Nipype: Opensource platform for unified and replicable interaction with existing neuroimaging tools

SS Ghosh[1], C Burns[2], D Clark[2], K Gorgolewski[3], YO Halchenko[4], C Madison[2], R Tungaraza[5], KJ Millman[2]
[1]MIT, Cambridge, MA, USA [2]U. California, Berkeley, CA, USA [3]U. Edinburgh, UK [4]Dartmouth College, Hanover, NH, USA [5]U. Washington, Seattle, WA, USA

Introduction

Current neuroimaging software offer users an incredible opportunity to analyze their data in different ways, with different underlying assumptions. However, this has resulted in a heterogeneous collection of specialized applications without transparent interoperability or a uniform operating interface. Nipype solves these issues by providing a uniform interface to existing neuroimaging software and by facilitating interaction between these packages within a single workflow.

Problems addressed

- **Optimal workflows.** Integrating different packages is tedious.
- **Developers nightmare.** No standard framework for disseminating tools across packages.
- **Knowledge distribution.** Training new personnel takes time.
- **Leveraging technology.** Many packages do not address computational efficiency.
- **Replicating research.** Method sections are often inadequate for replication or review.

Features

- **Multi-package support.** Uniform cross-package interface (supports SPM, FSL, FreeSurfer components).
- **Minimal programming knowledge.** A high level scripting interface to design workflows intended for non-programmers.
- **Parallel execution.** No additional coding necessary for parallel execution of processing stages.
- **Pedagogical.** Can be used as a teaching tool and facilitates user training.
- **Well documented.** Code documentation becomes user documentation.
- **Shared workflows.** Workflows separate execution stages from data and can be shared within and across labs.

Conclusion

- Provides an environment for interactive manipulation of data through a Python interface as well as for performing reproducible, distributed analysis using a pipeline system.
- Encourages scientific exploration of different algorithms and associated parameters.
- Simplifies the development of workflows within and between packages.
- Reduces the learning curve associated with understanding the algorithms, APIs and user interfaces of disparate packages.

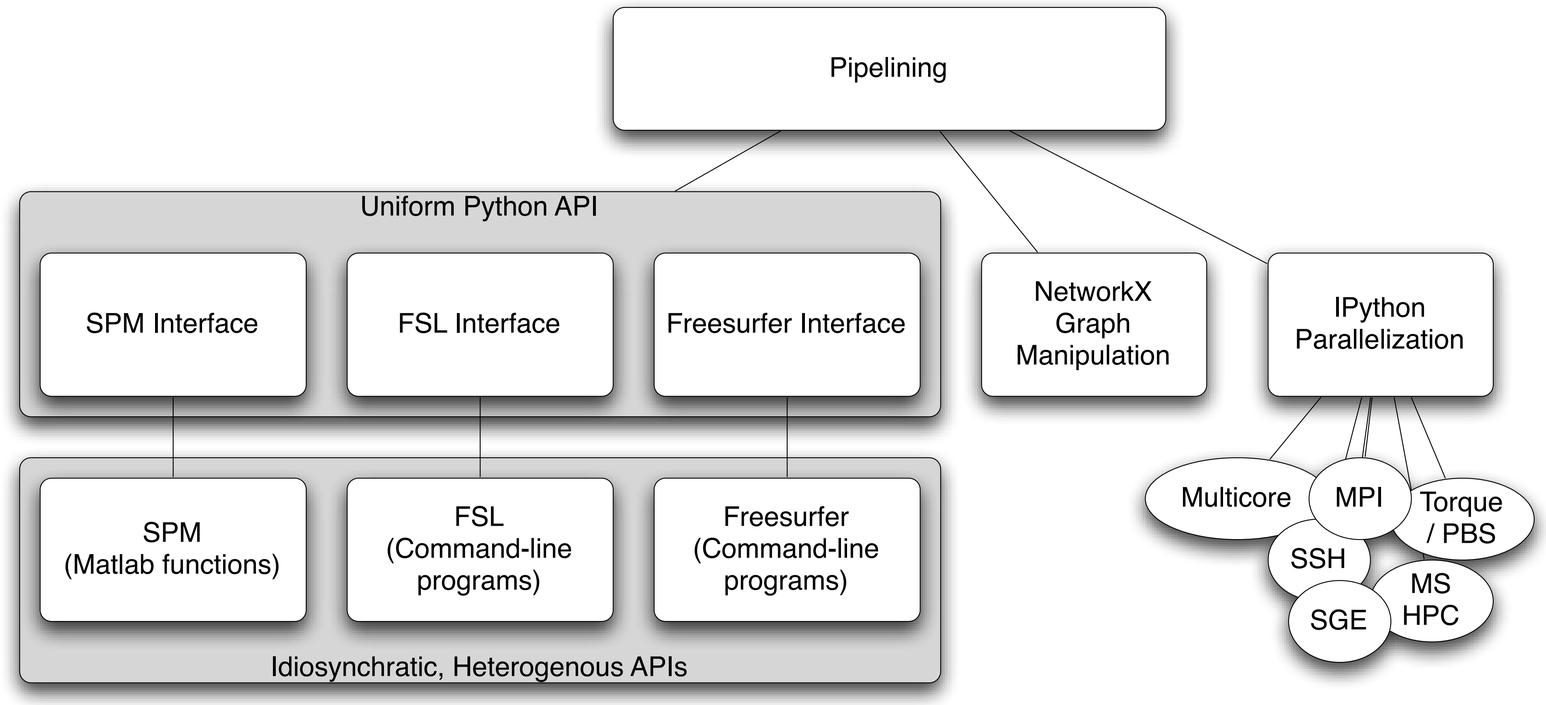
Engages the community	**C**ollaborative
Available for all	**O**pensource
Workflows are records	**R**eproducible
Minimizes redundancy	**E**fficient

Software dependencies

Nipy	`nipy.org`
IPython	`ipython.scipy.org`
Scipy,Numpy	`scipy.org`
NetworkX	`networkx.lanl.gov`
Traits	`www.enthought.com`

Also available from NeuroDebian (`neuro.debian.net`)

Component Architecture Diagram



Uniform package interface

```
In [5]: from nipype.interfaces.fsl import BET
In [6]: BET.help()

Inputs
------
Mandatory:
 in_file: input file to skull strip

Optional:
 args: Additional parameters to the command
 environ: Environment variables (default={})
 frac: fractional intensity threshold
 functional: apply to 4D fMRI data
  mutually exclusive: functional, reduce_bias
 mask: create binary mask image
...

Outputs
-------
out_file: path/name of skullstripped file
...

In [7]: bet = BET()
In [8]: res = bet.run(in_file='struct.nii', mask=True)
```

Simple workflow construction

Import components

```
from nipype.pipeline.engine import (Workflow,
                                    Node, Mapnode)
from nipype.interfaces.spm import Realign
from nipype.interfaces.fsl import Smooth
```

Define processes

```
realign = Node(interface=Realign(), name='realign')
smooth = MapNode(interface=Smooth(fwhm=6),
                 iterfield=['in_file'], name='smooth')
```

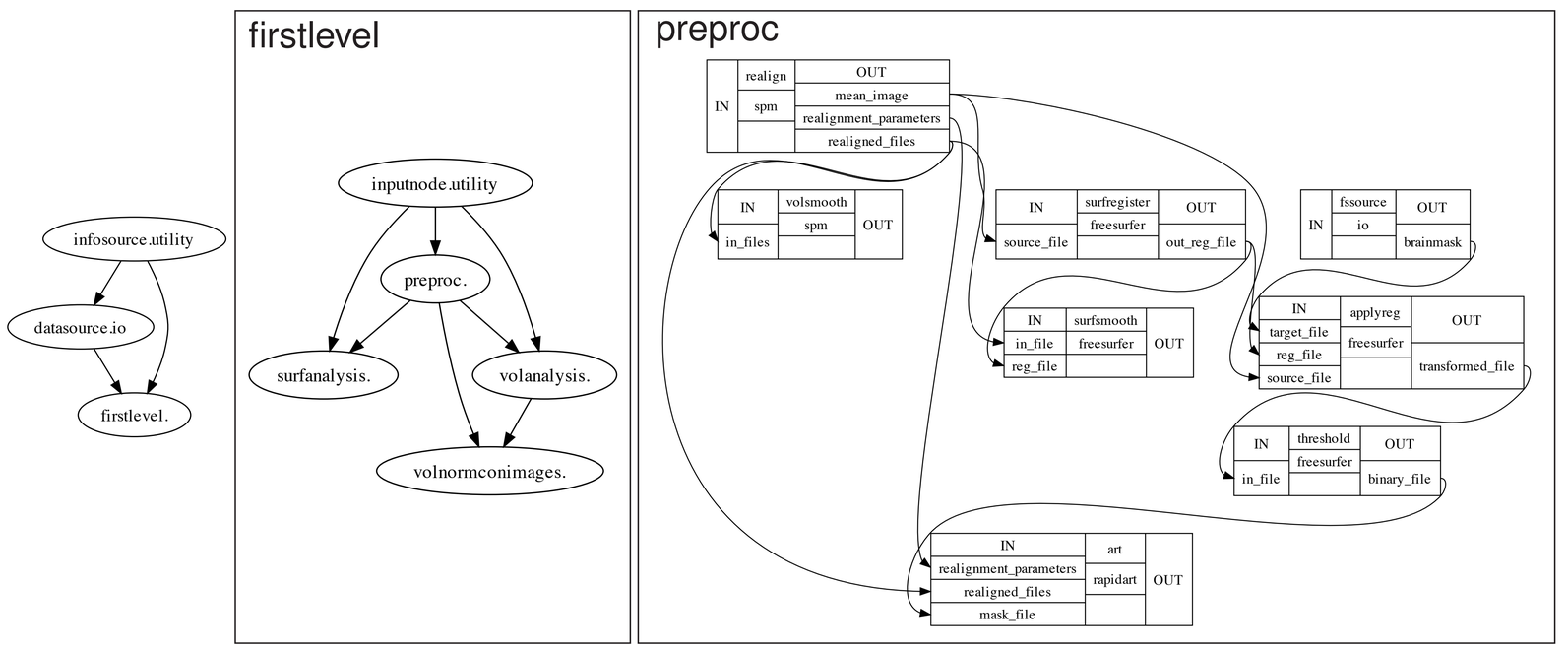
Construct workflow

```
preproc = Workflow()
preproc.connect(realign, 'realigned_files',
                smooth, 'in_file')
```

Execute workflow

```
preproc.inputs.realign.in_files = ['run1.nii', 'run2.nii']
preproc.base_dir = './workdir'
preproc.run()
```

Hierarchical workflow visualization



Parallel execution

- Leverages IPython for distributed computation.
- Non-dependent processes can be executed in parallel.
- No additional coding necessary.
- Example: SPM fMRI workflow on 69 subjects
  - Serial runtime: 36 h
  - Nipype runtime: 1 h 40 min on 40 cores.

Funding

This project was supported by NIH grants NIBIB R03 EB008673, NIMH R01 MH081909, and IBIC, U. of Washington, Seattle.

Future plans

- **More interfaces.**
  - Complete coverage of existing packages.
  - Newer registration algorithms (e.g., ANTS - U. Penn, CVS - MGH)
  - Integration with tools in Nipy, Dipy and Nitime.
  - Integrate with Slicer
- **Web interface.** A graphical interface to define workflows.
- **Workflow repository.** Create a centralized location for version controlled workflows.
- **Additional modalities.** Extend to PET, EEG/MEG, fNIRS.