FAIRly big: A framework for reproducible processing of large-scale data

Adina S. Wagner[*1], Laura K. Waite[*1], Małgorzata Wierzba[*1,2], Felix Hoffstaedter[1,3], Alexander Q. Waite[1], Benjamin Poldrack[1], Simon B. Eickhoff[1,3] & Michael Hanke[1,3]

[1] Institute of Neuroscience and Medicine, Brain & Behaviour INM-7, Research Center Jülich, Germany
[2] Laboratory of Brain Imaging, Nencki Institute of Experimental Biology, Polish Academy of Sciences, Warsaw, Poland
[3] Institute of Systems Neuroscience, Medical Faculty, Heinrich Heine University Düsseldorf, Germany

nencki institute of experimental biology
JÜLICH Forschungszentrum
hhu Heinrich Heine Universität Düsseldorf

The framework consists of the following steps. Steps 1-4 are set up by a bootstrap script based on user-input. Its main features are version control, ephemeral workspaces, and computational provenance capture

Big data? FAIRly difficult

Open **sharing & reusing** derivatives is the most viable way to extend previous work[1]. Still, the reproducible processing of **large-scale** or **sensitive** data, or with **proprietary** software poses research data management (**RDM**) **challenges**.

Storage & computational **demands** of large data strain the capabilities of compute infrastructure; The growing complexity of handling big data threatens the **trustworthiness** of its derivatives. Sensitive data can only be shared as open as its responsible use permits, and proprietary software obstructs recomputation. Sharing large-scale derivatives in bulk can make them as inaccessible as the original raw data due to size.

Data should not only be as **FAIR** as possible, but also handled in a **sustainable** manner that prioritizes **data sharing**, **transparency** & **reuse** by appropriate audiences. We present an open source framework built on DataLad[2] & containerization software[3] to **reproducibly process** & **share** big datasets.

1. Create a Dataset

```
$ datalad create ukb-vbm
```

Comprehensive version control

2. Link input data

```
$ datalad clone -d . \
   ${datastore}#~ukb-bids
```

DataLad **datasets**, Git-repository-based overlay structures, version control files of any size or type. They track and transport files in a distributed network of **clones** - lightweight dataset copies linked to their origin dataset - and can retrieve or drop registered, remote file content on demand with single file granularity, enabling data access without permanent storage demands. Datasets can link other datasets in arbitrarily deep hierarchies.

3. Link processing pipeline

```
$ datalad clone -d . \
   ${containerstore}#~cat code/cat
$ datalad containers-add \
   code/cat/... --name cat
```

Storage representation (RIA store)

```
/data/store/
  893/8de76-0302-45b5-9825-3c6ce3f3fffe/
  800/467ae-d521-11ea-8bf7-ac1f6bbc444e/
    branches/...
    config
    refs/...
    objects/...
    ...
    archives/archive.7z
```

Common representation

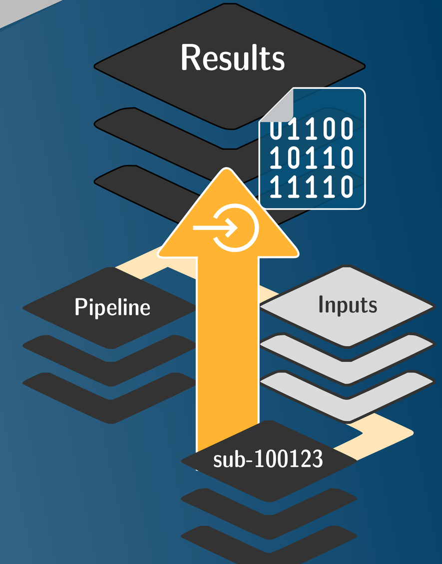
```
├── ses-2/
│   ├── anat/ ...
│   ├── dwi/  ...
│   ├── func/ ...
│   ├── non-bids/ ...
│   │   ├── dMRI/ ...
│   │   ├── fieldmap/ ...
│   │   ├── fMRI/ ...
│   │   ├── SWI/ ...
│   │   ├── T1/ ...
│   │   └── T2_FLAIR/ ...
│   └── swi/ ...
├── ses-3/
│   ├── anat/ ...
│   ├── dwi/ ...
│   ├── func/ ...
│   ├── non-bids/ ...
...
```

```
# clone one participant dataset from the store
$ datalad clone ria+file:///store#800467ae-d521-11ea-8bf7-ac1f6bbc444e sub-10..
$ cd sub-10..
# retrieve all of the subjects data from the archives in
# the store
$ datalad get .
```

Datasets have a **common representation** (a directory tree) familiar to users, and an **internal storage representation** (a RIA store[4]) that can host a dataset of arbitrary size and number of files in fewer than 25 inodes, with minimal server-side requirements, and optional content compression and encryption.

```
# draft a datalad (containers-)run
# command for analysis granularity
# of your choice (e.g., subject) &
# create its ephemeral workspace.
# Pick job scheduler and resources
```

Ephemeral workspaces



4. Develop compute job

5. Parallel execution

```
# job scheduler
$ condor_submit ...
$ sbatch ...
```

Author: Jane Doe <j.doe@fz-juelich.de>
AuthorDate: Wed Feb 10 18:05:30 2021 +0100

{   "chain": [],
    "cmd": "singularity exec -B {pwd} --cleanenv
            code/pipeline[...] sh -e -u -x -c [...]'
    "dsid": "8938de76-0302-45b5-9825-3c6ce3f3fffe",
    "exit": 0,
    "extra_inputs": ["code/pipeline/.datalad/environments/cat/image"],
    "inputs": ["inputs/ukb/sub-6.../ses-2/anat/sub-6..._ses-2_T1w.nii.gz",
     "code/cat_standalone_batch.txt",
     "code/finalize_job_outputs.sh"],
    "outputs": ["sub-6025043/ses-2"],
    "pwd": "."   }

In collaborative software development routines, parallel feature development happens in branches. Merging integrates them into main revision history. Similarly, our framework executes jobs in parallel on unique branches, which users merge to aggregate results.

For parallel computations, each compute job bootstraps an ephemeral (shortlived) workspace using job scheduling software (HTCondor, SLURM). Jobs retrieve all relevant elements (e.g., subsets of input data) to their workspace, capture analysis provenance on a unique branch, and push results back. As this potentially expands compressed or archives files on infrastructure with storage constrains, the number of concurrent jobs can be adjusted to fit available resources.

6. Result consolidation

```
# octopus-merge all "job" branches
$ git merge -m "Merge results" \
   $(git branch -al | grep 'job')
```

Computational provenance

Datasets can **record** and **re-execute** actionable **process provenance** about any file's genesis. It is created as a **machine-readable** record within the compute job, using either a datalad run command (for shell command execution) or datalad containers-run command for container invocations.

```
# recompute a previous computation
$ datalad rerun e035f896s45c9
```

```
# perform & capture a container execution in a job
$ datalad containers-run \
   -m "Compute subject ${subid}" \
   -n cat \
   --input "inputs/${subid}/*T1w.nii.gz" \
   --output "${subid}" \
   "<arguments for container invocation>"
```

Each job adds provenance to its job branch. Ephemeral workspaces ensure its completeness (defining all relevant processing elements), and thus **portability**. Beyond **transparency**, this allows consumers to rerun individual jobs on their own laptops and check for **computational reproducibility**.

UK-Biobank use case

Key workflow metrics:

41.180 T1-weighted brain images. Each compute job:
- processed one image for voxel-based morphometry[5]
- required one CPU hour
- needed 5 GB disk space
- needed 4 GB RAM
- created 4 output archives

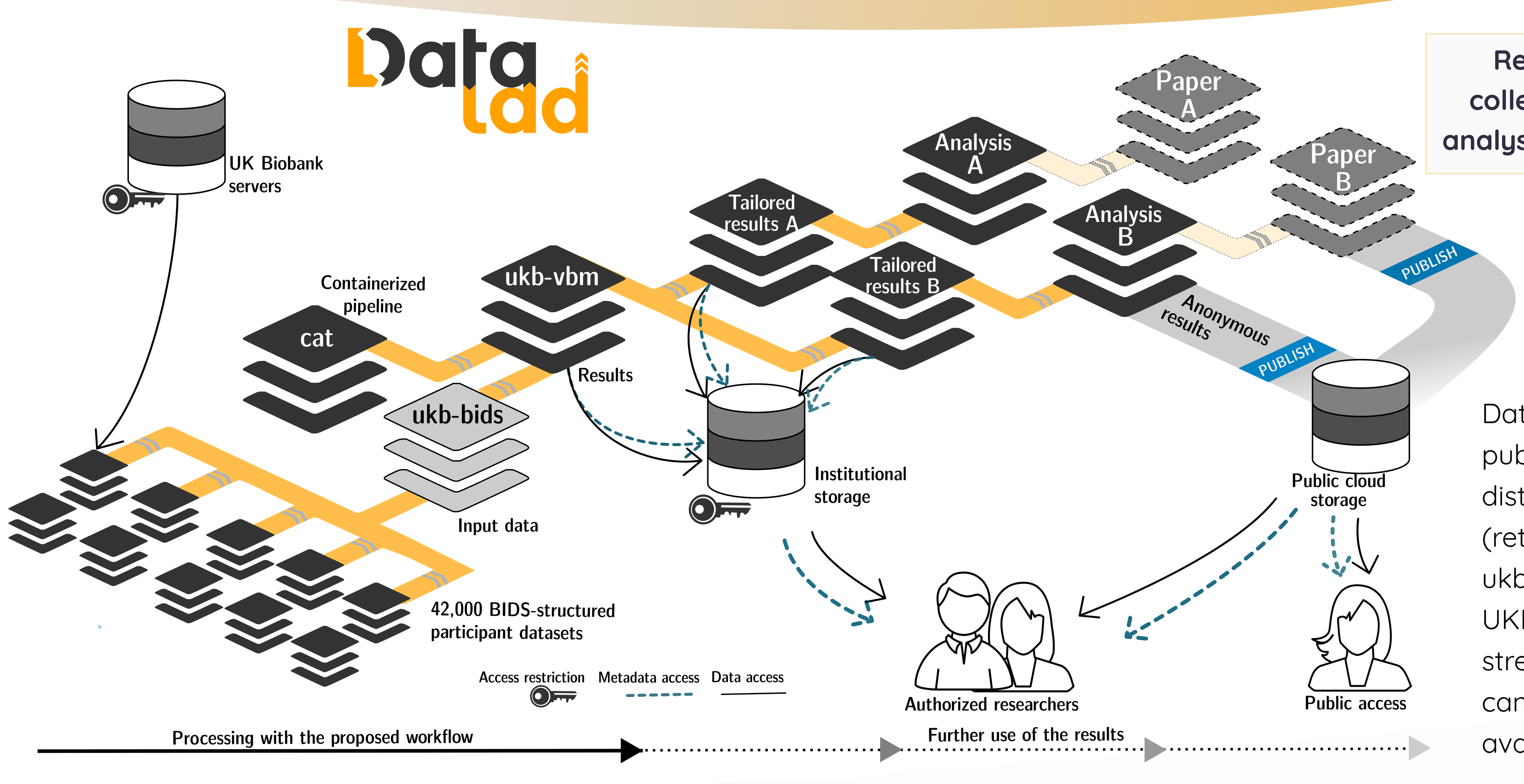
Key computing metrics:

Low disk space **HTC**:
- up to 600 jobs at a time
- Processing time: 6 weeks

Inode-constrained **HPC**:
- 3125 jobs at a time
- Processing time: 10 hours

Recomputations yielded >50% binary identical results with the exception of cortical projections.

Recompute your colleagues' multi-TB analysis - on your laptop!



Re-Use

DataLad Datasets ease public or private data distribution. Access to raw (retrieved by datalad-ukbiobank[6]) & processed UKB data is restricted, but streamlined. If legal, data can be made publicly available easily.

biobank application no. 41655

Code · Visualization

Funding
NATIONAL SCIENCE CENTRE POLAND
NSF 1912266 · NSF 1429999
Federal Ministry of Education and Research: BMBF 01GQ1905 · BMBF 01GQ1411
MOTOR SFB
HBP: H2020-EU.3.1.5.3, 945539 · H2020-EU.3.1.5.3, 826421

Legend
- File content
- Content identity and metadata availability
- Dataset with file content present
- Dataset with file content absent

References
[1] Craddock, C. et al. (2013). doi 10.3389/conf.fninf.2013.09.00041
[2] Halchenko, Y. O. et al. (2021). doi: 10.21105/joss.03262
[3] Kurtzer et al. (2017). doi: 10.1371/journal.pone.0177459
[4] Poldrack et al. (2021). doi: 10.7490/f1000research.1118494.1
[5] Gaser & Dahnke, http://www.neuro.uni-jena.de/cat
[6] Hanke et al. (2021) doi: 10.5281/zenodo.4773629

Talk · Paper · Tutorial