Containerizing `BIDSme` : A Reproducible Tool for BIDS Conversion

Bradley Spitz[1,2], Antoine Jacquemin[2], Nikita Beliy[2], Christophe Phillips[2]

[1]Télécom Physique Strasbourg, Université de Strasbourg, France
[2]GIGA – CRC Human Imaging, University of Liège, Liège, Belgium

Abstract

The "Brain Imaging Data Structure" (BIDS) has become a widely adopted standard for organizing and sharing neuroimaging datasets of various modalities. However, converting raw brain imaging data into BIDS framework remains a complex and time-consuming task. `BIDSme` is a semi-automated tool developed to streamline this conversion process, but until recently, it lacked the portability and accessibility needed for widespread adoption. This paper presents the containerization of `BIDSme` using Docker and Docker Compose, improving usability, reproducibility, and integration into existing platforms like Neurodesk. It also details the design choices, iterative refinements, and validation process that led to a flexible, lightweight, and user-friendly containerized application.

1 Introduction

The Brain Imaging Data Structure (BIDS) scheme [1] has emerged as a community-driven standard for organizing and sharing neuroimaging data, helping reproducibility and interoperability across studies and software tools [2]. However, transforming "raw" [1] neuroimaging data into a valid BIDS format remains a complex, error-prone, and often manual process. This stems from variability in possible acquisition protocols and outlier cases in the data?

`BIDSme` is an open-source Python application developed at the GIGA-CRC Human Imaging research unit to facilitate this "BIDS-ification" of neuroimaging data [3]. It provides a configurable, user-guided workflow for renaming, restructuring, and enriching datasets with metadata, allowing flexibility beyond rigid BIDS-ification tools [4].

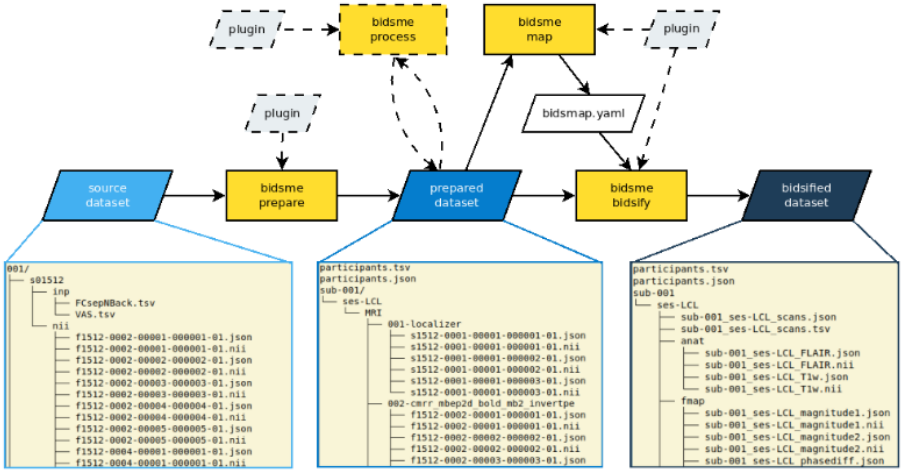

Figure 1: General workflow of `BIDSme` : from raw neuroimaging data to a BIDS-compliant dataset.

Despite its potential, `BIDSme` was not initially designed for broad distribution or easy deployment. It required local installation with multiple dependencies and lacked support for standardized, reproducible environments. This limited its accessibility for end-users, particularly those without technical expertise in Python or environment management.

To overcome these limitations, we propose here a containerized version of `BIDSme` using Docker [5], along with a simplified interface and integration into platforms like Neurodesk [6]. This paper outlines the containerization process, design choices, usability improvements, and validation strategies, aiming to make `BIDSme` a robust and portable tool for the neuroimaging community.

2 Software description

2.1 Overview of `BIDSme`

`BIDSme` is an open-source Python tool developed to facilitate the conversion of raw neuroimaging datasets into the Brain Imaging Data Structure (BIDS) format [1]. Unlike fully automated converters, `BIDSme` offers a semi-automated, user-guided approach, allowing flexibility to handle complex or heterogeneous datasets typically found in research settings.

Its modular design enables users to perform key steps such as renaming raw files, reorganizing directory structures, and mapping metadata fields, thereby reducing manual effort and minimiz-

[1] directly coming the acquisition system, or with minimal processing such DICOM-to-NIfTI conversion.

ing errors. The software supports a broad range of neuroimaging modalities and formats. To support new and experienced users, a tutorial and example datasets are also provided [7, 8].

2.2 Key functionalities

The core functionalities of `BIDSme` are implemented as four main commands:

- `prepare`: Organizes and preprocesses raw data by renaming files and arranging them into an intermediate directory structure that facilitates subsequent checks and BIDSification.
- `map`: Maps and transfers relevant metadata and filenames from the prepared data to the final BIDS-compliant structure.
- `bidsify`: Finalizes the conversion by applying BIDS-compliant naming conventions.
- `process`: Processes the bidsified dataset prior to final BIDSification. This command can be used to produce derivatives, perform data conversion, or execute anonymization while maintaining the advantage of recording identification through the use of `bidsmap.yaml`. Essentially, it performs all processing steps similar to BIDSification but without executing the BIDSification itself.

These operations can be executed sequentially or individually, providing users with fine control over the conversion workflow.

2.3 Limitations of the original implementation

Initially, `BIDSme` required users to manually install Python dependencies and set up the environment locally, which posed challenges in terms of portability, reproducibility, and ease of use. Dependency conflicts, differences in system configurations, and the need for command-line proficiency limited its accessibility to less technical users.

The containerization effort presented here addresses these issues by packaging `BIDSme` and all its dependencies into a portable and reproducible Docker image, enhancing usability through dual Command Line Interface (CLI) and JupyterLab modes [9], and facilitating integration into broader neuroimaging platforms.

3 Containerization approach

The containerization of `BIDSme` is guided by the motivation of ensuring its portability and reproducibility, by notably encapsulating all of its dependencies and environment configurations, finally leading to an easier deployment across diverse systems.

3.1 Base image selection

The process begins with the creation of our Dockerfile. The Dockerfile is a script defining how to assemble the container image, specifying the base environment, dependencies, code, configuration, and runtime behavior.

We first start the Dockerfile by choosing a base image. For `BIDSme`, a lightweight Python image such as **python:3.9-slim-bullseye** was chosen to provide a balance between minimal image size and broad compatibility. Using a slim variant minimizes the overall container size by excluding unnecessary packages and utilities, which is advantageous for efficient storage and faster image transfers.

3.2 Dependency management

A critical aspect of containerization is dependency management. `BIDSme` depends on both system-level libraries and a set of Python packages, which are clearly defined in its **setup.py** file.

At the system level, `BIDSme` requires essential neuroimaging tools, as well as libraries for GUI support and compression, among others. On the Python side, the core packages include **pandas** [10], **pyparsing** [11], **bidsschematools**, and several others. `BIDSme` also offers optional extras to tailor functionality according to user needs, such as **nibabel** [12], **pydicom** [13], **dcm2niix** [14], and **dicom_parser** [15], which are installed together to provide full feature support.

To guarantee that all these dependencies are correctly installed within the container, the Dockerfile retrieves the `BIDSme` source code and installs both the core packages and the extras in a single `pip install` command (e.g., `pip install ".[all]"`), ensuring that the container has all necessary components pre-installed for seamless execution.

3.3 Multi-stage build strategy

To optimize the final image size and separate the build environment from the runtime environment, a multi-stage build approach is employed, as shown in figure 2.

The first stage ("build stage"), see figure 2a, contains the base image selection, installs all system dependencies, clones the `BIDSme` repository, and performs the installation of Python packages including optional extras and JupyterLab for interactive use. The decision was made to directly clone the Git repository within the container build process. This approach provides full access to the latest version of the application without requiring any manual or redundant local copy prior to building the Docker image, thereby automating the containerization workflow.

The second stage ("runtime stage"), see figure 2b, starts from a fresh base image where only the minimal system dependencies are re-installed. It then copies the pre-installed Python environment from the "build stage", alongside necessary scripts, configuration and environment variables. This separation allows the final container to be cleaner and more secure by excluding unnecessary build tools and caches, while also preserving all required runtime functionalities.

```dockerfile
# ---------- Build stage ----------
ARG PYTHON_VERSION=3.9-alpine
FROM python:${PYTHON_VERSION} AS builder
LABEL maintainer="SPITZ Bradley <spitzbradley@gmail.com>"

# Compilation dependencies
RUN apk add --no-cache --virtual .build-deps \
        build-base cmake git nodejs npm \
        python3-dev tcl-dev tk-dev libffi-dev zlib-dev

# Compilation of dcm2niix
WORKDIR /tmp
RUN git clone --depth 1 https://github.com/rordenlab/dcm2niix.git && \
    cd dcm2niix && mkdir build && cd build && cmake .. && make && \
    cp bin/dcm2niix /usr/local/bin/


WORKDIR /mnt
COPY bidsme /mnt

# Installation of jupyterlab and BIDSme
RUN pip install --no-cache-dir --prefix=/install jupyterlab
RUN pip install --no-cache-dir --prefix=/install .
```

(a) Building stage

```dockerfile
# ---------- Runtime stage ----------
FROM python:${PYTHON_VERSION}

# Required system libraries at runtime
RUN apk add --no-cache tcl tk libffi zlib
sme_folder/projet BIDSme/prepared
COPY --from=builder /usr/local/bin/dcm2niix /usr/local/bin/
COPY --from=builder /install /usr/local
COPY --from=builder /mnt /mnt

# Copy entrypoint
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

# Create non-root user
RUN addgroup -S app && adduser -S app -G app
RUN chown -R app:app /mnt
USER app

# Environment variables
ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1 \
    JUPYTER_PORT=8888 \
    JUPYTER_TOKEN=""
WORKDIR /mnt
EXPOSE 8888
ENTRYPOINT ["/entrypoint.sh"]
CMD []
```

(b) Runtime stage

Figure 2: The multi-stage Dockerfile to optimize the image building process

3.4 Directory structure and mounting strategy

To support `BIDSme`'s file processing workflow and promote clarity across executions, a well-structured and consistent directory layout was established inside the container. All user-facing operations are carried out under a single shared workspace, typically mounted at `/mnt`, where input, output, and configuration files are organized.

The default expected structure includes the following subdirectories:

- `/mnt/rawdata` - contains the original neuroimaging files to be processed.
- `/mnt/prepared` - serves as the output directory for pre-processed data, ready for BIDSification.
- `/mnt/bidsified` - stores the final BIDS-compliant dataset.
- `/mnt/configuration` - holds configuration files and plugins such as `bidsmap.yaml` for flexible mapping logic.

This organization improves clarity, promotes automation, and aligns closely with `BIDSme`'s internal logic, see figure 1. To make this layout persistent beyond the life-cycle of the container, Docker volumes are used to bind host directories to the corresponding paths inside the container. Thus processed data and user configurations are not lost when the container stops or is removed.

3.5 Management of the user environment and permissions

To ensure compatibility with host file systems and avoid permission issues — especially when writing output data from within the container — a non-root user is created during the runtime stage of the Dockerfile. The user ID and group ID can be passed as build arguments to align with the host user's permissions. This helps prevent issues related to inaccessible files after processing and makes the container usable across different systems and setups.

3.6 Entry point and multi-interface support

To improve usability and offer more flexibility, a custom `entrypoint.sh` script was added to the container. This script detects the mode of execution requested by the user and automatically launches the appropriate interface: **Command Line Interface (CLI)**, **Interactive Shell**, or **JupyterLab**. This unified entry point simplifies container usage across different workflows and user preferences.

In addition, a preconfigured Jupyter notebook is included within the container. When JupyterLab is launched, this notebook serves as a starting point. Details on how each interface can be accessed and used are provided in Section 5. Overall, these various steps allow the application to be fully containerized and operational, resulting in a final image size of approximately 1GB.

4 Testing and validation

Once the container was built, because of the multitude of configurations and modes of functioning, some testing and validation was absolutely required.

4.1 Functional verification

Rigorous functional testing was performed to ensure that `BIDSme` 's core features operated correctly within the Dockerized environment. We relied on the example data [8] and tutorial [7] provided with `BIDSme` . The validation covered the full pipeline, including the commands:

- `prepare`: to restructure and rename raw data before BIDSifying them.

- `map`: to transform the prepared data into a BIDS-compatible structure using `bidsmap.yaml`.
- `bidsify` and `process`: for full BIDSification or additional post-processing steps.

These commands were executed both through the command-line interface and via Jupyter notebooks. The correctness of the output directories (`prepared/`, `bidsified/`) and the absence of runtime errors were used as criteria for validation.

4.2 Interface testing

To ensure consistency and reliability across usage modes, each interface — CLI, interactive shell, and JupyterLab — was tested independently. The CLI was validated using direct `docker run` commands, confirming that input/output volumes were properly mounted and outputs were written as expected. The JupyterLab interface was tested to ensure that notebooks could be executed end-to-end, and that preloaded configurations and paths were properly set up.

4.3 Data consistency and output validation

To confirm that the container did not alter the logic or quality of the original `BIDSme` tool: an example dataset [8] were processed through both local, i.e. non-containerized, and Dockerized versions of `BIDSme`. The resulting outputs were compared for structural consistency and file integrity using automated directory checksums and visual inspections of BIDS-compliant folders. No differences were detected between the two outputs, thus validating the correctness of the containerized version.

4.4 Reproducibility assurance

To test reproducibility, the same dataset was processed multiple times under identical container configurations across different machines, running Windows or Linux OS. The outputs were identical in both structure and content, demonstrating that the container provides a reproducible environment across platforms. This process also helped identify potential permission issues arising from Docker's default behavior, which often assigns `root` ownership to files and directories created within containers.

5 Usability and distribution enhancements

To improve the accessibility, maintainability, and practical deployment of the containerized version of `BIDSme`, several usability-focused tools and structures were developed and published alongside the Dockerfile.

5.1 Environment management via Docker Compose

To streamline usage and avoid the need to repeatedly type long and error-prone `docker run` commands with multiple volume mounts and environment variables, a Docker Compose-based setup was introduced. Two separate `docker-compose.yml` files were created:

- A **development version**, designed to launch the container in JupyterLab mode with persistent notebooks and environment variables useful for debugging and testing.
- A **production version**, meant for regular use and deployment, also with a persistent production notebook.

Depending on the options provided after the `lab` argument, the `entrypoint.sh` script and the `init_bidsme_lab.py` file dynamically initialize default paths by assigning them to well-defined

variables. This mechanism streamlines the workflow by simplifying file access and reducing manual configuration within the JupyterLab environment.

Moreover, the Docker Compose files automatically mount the required volume structure under `/mnt` and set key environment variables such as user ID, group ID, and/or JupyterLab port. This approach enhances reproducibility and minimizes the risk of user error during setup.

5.2 Helper scripts and build wrappers

To further simplify the image management, wrapper scripts were implemented for both building and running the container. For example:

- A `build.sh` script condenses the Docker build process, handling arguments such as the desired `BIDSme` version. The script also fetches and tags the correct Git commit from the `BIDSme` repository, ensuring the containerized version is tightly coupled to the underlying software version.
- `bidsme_prepare.sh` is a convenience script that automates the initialization of a dataset in a single step. It ensures correct directory creation, mounts, and command execution to perform the initial `prepare` phase in a quick and reliable way.

These scripts ease the task of contributors and users to generate consistent, reproducible containers while maintaining compatibility with upstream changes.

5.3 Multi-interface access modes

To accommodate different usage profiles and levels of technical expertise, the container supports three primary modes of operation, all unified under a single `entrypoint.sh`:

- the **CLI mode** allows direct execution of `BIDSme` commands such as `prepare`, `map`, or `bidsify` via Compose or Docker CLI.
- the **interactive shell mode** launches a bash shell into the container to explore its environment, test commands manually, or inspect intermediate files.
- the **JupyterLab mode** opens a web-based notebook interface with preloaded paths, allowing easier exploration, configuration, and execution for users less comfortable with the command line.

Depending on how the container is started (via CLI or Compose), the appropriate mode is automatically triggered by the `entrypoint.sh` logic, see Figure 3. A dedicated configuration notebook is also embedded in the container to assist users launching via JupyterLab.

For example, the JupyterLab interface in production mode can be launched using a full Docker command:

```
docker run -p 8888:8888 \
  -v $(pwd)/rawdata_prod:/mnt/rawdata \
  -v $(pwd)/prepared_prod:/mnt/prepared \
  -v $(pwd)/bidsified_prod:/mnt/bidsified \
  -v $(pwd)/configuration_prod:/mnt/configuration \
  bidsme:<version> lab prod
```

Alternatively, the same setup can be started using the predefined production Compose file:

```
docker compose -f docker-compose.prod.yml run --service-ports bidsme lab
```

Each method provides consistent behavior, with automatic initialization of environment variables and preloaded notebook paths for a smooth user experience. Notably, Docker Compose proves especially useful by reducing a long and error-prone multi-line Docker command to a single, concise instruction.

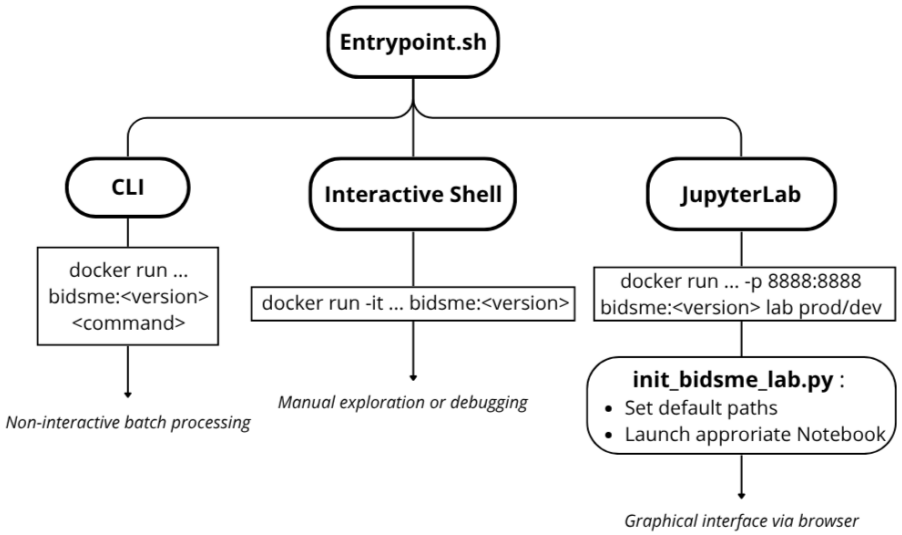

Figure 3: Dockerfile's entry point logic

5.4 Public distribution via GitHub

The full containerization project is hosted on GitHub [16], providing open access to the Dockerfile, helper scripts, and example configurations. This ensures transparency, version control, and easy collaboration across users and contributors. The repository also includes documentation to guide new users through installation, setup, and execution of the container.

6 Integration into Neurodesk

Neurodesk is an open-source, containerized computing environment designed to simplify and standardize neuroimaging data analysis [6]. Distributing `BIDSme` through Neurodesk would ensure a broader distribution and potentially easier integration in various data handling pipelines.

6.1 Integration process

The integration process was carried out by creating a YAML configuration file that describes the `BIDSme` application's specifications, metadata, and build instructions. This YAML file acts as the blueprint used by Neurodesk's automated build system:

```
name: bidsme
version: 1.9.3
architectures:
  - x86_64
categories:
  - data organisation
  - bids app
build:
```

```
kind: neurodocker
base-image: bradley987/bidsme:1.9.3
pkg-manager: apt
```

Unlike other Neurodesk applications that are built from a minimal Ubuntu base image, the integration of `BIDSme` relied on an existing Docker image that had already been containerized and optimized. This approach ensured a reliable and reproducible environment while reducing development overhead. The YAML file additionally specifies metadata such as documentation links, licensing, and an automated test script to verify that the command-line interface is available:

```
- test:
    name: testScript
    script: |-
      #!/bin/bash
      bidsme --help
```

The integration leverages Neurodesk's automated build pipeline, which processes the YAML file and generates the corresponding Singularity container image. The workflow includes GitHub submission, validation, container build, optimization, and final deployment into the Neurodesk registry. Since the integration is based on the preconfigured `bradley987/bidsme:1.9.3` Docker image, the build remains lightweight and easily updatable.

6.2 Usage and Validation

Within the Neurodesk environment, `BIDSme` is currently accessible through the module loading mechanism. This ensures that the tool is properly initialized within the containerized environment before use:

- **Module Loading**:

```
import module
await module.load('bidsme/1.9.3')
```

- **Command Execution** (in JupyterLab or terminal):

```
!bidsme --help
!bidsme prepare /path/to/data /path/to/output <options>
```

At the current stage, `BIDSme` cannot be directly imported as a Python package within Neurodesk native Python environment. Access is therefore limited to its CLI once the module is loaded, or through a JupyterLab notebook if launched afterwards.

Validation involved checking installation correctness, core functionality of dataset preparation and BIDSification, module-loading behavior, and user experience in both JupyterLab and terminal contexts. These tests confirmed that the tool runs reliably inside the Neurodesk environment.

6.3 Benefits and Impact

The successful integration of `BIDSme` into Neurodesk delivers multiple benefits for the neuroimaging research community:

- **Accessibility**: `BIDSme` can be accessed immediately without requiring manual installation
- **Reproducibility**: A consistent and controlled containerized environment ensures reproducible results
- **Collaboration**: Researchers share the same standardized BIDS conversion tools
- **Education**: Provides newcomers with preconfigured access to `BIDSme`, lowering the entry barrier
- **Maintainability**: Centralized updates via Neurodesk simplify long-term maintenance

Overall, this integration significantly improves the accessibility of `BIDSme` and facilitates the adoption of BIDS standards across diverse research environments.

7 Conclusions

The actual containerized version of `BIDSme` offers significant improvements in reproducibility and ease of deployment. By encapsulating all dependencies and configuration, it eliminates the variability often encountered when neuroimaging pipelines are run on different systems. Providing several ways to use the containerized app gives users the freedom to perform processing according to their preferences and needs, while reducing the tediousness of this kind of work.

Despite the advantages provided by the containerized setup, the final image remains relatively large (approximately 1 GB), which may pose challenges for environments with limited storage or bandwidth. Additionally, using the container without Docker Compose can be bothersome, as it requires manually specifying long and complex volume-mounting commands. Additionally, although containerization improves portability, a basic familiarity with Docker is still necessary for effective usage, which may represent a barrier for users without prior experience in container technologies.

`BIDSme` has already been successfully integrated into the Neurodesk ecosystem as a containerized application. This ensures accessibility and reproducibility across platforms, in line with Neurodesk's modular container-based approach. However, since `BIDSme` is a pure Python tool, keeping it inside a dedicated container is not strictly necessary. A more sustainable approach would be to make `BIDSme` directly available within Neurodesk's shared Python environment (via a virtual environment installation). This would reduce maintenance overhead, lower image size, and simplify updates, while still providing users with the same level of accessibility.

Future work will therefore focus on discussing with the Neurodesk team whether such an installation strategy could be adopted, ensuring that `BIDSme` remains fully integrated while avoiding

the unnecessary burden of maintaining a standalone container.

Acknowledgments

B. Spitz received financial support from the Erasmus+ program of the European Union and from the Région Grand Est (France), through its international mobility support program, to support his research internship at the GIGA – CRC Human Imaging, University of Liège, Belgium. C. Phillips is supported by the F.R.S.-FNRS., Belgium. The original development of `BIDSme` , as well as N. Beliy at the time, were supported by an Excellence of Science grant (#30446199, 2017, "MEMODYN, The journey of a memory: dynamics of learning and consolidation in maturation and ageing") from the F.R.S.-FNRS and the FWO, Belgium.

References

[1] K. J. Gorgolewski, T. Auer, V. D. Calhoun, R. C. Craddock, S. Das, E. P. Duff, G. Flandin, S. S. Ghosh, T. Glatard, Y. O. Halchenko, D. A. Handwerker, M. Hanke, D. Keator, X. Li, Z. Michael, C. Maumet, B. N. Nichols, T. E. Nichols, J. Pellman, J.-B. Poline, A. Rokem, G. Schaefer, V. Sochat, W. Triplett, J. A. Turner, G. Varoquaux, and R. A. Poldrack, "The Brain Imaging Data Structure, a Format for Organizing and Describing Outputs of Neuroimaging Experiments," *Scientific Data*, vol. 3, no. 1, p. 160044, Jun. 2016.

[2] R. A. Poldrack, C. J. Markiewicz, S. Appelhoff, Y. K. Ashar, T. Auer, S. Baillet, S. Bansal, L. Beltrachini, C. G. Benar, G. Bertazzoli, S. Bhogawar, R. W. Blair, M. Bortoletto, M. Boudreau, T. L. Brooks, V. D. Calhoun, F. M. Castelli, P. Clement, A. L. Cohen, J. Cohen-Adad, S. D'Ambrosio, G. de Hollander, M. de la Iglesia-Vayá, A. de la Vega, A. Delorme, O. Devinsky, D. Draschkow, E. P. Duff, E. DuPre, E. Earl, O. Esteban, F. W. Feingold, G. Flandin, A. Galassi, G. Gallitto, M. Ganz, R. Gau, J. Gholam, S. S. Ghosh, A. Giacomel, A. G. Gillman, P. Gleeson, A. Gramfort, S. Guay, G. Guidali, Y. O. Halchenko, D. A. Handwerker, N. Hardcastle, P. Herholz, D. Hermes, C. J. Honey, R. B. Innis, H.-I. Ioanas, A. Jahn, A. Karakuzu, D. B. Keator, G. Kiar, B. Kincses, A. R. Laird, J. C. Lau, A. Lazari, J. H. Legarreta, A. Li, X. Li, B. C. Love, H. Lu, E. Marcantoni, C. Maumet, G. Mazzamuto, S. L. Meisler, M. Mikkelsen, H. Mutsaerts, T. E. Nichols, A. Nikolaidis, G. Nilsonne, G. Niso, M. Norgaard, T. W. Okell, R. Oostenveld, E. Ort, P. J. Park, M. Pawlik, C. R. Pernet, F. Pestilli, J. Petr, C. Phillips, J.-B. Poline, L. Pollonini, P. R. Raamana, P. Ritter, G. Rizzo, K. A. Robbins, A. P. Rockhill, C. Rogers, A. Rokem, C. Rorden, A. Routier, J. M. Saborit-Torres, T. Salo, M. Schirner, R. E. Smith, T. Spisak, J. Sprenger, N. C. Swann, M. Szinte, S. Takerkart, B. Thirion, A. G. Thomas, S. Torabian, G. Varoquaux, B. Voytek, J. Welzel, M. Wilson, T. Yarkoni, and K. J. Gorgolewski, "The past, present, and future of the brain imaging data structure (BIDS)," *Imaging Neuroscience*, vol. 2, pp. imag–2–00 103, 03 2024. [Online]. Available: https://doi.org/10.1162/imag_a_00103

[3] N. Beliy, G. Hammad, and C. Phillips, "BIDSme." [Online]. Available: https://github.com/CyclotronResearchCentre/BIDSme

[4] N. Beliy, C. Guillemin, E. Pommier, G. Hammad, and C. Phillips, "BIDSme: Expandable BIDS-ifier of Brain Imagery Datasets," *Journal of Open Source Software*, vol. 8, no. 92, p. 5575, Dec. 2023.

[5] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[6] A. I. Renton, T. T. Dao, T. Johnstone, O. Civier, R. P. Sullivan, D. J. White, P. Lyons, B. M. Slade, D. F. Abbott, T. J. Amos, S. Bollmann, A. Botting, M. E. J. Campbell, J. Chang, T. G. Close, M. Dörig, K. Eckstein, G. F. Egan, S. Evas, G. Flandin, K. G. Garner, M. I. Garrido, S. S. Ghosh, M. Grignard, Y. O. Halchenko, A. J. Hannan, A. S. Heinsfeld, L. Huber, M. E. Hughes, J. R. Kaczmarzyk, L. Kasper, L. Kuhlmann, K. Lou, Y.-J. Mantilla-Ramos, J. B. Mattingley, M. L. Meier, J. Morris, A. Narayanan, F. Pestilli, A. Puce, F. L. Ribeiro, N. C. Rogasch, C. Rorden, M. M. Schira, T. B. Shaw, P. F. Sowman, G. Spitz, A. W. Stewart, X. Ye, J. D. Zhu, A. Narayanan, and S. Bollmann, "Neurodesk: an accessible, flexible and portable data analysis environment for reproducible neuroimaging," *Nature Methods*, vol. 21, no. 5, pp. 804–808, 2024. [Online]. Available: https://doi.org/10.1038/s41592-023-02145-x

[7] N. Beliy, "BIDSme tutorial." [Online]. Available: https://github.com/CyclotronResearchCentre/bidsme_tutorial

[8] ——, "BIDSme example dataset." [Online]. Available: https://github.com/CyclotronResearchCentre/bidsme_examples

[9] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter Notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87 – 90.

[10] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[11] P. T. McGuire, "pyparsing: A Python parsing module," GitHub, 2024, version 3.1.2. [Online]. Available: https://github.com/pyparsing/pyparsing

[12] M. Brett, C. Markiewicz, J. Millman, S. Ghosh, G. Varoquaux, O. Esteban, F. Peréz-García, F. Morency, and C. Burns, "nipy/nibabel: NiBabel Release 5.3.1," Oct 2024. [Online]. Available: https://nipy.org/nibabel/

[13] D. Mason, "SU-E-T-33: pydicom: an open source DICOM library," *Medical Physics*, vol. 38, no. 6, p. 3493, 2011.

[14] X. Li, P. S. Morgan, J. Ashburner, J. Smith, and C. Rorden, "The first step for neuroimaging data analysis: DICOM to NIfTI conversion," *Journal of Neuroscience Methods*, vol. 264, pp. 47–56, 2016.

[15] L. Peng, "DCM-parser: A Lightning DICOM Parser," https://github.com/PL97/DICOM_Parser, 2022.

[16] B. Spitz, N. Beliy, and C. Phillips, "BIDSme Containerization Project." [Online]. Available: https://github.com/CyclotronResearchCentre/BIDSme_containerisation