Chapter 9

Structural MRI and Computational Anatomy

Felix Hoffstaedter, Georgios Antonopoulos, and Christian Gaser

Abstract

Structural magnetic resonance imaging can yield highly detailed images of the human brain. In order to quantify the variability in shape and size across different brains, methods developed in the field of computational anatomy have proved exceptionally useful. For example, voxel-based morphometry is a popular method that involves segmenting magnetic resonance imaging scans into gray matter, white matter, and cerebrospinal fluid, and transforming individual brain shapes to a standard template space for comparative analysis. However, computational anatomy—when applied to brain data at scale—can be complex and computationally expensive. Furthermore, there are many possible pipelines that can be applied to structural brain data and for this reason it is important to follow best practices for reproducible neuroimaging analyses. This chapter demonstrates reproducible processing using the CAT12 (Computational Anatomy Toolbox) extension to SPM12 that focuses on voxel- and region-based morphometry. Through worked examples, we demonstrate three approaches to reproducible image analysis: "minimal", "intermediate", and a "comprehensive" protocol using the FAIRly big workflow based on DataLad. The comprehensive approach automatically facilitates parallel execution of whole dataset processing using container technology and also produces re-executable run records of each processing step to enable fully automatic reproducibility.

Key words Structural neuroimaging, Computational anatomy, Voxel-based morphometry, Reproducibility, FAIRly big workflow

1 Introduction

Since the growing availability of magnetic resonance imaging (MRI) machines in research facilities and hospitals more than two decades ago, robust noninvasive analysis of brain anatomy has become common practice in neuroscience research. Two fundamental use cases can be distinguished. First, anatomical brain mapping in relation to basic phenotypes, such as age and sex/gender. Second, the investigations of cognitive functioning and clinical conditions, which are associated with variations in brain morphology. Voxel-based morphometry (VBM) is one of the first and most commonly used method for quantification of brain tissue types

Robert Whelan and Hervé Lemaître (eds.), *Methods for Analyzing Large Neuroimaging Datasets*, Neuromethods, vol. 218, https://doi.org/10.1007/978-1-0716-4260-3_9,

(specifically gray matter volume; GMV) in humans [1] and animals [2].

VBM is based on the identification of specific brain tissue types—commonly gray and white matter (GM, WM) as well as CerebroSpinal Fluid (CSF), during a process called *segmentation*. In whole-brain MRI scans, tissue classification is performed by expressing every point/voxel in the brain as a probability or volume fraction of GM, WM, and CSF. Subsequently, individual brain shapes are transformed to match a standard template, often in MNI space [3], allowing a point-to-point comparison of corresponding regions across different brains. Furthermore, the local amount of deformation in every voxel—expressed by the Jacobian determinant—is utilized to *modulate* tissue probability/fraction to represent a proxy for local GMV. General comparability over studies is made possible by using the MNI space as a standard coordinate system combined with the use of common brain templates and brain parcellations. There are several widely used and well-tested publicly available software packages that, in principle, allow researchers of different experience levels to carry out structural MRI analysis without first becoming experts in programming or brain anatomy.

Recent studies have shown marked differences among VBM pipelines [4, 5]; however, robust effects of age or sex were detected by all evaluated software solutions. Similarly, coordinate-based meta-analysis of VBM studies each using different analysis software has shown consistent brain changes associated with neurodegenerative diseases. For example, a neuroimaging meta-analysis on frontotemporal dementia found consistent effects of neural atrophy over studies using different analysis software and brain templates [6]. Neuroscience, along with other natural sciences such as biology and psychology, faces the challenge of poor replicability in many studies due to small sample sizes and analytic flexibility (*see* also Chapter 4 for best practices in reproducible neuroimaging). Indeed, it has been shown that VBM analyses on moderately sized samples ($n \sim 300$) tend to overestimate effect sizes and significant effects are often not replicable [7]. Together with other evidence, this makes a strong case for conducting large-scale replication analyses even for well-known effects such as brain maturation [8], asymmetry [9] and aging [10].

In this chapter, we demonstrate the automatic structural processing of MRI images, using the AOMIC datasets [11] as an example. We describe a fully reproducible workflow [12] based on DataLad [13]. DataLad (*see* Chapter 2) is a powerful, open-source research data management software based on git (git-scm.com) and git-annex (git-annex.branchable.com); *see* also Chapter 5 (Garcia & Kelly) for git use. We utilize the robust and easy-to-use CAT12 (Computational Anatomy Toolbox, https://neuro-jena.github.io/cat) extension to SPM12 (www.fil.ion.ucl.ac.uk/spm) in

Matlab, also available without license costs as a standalone version (https://neuro-jena.github.io/enigma-cat12/#standalone) or as a Singularity container (https://github.com/inm7-sysmed/ENIGMA-cat12-container). CAT12 covers diverse morphometric analysis methods such as VBM, surface-based morphometry (SBM), deformation-based morphometry (DBM), and label- or region-based morphometry (RBM). For a brief description of CAT12, *see* https://neuro-jena.github.io/cat12-help/#basic_vbm); for a detailed description of CAT12, *see* https://neuro-jena.github.io/cat12-help/#process_details. As the focus of this chapter is the reproducible processing of large cohorts, the analysis is limited to showcases of preprocessing for VBM and RBM analysis.

2 Methods Overview

2.1 Starting Point of the Data

Note
CAT12 will produce many different output images (e.g., local tissue estimates at voxel level of 1.5 mm^3 by default), as well as atlas based regional volume estimates. Users should be aware that many VBM pipelines—including CAT12—can output files into one or multiple folders, which creates complexity for larger datasets.

The starting point for volumetric analyses of brain tissues typically uses a T1-weighted contrast MRI sequence, which highlights cortical gray and white matter. Additionally, the T2-weighted contrast is essential for brain lesion mapping in clinical analysis. Even in smaller samples ($n < 50$ subjects), disease-related effects (e.g., neurodegeneration) can be reliably detected if the effect size is large. However, in order to improve generalization, larger samples ($n > 300$) should be used [7]. As described in other chapters in this book, the importance of standardized data structures and formats should not be underestimated. It is strongly recommended to adhere to the BIDS standard for input data and carefully plan the output structure especially for very large datasets (*see* Chapter 4).

2.2 Data Storage and Computing

CAT12 is based on Statistical Parametric Mapping (SPM) software, originally running on Matlab, however there is an Octave (7.3) version and a compiled standalone version available which runs without a Matlab License via Matlab Runtime (MCR, Matlab Compiler Runtime, *see* Resources). To enhance reproducibility, we also provide a recipe for fully automatic creation of a Singularity container (https://apptainer.org/; *see* Glossary). As MCR cannot

be publicly shared, CAT12 Singularity containers are privately available upon reasonable and unreasonable requests. We provide example setups for Linux and macOS without container environments for local data processing, as well as a Linux-based setup deploying Singularity with template scripts for the optional use of a job scheduler. CAT12 can run even on older laptops with four CPUs and 8 GB RAM, requiring up to 20 min of compute time per T1-weighted image, including cortical surface extraction. The ideal compute environment for full scalability is a compute cluster running a flavor of Linux providing recent versions of DataLad (≥0.17), Singularity (≥2.5) and a workload scheduling system like HTCondor or SLURM. Regarding data storage, the amount of data generated during CAT12 preprocessing varies based on the input data. For instance, with a standard T1-weighted MRI scan of 6.5 MB, the output generated by CAT12 will be approximately 60 MB per scan.

2.3 Coding Knowledge

The example setup in this chapter should be executable for beginners with basic command line experience given the correct platform setup (*see* above).

2.4 Computational Expense

Standard preprocessing of data from 100 subjects (×0.3h) takes 30 core hours and parallelized on a standard desktop computer with four cores, will take only 7.5 h. Parallelization (*see* Glossary) of individual jobs running independently at the same time can be done directly by CAT12 standalone or automatically by the example workflow locally using GNU-parallel or a given scheduler software on a compute cluster.

3 Worked Example

The following demonstration of structural image analysis starts with a simple approach to reproducible image processing, using a two-step protocol to improve result tracking and automatic process documentation. Initially, DataLad is used as a download tool to provide access to the input data and analysis code (as described in Chapter 2). Secondly, the results of processing are periodically captured by saving snapshots of the outcome of each processing step. We also described a way to automatically track the whole image analysis pipeline by using the "FAIRly big workflow" [12].

As with most standard data processing workflows, we need:

(A) Input Data

(B) Processing Software/ Pipeline

(C) Code to execute B on A

3.1. *A minimal approach to reproducible image analysis consists in* (A) providing the *Input Data*, here MR images and added metadata about how they were acquired (B) documenting the software versions used in the *Processing Pipeline* or provide precise instructions to build the used software environment (C) best practice is to share the analysis *Code* as a Git repository or as download on a public hosting site like Zenodo or GitHub, together with *Results Images and Statistics* (*see* Chapter 4). The following commands work on any Linux OS and macOS system.

With DataLad installed, Users are able to access the input data (A)—here the AOMIC-PIOP2 dataset—by simply cloning it from OpenNeuro to their compute environment using one command. It is recommended to clone the data into a dedicated project folder.

```
datalad clone https://github.com/OpenNeuroDatasets/ds002790.
git AOMIC-PIOP2
```

The dataset comprises the raw data in BIDS format and many processed derivatives, but one should be aware that only the metadata is directly available via this clone. The user will also have to download the file content for the actual CAT12 processing, using DataLad commands. The following lines of code describe how to download one subject's T1w data, then copy it to a CAT12_derivatives folder for processing and finally drop the file content from the input dataset after use to minimize the storage footprint.

```
 # download the data
  datalad get -d AOMIC-PIOP2 AOMIC-PIOP2/sub-0111/anat/sub-
0111_T1w.nii.gz
 # create an output directory and copy the T1w file there
 mkdir -p CAT12_derivatives/TEST_sub-0111
 cp AOMIC-PIOP2/sub-0111/anat/sub-0111_T1w.nii.gz CAT12_deri-
vatives/TEST_sub-0111/
 # delete/drop the local version of the file as we can get it
back anytime
  datalad drop --what filecontent --reckless kill -d AOMIC-
PIOP2 AOMIC-PIOP2/sub-0111
```

Now, the software environment is setup (B) for CAT12 standalone with Matlab Compiler Runtime (MCR). CAT12 (e.g., CAT12.8.1_r2042_R2017b_MCR) is downloaded as described here and unpacked to the working directory. Then the matching MCR (version R2017b 9.3) is downloaded and installed from here. The software setup should be tested by starting the now available standalone version of SPM12 including CAT12. In the CAT12

folder, the "run_spm12.sh" script runs by adding the full path to the MCR installation as argument. With the following command, the standard SPM12 graphical user interface is initiated and under the Toolbox button "cat12" will be started already in expert mode, again without a full Matlab license. The command needs the *full path* to MCR, but in reproducible scripts it is generally recommended to use relative paths from the dataset or code folder instead of full paths as the latter is machine- and user-specific.

```
# start SPM12 graphical user interface in PET/VBM mode
./run_spm12.sh </FULLPATH/MCR/v93>
```

A test subject can reproducibly be run via the command line interface, using the CAT12 standalone syntax, which is well described on the CAT12 website. The location of the MCR setup and the CAT12 *version of choice* can be provided via environmental variables with *full paths* as follows:

```
# provide full paths to pipeline setup and make it available
MCRROOT=<//FULLPATH/MCR/v93>
SPMROOT=<//FULLPATH/CAT12.8.1_r2042_R2017b_MCR_Linux>
export MCRROOT SPMROOT
```

Now, the first test processing is run by executing the *cat_standalone.sh* script together with a batch file (*-b*) that describes the settings of how the target T1w scan will be segmented:

```
# start CAT12 standalone with simple segment batch for example
subject
 ./CAT12.8.1_r2042_R2017b_MCR_Linux/standalone/cat_standa-
lone.sh \
 -b ./CAT12.8.1_r2042_R2017b_MCR_Linux/standalone/cat_standa-
lone_segment.m \
 CAT12_derivatives/TEST_sub-0111/sub-0111_T1w.nii.gz
```

CAT12 segmentation takes <20 min per subject and produces ~60 MB of data in the following output structure:

```
CAT12_derivatives
└── TEST_sub-0111
├── label
│   ├── catROI_sub-0111_T1w.mat
│   └── catROI_sub-0111_T1w.xml
├── mri
│   ├── mwp1sub-0111_T1w.nii
│   ├── mwp2sub-0111_T1w.nii
│   ├── p0sub-0111_T1w.nii
│   ├── wmsub-0111_T1w.nii
```

```
│   └── y_sub-0111_T1w.nii
├── report
│   ├── catlog_sub-0111_T1w.txt
│   ├── catreportj_sub-0111_T1w.jpg
│   ├── catreport_sub-0111_T1w.pdf
│   ├── cat_sub-0111_T1w.mat
│   └── cat_sub-0111_T1w.xml
└── sub-0111_T1w.nii.gz
```

The *report* folder contains full processing logs in different file formats and a pdf/jpg report sheet for quick quality assessment. For VBM analyses in the *mri* folder, modulated gray (mwp1) and white matter (mwp2) images are provided in template space. Also available are a partial volume image (p0) in native subject space, a denoised normalized brain image (wm) in template space and the transformation/warp field (y_) from native to template space. Regional volume estimates of several brain parcellations are found in the *label* folder.

For convenience, *bootstrap scripts* are provided to be cloned from here for *Linux (and macOS)* that will do all of the above reproducibly from scratch. The following commands will download those to our workspace directory and run them.

```
# clone Computational Anatomy Tutorial from OSF
 datalad clone https://osf.io/ydxw7/ ca_tutorial
```

The script ca_minimal.sh will set up the environment in the current working directory and run the test subject as described. It is recommended to have a close look at the script to follow what is happening.

```
./ca_tutorial/ca_minimal.sh
```

After carefully checking the output of the test subject, parallelization can then be tested by executing the prepared script, which will process nine subjects with three in parallel in the background.

```
 ./run_3x3catjobs.sh
```

3.2. *An intermediate approach to reproducible image analysis is to share not only the list of ingredients but access to* (A) *Input Data*, (B) *Processing Pipeline* setup, and (C) *Code*. An emerging standard is the publication of intermediate processing results for re-execution of statistical analysis and figure creation, which is particularly useful in genetic or big data modeling analysis [14]. Here, scripts for automatic execution

of all processing steps are provided. The main difference to the minimal approach consists in the creation of a DataLad dataset to capture the ingredients A/B/C and intermediate results, which is extensively and clearly described in the *DataLad handbook*. DataLad uses Git under the hood (*see* Chapter 5) to track not only code but also arbitrarily sized data providing a re-executable framework to reproduce the exact succession of computations captured in the git history. To avoid redundancies in this tutorial, only additional commands are documented. Of note, the MCR setup cannot be shared due to the license conditions and must be installed individually, so it is not part of the DataLad dataset.

```
# define CAT12 version
 CAT12_version="CAT12.8.1_r2042_R2017b_MCR_Linux"
 # create Datalad yoda dataset for intermediate cat12 pipeline
and enter
 datalad create -c yoda intermed_cat12
 cd intermed_cat12
 # add input dataset as subdataset (!) from OpenNeuro AOMIC-
PIOP2
 datalad clone -d . https://github.com/OpenNeuroDatasets/
ds002790.git inputs/AOMIC-PIOP2
 # setup CAT12 standalone pipeline in code/ folder
 wget "https://www.neuro.uni-jena.de/cat12/${CAT12_version}.
zip"
 unzip ${CAT12_version}.zip -d code
 rm -f ${CAT12_version}.zip
 # register the CAT12.8.1 standalone pipeline in the dataset
 datalad save -m "add CAT12.8.1 r2042 pipeline with git" code/
```

After providing the location of both CAT12 and MCR setup and preparing the T1w image in a derivatives folder, execute the example subject and capture result with DataLad:

```
 # provide full paths to pipeline setup and make it available
 MCRROOT=<//FULLPATH/MCR/v93>
 SPMROOT=<//FULLPATH/CAT12.8.1_r2042_R2017b_MCR_Linux>
 export MCRROOT SPMROOT


 # download the data
 datalad get inputs/AOMIC-PIOP2/sub-0111/anat/sub-0111_T1w.
nii.gz
 # create an outputs directory and copy the T1w file there
 mkdir -p CAT12_derivatives/TEST_sub-0111
 cp inputs/AOMIC-PIOP2/sub-0111/anat/*T1w.nii.gz CAT12_deri-
```

```
vatives/TEST_sub-0111/
 # delete/drop the local version of the file as we can get it
back anytime
 datalad drop --what filecontent --reckless kill inputs/AOMIC-
PIOP2/sub-0111

 # execute CAT12 standalone with MCR
 ./code/${CAT12_version}/standalone/cat_standalone.sh \
  -b  ./code/${CAT12_version}/standalone/cat_standalone_seg-
ment.m \
  -a  "matlabbatch{1}.spm.tools.cat.estwrite.output.surface  =
0" \
 CAT12_derivatives/TEST_sub-0111/sub-0111_T1w.nii.gz
  # track CAT12 derivatives and standalone setup in datalad
dataset
  datalad save -m "save test subject & standalone setup"
CAT12_derivatives code
```

Now, the dataset contains all necessary parts ABC (and MCR outside) to conduct reproducible, full-scale VBM processing of the AOMIC PIOP2 dataset and registering all results. CAT12 also includes a feature to locally parallelize processing over subjects, wrapping the former command in another script. All T1w images have to be prepared and copied into the CAT12_derivatives folder before running CAT12:

```
  # download first 9 subjects to process 3x in parallel
(sub-0001..9; 3x3jobs)
 datalad get inputs/AOMIC-PIOP2/sub-000*/anat/*T1w.nii.gz


# copy T1w images of 9 subjects to CAT_derivatives folder
 for sub in inputs/AOMIC-PIOP2/sub-000*; do
 sub=$(basename $sub); mkdir -p CAT12_derivatives/${sub};
  cp  inputs/AOMIC-PIOP2/${sub}/anat/*T1w.nii.gz  CAT12_deriva-
tives/${sub};
 done
# run CAT12 3x in parallel (-p 3) writing logs to folder (-l)
 ./code/${CAT12_version}/standalone/cat_parallelize.sh -p 3 -l
. -c "
 ./code/${CAT12_version}/standalone/cat_standalone.sh \
  -b  ./code/${CAT12_version}/standalone/cat_standalone_seg-
ment.m" \
 CAT12_derivatives/sub-*/*T1w.nii.gz
 # track CAT12 derivatives in datalad dataset after processing
  datalad save -m "save CAT12 derivatives for 9 subjects"
CAT12_derivatives
```

This whole procedure is written and described in the following *bootstrap script*.

```
./ca_tutorial/ca_intermed.sh
```

After carefully checking the output of the test subject, parallelization can be tested be executing the prepared script from within the dataset, which will process nine subjects with three in parallel in the background and capture the outcome in the DataLad dataset.

```
cd intermed_cat12
./code/run_3x3catjobs.sh
```

3.3. *Finally, a comprehensive approach to reproducible image analysis is the FAIRly big workflow* [11], which consists of an automatic setup procedure via a *bootstrap script*, which tracks all ingredients ABC as well as the results of each processing step. Additionally, the workflow contains automatic means for parallel execution of whole dataset processing using container technology and produces re-executable run records of each processing step to enable fully automatic reproducibility. Of note, for building the Singularity container *sudo rights* are needed. For this more complex workflow the following additional dependencies are needed:

GNU-parallel	
datalad-container extension:	pip install datalad-container
Singularity container	

To properly introduce the workflow, a few essential concepts are presented below, which are directly taken from the original publication: "*FAIRly big: A framework for computationally reproducible processing of large-scale data*" [12], with permission of the authors.

DataLad Dataset

DataLad's core data structure is the dataset. On a technical level, it is a joint Git/git-annex (REF) repository. Conceptually, it is an overlay data structure that is particularly suited to address data integration challenges. It enables users to version control files of any size or type, track and transport files in a distributed network of dataset clones, as well as record and re-execute actionable process provenance on the genesis of file content. DataLad datasets have the ability to retrieve or drop registered, remote file content on demand with single file granularity. This is possible based on a lean record of file identity and file availability (via checksum and URLs) irrespective of the true file size. A user does not need to be aware of the actual download source of a file's content, as precise file identity

is automatically verified regardless of a particular retrieval method, and the specification of redundant sources is supported. These technical features enable the implementation of infrastructure-agnostic data retrieval and deposition logic in user code.

A Clone (Git concept) is a copy of a DataLad dataset that is linked to its origin dataset and its history. The clones are lightweight and can typically be obtained within seconds, as they are primarily comprised of file identity and availability records. DataLad enables synchronization of content between clones and, hence, the propagation of updates.

A Branch (Git concept) is an independent segment of a DataLad dataset's history. It enables the separation of parallel developments based on a common starting point. Branches can encompass arbitrarily different modifications of a dataset. In a typical collaborative development or parallel processing routine, changes are initially introduced in branches and are later consolidated by merging them into a mainline branch.

Nesting A DataLad dataset can also contain other DataLad datasets. Analog to file content, this linkage is implemented using a lightweight dataset identity and availability record (based on Git's submodules). This nesting enables flexible (re-)use of datasets in a different context. For example, it allows for the composition of a project directory from precisely versioned, modular units that unambiguously link all inputs of a project to its outcomes. Nesting offers actionable dataset linkage at virtually no disk space cost, while providing the same on-demand retrieval and deposition convenience as for file content operations because DataLad can work with a hierarchy of nested datasets as if they are a single monolithic repository. When a DataLad dataset B is nested inside DataLad dataset A, we also refer to A as the superdataset and to B as a subdataset. A superdataset can link any number of subdatasets, and datasets can simultaneously be both super- and subdataset.

RIA Store A file system-based store for DataLad datasets with minimal server-side software requirements (in particular no DataLad, no git-annex, and Git only for specific optional features) (REF). These stores offer inode minimization (using indexed 7-zip archives). A dataset of arbitrary size and number of files can be hosted while consuming fewer than 25 inodes, while nevertheless offering random read access to individual files at a low and constant latency independent of the actual archive size. Combined with optional file content encryption and compression, RIA ("Remote Indexed Archive") stores are particularly suited for staging large-scale, sensitive data to process on HPC resources.

The following script automatically sets up the FAIRly Big workflow for fully reproducible processing of large-scale structural

MRI data tracking results and execution with DataLad enabling automatic re-execution of CAT12 processing with "datalad run":

```
./ca_tutorial/ca_FAIRlyBig.sh
```

The main differences to the other approaches in 3.1 and 3.2 are the use of a Singularity container as pipeline and directly tracking each compute job with DataLad in separate ephemeral *clones* of the entire workflow setup. This is made possible by pushing the workflow setup as *DataLad dataset* to local repositories called *RIA store* (*see* Above). The workflow dataset contains all Code **(C)** and references both the Input Data **(A)** and the containerized Processing Pipeline **(B)** as *nested* subdatasets ready to be cloned. Two identical repositories are created for the workflow dataset, with an input *RIA store* to clone from and an output *RIA store* to push processing results to. For each compute job, an independent clone of the workflow from the *input RIA* is created in a separate, temporal location (e.g., tmp/), where the compute job automatically downloads the specific input data needed, and runs CAT12 in an individual container instance. Processing results are tracked in a job-specific *branch* and pushed to the output *RIA* before the temporal clone is deleted, which makes sure that no partial results will be saved in the final *DataLad dataset*. This full separation of individual compute jobs enables a full parallelization of the workflow, while tracking input, pipeline, code and all results in one place. Template submission scripts have been built for HTCondor and SLURM job scheduling systems, which can be tailored to any compute cluster setup available to the User. With the setup of the AOMIC-PIOP2_cna_cat12.8.1 analysis dataset, instructions for running a test subject are printed in the terminal. Here we use the ENIGMA CAT12 processing batch, which conducts more comprehensive data processing and produces more output.

```
# enter into fully setup Datalad dataset:
cd AOMIC-PIOP2_cna_cat12.8.1
# run test subject in the FAIRly Big workflow
./code/process.sub sub-0222
```

After successful processing, the results dataset has to be consolidated by running the available results.merger script, which merges all individual job *branches* into the main *branch* by performing an octopus merge. The FAIRly big workflow saves/pushed all processing output directly in the *output RIA*, of which the dataset is a mere clone. The consolidation script updates the local dataset by pulling the full git history from the output RIA, collecting all subjects and checking their file availability in the RIA.

```
./code/results.merger
```

This consolidation step reveals that "No known copies exist" of a few files like denoised input images and in particular of the spatial transformations from subject to template space (y_*) and the inverse (iy_*). This was an explicit decision setting up the workflow for a large dataset, as those files are comparably big in size and rarely used, as all images for standard analyses are available in template space. CAT12 processing was tracked with DataLad containers-run including a comprehensive run record in the git history necessary for datalad rerun to fully automatically re-execute the workflow creating the missing files. The following command will rerun the last subject/job and make all files available:

```
# rerun CAT12 job by using the latest commit hash
datalad rerun $(git rev-parse HEAD)
```

Data of the test subject should be inspected by downloading the data from the *output RIA:*

```
datalad get sub-0222
```

The CAT12 ENIGMA pipeline creates many more files, which are documented in the comprehensive CAT12-help including surface projection and cortical thickness estimations of different surface atlases.

When the results of the test subject are satisfactory, again nine subjects with three in parallel can be started in the background using the following command.

```
parallel -j3 ./code/process.sub sub-000{} ::: {1..9}
```

The whole dataset can be submitted for local processing with four subjects in parallel (-j4) using GNU-parallel, which takes care that of all 200 subjects only four will run at any given moment.

```
parallel -j4 ./code/process.sub {/} ::: inputs/AOMIC-PIOP2/
sub*
```

The *output RIA* is locally hosted in this example but can also be stored anywhere on a file system or the web using OSF or other web hosting services. In this example the RIA looks as follows:

```
dataladstore
├── 954
│   └── 6b411-702b-4839-8773-50101d8f2cd9
├── alias
│   ├── AOMIC-PIOP2_ca_cat12.8.1 -> ../954/6b411-702b-4839-
8773-50101d8f2cd9
│   └── cat12.8_container -> ../e64/188f5-1330-4729-90a6-
7dd4ba71d529
```

```
├── e64
│   └── 188f5-1330-4729-90a6-7dd4ba71d529
├── error_logs
├── inputstore
│   ├── 954
│   ├── alias
│   ├── error_logs
│   └── ria-layout-version
└── ria-layout-version
```

The datasets can be cloned from RIA by using the alias in the following commands:

```
datalad clone "ria+file://<FULLPATH>/dataladstore#~AOMIC-PIOP2_ca_cat12.8.1"
 datalad clone "ria+file://<FULLPATH>/dataladstore#~cat12.8_-container"
```

To push the whole dataset to OSF the datalad-osf extension is needed and after setting up the credentials any dataset can be published on OSF with the data present in the local clone:

```
 datalad create-sibling-osf --title AOMIC-PIOP2_ca_cat12.8.1 -s osf \
 --mode export --category data --tag reproducibility --public
 datalad get .
 datalad push --to osf
```

For comparison the fully processed dataset can be cloned using the following command:

```
 datalad clone https://osf.io/9gtsf/ AOMIC-PIOP2_ca_cat12.8.1
```

Statistical data analyses are described well in the CAT12 help including detailed instructions about how to use the CAT12 graphical user interface. For very large datasets, the amount of RAM and compute time needed for estimating large GLMs can increase immensely, but the model setup is largely identical to smaller dataset.

4 Conclusion

Here, we have demonstrated how to use CAT12 to produce reproducible computational anatomy pipelines. We have shown how analyses can be scaled up to, in theory, analyze thousands of MRI images in parallel while keeping tracking of input files and maintaining the pipeline, code and all results in one place.

References

1. Ashburner J, Friston KJ (2000) Voxel-based morphometry—the methods. NeuroImage 11(6):805–821. https://doi.org/10.1006/nimg.2000.0582
2. McLaren DG, Kosmatka KJ, Kastman EK, Bendlin BB, Johnson SC (2010) Rhesus macaque brain morphometry: a methodological comparison of voxel-wise approaches. Methods 50(3):157–165. https://doi.org/10.1016/j.ymeth.2009.10.003
3. Evans AC, Kamber M, Collins DL, MacDonald D (1994) An MRI-based probabilistic atlas of neuroanatomy. In: Shorvon SD, Fish DR, Andermann F, Bydder GM, Stefan H (eds) Magnetic resonance scanning and epilepsy. Springer US, Boston, pp 263–274
4. Antonopoulos G, More S, Raimondo F, Eickhoff SB, Hoffstaedter F, Patil KR (2023) A systematic comparison of VBM pipelines and their application to age prediction. NeuroImage 279:120292. https://doi.org/10.1016/j.neuroimage.2023.120292
5. Zhou X, Wu R, Zeng Y, Qi Z, Ferraro S, Xu L, Zheng X, Li J, Fu M, Yao S, Kendrick KM, Becker B (2022) Choice of Voxel-based Morphometry processing pipeline drives variability in the location of neuroanatomical brain markers. Commun Biol 5(1):913. https://doi.org/10.1038/s42003-022-03880-1
6. Kamalian A, Khodadadifar T, Saberi A, Masoudi M, Camilleri JA, Eickhoff CR, Zarei M, Pasquini L, Laird AR, Fox PT, Eickhoff SB, Tahmasian M (2022) Convergent regional brain abnormalities in behavioral variant frontotemporal dementia: a neuroimaging meta-analysis of 73 studies. Alzheimers Dement 14(1):e12318. https://doi.org/10.1002/dad2.12318
7. Kharabian Masouleh S, Eickhoff SB, Hoffstaedter F, Genon S, Alzheimer's Disease Neuroimaging Initiative (2019) Empirical examination of the replicability of associations between brain structure and psychological variables. eLife 8:e43464. https://doi.org/10.7554/eLife.43464
8. Bennett CM, Baird AA (2006) Anatomical changes in the emerging adult brain: a voxel-based morphometry study. Hum Brain Mapp 27(9):766–777. https://doi.org/10.1002/hbm.20218
9. Luders E, Gaser C, Jancke L, Schlaug G (2004) A voxel-based approach to gray matter asymmetries. NeuroImage 22(2):656–664. https://doi.org/10.1016/j.neuroimage.2004.01.032
10. Good CD, Johnsrude IS, Ashburner J, Henson RNA, Friston KJ, Frackowiak RSJ (2001) A Voxel-Based Morphometric Study of Ageing in 465 Normal Adult Human Brains. NeuroImage 14(1):21–36. https://doi.org/10.1006/nimg.2001.0786
11. Snoek L, Van Der Miesen MM, Beemsterboer T, Van Der Leij A, Eigenhuis A, Steven Scholte H (2021) The Amsterdam Open MRI Collection, a set of multimodal MRI datasets for individual difference analyses. Sci Data 8(1):85. https://doi.org/10.1038/s41597-021-00870-6
12. Wagner AS, Waite LK, Wierzba M, Hoffstaedter F, Waite AQ, Poldrack B, Eickhoff SB, Hanke M (2022) FAIRly big: a framework for computationally reproducible processing of large-scale data. Sci Data 9(1):80. https://doi.org/10.1038/s41597-022-01163-2
13. Halchenko Y, Meyer K, Poldrack B, Solanky D, Wagner A, Gors J, MacFarlane D, Pustina D, Sochat V, Ghosh S, Mönch C, Markiewicz C, Waite L, Shlyakhter I, De La Vega A, Hayashi S, Häusler C, Poline J-B, Kadelka T, Skytén K, Jarecka D, Kennedy D, Strauss T, Cieslak M, Vavra P, Ioanas H-I, Schneider R, Pflüger M, Haxby J, Eickhoff S, Hanke M (2021) DataLad: distributed system for joint management of code, data, and their relationship. JOSS 6(63):3262. https://doi.org/10.21105/joss.03262
14. Bethlehem RAI, Seidlitz J, White SR, Vogel JW, Anderson KM, Adamson C, Adler S, Alexopoulos GS, Anagnostou E, Areces-Gonzalez-A, Astle DE, Auyeung B, Ayub M, Bae J, Ball G, Baron-Cohen S, Beare R, Bedford SA, Benegal V, Beyer F, Blangero J, Blesa Cábez M, Boardman JP, Borzage M, Bosch-Bayard JF, Bourke N, Calhoun VD, Chakravarty MM, Chen C, Chertavian C, Chetelat G, Chong YS, Cole JH, Corvin A, Costantino M, Courchesne E, Crivello F, Cropley VL, Crosbie J, Crossley N, Delarue M, Delorme R, Desrivieres S, Devenyi GA, Di Biase MA, Dolan R, Donald KA, Donohoe G, Dunlop K, Edwards AD, Elison JT, Ellis CT, Elman JA, Eyler L, Fair DA, Feczko E, Fletcher PC, Fonagy P, Franz CE, Galan-Garcia L, Gholipour A, Giedd J, Gilmore JH, Glahn DC, Goodyer IM, Grant PE, Groenewold NA, Gunning FM, Gur RE, Gur RC, Hammill CF, Hansson O, Hedden T, Heinz A, Henson RN, Heuer K, Hoare J, Holla B, Holmes AJ, Holt R, Huang H, Im K, Ipser J, Jack CR,

Jackowski AP, Jia T, Johnson KA, Jones PB, Jones DT, Kahn RS, Karlsson H, Karlsson L, Kawashima R, Kelley EA, Kern S, Kim KW, Kitzbichler MG, Kremen WS, Lalonde F, Landeau B, Lee S, Lerch J, Lewis JD, Li J, Liao W, Liston C, Lombardo MV, Lv J, Lynch C, Mallard TT, Marcelis M, Markello RD, Mathias SR, Mazoyer B, McGuire P, Meaney MJ, Mechelli A, Medic N, Misic B, Morgan SE, Mothersill D, Nigg J, Ong MQW, Ortinau C, Ossenkoppele R, Ouyang M, Palaniyappan L, Paly L, Pan PM, Pantelis C, Park MM, Paus T, Pausova Z, Paz-Linares D, Pichet Binette A, Pierce K, Qian X, Qiu J, Qiu A, Raznahan A, Rittman T, Rodrigue A, Rollins CK, Romero-Garcia R, Ronan L, Rosenberg MD, Rowitch DH, Salum GA, Satterthwaite TD, Schaare HL, Schachar RJ, Schultz AP, Schumann G, Schöll M, Sharp D, Shinohara RT, Skoog I, Smyser CD, Sperling RA, Stein DJ, Stolicyn A, Suckling J, Sullivan G, Taki Y, Thyreau B, Toro R, Traut N, Tsvetanov KA, Turk-Browne NB, Tuulari JJ, Tzourio C, Vachon-Presseau É, Valdes-Sosa MJ, Valdes-Sosa PA, Valk SL, Van Amelsvoort T, Vandekar SN, Vasung L, Victoria LW, Villeneuve S, Villringer A, Vértes PE, Wagstyl K, Wang YS, Warfield SK, Warrier V, Westman E, Westwater ML, Whalley HC, Witte AV, Yang N, Yeo B, Yun H, Zalesky A, Zar HJ, Zettergren A, Zhou JH, Ziauddeen H, Zugman A, Zuo XN, 3R-BRAIN, AIBL, Rowe C, Alzheimer's Disease Neuroimaging Initiative, Alzheimer's Disease Repository Without Borders Investigators, Frisoni GB, CALM Team, Cam-CAN, CCNP, COBRE, cVEDA, ENIGMA Developmental Brain Age Working Group, Developing Human Connectome Project, FinnBrain, Harvard Aging Brain Study, IMAGEN, KNE96, The Mayo Clinic Study of Aging, NSPN, POND, The PREVENT-AD Research Group, Binette AP, VETSA, Bullmore ET, Alexander-Bloch AF (2022) Brain charts for the human lifespan. Nature 604(7906):525–533. https://doi.org/10.1038/s41586-022-04554-y