BIDS Apps: Improving ease of use, accessibility and reproducibility of neuroimaging data analysis methods

Krzysztof J. Gorgolewski (krzysztof.gorgolewski@gmail.com)[1], Fidel Alfaro-Almagro (falmagro@fmrib.ox.ac.uk)[13],Tibor Auer (tibor.auer@rhul.ac.uk)[15], Pierre Bellec (pierre.bellec@criugm.qc.ca)[18,21], Mihai Capotă (mihai.capota@intel.com)[20], M. Mallar Chakravarty (mallar@cobralab.ca)[22,23], Nathan W. Churchill (nchurchill.research@gmail.com)[25], R. Cameron Craddock (ccraddock@nki.rfmh.org)[9,10], Gabriel A. Devenyi (gdevenyi@gmail.com)[22,23], Anders Eklund (anders.eklund@liu.se)[2,3,4], Oscar Esteban (phd@oscaresteban.es)[1], Guillaume Flandin (g.flandin@ucl.ac.uk)[8], J. Swaroop Guntupalli (swaroopgj@gmail.com)[12], Mark Jenkinson (mark@fmrib.ox.ac.uk)[13], Anisha Keshavan (anisha.keshavan@ucsf.edu)[11], Gregory Kiar (gkiar@jhu.edu)[5,6], Pradeep Reddy Raamana (praamana@research.baycrest.org)[16,17], David Raffelt (david.raffelt@florey.edu.au)[7], Christopher J. Steele (steele.christopher.j@gmail.com)[22,23], Pierre-Olivier Quirion (poq@criugm.qc.ca)[18], Robert E. Smith (robert.smith@florey.edu.au)[7], Stephen C. Strother (sstrother@research.baycrest.org)[16,26], Gaël Varoquaux (gael.varoquaux@inria.fr)[14], Tal Yarkoni (tyarkoni@utexas.edu)[19], Yida Wang (yida.wang@intel.com)[20], Russell A. Poldrack (russpold@stanford.edu)[1]

1 Department of Psychology, Stanford University, Stanford, CA, 94305
2 Department of Biomedical Engineering, Linköping University, Linköping, Sweden
3 Department of Computer and Information Science, Linköping University, Linköping, Sweden
4 Center for Medical Image Science and Visualization (CMIV), Linköping University, Linköping, Sweden
5 Center for Imaging Science, Johns Hopkins University, Baltimore, MD
6 Department of Biomedical Engineering, Johns Hopkins University, Baltimore, MD
7 Florey Institute of Neuroscience and Mental Health, Melbourne, Victoria, Australia
8 Wellcome Trust Centre for Neuroimaging, London, UK
9 Computational Neuroimaging Lab, Center for Biomedical Imaging and Neuromodulation, Nathan S. Kline Institute for Psychiatric Research, Orangeburg, NY
10 Center for the Developing Brain, Child Mind Institute, New York, NY
11 UC Berkeley-UCSF Graduate Program in Bioengineering, San Francisco, CA, US
12 Department of Psychological and Brain Sciences, Dartmouth College, Hanover, NH, US
13 Oxford Centre for Functional Magnetic Resonance Imaging of the Brain (FMRIB), Oxford University, UK.
14 Parietal team, INRIA Saclay Ile-de-France, Palaiseau, France
15 Department of Psychology, Royal Holloway University of London. Egham, UK
16 Rotman Research Institute, Baycrest Health Sciences, Toronto, ON, Canada.
17 Department of Medical Biophysics, University of Toronto, Toronto, ON, Canada.
18 Centre de Recherche de l'Institut Universitaire Gériatrique de Montréal
19 Department of Psychology, University of Texas at Austin, Austin, TX
20 Parallel Computing Lab, Intel Corporation, Santa Clara, CA & Hillsboro, OR
21 Department of computer science and operations research, Université de Montréal, CA
22 Douglas Mental Health University Institute, McGill University, Montreal, CA
23 Department of Psychiatry McGill University, Montreal, CA
24 Department of Biological and Biomedical Engineering, McGill University, Montreal, CA
25 Keenan Research Centre of the Li Ka Shing Knowledge Institute, St. Michael's Hospital
26 Department of Medical Biophysics, University of Toronto, Toronto, ON, Canada

Abstract

In this work, we introduce a framework for creating, testing, versioning and archiving portable applications for analyzing neuroimaging data organized and described in compliance with the Brain Imaging Data Structure (BIDS). The portability of these applications (BIDS Apps) is achieved by using container technologies that encapsulate all binary and other dependencies in one convenient package. BIDS Apps run on all three major operating systems with no need for complex setup and configuration and thanks to the richness of the BIDS standard they require little manual user input. Previous containerized data processing solutions were limited to single user environments and not compatible with most multi tenant High Performance Computing systems. BIDS Apps overcome this limitation by taking advantage of the Singularity container technology. As a proof of concept, this work is accompanied by 18 ready to use BIDS Apps, packaging a diverse set of commonly used neuroimaging algorithms.

Introduction

The last 25 years have witnessed a proliferation of methods for imaging the human brain (including structural, diffusion and functional Magnetic Resonance Imaging, Positron Emission Tomography, Electroencephalography and Magnetoencephalography). These methods have been accompanied by literally thousands of different algorithms for signal denoising, normalization, feature extraction, and statistical analysis. Modern analysis pipelines for neuroimaging often consist of dozens of steps and rely on software developed by multiple external groups with each group often developing their own idiosyncratic parameter settings even when using the same software packages. The increasing complexity of neuroimaging data analysis has led to many discoveries, and the flexibility provided by the plethora of feature extraction methods has allowed cognitive and clinical neuroscientists to develop new theories about the relationships between brain and behavior in healthy and diseased populations.

However, due to the intrinsic heterogeneity of scientific software (arising from the fact that it rarely is developed for widespread distribution), installing, configuring, and running many of the available methods is often difficult. Most neuroimaging software packages run natively on Linux (Hanke & Halchenko 2011) and (to a lesser extent) Mac OS X; however, Windows users are often left without that option. Operating system aside, many scientific packages depend on external libraries (often requiring a particular, sometimes outdated version), and require complex configurations of environment variables and/or data files.

There have been some attempts in the field of neuroinformatics to solve this issue. The most notable is the NeuroDebian project (Halchenko & Hanke 2012), which provides Debian and Ubuntu Linux distributions with packages containing many of the popular neuroimaging software tools. Installing packages prepared by the NeuroDebian team is very easy, as it can be performed with the built-in Debian/Ubuntu package management system. However, this solution only applies to Linux systems, as Mac OS X and Windows users are required to install Linux inside a Virtual Machine (VM). Additionally, creating new Debian/Ubuntu packages is a non-trivial task, which may limit the rate at which new software is added to NeuroDebian.

Another consequence of these deployment and installation issues is that they make it very difficult to perfectly recreate analysis pipelines, which exacerbates reproducibility issues in neuroimaging. This is partly due to journal space limitations, which typically preclude a full accounting of the scientific software stack used to generate results. But even knowing the versions of all of the software tools employed in an analysis is rarely sufficient to completely reproduce a workflow. Studies have shown that operating system type, version, and even hardware architecture can influence results in a significant manner (Mackenzie-Graham et al. 2008; Gronenschild et al. 2012; Glatard et al. 2015). Such issues are troubling in and of themselves, and call for thorough investigations into the sources of this variability. They also raise serious practical concerns for extended longitudinal studies that need to maintain the same software (and sometimes even hardware) stack along the time span of an experiment (e.g. a software upgrade midway through the analysis could lead to spurious differences between groups processed at different points in time).

One potential solution to ease the deployment problem proposed in the bioinformatics community is to create Virtual Machines (VMs) capturing all of the necessary dependencies for a workflow (Angiuoli et al. 2011; Krampis et al. 2012). Running a Virtual Machine does however come with significant performance overhead; furthermore, only a few High Performance Computing (HPC) systems (which have become the primary computational resource for many academics in the last decade) allow their users to run VMs on large clusters. Building on the virtual machine concept, a more lightweight solution has recently become more prevalent in the industry, known as 'operating-system-level-virtualization' or (more commonly) *containers*. In contrast to VMs, containers share the same kernel with the host operating system and thus deliver much better performance. The bioinformatics community has once again been at the forefront of adoption of this new technology, with the aim of improving research reproducibility (Folarin et al. 2015; Moreews et al. 2015; Devisetty et al. 2016; Belmann et al. 2015). Most proposed solutions have been based on a particular implementation of the container concept: 'Docker', which, due to its kernel and security requirements, is difficult or impossible to use in a multi-tenant environment such as an HPC system (which is often the most cost effective computational resource available to researchers).

Finally, most existing neuroimaging data processing pipelines expect input datasets to be organized and described in different and idiosyncratic ways. To account for variability in input data organization and a lack of consensus in terms of metadata description, data processing workflows often require users to input metadata manually - in a different fashion for each pipeline. This expandable step can cause errors and lead to incorrect results, while also making it harder to integrate processing pipelines into automated analysis platforms such as those developed to provide "Science as a Service" (Jordan et al. 2011).

After careful evaluation of existing solutions for the specific problems faced by the neuroimaging community, we have developed a framework for sharing and executing neuroimaging analysis pipelines that improves ease of use, accessibility and reproducibility for users of all three major operating systems, as well as for researchers using HPC or cloud computing systems. The framework overcomes Docker's inability to run on multi-tenant HPC systems by capitalizing on the Singularity (Kurtzer 2016) container technology developed specifically for HPC use. To minimize the number of manual inputs required from researchers--and hence reduce the number of errors arising from misinterpretation of those inputs--the framework capitalizes on the recently introduced Brain Imaging Data Structure (BIDS) standard (Gorgolewski et al. 2016) for organizing and describing datasets. Correspondingly, the proposed framework is named BIDS Apps. We describe here the anatomy of a BIDS App, the infrastructure used for building, testing and archiving BIDS Apps, as well as steps necessary to run BIDS Apps in various scenarios.

Results

What is a BIDS App?

A BIDS App is a container image[1] capturing a neuroimaging pipeline that takes a BIDS-formatted dataset as input. Each BIDS App has the same core set of command line arguments, making them easy to run and integrate into automated platforms. BIDS Apps are constructed in a way that does not depend on any software outside of the container mage other than the container engine.

BIDS Apps rely upon two technologies for container computing:

1. Docker - for building, hosting as well as running containers on local hardware (running Windows, Mac OS X or Linux) or in the cloud.
2. Singularity - for running containers on HPCs (Kurtzer 2016).

BIDS Apps are deposited in the Docker Hub (http://hub.docker.com) repository, making them openly accessible. Each app is versioned and all of the historical versions are available to download. By reporting the BIDS App name and version in a manuscript, authors can provide others with the ability to exactly replicate their analysis workflow.

Docker is used for its excellent documentation, maturity, and the Docker Hub service for storage and distribution of the images. Docker containers are easily run on personal computers and cloud services. However, the Docker Engine was originally designed to run different components of web services (HTTP servers, databases etc.) using cloud resources. Docker thus requires root or root-like permissions, as well as modern versions of Linux kernel (to perform user mapping and management of network resources); though this is not a problem in context of renting cloud resources (which are not shared with other users), it makes it difficult or impossible to use in a multi-tenant environment such as an HPC system, which is often the most cost effective computational resource available to researchers. Singularity, on the other hand, is

[1] Images vs. containers. A container image is a serialization of binary dependencies and can be used to run containers. In other words a container is a particular instantiation of an image. There can be many running containers of the same image.

a unique container technology designed from the ground up with the encapsulation of binary dependencies and HPC use in mind. Its main advantage over Docker is that it does not require root access for container execution and thus is safe to use on multi-tenant systems. In addition, it does not require recent Linux kernel functionalities (such as namespaces, cgroups and capabilities), making it easy to install on legacy systems.

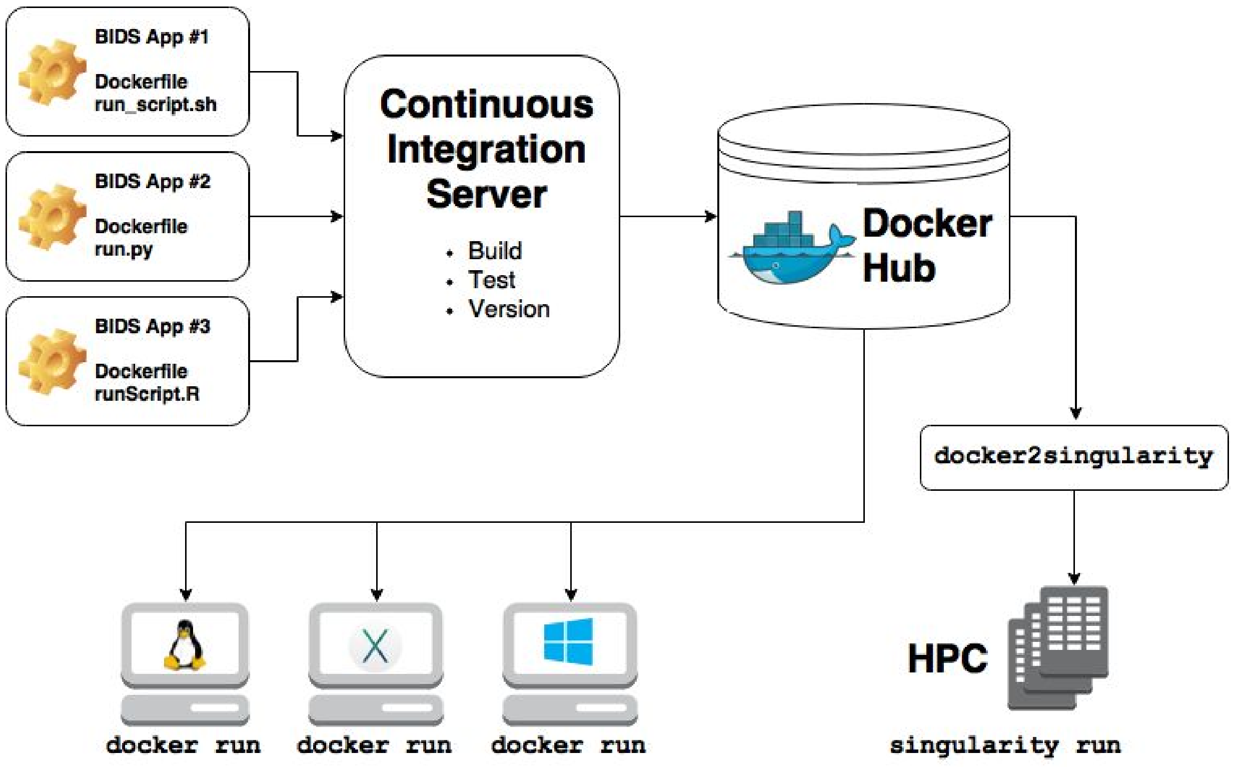

Figure 1. Overview of the creation and use of BIDS Apps.

How to create a BIDS App?

BIDS Apps Forge

Inspired by the conda-forge project (https://conda-forge.github.io/), BIDS Apps development is centered around a GitHub organization (http://github.com/BIDS-Apps) that maintains a code repository for each BIDS App (see Figure 1). Every repository hosts a Dockerfile describing how to build the container image, a lightweight wrapper for providing a unified command line interface as well as parsing the BIDS input, and brief documentation. BIDS Apps are an evolution of lightweight wrappers such as the Interface class developed within Nipype (Gorgolewski et al. 2011), with the added advantage of being programming language agnostic. Since BIDS Apps merely serve the purpose of capturing dependencies and providing a unified way of calling the relevant program (through the command-line interface), the repositories in the github.com/BIDS-Apps organization do not actually host the source code or data of the pipelines and workflows being wrapped: these are contained in the built Docker image stored on

DockerHub. Building a BIDS App starts with creating a new repository and populating it with a Dockerfile and run script.

Dockerfile creation

A *Dockerfile* is a script written in a domain specific language that describes the steps necessary to build a Docker image. The Docker project provides excellent documentation and a tutorial on how to write Dockerfiles (https://docs.docker.com/engine/reference/builder/). However, because the container images built in the BIDS-Apps organization are ultimately intended to alternatively run under Singularity, there are some additional requirements that must be followed:

- Apps cannot rely on having elevated permissions inside the container image (in contrast to Docker, processes inside a Singularity container run with the privileges of the user running the container).
- Environment variables must be set using the ENV statement within the relevant Dockerfile, rather than relying on config files (such as /root/.bashrc).
- Apps should not write anywhere outside of /tmp, $HOME, and the specific output folder provided as a command-line argument.

To facilitate the process of writing Dockerfiles, we have created a set of templates that include installation steps for the most popular neuroimaging tools (FSL, FreeSurfer, AFNI, ANTs etc.): https://github.com/BIDS-Apps/dockerfile-templates. Those templates are also available as container images that can be used directly as a base for new BIDS Apps dockerfiles using the FROM statement.

Command-line interface

To improve user experience and ability to integrate BIDS Apps into various computational platforms, each App follows a set of core command-line arguments:

```
runscript input_dataset output_folder analysis_level
```

For example:

```
runscript /data/ds114 /scratch/outputs participant
```

- `input_dataset` provides a path to the dataset to be analyzed (read-only), which must conform to the BIDS standard

- `output_folder` is the folder where results of the analysis will be stored

- `analysis_level` denotes the stage of the analysis that will be performed

To facilitate easy and efficient execution, analyses in BIDS Apps can be split into stages (see Materials and Methods). In the simplest design, an App would run in two stages: 'participant' and 'group'. The 'participant' stage runs first level analysis that can be performed independently for each subject in the dataset (and thus can be executed in parallel). Analysis may optionally be restricted to a subset of participants using the `--particiapant_label` argument. The group level analysis runs on the outputs from the participant level analysis and cannot be split into independent parallel jobs. This scheme is inspired by the MapReduce programming model (Dean & Ghemawat 2008). Multiple such MapReduce steps can be defined for a single pipeline (full specification of the command-line scheme can be found in the Supplementary Materials).

There are no restrictions on what language is used to write the wrapper script as long as it conforms to the prescribed interface. However, to make it easier to generate new BIDS Apps we have created a basic implementation in Python that can be imported into a new script and filled with App-specific options: https://github.com/BIDS-Apps/bidscmd. We also provide several utilities that make it easier to work with BIDS-compatible directory structures--most notably, the PyBIDS Python package (https://github.com/INCF/pybids), which provides tools for simple but powerful logical queries over entities defined in the BIDS specification (e.g., retrieving a list of all unique subjects; getting the fieldmap files for all subjects with a valid first scanning run; etc.).

In addition to conforming to a standardized command-line argument scheme, run scripts are also responsible for validation of the input data before running any analysis. To facilitate the process we have developed a command line validator that checks whether the input datasets are compliant with the BIDS standard https://github.com/INCF/bids-validator. Because not all BIDS compatible datasets can be analyzed by all BIDS Apps (e.g. a surface reconstruction pipeline requires a high-resolution T1 weighted image), the validator can be configured to reject datasets with particular properties. Integrating the validator is as easy as calling an external command; Dockerfiles and container images with the validator pre-installed are also available. BIDS Apps developers can also choose to implement validation steps themselves if the requirements of their pipelines cannot be easily checked by the standard validator.

Building and testing container images

For each BIDS App a Docker image is built and run on a set of lightweight example BIDS datasets (Gorgolewski et al. 2013). This execution tests that the command-line interface and BIDS support are correctly implemented. The tests are run forcing read-only containers, to ensure compatibility with Singularity (which imposes read-only mode). If the Docker image builds successfully and passes all tests, it is assigned a unique version (based on the tag obtained from the corresponding GitHub repository) and uploaded to Docker Hub with a version tag. Since version tags are unique and Docker images stored on Docker Hub are never overwritten, all historical versions of each BIDS App will always be accessible. Building, testing and archiving of BIDS Apps is performed automatically through an Continuous Integration service (CircleCI). Automation of testing and versioning improves reliability due to minimization of human errors.

To facilitate the process of creating new BIDS Apps we have made an example App that can also be used as a template for new Apps: https://github.com/BIDS-Apps/example. Additional documentation and tutorials are available at http://bids-apps.neuroimaging.io. Developers seeking help are also encouraged to subscribe to the bids-app-dev mailing list at https://groups.google.com/d/forum/bids-apps-dev.

Available BIDS Apps

At the moment there are 18 BIDS Apps (see Table 1) in the repository, most of which were developed by participants in the 2016 sprint (see Materials and Methods). The Apps span different imaging modalities (structural, functional and diffusion MRI) as well as languages (Python, C++, MATLAB/Octave, OpenCL).

App name	Description	Applicable modalities	References
example	Example App that also serves as a template for new apps. Calculates intracranial volume.	T1w	n/a
Freesurfer	Surface extraction, longitudinal pipeline and study specific template calculation using FreeSurfer.	T1w	(Fischl 2012)
ndmg	One-click reliable and reproducible pipeline for T1w + DWI weighted MRI connectome estimation.	T1w, DWI	(Kiar et al. 2016)
BROCCOLI	Fast fMRI analysis on many-core CPUs and GPUs.	T1w, fMRI	(Eklund et al. 2014)
FibreDensityAndCrosssection	Fixel-Based Analysis (FBA) of Fibre Density and Fibre Cross-section	DWI	(Raffelt et al. 2015; Raffelt et al. 2016)
SPM	Statistical Parametric Mapping	T1w, fMRI	(Friston 2007)
MRIQC	Quality Assessment of structural and functional MRI	T1w, fMRI	In preparation
FMRIPREP	A generic fMRI preprocessing pipeline providing results robust to the input data quality as well as informative reports.	T1w, fMRI	In preparation

Quality Assessment Protocol	Quality Assessment of structural and functional MRI.	T1w, fMRI	(Zarrar et al. 2015)
Configurable Pipeline for the Analysis of Connectomes	Pipeline for high throughput processing and analysis of structural and functional MRI data.	T1w, fMRI	(Craddock et al. n.d.)
Hyperalignment	Computes hyperalignment transformations for functional alignment.	fMRI	(Guntupalli et al. 2016)
mindboggle	Pipeline to improve the accuracy, precision, and consistency of automated labeling and shape analysis of human brain image data.	T1w	(Klein & Tourville 2012)
MRtrix3 connectome	Robust generation and statistical analysis of structural connectomes estimated from diffusion tractography.	T1w, DWI	(Smith et al. 2015)
nilearn	Extraction of time-series and connectomes for population analysis.	fMRI	(Abraham et al. 2014)
automatic analysis (aa)	Neuroimaging pipeline system written in Matlab.	T1w, T2w, fMRI, DWI	(Cusack et al. 2014)
Niak Preprocessing	Noise reduction, segmentation, coregistration, motion estimation, resampling.	fMRI	(Bellec et al. 2012)
HPCPipelines	Anatomical and functional preprocessing pipelines used in the Human Connectome Project.	T1w, T2w, fMRI	(Glasser et al. 2013; Smith et al. 2013)
BrainIAK-SRM	Functional alignment using Shared Response Model implementation from the Brain Imaging Analysis Kit.	fMRI	(Chen et al. 2015)

Table 1. List of currently available BIDS Apps.

Running a BIDS App locally

Running a BIDS App on a local system can be performed using Docker, which is easy to install on all three major operating systems. To run the first stage of the example BIDS App for participant number 01 the user needs to open a console (terminal or cmd) and type:

```
docker run -ti --rm \
    -v /Users/cajal/data/ds005:/bids_dataset:ro \
    -v /Users/cajal/outputs:/outputs \
    bids/example:0.0.4 \
    /bids_dataset /outputs participant --participant_label 01
```

Where /Users/cajal/data/ds005 is the path to the input dataset and /Users/cajal/outputs the path where results should be stored. If the BIDS App was not run before on this machine, the Docker image will be automatically downloaded from the Docker Hub.

Running a BIDS App on a cluster (HPC)

On many academic clusters, singularity can be used to run containers[2]. In these setting, to run a BIDS App, it first needs to be saved to an Singularity-compatible image file. This step needs to be performed outside of the cluster (for example on a laptop) and requires Docker:

```
docker run --privileged -ti --rm  \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v D:\singularity_images:/output \
    filo/docker2singularity \
    bids/example:0.0.4
```

Where D:\singularity_images is a path where the Singularity image will be stored. After transferring the .img file to a cluster it can be run like any other executable:

```
./bids_example-0.0.4.img /bids_dataset /outputs participant --participant_label 01
```

Discussion

We have proposed a new way of distributing easy-to-use, reproducible neuroimaging analysis workflows that can run on all three major operating systems as well as multi-tenant clusters. Each BIDS App encapsulates all of its binary dependencies providing the means for

[2] Singularity software needs to be installed on the cluster for users to be able to use Singularity images. However, due to its minimal dependencies and security concerns Singularity is more likely to be approved for multi-tenant systems usage than Docker.

reproducible analysis as well as an ultimate source of provenance information. Thanks to the BIDS standard for the organization of input data, errors caused by manually provided metadata are minimized. Finally, the unified command-line interface structure combined with flexible MapReduce-style execution schemes lends BIDS Apps to easy integration into data analysis platforms as well as efficient execution on computational clusters independently of the particular scheduling software. To support the

To prove the viability of the BIDS Apps concept we have developed 18 Apps representing a diverse set of neuroimaging software originating from many different labs. We are expecting that the number of available apps will grow in the future. At the time of writing this paper there were two more Apps being developed (see Table 2). We are actively encouraging neuroimaging methods developers to deploy their tools as BIDS App to further growing the library of available pipelines.

App name	**Description**	**Applicable modalities**	**References**
OPPNI	Optimization of Preprocessing Pipelines for NeuroImaging, for analysis of fMRI data	fMRI	(Strother et al. 2002; Churchill, Oder, et al. 2012; Churchill, Yourganov, et al. 2012; Churchill et al. 2015)
MAGeTbrain	Segmentation and surface-based morphometry tool for hippocampus, amygdala, basal ganglia, thalamus, and cerebellum	T1w (optional T2w, potentially others)	(Chakravarty et al. 2015; Pipitone et al. 2014; Park et al. 2014; Chakravarty et al. 2013)

Table 2. List of BIDS Apps currently in development.

Even though similar solutions based on Docker have been proposed in the past, none have addressed the problem of running Docker containers on HPCs. For example, a study evaluating computational overhead running Docker images on an HPC (Di Tommaso et al. 2015) had to limit access to the Docker-enabled cluster only to “trusted” users due to security concerns (personal communication). The solution proposed here combines mature container building tools provided for multiple operating systems by the Docker project with HPC compatibility through Singularity.

Additionally, in contrast to previous proposals, the presented solution puts a strong emphasis on clear versioning of container images and the ability to access all previous versions. We envisage that this feature will be very valuable in the context of longitudinal studies (where the same software stack needs to be used over many years) as well as for accurately reporting a set of computational methods in a publication and later replicating them (which can be achieved by simply referencing a BIDS App with a corresponding version). Strict versioning of BIDS Apps is achieved by careful management of Docker images on Docker Hub via a Continuous Integration service which also is responsible for testing the Docker images, further reducing potential errors.

It is also worth noting that even though containers increase the reproducibility of scientific results, they do not solve the problem of sensitivity of some results to different operating systems, architectures or third party libraries. In particular, numerical and statistical instabilities create artificial dependence of results on hardware details that can only be addressed algorithmically. Further research is necessary to assess the robustness of published results to these factors. BIDS Apps can help in this endeavour to a certain extent - for example, a single analysis can be run using different versions of the same BIDS App to see the variance in results (different flavours of of the same App using different Linux distributions could be created for this purpose). However this approach has limits, further work is needed to better understand the potential for variance in results due to different Linux kernel versions across different systems and hardware architectures, since containers do not encapsulate this.

Another advantage of container-based solutions is that the user can run software in an almost identical software ecosystem as the one used for its development. This reduces the number of problems experienced by users due to nuanced differences in system configuration, such as system libraries or software versions. It also makes maintaining the software significantly easier for the developers, who do not need to support a variety of different configurations, and can easily reproduce errors.

Even though BIDS Apps as a form of neuroimaging software distribution scheme can be perceived as performing a similar task as the NeuroDebian project, the two initiatives in fact complement each other. Many of the example BIDS Apps presented in this manuscript use Debian as their base distribution and benefit from the ease of installation provided by the NeuroDebian project. It not only makes Dockerfiles shorter and easier to maintain, but due to the network of NeuroDebian mirror servers the build process is more reliable than then downloading software from their original locations. On the other hand the NeuroDebian project benefits from the BIDS App distribution scheme by exposing software previously limited only to Debian based distribution to all flavours of Linux. This is important considering that many HPCs run on RedHat and CentOS Linux distributions rather than Debian.

Future work will involve engaging more developers of neuroimaging methods to create BIDS Apps. Additionally there are plans for developing a repository that would be independent of Docker Hub to provide Open Container Consortium as well as Singularity-compatible images. This would provide improved sustainability of the project, but also remove the conversion step (from Docker to Singularity) currently necessary for running BIDS Apps on clusters.

Work is currently underway to facilitate the integration of BIDS Apps in other platforms such as XNAT (Marcus et al. 2007), CBRAIN (Sherif et al. 2014), or cloud based services like Amazon

Web Service (AWS). The apps will include a machine readable description of input arguments, their descriptions and acceptable values (based on the Boutiques application descriptor (Glatard et al. 2015). Another benefit of such a description is to allow developers integrating BIDS Apps into their platforms to automatically generate user interfaces, and to improve validation of input parameters.

Singularity is not the only solution proposed to handle containers on HPCs. “Shifter” (as of September 2016 only available as pre-release beta version) has been discussed in the context of running containerized academic software (Hale et al. 2016). The principles behind Shifter are similar to Singularity, but Shifter also attempts to tackle the problem of managing the container images. Because of this, it depends on several services (Redis and MongoDB servers, worker processes, etc.) that make setting it up and maintaining it more involved than Singularity. However, despite the differences all BIDS Apps would be able to run on clusters running Shifter.

It is also worth mentioning that even though the proposed framework is focused on analysis of neuroimaging data, a similar scheme could be applied to other types of data. The only element that is specific to neuroimaging in BIDS Apps is the format of the input data. Other fields with well established data standards can easily adopt the same way of constructing, testing and archiving pipelines with their software dependencies.

BIDS Apps is a framework to help neuroimaging practitioners deploy the complex data processing workflows that they require in their research. Leveraging existing technologies such as GitHub and CircleCI and extensively using *containers* (Docker and Singularity), the proposed ecosystem automatically generates the appropriate container images with a minimized impact on the researcher’s development flow. This paper shows how these apps are created, stored, and executed either locally or in HPCs. To maximize interoperability and reduce manual metadata handling, BIDS Apps require that input data are in the BIDS organization format. The ultimate goal of BIDS Apps is reproducibility, thus this framework is particularly focused on archiving and versioning of neuroimaging workflows.

Materials and Methods

Engaging the community

To kickstart the repository of BIDS Apps, a four day long coding sprint was organized at Stanford University in August 2016. Leading neuroimaging methods and workflow developers were invited to learn about BIDS, Docker, and Singularity. In addition the sprint was advertised on the Stanford Center for Reproducible Neuroscience and Twitter, and outreach was performed during the 2016 OHBM Meeting in Geneva. The workshop consisted of one day of training; during the remaining three days, hands-on support was provided by experienced Docker developers.

Command line specification

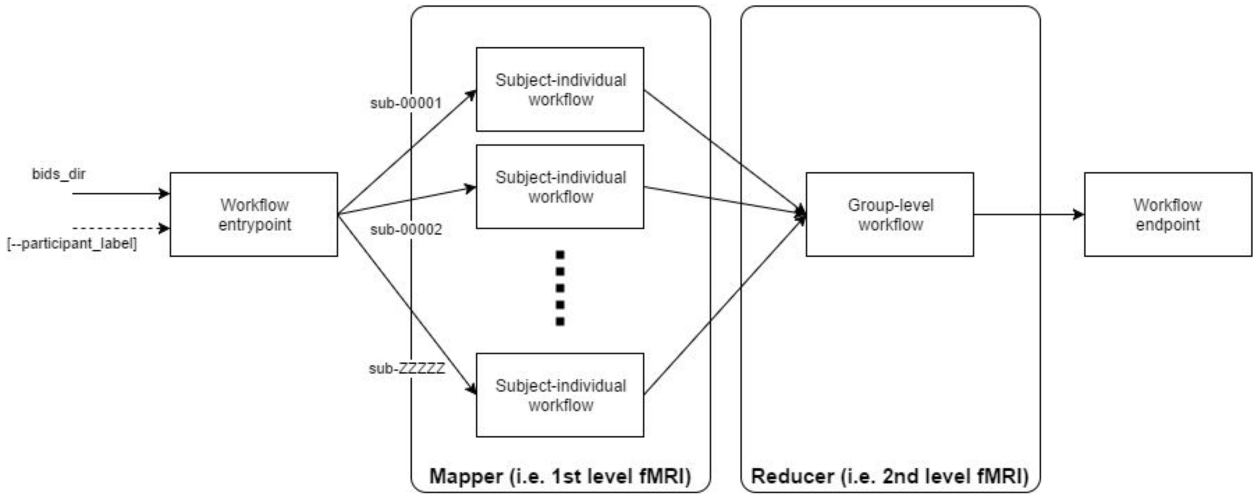
bids_dir
[--participant_label]
Workflow entrypoint
sub-00001
sub-00002
sub-ZZZZZ
Subject-individual workflow
Subject-individual workflow
Subject-individual workflow
Group-level workflow
Workflow endpoint
Mapper (i.e. 1st level fMRI)
Reducer (i.e. 2nd level fMRI)

Figure 2. The overall structure of workflows: workflows will need to decouple the individual level analysis (process independent subjects) from the group-level analysis. For the analysis of individual subjects, the workflow will require an understanding of the BIDS structure so that the required inputs for the designated subject are found. The optional reducer module will take up from results generated in the mapper and generate a group output. The overall workflow has an entrypoint and an endpoint responsible of setting-up the map-reduce tasks and the tear-down including organizing the outputs for its archiving, respectively.

This approach to run our workflows requires sticking with three *standards*: 1) a common command-line interface, 2) a Docker container to ensure portability, and 3) a standard for organizing input data. Containers created this way can be easily integrated in OpenfMRI as well as other data analysis platforms. Thanks to Docker to Singularity conversion they can also be easily run on High Performance Computers (clusters) without the need to install all of the dependencies.

Each workflow/pipeline will be run independently for each subject (the map step). Results of this execution (arrange in whatever way the pipeline prefers) can be optionally processed in a group level analysis (reduce step).

Command line interface

Each pipeline should have a simple wrapper script used to run it. The script should be command line interface and accept the following command line arguments (minimally):

1) For the participant level (aka map) step:

- `bids_dir` - (positional argument #1) the directory with the input dataset formatted according to the BIDS standard. This directory is read only.
- `output_dir` - (positional_argument #2) the directory where the output files should be stored. This is the only directory the pipeline should write to. Can be used to store

intermediate files, but they should be removed after the pipeline finishes. This directory is shared across all of the participant level jobs - it's up to the script to create subfolders for each subject.

- `“participant”` - (positional_argument #3) indicates that this is a participant level analysis.
- `--participant_label` - (optional) label of the participant 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.

Example:

Run processing for every subject independently (map step). Each of these operations can be performed in parallel. There are no restrictions or specification of how data should be organized inside the output_dir

```
./my_pipeline /data/my_dataset /scratch/outputs participant
--participant_label 01

./my_pipeline /data/my_dataset /scratch/outputs participant
--participant_label 02

./my_pipeline /data/my_dataset /scratch/outputs participant
--participant_label 03

...
```

2) (Optional) For the group level (aka reduce) step:

- `bids_dir` - (positional argument #1) the directory with the input dataset formatted according to the BIDS standard. This directory is read only.
- `output_dir` - (positional_argument #2) the directory where the output files should be stored. This is the only directory the pipeline should write to. Can be used to store intermediate files, but they should be removed after the pipeline finishes. This directory is the same one that was used in the participant level analysis and should be prepopulated with participant level results before running the group level.
- `“group”` - (positional_argument #3) indicates that this is a group level analysis.
- `--participant_label` - (optional) labels of the participants 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. This can be useful if you want to do a group level analysis on a subset of all participants in your dataset

Example:

```
./my_pipeline /data/my_dataset /scratch/outputs group
```

The script can also accept other arguments specific to your pipeline (see `--template_name` in FreeSurfer App). Mind that the same set of extra arguments will be passed to the map (single subject level) and the reduce (group level) stage.

Advanced use cases

- Multiple map reduce steps. In case your pipeline needs to do multiple map reduce steps the analysis_level (third positional argument) can take additional arguments: `participant2`, `group2`, `participant3`, `group3` etc. In the description of your app please specify how many map reduce steps are necessary.
- Within-job multi CPU parallelization. Each job (on any given level: participant or group) might be run on a multi CPU machine; in such case a parameter `--n_cpus` followed by an integer will be passed with the number of CPUs available.
- If your app is capable of adapting its workflow depending on how much memory is available on the environment it is running on you can implement an optional `--mem_mb` flag. When running your app the execution system will pass available memory in megabytes.

References

Abraham, A. et al., 2014. Machine learning for neuroimaging with scikit-learn. *Frontiers in neuroinformatics*, 8, p.14. Available at: http://dx.doi.org/10.3389/fninf.2014.00014.

Angiuoli, S.V. et al., 2011. CloVR: a virtual machine for automated and portable sequence analysis from the desktop using cloud computing. *BMC bioinformatics*, 12, p.356. Available at: http://dx.doi.org/10.1186/1471-2105-12-356.

Bellec, P. et al., 2012. The pipeline system for Octave and Matlab (PSOM): a lightweight scripting framework and execution engine for scientific workflows. *Frontiers in neuroinformatics*, 6, p.7. Available at: http://dx.doi.org/10.3389/fninf.2012.00007.

Belmann, P. et al., 2015. Bioboxes: standardised containers for interchangeable bioinformatics software. *GigaScience*, 4, p.47. Available at: http://dx.doi.org/10.1186/s13742-015-0087-0.

Chakravarty, M.M. et al., 2013. Performing label-fusion-based segmentation using multiple automatically generated templates. *Human brain mapping*, 34(10), pp.2635–2654. Available at: http://dx.doi.org/10.1002/hbm.22092.

Chakravarty, M.M. et al., 2015. Striatal shape abnormalities as novel neurodevelopmental endophenotypes in schizophrenia: a longitudinal study. *Human brain mapping*, 36(4), pp.1458–1469. Available at: http://dx.doi.org/10.1002/hbm.22715.

Chen, P.-H. (cameron) et al., 2015. A Reduced-Dimension fMRI Shared Response Model. In C. Cortes et al., eds. *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., pp. 460–468. Available at:

http://papers.nips.cc/paper/5855-a-reduced-dimension-fmri-shared-response-model.pdf.

Churchill, N.W. et al., 2015. An Automated, Adaptive Framework for Optimizing Preprocessing Pipelines in Task-Based Functional MRI. *PloS one*, 10(7), p.e0131520. Available at: http://dx.doi.org/10.1371/journal.pone.0131520.

Churchill, N.W., Yourganov, G., et al., 2012. Optimizing preprocessing and analysis pipelines for single-subject fMRI: 2. Interactions with ICA, PCA, task contrast and inter-subject heterogeneity. *PloS one*, 7(2), p.e31147. Available at: http://dx.doi.org/10.1371/journal.pone.0031147.

Churchill, N.W., Oder, A., et al., 2012. Optimizing preprocessing and analysis pipelines for single-subject fMRI. I. Standard temporal motion and physiological noise correction methods. *Human brain mapping*, 33(3), pp.609–627. Available at: http://dx.doi.org/10.1002/hbm.21238.

Craddock, C. et al., Towards Automated Analysis of Connectomes: The Configurable Pipeline for the Analysis of Connectomes (C-PAC). *Frontiers in neuroinformatics*, (42). Available at: http://www.frontiersin.org/neuroinformatics/10.3389/conf.fninf.2013.09.00042/full.

Cusack, R. et al., 2014. Automatic analysis (aa): efficient neuroimaging workflows and parallel processing using Matlab and XML. *Frontiers in neuroinformatics*, 8, p.90. Available at: http://dx.doi.org/10.3389/fninf.2014.00090.

Dean, J. & Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1), p.107. Available at: http://dx.doi.org/10.1145/1327452.1327492.

Devisetty, U.K. et al., 2016. Bringing your tools to CyVerse Discovery Environment using Docker. *F1000Research*, 5. Available at: http://f1000research.com/articles/5-1442/v1/pdf [Accessed August 17, 2016].

Di Tommaso, P. et al., 2015. The impact of Docker containers on the performance of genomic pipelines. *PeerJ*, 3, p.e1273. Available at: http://dx.doi.org/10.7717/peerj.1273.

Eklund, A. et al., 2014. BROCCOLI: Software for fast fMRI analysis on many-core CPUs and GPUs. *Frontiers in neuroinformatics*, 8, p.24. Available at: http://dx.doi.org/10.3389/fninf.2014.00024.

Fischl, B., 2012. FreeSurfer. *NeuroImage*, 62(2), pp.774–781. Available at: http://dx.doi.org/10.1016/j.neuroimage.2012.01.021.

Folarin, A.A., Dobson, R.J.B. & Newhouse, S.J., 2015. NGSeasy: a next generation sequencing pipeline in Docker containers. *F1000Research*, 4. Available at: http://f1000research.com/articles/4-997/v1/pdf [Accessed August 19, 2016].

Friston, K.J., 2007. *Statistical Parametric Mapping: The Analysis of Functional Brain Images*, Elsevier/Academic Press. Available at: http://books.google.com/books?id=KZZcLqkxhFIC.

Glasser, M.F. et al., 2013. The minimal preprocessing pipelines for the Human Connectome

Project. *NeuroImage*, 80, pp.105–124. Available at: http://dx.doi.org/10.1016/j.neuroimage.2013.04.127.

Glatard, T. et al., Boutiques: an application-sharing system based on Linux containers. In *Neuroinformatics 2015*. Neuroinformatics 2015. Available at: http://www.frontiersin.org/neuroscience/10.3389/conf.fnins.2015.91.00012/full.

Glatard, T. et al., 2015. Reproducibility of neuroimaging analyses across operating systems. *Frontiers in neuroinformatics*, 9. Available at: http://journal.frontiersin.org/article/10.3389/fninf.2015.00012/abstract [Accessed April 8, 2015].

Gorgolewski, K. et al., 2011. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. *Frontiers in neuroinformatics*, 5(August), p.13. Available at: http://dx.doi.org/10.3389/fninf.2011.00013.

Gorgolewski, K.J. et al., 2013. A test-retest fMRI dataset for motor, language and spatial attention functions. *GigaScience*, 2(1), p.6. Available at: http://dx.doi.org/10.1186/2047-217X-2-6.

Gorgolewski, K.J. et al., 2016. The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. *Scientific data*, 3, p.160044. Available at: http://dx.doi.org/10.1038/sdata.2016.44.

Gronenschild, E.H.B.M. et al., 2012. The Effects of FreeSurfer Version, Workstation Type, and Macintosh Operating System Version on Anatomical Volume and Cortical Thickness Measurements S. Hayasaka, ed. *PloS one*, 7(6), p.e38234. Available at: http://dx.doi.org/10.1371/journal.pone.0038234.

Guntupalli, J.S. et al., 2016. A Model of Representational Spaces in Human Cortex. *Cerebral cortex* , 26(6), pp.2919–2934. Available at: http://dx.doi.org/10.1093/cercor/bhw068.

Halchenko, Y.O. & Hanke, M., 2012. Open is not enough. Let' s take the next step: An integrated, community-driven computing platform for neuroscience. *Frontiers in neuroinformatics*, 6(22). Available at: http://dx.doi.org/10.3389/fninf.2012.00022.

Hale, J.S. et al., 2016. Containers for portable, productive and performant scientific computing. *arXiv [cs.DC]*. Available at: http://arxiv.org/abs/1608.07573.

Hanke, M. & Halchenko, Y.O., 2011. Neuroscience runs on GNU / Linux. *Frontiers in neuroinformatics*, 5(July), pp.7–9. Available at: http://dx.doi.org/10.3389/fninf.2011.00008.

Jordan, C., Skidmore, E. & Dooley, R., 2011. The iPlant collaborative: cyberinfrastructure for plant biology. *Frontiers in plant science*. Available at: http://journal.frontiersin.org/article/10.3389/fpls.2011.00034/full.

Kiar, G. et al., 2016. *ndmg: NeuroData's MRI Graphs pipeline*, Zenodo. Available at: http://zenodo.org/record/60206.

Klein, A. & Tourville, J., 2012. 101 labeled brain images and a consistent human cortical labeling protocol. *Frontiers in neuroscience*, 6, p.171. Available at: http://dx.doi.org/10.3389/fnins.2012.00171.

Krampis, K. et al., 2012. Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community. *BMC bioinformatics*, 13, p.42. Available at: http://dx.doi.org/10.1186/1471-2105-13-42.

Kurtzer, G.M., 2016. Singularity 2.1.2 - Linux application and environment containers for science. Available at: http://dx.doi.org/10.5281/zenodo.60736.

Mackenzie-Graham, A.J. et al., 2008. Provenance in neuroimaging. *NeuroImage*, 42(1), pp.178–195. Available at: http://dx.doi.org/10.1016/j.neuroimage.2008.04.186.

Marcus, D.S. et al., 2007. The extensible neuroimaging archive toolkit. *Neuroinformatics*, 5(1), pp.11–33. Available at: http://www.springerlink.com/index/U58X72728112X367.pdf [Accessed August 14, 2014].

Moreews, F. et al., 2015. BioShaDock: a community driven bioinformatics shared Docker-based tools registry. *F1000Research*, 4, p.1443. Available at: http://dx.doi.org/10.12688/f1000research.7536.1.

Park, M.T.M. et al., 2014. Derivation of high-resolution MRI atlases of the human cerebellum at 3T and segmentation using multiple automatically generated templates. *NeuroImage*, 95, pp.217–231. Available at: http://dx.doi.org/10.1016/j.neuroimage.2014.03.037.

Pipitone, J. et al., 2014. Multi-atlas segmentation of the whole hippocampus and subfields using multiple automatically generated templates. *NeuroImage*, 101, pp.494–512. Available at: http://dx.doi.org/10.1016/j.neuroimage.2014.04.054.

Raffelt, D.A. et al., 2015. Connectivity-based fixel enhancement: Whole-brain statistical analysis of diffusion MRI measures in the presence of crossing fibres. *NeuroImage*, 117, pp.40–55. Available at: http://dx.doi.org/10.1016/j.neuroimage.2015.05.039.

Raffelt, D.A. et al., 2016. Investigating White Matter Fibre Density and Morphology using Fixel-Based Analysis. *NeuroImage*. Available at: http://dx.doi.org/10.1016/j.neuroimage.2016.09.029.

Sherif, T. et al., 2014. CBRAIN: a web-based, distributed computing platform for collaborative neuroimaging research. *Frontiers in neuroinformatics*, 8, p.54. Available at: http://dx.doi.org/10.3389/fninf.2014.00054.

Smith, R.E. et al., 2015. The effects of SIFT on the reproducibility and biological accuracy of the structural connectome. *NeuroImage*, 104, pp.253–265. Available at: http://dx.doi.org/10.1016/j.neuroimage.2014.10.004.

Smith, S.M. et al., 2013. Resting-state fMRI in the Human Connectome Project. *NeuroImage*. Available at: http://dx.doi.org/10.1016/j.neuroimage.2013.05.039.

Strother, S.C. et al., 2002. The quantitative evaluation of functional neuroimaging experiments: the NPAIRS data analysis framework. *NeuroImage*, 15(4), pp.747–771. Available at:

http://dx.doi.org/10.1006/nimg.2001.1034.

Zarrar, S. et al., 2015. The Preprocessed Connectomes Project Quality Assessment Protocol - a resource for measuring the quality of MRI data. *Frontiers in neuroscience*, 9. Available at: http://www.frontiersin.org/Community/AbstractDetails.aspx?ABS_DOI=10.3389/conf.fnins.2015.91.00047.