NAMNIs: **N**euromodulation **A**nd **M**ultimodal **N**euro**I**maging **s**oftware

Temmuz Karali[1, 2], Frank Padberg[1], Valerie Kirsch[3], Sophia Stoecklein[2], Peter Falkai[1], Daniel Keeser[1, 2]

2019

[1]Department of Psychiatry and Psychotherapy, University Hospital LMU, Munich, Germany
[2]Department of Radiology, University Hospital LMU, Munich, Germany
[3]Department of Neurology, University Hospital LMU, Munich, Germany

Abstract

The basic requirements in neuroimaging research are rapidly changing due to the growing size of data sets and multimodal approaches with ever more complex, time-consuming, diverse and fast-moving standards. NAMNIs offers a ready-made, open-source pipeline for pre- and post-processing of multimodal neuroimaging and neuromodulation data with a strong focus on reproducibility and multi-platform parallelization support.

1 Introduction

NAMNIs consists of a processing pipeline for multimodal magnetic resonance imaging (MRI) data analysis using parallel processing. It performs various pre-processing steps with the aforementioned data that calculate relevant metrics such as the number of activated voxels (spatial extent), within regions of interest (ROIs) effects, ROI-to-whole-brain calculations, probabilistic values, motion parameters, connectivity strength values (standardized in z scores), and more. These steps are performed in parallel with different topologies to enable the processing of large data sets in a reasonable time. It is applicable to High Performance Computing (HPC) and is provided as research software which is free and open-source software (FOSS). Already in the proof-of-concept and prototype phase, the project was published as abstracts of international conferences [1] [2].

2 Implementation Details and Design Choices

In order to reduce reproducibility issues across different systems, a design choice was made to support only Linux-based operating systems and the 64-bit system architecture. This design choice is important in order to achieve reproducibility goals using containerization solutions in the future versions of the software, as elaborated in section 5. This way, assuming that same NAMNIs container is deployed across different systems, it is possible to reproduce the same results, apart from the variance caused by different operating system kernels and different hardware. The operating system kernel factor could be eliminated by deploying NAMNIs in a virtual machine, however it would require substantially more effort on the user side to set up, and would only be possible with a performance disadvantage. The hardware factor cannot be realistically eliminated; therefore it is recommended that some hardware details about the system(s) used for analyses are mentioned in publications.

In order to offer NAMNIs completely as free and open source software (FOSS), a design choice was made to exclusively use FOSS as dependencies of NAMNIs, regardless of whether these are functionality dependencies or platform dependencies. This provides a decisive advantage for users and developers who want to base their platform exclusively on FOSS in order to achieve maximum transparency as open-science best practice [3]. In the literature there are several research software approaches that are FOSS, but depend on non-free, propetiary software packages, making it impossible to use these software packages in a fully open source research environment. It was preferred that NAMNIs does not have such limitations and can be used in accordance with open-science best practices.

Functionality dependencies NAMNIs code makes extensive use of third-party dependencies to provide core functionality (i. e. functionality dependencies). The most important third-party dependencies of NAMNIs are listed below.

- AFNI [4]
- ciftify [5]
- freesurfer [6]
- FSL [7]
- numpy [8]
- pandas [9]
- pybids [10]
- scipy [11]
- seaborn [12]
- SIMNIBS [13]

Platform dependencies NAMNIs codebase was programmed in "pure" python3 [14]. python3 (version 3.7 or higher) is therefore the main platform dependency of NAMNIs. Despite cross-platform support of python [14], a design choice was made not to support any of the earlier versions of python in order to make use the new features offered by python 3.7 such as dataclasses, f-strings, ordered dictionaries by default, and more [14] without having to rely on fallbacks, backports or self-implementations for older versions. This allows for a cleaner code-base that is more intuitive and less error-prone. Optional platform dependencies of NAMNIs are listed below.

- SLURM [15] for high-performance computing (HPC) parallelization support

3 Methods and Results

3.1 Reproducibility-optimized Workflow

Each combination of raw data and configuration data is defined as a *NAMNIs workflow*. The goal is to enable standardization at all levels of user input to ensure better reusability and reproducibility. This workflow is fully BIDS compatible and utilizes `rawdata/` and `code/` components in the specification [16].

Raw data BIDS [16] is the only compatible input raw data format. A NAMNIs workflow can make use of two of eight data types defined in the BIDS specification as of version 1.2.1 [16]: `func` (task based and resting state functional MRI) and `anat` (structural imaging such as T1, T2, etc.). A minimal workflow requires only anatomical data, but depending on the configuration, all data types can be used. Three more data types are planned to be supported in the future versions, as mentioned in section 5. Using this raw data, a NAMNIs workflow can be configured to perform processing tasks. A default workflow function is pre-defined for each of these tasks, though in a modular design it is possible for software developers to create their own workflow functions, if preferred.

Configuration data A common framework for parsing configuration files provides the capacity for a configuration file to define all relevant variables of a certain run on a given dataset. Configuration input is also standardized using JSON files, an example of which is elaborated in section 4. This input will be further formalized for robustness in the future versions, as mentioned in section 5.

3.2 Parallelization

Complex parallelization is achieved by incorporating a job scheduler (SLURM [15]) in high performance clustered environments. Job arrays of SLURM are used to manage nested parallel job structures accordingly. This component will be further optimized in the future versions, as mentioned in section 5. Simpler parallelization (i.e. subject-level parallelization based on a fixed multiprocessing pool) is also supported on single-machine systems.

3.3 Overview of the Pipeline Steps

A non-exhaustive list of the pipeline steps for each modality is summarized in the corresponding subsections.

3.3.1 `anat`

1. In the first step, `fslreorient2standard` [7] is called to orient all MR MPRAGE imaging data correctly to match the orientation of the MNI152 standard template.
2. The re-aligned MPRAGE images are brain extracted using `bet` [7] with the specific parameters as elaborated below (note that these parameters may vary depending on scanner type and MPRAGE scan MR acquisition settings).

   `-R` enable robust brain centre estimation (several iterations)

   `-f 0.45` fractional intensity threshold

   `-g` 0
3. Afterwards, a binary mask is created using `fslmaths` [7].
4. With the help of `fast` [7], the brain is segmented into the tissue groups CSF (0), gray matter (1), and white matter (2). This fully-automated tool is based on a hidden Markov random field model and an associated expectation-maximization algorithm.
5. Both linear and non-linear registrations were used to register the atlas on the individual MPRAGE images. 12 degrees of freedom (DOF) were used.
6. The transformation/deformation field is inverted.
7. The total volume for the tissue CSF (0), GM (1) and WM (2) is calculated for each subject.
8. Individual MPRAGE brain atlases are generated using `applywarp` [7]. The atlases are read out starting from the fourth dimension of the atlas file with the help of the `fslval` tool [7]. All regions are extracted individually for the GM and WM for the volume and voxel number; `fslmaths -mas` [7] and `fslstats -V` [7] are used for this.

3.3.2 `func`

1. In the first step, `3dresample` and `3dvolreg` [4] is called to correctly orient and subsequently motion-correct all data.
2. Afterwards, affine registration is conducted using individual MPRAGE images, which has been re-aligned with `fslreorient2standard` [7] and brain extracted with `bet` [7]. 6 degrees of freedom (DOF) were used.
3. The transformation/deformation field is inverted.
4. Afterwards, a binary mask is created using `fslmaths` [7].
5. Individual brain atlases are generated using `applywarp` [7]. The atlases are read out starting from the fourth dimension of the atlas file with the help of the `fslval` tool [7].
6. All regions are extracted individually for the voxel number and z-value; `fslmeants` [7] and `fslstats -V` [7] are used for this.

4 Usage Example

Suppose one would like to generate all pre-processing outputs offered by NAMNIs, and nothing else. It is easy to notice if every top-level object in the configuration file besides `globals` were to be collapsed into a single on-off switch, the configuration file would look like the code snippet below.

```
{
    "anat": {
        "enabled": true
    },
    "func": {
        "enabled": true
    },
    "globals": {}
}
```

An actual configuration file is merely an elaboration of the basic structure above, since modalities are exactly what constitutes the core of a NAMNIs configuration file. It is also easy to notice that all boolean values are preset to `false`, meaning the only values that had to be changed for this example are those listed above.

Note that there is another parameter outside of `globals` that must be set: `fsl_config_file` path in `anat` section. Here, the path for a FSL configuration file [7] that is compatible with the standard template of choice must be provided.

At this point, only `globals` section is not set. In accordance with the previous code snippet, represented below is a simplified `globals` section, with all of its subsections as empty objects.

```
{
    "globals": {
        "atlases": {},
        "standard_template_file": {}
    }
}
```

`standard_template_file` must be a valid path to the standard template of choice. Objects in the `atlases` section must contain a path to the atlas file, and a list of regions. The list of regions can be provided either as a JSON list or in a seperate plain text file, whose path must be specified at the place of the aforementioned JSON list. Ideally, this text file lists the name of each region in each line. Below the previous code snippet is extended into a configuration with two atlas definitions, each defined by one of the aforementioned methods for listing regions.

```
{
    "globals": {
        "atlases": {
            "atlas0": {
                "file": {
                    "__relative_path__": "code/atlas/atlas0.nii.gz"
                },
                "regions": {
                    "__relative_path__": "code/atlas/atlas0.txt"
                }
            },
            "atlas1": {
                "file": {
                    "__relative_path__": "code/atlas/atlas1.nii.gz"
                },
                "regions": [
                    "region0",
                    "region1"
                ]
            }
        },
        "standard_template_file": {}
    }
}
```

5 Future Perspectives

NAMNIs is expected to undergo extensive further development, including changes at the technical level of software implementation. It goes beyond the scope of this section to provide an exhaustive list of all future prospects of this project. Below is a summary of some of the key features expected to be provided as part of NAMNIs.

- Introduce pipeline steps for `dwi`, `fmap`, and `beh` modalities, as defined in the BIDS specification as of version 1.2.1 [16].
- Provide a "one-button" solution by offering the ability to configure post-processing steps (analogous to pre-processing steps) in the uniform configuration file.

- Containerization addresses the challenges on the technical side of reproducibility [17]. In order to achieve maximum reproducibility, distribute NAMNIs using versioned containers with all its dependencies included and versioned.
- Formalize the processing of configuration data using a specific JSON schema.
- Add support for tDCS and rTMS neuromodulation simulation tasks.
- Add support BIDS conform output data (analogous to input data).
- Implement substantially more efficient parallelization support using job dependencies as provided by SLURM [15].

6 Conclusion

The reproducibility crisis in scientific papers [18] and ever-growing requirements in computational resources show that reproducible workflows with strong parallelization support will be a requirement for any neuroimaging lab in the future. NAMNIs offers a ready-made, cost-free, and open source alternative. NAMNIs is planned to be published also in the form of a peer-reviewed publication.

7 References

[1] Karali et al. (2017): LMU Scripts • Ready-Made HPC-Applicable Pipeline for Structural and Functional Data Analyses. 23th Human Brain Mapping Congress Vancouver, Canada.

[2] Karali et al. (2019): NAMNIs: Neuromodulation And Multimodal NeuroImaging scripts. 25th Human Brain Mapping Congress Rome, Italy.

[3] Halchenko et al. (2015): Four aspects to make science open "by design" and not as an afterthought. GigaScience 4:31. DOI:s13742-015-0072-7

[4] Cox (1996): AFNI: Software for Analysis and Visualization of Functional Magnetic Resonance Neuroimages. Computers and Biomedical Research 29:162-173.

[5] Dickie et al. (2019): edickie/ciftify: Fix to ciftify_meants and new ciftify_dlabel_to_vol script (Version 2.3.3). Zenodo. DOI:10.5281/zenodo.3369937

[6] Fischl (2012): FreeSurfer. NeuroImage, Volume 62, Issue 2, Pages 774-781, ISSN 1053-8119. DOI:10.1016/j.neuroimage.2012.01.021.

[7] Jenkinson et al. (2012): FSL. NeuroImage 62:782-790.

[8] van der Walt et al. (2011): The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30, DOI:10.1109/MCSE.2011.37 (publisher link)

[9] Reback et al. (2019): pandas-dev/pandas: v0.25.3 (Version v0.25.3). Zenodo. DOI:10.5281/zenodo.3524604

[10] Tal et al. (2019): PyBIDS: Python tools for BIDS datasets (Version 0.9.4). Zenodo. DOI:10.5281/zenodo.3458537

[11] Virtanen et al. (2019): SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. preprint arXiv:1907.10121

[12] Waskom et al. (2018): mwaskom/seaborn: v0.9.0 (July 2018) (Version v0.9.0). Zenodo. DOI:10.5281/zenodo.1313201

[13] Thielscher et al. (2015): Field modeling for transcranial magnetic stimulation: a useful tool to understand the physiological effects of TMS? IEEE EMBS 2015, Milano, Italy

[14] van Rossum et al. (1991): Interactively Testing Remote Servers Using the Python Programming Language, CWI Quarterly, Volume 4, Issue 4, Amsterdam, pp 283–303.

[15] Yoo et al. (2003): SLURM: Simple Linux Utility for Resource Management. Job Scheduling Strategies for Parallel Processing. JSSPP 2003. Lecture Notes in Computer Science, vol 2862.

[16] Gorgolewski et al. (2016): The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments, Scientific Data, 3, 160044.

[17] Boettiger (2015): An introduction to Docker for reproducible research. Operating Systems Review, 49, 71-79.

[18] Nature Editors (2012): Must try harder. Nature. 483, 7391, 509–509.