` format. The thickness, curvature and surface area are computed using the same methods as cortical surfaces, based on the surface geometry files, and are provided in the `T1w` space. The gyrification metric is the ratio of native to unfolded surface area, or equivalently, the scaling or distortion factor when unfolding:

```
sub-{subject}
 └── surf
     └── sub-001_hemi-{L,R}_space-T1w_den-0p5mm_label-hipp_{thickness,curvature,
↪gyrification}.shape.gii
     └── sub-001_hemi-{L,R}_space-T1w_den-0p5mm_label-dentate_{curvature,gyrification}.
↪shape.gii
```

These metrics are shown in both folded and unfolded space in the images below. Note that these results are from group-averaged data and so individual subject maps may show considerably more variability.



surface labels

The subfield labels from unfolded atlases are also provided for each subject, in `.label.gii` format. Analogous to the volume-based labels, the name of the atlas (default: `bigbrain`) is in the file name.

```
sub-{subject}
 └── surf
     └── sub-001_hemi-{L,R}_space-T1w_den-0p5mm_label-hipp_atlas-bigbrain_subfields.
↪label.gii
```

cifti files

In addition to lateralized `.shape.gii` and `.label.gii` metrics and labels, we also provide data mapped to hippocampi from hemispheres in a single file using the corresponding CIFTI formats, `.dscalar.nii` and `.dlabel.nii`. Note: since CIFTI does not support hippocampus surfaces (yet), we make use of the `CORTEX_LEFT` and `CORTEX_RIGHT` labels for the hippocampal surfaces.

```
sub-{subject}
 └── surf
     ├── sub-001_space-T1w_den-0p5mm_label-{hipp,dentate}_{thickness,curvature,
↪gyrification}.dscalar.nii
     └── sub-001_space-T1w_den-0p5mm_label-hipp_atlas-bigbrain_subfields.dlabel.nii
```

spec files

Finally, these files are packaged together for easy viewing in Connectome Workbench, `wb_view`, in the following `.spec` files, for each hemisphere and structure separately, and combined:

```
sub-{subject}
 └── surf
     ├── sub-001_hemi-{L,R}_space-T1w_den-0p5mm_label-{hipp,dentate}_surfaces.spec
     └── sub-001_space-T1w_den-0p5mm_label-{hipp,dentate}_surfaces.spec
```

New: label-dentate

HippUnfold v1.0.0 introduces `label-dentate` files which represent a distinct surface making up the dentate gyrus (reflecting its distinct topology from the rest of the cortex). The rest of the surfaces are given the name `label-hipp` to differentiate them from these new files.

These are illustrated in the following image (orange represents the usual hippocampal midthickness surface, while violet shows the new `dentate` surface):

Note that the dentate uses the same unfolding methods as the rest of the hippocampus, but with several caveats. Given its small size, its boundaries are not easily deliminated and so `inner`, `outer`, and `thickness` gifti surfaces are omitted. Furthermore, Laplace coordinates and therefore vertex spacing are not guaranteed to be topologically equivalent as they are obtained through volumetric registration with the template shape injection step of this workflow.

Corresponding `coords` and `warp` files are also generated.

New: myelin maps

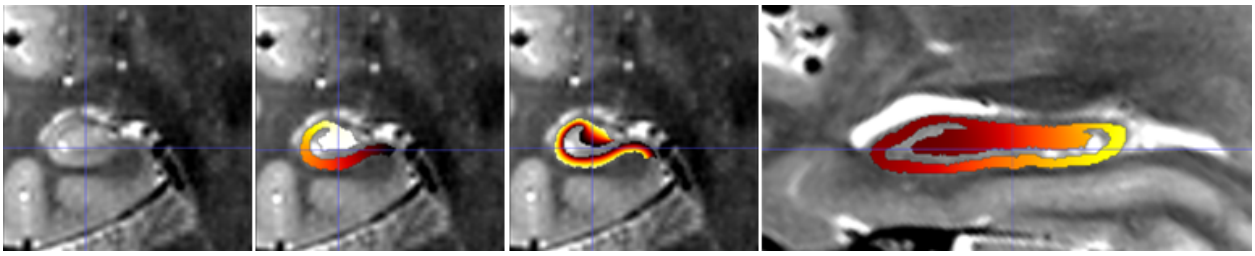
If your dataset has T1w and T2w images (and you are using `--modality=T1w` or `--modality=T2w`), then you can enable the generation of myelin maps as the ratio of T1w over T2w images. This division is done in the `corobl` space, and provides `myelin.shape.gii` surface metrics, and also includes these in the CIFTI and spec files.

This option is enabled with the `--generate-myelin-maps` command-line option.

coords

Hippunfold also provides images that represent anatomical gradients along the 3 principal axes of the hippocampus, longitudinal from anterior to posterior, lamellar from proximal (dentate gyrus) to distal (subiculum), and laminar from inner (SRLM) to outer. These are provided in the images suffixed with `coords.nii.gz` with the direction indicated by `dir-{direction}` as `AP`, `PD` or `IO`, and intensities from 0 to 1, e.g. 0 representing the Anterior end and 1 the Posterior end.

Here is an example showing coronal slices of the hippocampus with the PD, IO, and AP (sagittal slice) overlaid.



Note that these images have been resampled to `space-corobl` which is the space in which most processing is done internally. These can be seen in the `work/` output directory or specified as a possible output space.

warps

ITK transforms to warp images between the `T1w` space to the `unfolded` space, are provided for each hippocampus:

```
sub-{subject}
 └── seg
     └── sub-001_hemi-R_from-T1w_to-unfold_mode-image_xfm.nii.gz
     └── sub-001_hemi-R_from-unfold_to-T1w_mode-image_xfm.nii.gz
```

These are ITK transforms that can transform any image that is in `T1w` space (can be any resolution and FOV, as long as aligned to `T1w`), to the `unfolded` hippocampal volume space, and vice-versa. You can use the warp itself as a reference image, e.g.:

```
antsApplyTransforms -d 3 \
-i sub-001_space-T1w_FA.nii.gz \
-o sub-001_hemi-L_space-unfolded_FA.nii.gz \
-t sub-001_hemi-L_from-T1w_to-unfold_mode-image_xfm.nii.gz \
-r sub-001_space-unfolded_refvol.nii.gz
```

Additional Files

The top-level `PATH_TO_OUTPUT_DIR` contains additional folders:

```
├── hippunfold
├── logs
├── work
└── .snakemake
```

The hidden `.snakemake` folder contains a record of the code and parameters used, and paths to the inputs.

Workflow steps that write logs to file are stored in the `logs` subfolder, with file names based on the rule wildcards (e.g. subject, hemi, etc..).

Intermediate files are stored in the `work` folder. These files and folders, similar to results, are generally named according to BIDS. This folder will have `tar.gz` files for each subject, unless the `--keep_work` option is used.

If the app is run in workflow mode (`--workflow-mode/-W`) which enables direct use of the `snakemake` CLI to run hippunfold, the `hippunfold` and `work` folders will be placed in a `results` folder.

4.4.12 Visualization

Freeview (volumes and surfaces)

Freeview is a powerful viewer that works well with both volume and surface data. It (currently) has several quirks worth noting, such as:

- All surfaces should be loaded before volumes (otherwise their orientation will be incorrect).
- Loading surface metric data can be difficult, see the examples below.
- Crashes can still occur occasionally, especially when trying to load surfaces or surface overlay data with the incorrect buttons.

The example below will act as a guide to avoid common mistakes:

- open Freeview by simple typing `freeview` in the command line.
- Add a hippunfold surface: File > Load Surface > Navigate to a surface (eg. `hippunfold/sub-01/surf/sub-02_hemi-L_space-T1w_den-0p5mm_label-hipp_midthickness.surf.gii`).
- Add the correponding T1w image: File > Load Volume > Navigate to a volume (eg. `hippunfold/sub-01/anat/sub-02_desc-preproc_T1w.nii.gz`). You should now see a hippocampal surface projected onto the coronal, sagittal, and axials views over a T1w image. You should also see a 3D model of the hippocampus. You can toggle visibility of each file by unticking or ticking it in the left panel. You can also adjust the ordering ov overlays here.
- Add shape data as an overlay on the midthickness surface: With the surface file selected, use the left information panel to select Overlay > Load Generic and navigate to a metric file (eg. `hippunfold/sub-01/surf/sub-02_hemi-L_space-T1w_den-0p5mm_label-hipp_gyrification.shape.gii`). You should now have a Configure button which you can use to adjust the windowing and colormap of this data.
- Subfield labels (`.label.gii` files) can be loaded similarly to metric data, but using the Annotation button on the left panel instead of Overlay.
- Add a segmentation image (eg. `hippunfold/sub-01/anat/sub-01_hemi-L_space-cropT1w_desc-subfield_dseg.nii.gz`). On the left panel, set Colormap > Lookup Table and then Select Lookup Table > Load

Lookup Table > Navigate to `YOUR_HIPPUNFOLD_INSTALLATION_DIRECTORY/hippunfold/resources/desc-subfields_freeview_desg.tsv`. You should now see standardized subfield colours and names in the bottom panel when you mouse over a given subfield.

Other visualization tips and tricks:

- Consider adding dentate surfaces (`label-dentate`), unfolded surfaces (`space-unfolded`), or other overlay data (eg. `thickness`).
- Ensure surfaces and metric data are sampled with the same number of vertices (eg. `label-hipp` and den-'0p5mm'), and note that folded and unfolded surfaces will appear far apart in the 3D viewer and so you may need to zoom out quite far to navigate to them.
- Any metric data loaded on a folded (eg. `space-T1w`) surface can also be viewed on an unfolded surface that has the same number of vertices.
- Add Laplace coordinates (eg. `hippunfold/sub-01/coords/sub-01_dir-AP_hemi-L_space-cropT1w_label-hipp_desc-nii.gz`) over your T1w image, then set the minimum windowing to 0.001 and tick the box `Clear background` in the left panel to maintain visibility of the T1w underlay.
- Note that `space-T1w` and `space-cropT1w` should appear in equivalent positions in Freeview, despite having a different field of view and resolution. If loading these data into Matlab or Python, you will need to first resample these images to the same reference image in order to index voxels from equivalent points.

HippUnfold Toolbox

The HippUnfold Toolbox provides examples and functions for mapping data onto hippocampal surfaces, plotting surfaces, and performing comparisons or statistical tests between subjects. Note that this can be done in other programs, like Connectome workbench, but these Python & Matlab tools should give an idea of how this can be done in a fully customizable fashion.

ITK-SNAP (volumes)

ITK-SNAP is a lightweight tool able to quickly open volumes, and is ideal for manual segmentation or edits. Segmentation images (`_dseg.nii.gz`) can be loaded as overlays and ITK-SNAP will create a 3D rendering of the contours of each label. Use the `space-cropT1w` or `space-cropT2w` images in the `anat/` folder to visualize one hemisphere at a time.

Connectome Workbench (surfaces)

Connectome Workbench view tool allows for advanced visualization of surfaces and surface data. Loading one of the `.spec` files produced by HippUnfold into `wb_view` will allow you to visualize the hippocampus in native and unfolded configurations, and also overlay metric and label data (e.g. subfields and thickness).

4.4.13 Quality Control

A complex pipeline can have multiple points of failure, so its imprtant to check the QC folder to ensure results make sense. This will contain (for example) the following files:

```
hippunfold/
└── sub-001
    └── qc
        ├── sub-001_from-T1w_to-CITI168_regqc.png
        ├── sub-001_hemi-L_desc-unetf3d_dice.tsv
        ├── sub-001_hemi-L_space-cropT1w_desc-subfields_atlas-histologyReference2023_
↪dseg.png
        ├── sub-001_hemi-L_space-T1w_den-0p5mm_label-dentate_desc-subfields_midthickness.
↪surf.png
        └── sub-001_hemi-L_space-T1w_den-0p5mm_label-hipp_desc-subfields_midthickness.
↪surf.png
```

These files will be mirrored in the right (`hemi-R`) hemisphere, but we will skip that here for brevity.

Automated checks

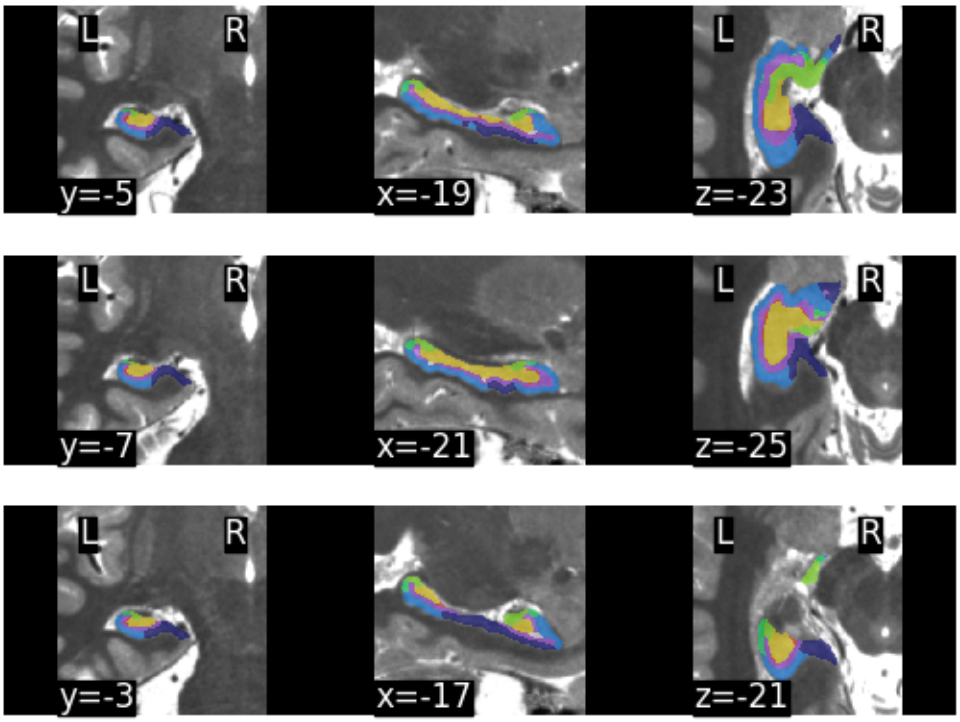
A very fast quality check is to simply look at `sub-001_hemi-L_desc-unetf3d_dice.tsv` and ensure that this number is greater than `0.7`. Typically this criterion can be applied to all subjects, and subjects <`0.7` can simply be automatically discarded. A more in-depth explanation for this is provided at the bottom of this page.

To better understand some common points of failure, let's walk through the other files contained here:

Catch-all failures

A good way to be sure that nothing has failed is also to look at the final results, as in the following snapshot images.

`sub-001_hemi-L_space-cropT1w_desc-subfields_atlas-histologyReference2023_dseg.png`:



Here we see a volumetric output of subfield parcellations. If you have some hippocampal expertise then you may be able to see right away whether the results make sense. Some common errors include missed parts of the hippocampus and, sometimes, parts of collateral sulcus are included as part of the hippocapus. This is because of gross or "catastrophic" failures in the UNet tissue classification, which should be caught be the above automated check. Still, this collage of sample slices should give a good sense for whether/how the segmentation has failed.

`sub-001_hemi-L_space-T1w_den-0p5mm_label-hipp_desc-subfields_midthickness.surf.png`:

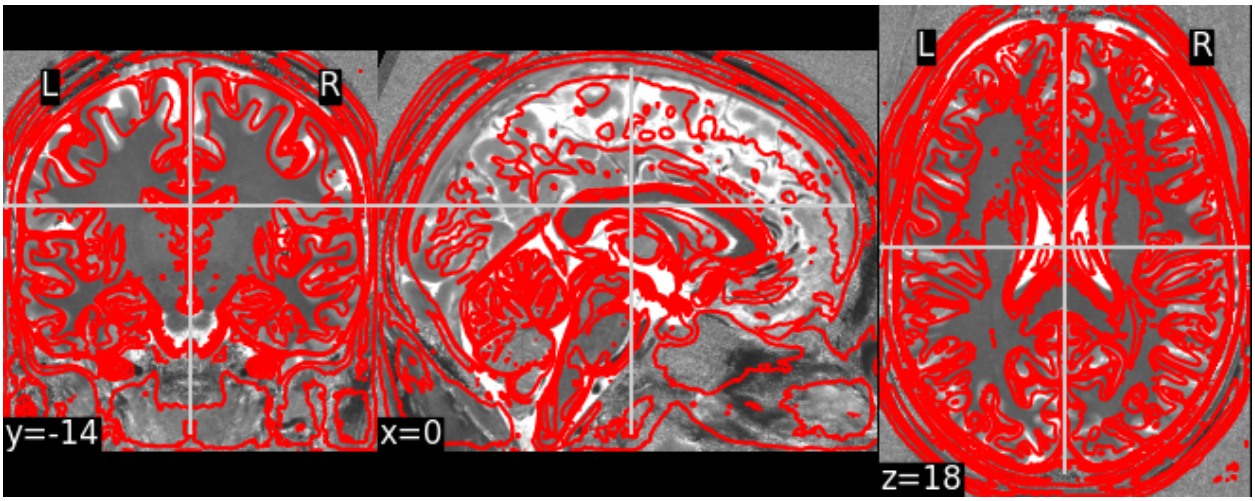
`sub-001_hemi-L_space-T1w_den-0p5mm_label-dentate_desc-subfields_midthickness.surf.png`:

In these two images we see the output hippocampal and dentate gyrus surfaces, which should roughly resemble those shown in the HippUnfold manuscript (though they may have some differences in shape and/or sulcal/gyral patterning due to inter-individual variance!).

Sometimes the dentate gyrus surface may show a few large triangles that obscure the rest of the surface, as in the example here. This typically arises when just a few vertices are badly misplaced, and so typically this doesn't have much impact on quantitative results but it can look quite ugly in a figure. This can arise because the structure is small and contains few voxels, none of which have a laplace field close to a given value and so the xyz coordinate of a given laplace value is extrapolated badly. We will try to fix this issue in future releases.

Registration/cropping failures

Though it seems simple, just getting the image roughly aligned to a template so we can then crop around the left and right hippocampi for subsequent processing (in `space-corobl`) can sometimes fail. We can examine this in the file `sub-001_from-T1w_to-CITI168_regqc.png`:



In this image, the subject tissue boundaries (e.g. grey matter, white matter, CSF, skull) are overlaid on a standard template. This is a linear alignment, so while gyri/sulci may not be well aligned, we should still see fairly good overlap that is sufficient to derive the position of the hippocampus in the image. This most often fails if the input images are in an unexpected orientation (e.g. this sometimes arises in ex-vivo scanning), and can sometimes be imporved by first running FSL's reorient2std.

Checking the automated check

The `Dice > 0.7` rule is a good automatic check because HippUnfold typically shows high sensitivity to details in hippocampal structure, but when it fails it often does so catastrophically. These failures are very obvious, and can often be shown by this simple Dice score that compares a whole mask of the hippocampus to a whole mask of a hippocampus generated by a more conventional but less precise measure: diffeomorphic registration to a standard template. The Dice score should be good between the two methods, but will never be close to perfect (>`0.9`) since the HippUnfold method will differ as its more precise.

This issue typically only occurs in ~1% of cases.

4.4.14 Contributing to Hippunfold

Hippunfold python package dependencies are managed with Poetry, which you'll need installed on your machine. You can find instructions on the poetry website.

HippUnfold also has a number of dependencies outside of python, including popular neuroimaging tools like `wb_command`, `ANTs`, `c3d`, and others listed in https://github.com/khanlab/hippunfold_deps. Thus we strongly recommend running HippUnfold with the `--use-singularity` flag, which will pull this container automatically and use it when required, unless you are comfortable using all of these tools yourself.

Note: These instructions are only recommended if you are making changes to HippUnfold code to commit back to this repository, or if you are using Snakemake's cluster execution profiles. If not, it is easier to run HippUnfold when it is packaged into a single singularity container (e.g. `docker://khanlab/hippunfold:latest`).

Set-up your development environment:

Clone the repository and install dependencies and dev dependencies with poetry:

```
git clone http://github.com/khanlab/hippunfold
cd hippunfold
poetry install
```

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

Then, you can run hippunfold with:

```
poetry run hippunfold
```

or you can activate a virtualenv shell and then run hippunfold directly:

```
poetry shell
hippunfold
```

You can exit the poetry shell with `exit`.

Note: you can alternatively use `pip install` to install dependencies.

Running code format quality checking and fixing:

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

```
poetry run poe
```

We use `black` and `snakefmt` to ensure formatting and style of python and Snakefiles is consistent. There are two task runners you can use to check and fix your code, and can be invoked with:

```
poetry run poe quality_check
poetry run poe quality_fix
```

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

Dry-run testing your workflow:

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

You can use the hippunfold CLI to perform a dry-run of the workflow, e.g. here printing out every command as well:

```
hippunfold test_data/bids_singleT2w test_out participant --modality T2w --use-
↪singularity -np
```

As a shortcut, you can also use `snakemake` instead of the hippunfold CLI, as the `snakebids.yml` config file is set-up by default to use this same test dataset, as long as you run snakemake from the `hippunfold` folder that contains the `workflow` folder:

```
cd hippunfold
snakemake -np
```

Instructions for Compute Canada

This section provides an example of how to set up a `pip installed` copy of HippUnfold on Compute Canada's `graham` cluster.

Setting up a dev environment on graham:

Here are some instructions to get your python environment set-up on graham to run HippUnfold:

1. Create a virtualenv and activate it:

   ```
   mkdir $SCRATCH/hippdev
   cd $SCRATCH/hippdev
   module load python/3.8
   virtualenv venv
   source venv/bin/activate
   ```

2. Install HippUnfold

   ```
   git clone https://github.com/khanlab/hippunfold.git
   pip install hippunfold/
   ```

3. To run Hippunfold on Graham as a member of the Khan lab, please configure the neuroglia-helpers with the khanlab profile.

4. To avoid having to download trained models (see section *below*), you can set an environment variable in your bash profile (~/.bash_profile) with the location of the trained models. For Khan lab's members, the following line must be add to the bash profile file:

   ```
   export HIPPUNFOLD_CACHE_DIR="/project/6050199/akhanf/opt/hippunfold_trained_models"
   ```

Note: make sure to reload your bash profile if needed (`source ~./bash_profile`).

5. For an easier execution in Graham, it's recommended to also install cc-slurm snakemake profile for cluster execution with slurm.

Note if you want to run hippunfold with modifications to your cloned repository, you either need to pip install again, or run hippunfold the following, since an `editable` pip install is not allowed with pyproject:

```
    python <YOUR_HIPPUNFOLD_DIR>/hippunfold/run.py
```

Running hippunfold jobs on graham:

Note that this requires neuroglia-helpers for regularSubmit or regularInteractive wrappers, and the cc-slurm snakemake profile for cluster execution with slurm.

In an interactive job (for testing):

```
regularInteractive -n 8
hippunfold <PATH_TO_BIDS_DIR> <PATH_TO_OUTPUT_DIR> participant \
--participant_label 001 -j 8 --modality T1w --use-singularity \
--singularity-prefix $SNAKEMAKE_SINGULARITY_DIR
```

Where:

- `--participant_label 001` is used to specify only one subject from a BIDS directory presumeably containing many subjects.
- `-j 8` specifies the number of cores used
- `--modality T1w` is used to specify that a T1w dataset is being processed
- `--singularity-prefix $SNAKEMAKE_SINGULARITY_DIR` specifies the directory in which singularity images will be stored. The environment variable is created when installing neuroglia-helpers.

Submitting a job (for larger cores, more subjects), still single job, but snakemake will parallelize over the 32 cores:

```
regularSubmit -j Fat \
hippunfold PATH_TO_BIDS_DIR PATH_TO_OUTPUT_DIR participant  -j 32 \
--modality T1w --use-singularity --singularity-prefix $SNAKEMAKE_SINGULARITY_DIR
```

Scaling up to ~hundred subjects (needs cc-slurm snakemake profile installed), submits 1 16core job per subject:

```
hippunfold PATH_TO_BIDS_DIR PATH_TO_OUTPUT_DIR participant \
--modality T1w --use-singularity --singularity-prefix $SNAKEMAKE_SINGULARITY_DIR \
--profile cc-slurm
```

Scaling up to even more subjects (uses group-components to bundle multiple subjects in each job), 1 32core job for N subjects (e.g. 10):

```
hippunfold PATH_TO_BIDS_DIR PATH_TO_OUTPUT_DIR participant \
--modality T1w --use-singularity --singularity-prefix $SNAKEMAKE_SINGULARITY_DIR \
--profile cc-slurm --group-components subj=10
```

Running hippunfold jobs on the CBS server

1. Clone the repository and install dependencies and dev dependencies with poetry:

```
git clone http://github.com/khanlab/hippunfold
cd hippunfold
poetry install
```

If poetry is not installed, please refer to the installation documentation. If the command poetry is not found, add the following line to your bashrc file located in your home directory (considering that the poetry binary is located under `$HOME/.local/bin`:

```
export PATH=$PATH:$HOME/.local/bin
```

2. To avoid having to download containers and trained models (see section *below*), add the `$SNAKEMAKE_SINGULARITY_DIR` and `$HIPPUNFOLD_CACHE_DIR` environment variables to the bashrc file. For Khan lab's members, add the following lines:

```
export SNAKEMAKE_SINGULARITY_DIR="/cifs/khan/shared/containers/snakemake_containers
↪"
export HIPPUNFOLD_CACHE_DIR="/cifs/khan/shared/data/hippunfold_models"
```

3. HippUnfold might be executed using `poetry run hippunfold <arguments>` or through the `poetry shell` method. Refer to previous section for more information in regards to execution options.

4. On the CBS server you should always set your output folder to a path inside `/localscratch`, and not your home folder or a `/srv` or `/cifs` path, and copy the final results out after they have finished computing. Please be aware that the CBS server may not be the most efficient option for running a large number of subjects (since you are limited in processing cores vs a HPC cluster).

5. If you are using input files in your home directory (or in your `graham` mount in your home directory), you may also need to also add the following to your bashrc file (Note: this will become a default system-enabled option soon)

```
export SINGULARITY_BINDPATH="/home/ROBARTS:/home/ROBARTS"
```

Deep learning nnU-net model files

The trained model files we use for hippunfold are large and thus are not included directly in this github repository, and instead are downloaded from Zenodo releases.

For HippUnfold versions earlier than 1.3.0 (< 1.3.0):

If you are using the docker/singularity container, `docker://khanlab/hippunfold`, they are pre-downloaded there, in `/opt/hippunfold_cache`.

If you are not using this container, you will need to download the models before running hippunfold, by running:

```
hippunfold_download_models
```

This console script (installed when you install hippunfold) downloads all the models to a cache dir on your system, which on Linux is typically `~/.cache/hippunfold`. To override this, you can set the `HIPPUNFOLD_CACHE_DIR` environment variable before running `hippunfold_download_models` and `hippunfold`.

NEW: For HippUnfold versions 1.3.0 and later (>= 1.3.0):

With the addition of new models, including all models in the container was not feasible and a change was made to **not include** any models in the docker/singularity containers. In these versions, the `hippunfold_download_models` command is removed, and any models will simply be downloaded as part of the workflow. As before, all models will be stored in the system cache dir, which is typically `~/.cache/hippunfold`, and to override this can set the `HIPPUNFOLD_CACHE_DIR` environment variable before running `hippunfold`.

If you want to pre-download a model (e.g. if your compute nodes do not have internet access), you can run simply run `download_model` rule in HippUnfold e.g.:

```
hippunfold BIDS_DIR OUTPUT_DIR PARTICIPANT_LEVEL --modality T1w --until download_model -
↪c 1
```

Overriding Singularity cache directories

By default, singularity stores image caches in your home directory when you run `singularity pull` or `singularity run`. As described above, hippunfold also stores deep learning models in your home directory. If your home directory is full or otherwise inaccessible, you may want to change this with the following commands:

```
export SINGULARITY_CACHEDIR=/YOURDIR/.cache/singularity
export SINGULARITY_BINDPATH=/YOURDIR:/YOURDIR
export HIPPUNFOLD_CACHE_DIR=/YOURDIR/.cache/hippunfold/
```

If you are running `hippunfold` with the `--use-singularity` option, hippunfold will download the required singularity containers for rules that require it. These containers are placed in the `.snakemake` folder in your hippunfold output directory, but this can be overriden with the Snakemake option: `--singularity-prefix DIRECTORY`