Heinrich Heine
Universität
Düsseldorf

Reconceptualizing neural function as high-dimensional brain state dynamics

Inaugural-Dissertation

zur Erlangung des Doktorgrades der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Adina S. Wagner
aus Wolfsburg

Düsseldorf, Oktober 2023

aus dem Institut für Experimentelle Psychologie
der Heinrich Heine Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Berichterstatter:

1. Prof. Dr. Gerhard Jocham

2. PD Dr. Jan Hirschmann

Tag der mündlichen Prüfung:

Abstract

Human brain mapping has focused on ascribing function to specific brain areas or networks. But neural signals can be re-conceptualized from blobs of activity at any one time to dynamic and evolving trajectories through the high dimensional space spanned by the sensors that pick it up. Over the last decade, this approach generated novel insights in the study of cognitive processes, and a variety of methods were formalized that build up on it, among them the shared response model. In this work, the shared response model was used to study a cognitive process that has yet evaded our full understanding, namely that of working memory maintenance, the offline representation of information in absence of its original item. Using magnetoencephalography data from a delayed decision making task, this work explored lower-dimensional subspaces of the high dimensional neural data to find latent factors representing fundamental trial properties during maintenance. The analyses reveal ample evidence for preparatory decision signals during maintenance instead of decision-relevant stimulus properties, and establish the shared response model as a viable methodological choice for magnetoencephalography data. However, the study of neural processes is not isolated from the social, organizational, and technical challenges underlying science. Thus, this work further presents several projects in the area of research data management and research software engineering that laid foundational steps to enable trustworthy reusable project outcomes, from a software tool for data management and a community-based documentation effort to improve scientific conduct, to pragmatic strategies for research data management and the technical implementation of a scalable framework for reproducible data analysis. While the findings on neural data for now only pertain to a single dataset and experiment, the technical solutions constitute widely reusable tools and methods for standard challenges in neuroscience and beyond.

Zusammenfassung

Die Kartierung des menschlichen Gehirns hat sich darauf konzentriert, bestimmten Gehirnbereichen oder Gruppen funktional vernetzer Hirnareale eine Funktion zuzuordnen. Eine alternative Betrachtungsweise ist es jedoch, neuronale Signale als dynamische und sich entwickelnde Trajektorien durch hochdimensionale Signalräume zu rekonzeptualisieren. In den letzten zehn Jahren hat dieser Ansatz zu neuen Erkenntnissen bei der Untersuchung kognitiver Prozesse geführt, und eine Reihe neuartiger Methoden etabliert – unter ihnen das Shared Response Model. In der vorliegenden Arbeit wird das Shared Response Model genutzt, um einen kognitiven Prozess zu untersuchen, der sich bisher unserem vollständigen Verständnis entzogen hat: Die Aufrechterhaltung des Arbeitsgedächtnisses, d. h. die Bereithaltung von Informationen in Abwesenheit des ursprünglichen Reizes. In dieser Arbeit wurden daher anhand von Magnetoenzephalographie-Aufnahmen eines Experiments zur verzögerten Entscheidungsfindung niederdimensionale Unterräume der hochdimensionalen neuronalen Daten untersucht, um latente Faktoren zu finden, die grundlegende Experimentcharakteristika im Arbeitsgedächtnis abbilden. Die vorgestellten Datenanalysen liefern Belege dafür, dass vorbereitende Entscheidungssignale anstelle von entscheidungsrelevanten Stimuluscharakteristika im Arbeitsgedächtnis aufrecht erhalten werden, und etablieren das Shared-Response-Modell als anwendbar auf Magnetoenzephalographiedaten. Analysen neuronaler Prozesse sind jedoch nicht von den sozialen, organisatorischen und technischen Herausforderungen wissenschaftlicher Projekte isoliert. Daher werden in dieser Arbeit auch mehrere Projekte aus den Bereichen Forschungsdatenmanagement und Forschungssoftwareentwicklung vorgestellt, mit denen vertrauenswürdige, wiederverwendbare Forschungsresultate erstellt werden können – von einem Softwaretool und einem Dokumentationsprojekt zur Verbesserung von Forschungsdatenmanagement bis hin zu pragmatischen Datenmanagementstrategien und der technischen Umsetzung eines skalierbaren computationalen Frameworks für reproduzierbare Datenanalysen. Während sich die Erkenntnisse zum Arbeitsgedächtnis vorerst nur auf einen einzigen Datensatz und ein einziges Experiment beziehen, stellen die technischen Lösungen weithin wiederverwendbare Werkzeuge und Methoden für Standardaufgaben in den Neurowissenschaften und darüber hinaus dar.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

ABCD Adolescent Brain Cognitive Development Study. 19

API Application Programming Interface. 28

BIDS Brain Imaging Data Structure. 18

CAT Computational Anatomy Toolbox. 46

DFG German Research Foundation. 27

EEG electroencephalography. 14

EOG electrooculography. 56

fMRI functional magnetic resonance imaging. 14

FZJ Research Center Jülich. 11

GDPR General Data Protection Regulation. 19

GM gray matter. 47

GUI graphical user interface. 24

HCP Human Connectome Project. 19

HHU Heinrich Heine University Düsseldorf. 11

HIPAA Health Insurance Portability and Accountability Act. 19

HPC high performance computing. 41

HTC high-throughput computing. 46

ICA independent component analysis. 60

MEG magnetoencephalography. 11

MIT Massachusetts Institute of Technology. 29

MRI magnetic resonance imaging. 19

MSE mean squared error. 49

OHBM Organization for Human Brain Mapping. 29

OSF Open Science Framework. 22

PCA principal component analysis. 15

PET positron emission tomography. 53

RDM research data management. 12

RSE research software engineering. 12

SRM shared response model. 64

SSS Signal Space Separation. 58

TIV total intracranial volume. 47

tSSS spatiotemporal Signal Space Separation. 58

UKB UK Biobank. 19

VBM voxel-based morphometry. 46

WM white matter. 47

1 Introduction

> An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.
>
> *(John Claerbout, rephrased by Buckheit & Donoho)*

In natural settings, value-based decisions under risk often have to be made without visual presence of competing alternatives. Do I commit to buying the red bicycle in the store I'm currently in, or should I return to the previous bike shop and pick the blue one? Do I want to rent the flat I viewed last Sunday, or the apartment I saw on Tuesday? In such situations, relevant properties of alternative options such as their subjective value have to be encoded, held in working memory through a maintenance (or "delay") period, and retrieved in order to form a categorical choice.
To understand the neural processes underlying behavior in these situations, the neural activity in the delay period and how it might represent or maintain relevant properties were a focus of research in the last decades. Although several popular theories emerged over time and were either refuted or refined, and the field has progressed from studying spiking activity in single cells to computationally elaborate models of spatially and temporally distributed activity, a comprehensive understanding is yet lacking (Sreenivasan and D'Esposito, 2019). The main goal in the conception of this research project has been to make a contribution to the study of brain state dynamics in working memory, a recently emerging theory that reconceptualizes neural activity as a trajectory through a high-dimensional neural space. To this end, I was given access to a pre-existing magnetoencephalography (MEG) dataset from human participants that played an experiment with a delayed decision making task, acquired by Kaiser, Gründler, and Jocham (2016), the *memento* dataset.
This research endeavor posed several challenges. One lay in acquiring the relevant information to work productively and confidently with this dataset. Data was captured at the Clinic for Neurology at the Otto-von-Guericke University Magdeburg, an institution to which I did not have access or first-hand knowledge about. The original lead investigators of the project had switched their focus to other projects by the time I started working with it, and both had left their research positions by the time I finished. Obtaining their first-hand knowledge was thus impeded not merely by fading memory over time, but also as their progression to other projects and out of academia naturally limited contact. While the dataset had been processed extensively already, the data was in an idiosyncratic organization, many existing analyses were in an interim state, and the project, excluding raw and processed data, was a collection totaling 20.761 files. Thus, a prerequisite of my analyses was to prepare the dataset such that it became usable for me, and ideally also to future other researchers that were not involved in its acquisition or processing.
A different challenge lay in the processing setup. The data were acquired and partially analyzed on infrastructure of the University of Magdeburg. After a change of affiliations, storage and further analyses were moved to the computational cluster of the Heinrich Heine University Düsseldorf (HHU). My own analyses, finally, were done on computational infrastructure at the Institute of Neuroscience and Medicine at the Research Center Jülich (FZJ). Typically, changes of computational infrastructure are detrimental to the utility of the processing and analysis scripts. Thus, an explicit aim of all processing was portability, such that research outcomes yielded on FZJ infrastructure were readily usable on servers at the HHU or elsewhere.
Finally, the to-be-conducted analyses were highly exploratory. Although the dataset had not been published before, a large amount of analyses had already been conducted. But with the exception of

a poster presentation on preliminary decoding results (Kaiser, Gruendler, et al., 2018), all analyses yielded null results and remained unpublished. Thus, in an attempt to extend previous analyses and search for remaining insights in the data with novel methodology, I set out to conduct analyses rooted in functional alignment and dimensionality reduction that have yet rarely been used on MEG data. Maintaining routines for reproducibility and reusability, however, is not trivial when the analyses are not mapped out entirely and the necessary software for the analyses is in flux. Keeping all analyses transparent, self-descriptive to others, and reproducible became a guiding principle.
Although the challenges I outlined above shaped the course of the project and gave way to productive projects to solve them, they are by no means uncommon in scientific endeavors. Science is incremental and its outcomes go beyond journal articles – code, data, results, or tools of previous finished or unfinished projects are reused or extended in later activities (Mons, 2018), and the people who start and finish a project are not necessarily the same (Puce and Hämäläinen, 2017). Science is also increasingly collaborative and spatially distributed (Csomós et al., 2020). Benefiting from differential expertise of several collaborators requires mutual understanding of a project and its components. But the relevant research data management skills are not commonly part of research training curricula (Grisham et al., 2016). Nevertheless, over several decades already, data shared across institutions or even openly has been a corner stone of neuroscientific progress (e.g., Ferguson et al., 2014; Niso, Rogers, et al., 2016), and this trend is increasing (Gorgolewski and Poldrack, 2016). A common challenge is thus to keep projects in such a state that they can be understood without the expertise of their original authors (Puce and Hämäläinen, 2017). This makes technical and organizational elements of scientific practice, such as research data management (RDM) and research software engineering (RSE), a fundamental prerequisite. Going sequentially through the different challenges and solutions to conducting reproducible brain state analyses, this thesis summarizes my conjunct work on research data management solutions, research software engineering practices, and their application in a scientific project that spanned several institutions and generations of researchers.
To lay the foundations for this, the remainder of this chapter establishes a number of prerequisites. The first section 1.1 outlines a brief overview of the literature in the field of working memory maintenance, and deduces the choice of analyses methods for dimensionality reduction for the study of brain states in Chapter 4. Afterwards, section 1.2 introduces the topic of research data management, and highlights central concepts and tools that will be a focus in the upcoming chapters 2 and 3. As I will lay out in this section and in upcoming chapters, RDM is a foundational element within good scientific practice, and an important prerequisite for computational neuroscience.

1.1 Brain state spaces

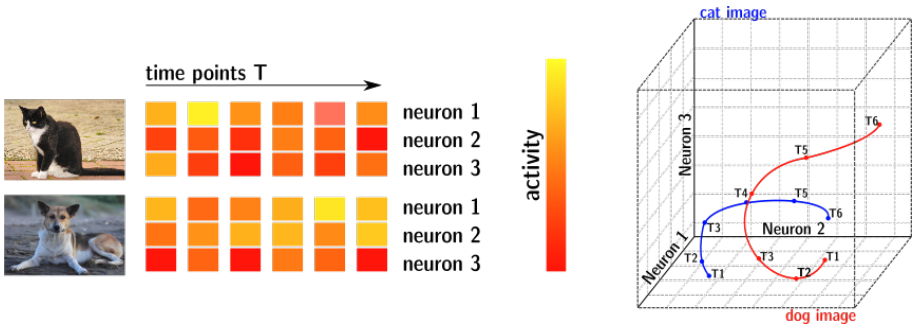
The concept of *brain states*, global neural activity patterns that are associated with distinct cognitive processes or behavioral states, has emerged in the study of various brain functions, from sleep (Lee and Dan, 2012), to motor processes (Pfurtscheller et al., 2008), or functional connectivity (Finn et al., 2017). Over the course of the last decades, it also emerged in the study of working memory. Early neuroimaging and electrophysiology studies suggested persistent neural activity ("spiking") in specific brain areas during delay periods as a mechanism of working memory maintenance (Goldman-Rakic, 1995). Sustained, above-baseline activity that starts during a sample presentation, lasts through the memory delay, and returns to base line activity after a response were found in prefrontal regions in human (e.g., Courtney et al., 1997) and non-human primates (e.g., Funahashi

et al., 1989; Fuster and Alexander, 1971; Miller et al., 1996). And single-cell recordings from the prefrontal cortex in monkeys unveiled that individual neurons display activity that is selective to specific task-relevant aspects, such as task rules, spatial location, or stimulus features (Wallis et al., 2001; White and Wise, 1999), suggesting the existence of neurons that uphold working memory content with a fixed selectivity for certain properties. Others have pointed out, however, that persistent spiking in prefrontal areas could reflect a variety of other different processes, including decision making (Curtis and Lee, 2010) or anticipation of the probe (Nobre and Stokes, 2011), that the high metabolic costs of action potentials is too energetically expensive to hold information in a spiking form (Attwell and Laughlin, 2001), and that distractor tasks are able to remove the persistent activity without an impact on later retrieval, questioning its role in working memory maintenance (LaRocque et al., 2013; Lewis-Peacock et al., 2015).

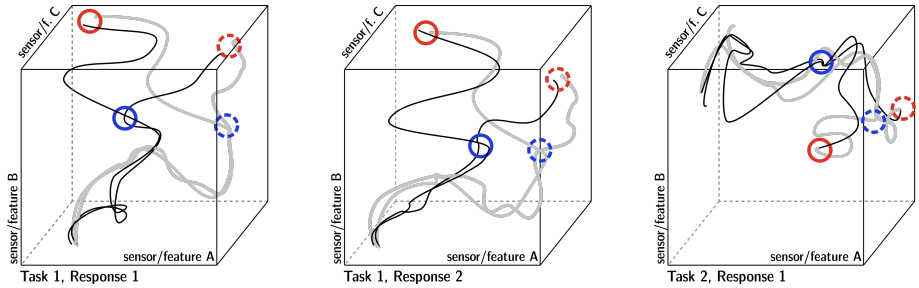
The idea of neural selectivity to specific task-aspects was refined in the adaptive coding framework (Duncan, 2001): Instead of a persistent, fixed selectivity, it postulates that neural responses are temporarily tuned to the particular task. This theory was able to explain how a recording of spiking activity in the same group of neurons relates to object information in one task but a location distinction in the next task (Duncan, 2001). Mongillo et al. (2008) proposed a highly influential theory according to which short-term neural plasticity changes are the underlying mechanism of working memory maintenance. According to it, working memory maintenance is established via increased residual calcium levels at the presynaptic terminals of neurons, which causes a short-term synaptic facilitation, akin to synaptic weights that link neurons coding for a working memory item. With this facilitation, the memory can be transiently held for about 1s without enhanced spiking activity in a network of neurons. Building up on this theory, Stokes (2015) proposed the dynamic coding framework, in which working memory is mediated by rapid transitions in such "activity silent" neural states. While these states mediate flexible, context-dependent processing, they do not emerge as constant activity and rather as "hidden states", appearing as altered response sensitivities of neural networks, established via short-term and long-term synaptic plasticity and temporal functional connectivity changes that influence the response to stimuli. An explanation why spiking activity is nevertheless found during delay periods comes from findings that a task-irrelevant read-out or "ping" signal is able to reactivate the neural assembly (Trübutschek et al., 2017). This raises the possibility that previous findings of neuronal spiking might have been similar readouts from an activity silent population after task-irrelevant non-specific inputs (Wolff et al., 2017). Alternatively, Fiebig and Lansner (2017) hypothesized that occasional spiking of memory-encoding neurons is needed to refresh the activity-silent states.

The idea that distributed representations that selectively favor information relevant to the current task emerge on the level of neural populations was central in other theories as well. Rigotti et al. (2013) proposed the concept of mixed selectivity, in which information is distributed across neurons through non-linear and high-dimensional representations even when it is not observable in individual cells. The rise of multivariate methods such as decoding analyses gave way to thorough investigations of distributed neural patterns. King and Dehaene (2014) formalized this with the temporal generalization method. In this method, several classifiers are trained on different time slices of training data each, and tested on all available times in the test data, thus revealing whether the neural code is stable or dynamically evolving. Interestingly, studies decoding working memory content found that decodable patterns can be non-stationary. Meyers et al. (2008) decoded object categories from electrophysiological data, and yielded worse decoding the more temporally distant training and testing times slices were apart. Stokes et al. (2013) used temporal cross-correlation analyses to show that population responses in the prefrontal cortex evolve during a memory delay. These representations of task-relevant stimulus information in temporarily and spatially

distributed activity across neurons were termed dynamic population coding (Sreenivasan, Curtis, and D'Esposito, 2014) (see Figure 1.1a).
The theory that cognitive processes evolve through different brain states as dynamic neural trajectories is in line with non-stable representations (Buonomano and Maass, 2009). These trajectories emerge by conceptualizing neural responses as vectors in an N-dimensional space that evolves over time: The axes of the high-dimensional space correspond to measurements, such as raw or transformed data from MEG/electroencephalography (EEG) sensors or functional magnetic resonance imaging (fMRI) voxels. Each time point in the acquisition becomes a point in the high-dimensional space spanned by all sensors, determined by the measurement in each sensor, such as activation strength. In geometric terms, this point is an N-dimensional vector. Neural trajectories emerge by tracing the activity on these axes over multiple time points. Common or distinct brain states could then become evident as converging or diverging trajectories (see Figure 1.1b).



(a) The dynamic coding framework



(b) Idealized neural trajectories through state-spaces

Figure 1.1: A schematic illustration of dynamic coding in neural populations as trajectories through high dimensional spaces. The left side of a) depicts a population of three neurons (squares), and their activity (color) over six time points time in response to two visual stimuli. The neural code in this population is not stationary, but dynamic: Each stimulus evokes a distinct response, but within this response, each time point has a different pattern of neural activity, too. By conceptualizing activity patterns over time as vectors, these patterns can be visualized in a multi-dimensional space (right). This space has as many dimensions as there are measurement sites (e.g., neurons, sensors, voxels, or electrodes). Dynamic population codes create trajectories through this space (adapted from Meyers, 2018, Fig. 2). b) shows an idealized illustration of neural trajectories in different states for two decision alternatives (blue) and two response alternatives (red), for two tasks in a delayed-response-mapping paradigm. Assigning meaning to the axes of the state-space may yield insights about brain states in different experiment conditions. Adapted from Jocham, Krauel, Hanke (unpublished).

To bridge the gap to theories of activity silent states, the neural responses that form dynamic trajectories are not solely determined by external stimuli, but also internal state changes of the network, such as the strength of synaptic connections, or excitatory or inhibitory influences from other networks (Buonomano and Maass, 2009). Distinct neural states were often hypothesized to reflect steps in mental processing (e.g., Seidemann et al., 1996). Muhle-Karbe et al. (2021), for example, proposed a hierarchy of functional states in working memory, with items relevant for a pending decision in the most active state, and items relevant only for later use in latent states. But studies were also able to extract external stimulus information from the trajectories of transient states, for example odors (Mazor and Laurent, 2005). The study of these neural states and their transition has emerged as its own field, employing multivariate methods such as decoding across neural populations, hidden Markov models (HMMs) (Rainer and Miller, 2000) to model ensemble activity as a sequence of constantly shifting neural states (Vidaurre et al., 2016), or functional alignment (Haxby, Guntupalli, Connolly, et al., 2011) to find common representations across idiosyncratic neural responses.

In parallel, dimensionality-reduction gained popularity in the analysis of neural population data (Cunningham and Yu, 2014). Though partially rooted in attempts to make analyses computationally feasible, a central underlying justification of dimensionality reduction approaches is that the measured neural activity is too complex compared with the few relevant task or stimulus aspects that require encoding, and that the neural signal of interest must be embedded in a subspace of the high-dimensional neural space. Santhanam et al. (2009) for example employed a decoding approach based on factor-analysis to reduce noise in high-dimensional neural recordings to improve the performance of neural prostheses, which otherwise suffered from the neural variability introduced by changes in attentional state or wakefulness. The motor signal of interest thus lay embedded in a high-dimensional space containing variability from unrelated cognitive sources, and identifying this subspace improved the study of the signal of interest. Fittingly, the term "effective dimensionality" of population activity emerged in the literature to identify shared components of collective dynamics that reflect task variables (Jazayeri and Ostojic, 2021). Investigations of dynamic or stable population codes for working memory have been done in conjunction with dimensionality reduction methods, too, and yielded interesting observations. Murray et al. (2017) used principal component analysis (PCA) on electrophysiology recordings acquired from primate prefrontal cortex during a working memory task. They found a lower-dimensional subspace in the high-dimensional state space in which stimulus representations were stable across the cue and delay epochs. Machens et al. (2010) likewise used PCA in a working memory task where non-human primates compared the frequency of two vibrations to identify a 6-dimensional subspace in which this tactile stimulus information is represented.

Studying working memory maintenance using dimensionality reduction methods and the concept of neural trajectories through a state-space is thus a promising approach for new insights. It re-conceptualizes neural function from averaged spiking activity that aims to draw conclusions about the brain area of a cognitive process, to dynamics in multi-dimensional spaces that might bear the potential to draw conclusions about the lower-dimensional components our cognition is built up on. The path from data acquisition or raw data to such a data analysis is, however, more complex than the method section of the final paper makes it appear. Before this thesis continues with the analyses of neural trajectories in working memory, the upcoming two chapters and the remainder of the introduction are dedicated to concepts and projects around RDM that were ultimately relevant pillars of the project.

1.2 Research data management

A foundation of scientific insight lies in research data management. Research data encompasses everything that is produced in the life span of a research project. From raw data acquisitions, preprocessed or otherwise standardized datasets, to software, analysis scripts, results, figures, compiled reports, and other final or intermediate research outcomes. To disambiguate the term "research data" from the smaller-scoped meaning of "data" in colloquial language (the outcome of a data acquisition in an experiment), it is also referred to as "research objects", and, in case it exists in purely electronic form, as "digital research objects" (Bechhofer, De Roure, et al., 2010).
Typically, research objects have a much longer life span than the project that creates them. Science is an incremental process that produces and builds up on more than just published journal articles (Mons, 2018), and code, data, results, or tools of previous finished or unfinished projects fuel new undertakings. RDM describes the handling of these research objects through their entire life cycle: from curation, use, publication and sharing, archiving to re-use or destruction (Figure 1.2). Ultimately, it ensures that research objects are preserved to act as an evidence base for findings, and as a discoverable resource for further reuse.

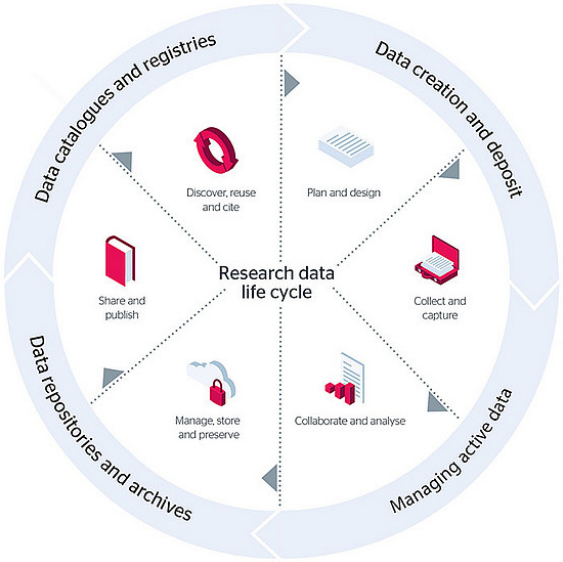


Figure 1.2: The life cycle of digital research objects. License: JISC Research data management toolkit (www.jisc.ac.uk/guides/rdm-toolkit), CC-BY-ND.

1.2.1 The FAIR guiding principles for scientific data management and stewardship

Since their publication, the so-called FAIR principles (Wilkinson et al., 2016) have become guidelines for RDM efforts for digital research objects. They describe four measurable properties of research data that – the better they are fulfilled – serve the ultimate goal to improve the discoverability and reusability of data by machines and humans alike (Wilkinson et al., 2016). The FAIRness of data can be improved on each of the four related but separable major characteristics that are behind the FAIR acronym (taken verbatim from Wilkinson et al.):

- **F**indable: To be Findable, (meta)data are assigned a globally unique and persistent identifier (F1); data are described with rich metadata (F2); metadata clearly and explicitly include the identifier of the data it describes (F3); (meta)data are registered or indexed in a searchable resource (F4)
- **A**ccessible: To be Accessible, (meta)data are retrievable by their identifier using a standardized communications protocol (A1); the protocol is open, free, and universally implementable (A1.1); the protocol allows for an authentication and authorization procedure, where necessary (A1.2); metadata are accessible, even when the data are no longer available (A2)
- **I**nteroperable: To be Interoperable, (meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation (I1); (meta)data use vocabularies that follow FAIR principles (I2); (meta)data include qualified references to other (meta)data (I3);
- **R**eusable: To be Reusable, meta(data) are richly described with a plurality of accurate and relevant attributes (R1); (meta)data are released with a clear and accessible data usage license (R1.1); (meta)data are associated with detailed provenance (R1.2); (meta)data meet domain-relevant community standards (R1.3)

A pivotal factor for the importance of the FAIR principles is the increasing digitization of research practice, and with it, a rapid growth in research data (German Research Foundation, 2020). Innovation relies on the ability to integrate new and existing data, and where the amount of data to discover, understand, and consolidate requires automation, research objects themselves need to become self-descriptive. For this, the term "machine-actionable" is central to the FAIR principles. It is used to describe an ideal state in which a digital object provides detailed information (metadata) to an autonomously-acting, computational data explorer in two possible contexts: For one, as metadata about the object ("What is it?"), and second, as metadata about its content ("How do I process it?") (Wilkinson et al., 2016). Notably, the FAIR principles do not suggest specific metadata standards, implementations, or technology choices. They do highlight the importance of community-endorsed vocabularies, however, and advocate to either extend them if they do not yet include the attributes required for rich annotations, or create suitable new community-endorsed vocabularies.
The end state of FAIR RDM for any digital research object is a high quality digital publication that facilitates and simplifies its discovery, evaluation, and reuse in downstream studies. This reusability refers to the further use or utilization inside or outside its original research context, by the original author or different actors. It is arguably the most important feature of a research object for various stakeholders: It can make research more efficient, foster collaborations, and speed up progress, and as it can reduce duplicate efforts it constitutes an economical benefit for funders as well as the public and industry (Nationale Forschungsdaten Infrastruktur, 2022). Considering additional time spent, cost of redundant storage, licensing costs, research retraction, double funding, and missed potential for economic growth, the European Commission estimates that the cost of not having FAIR data lies between 10 and 26 billion euro per year (European Commission and Directorate-General for Research and Innovation, 2019). The largest position in their cost-benefit analysis is the time wasted by researchers when non-FAIR data takes more time to find, retrieve, clean, integrate, and peer review. The remainder of this chapter will outline RDM requirements and solutions in the field of neuroimaging that aim to increase reusability. They will come into practice in the preparation of the dataset in Chapter 4 as reusing previous findings has played a central part in this thesis as well.

But one aspect of reusability is reproducibility to establish trust, which is a more important side effect of good data management than a cursory reading of the FAIR principles implies. The topic of reproducibility will thus also be continued later, and it is central to Chapter 3.

1.2.2 Research data management in neuroimaging

Neuroimaging data poses a number of additional challenges for research data management. For one, the field is characterized by complex datasets. They typically encompass different modalities (such as imaging, electro-physiological, and behavioral measurements), and often entail several recording sessions. The Brain Imaging Data Structure (BIDS) (Gorgolewski, Auer, et al., 2016) is a community-endorsed standard for organizing and describing neuroimaging data, and is widely considered as a successful solution for data standardization in such datasets. It defines common and modality specific schemes for file names and file organization, file formats, and metadata to accompany raw or derived data, i.e., both original acquisitions as well as the outputs of common processing pipelines. BIDS has widespread and growing support for different neuroimaging modalities, and is made a common prerequisite by neuroscientific data portals such as OpenNeuro (Markiewicz et al., 2021) or processing tools such as BIDSApps (Gorgolewski, Alfaro-Almagro, et al., 2017) or fMRIprep (Esteban et al., 2019). An example of the *memento* MEG dataset, organized idiosyncratically or structured according to BIDS, is shown in Figure 1.3.

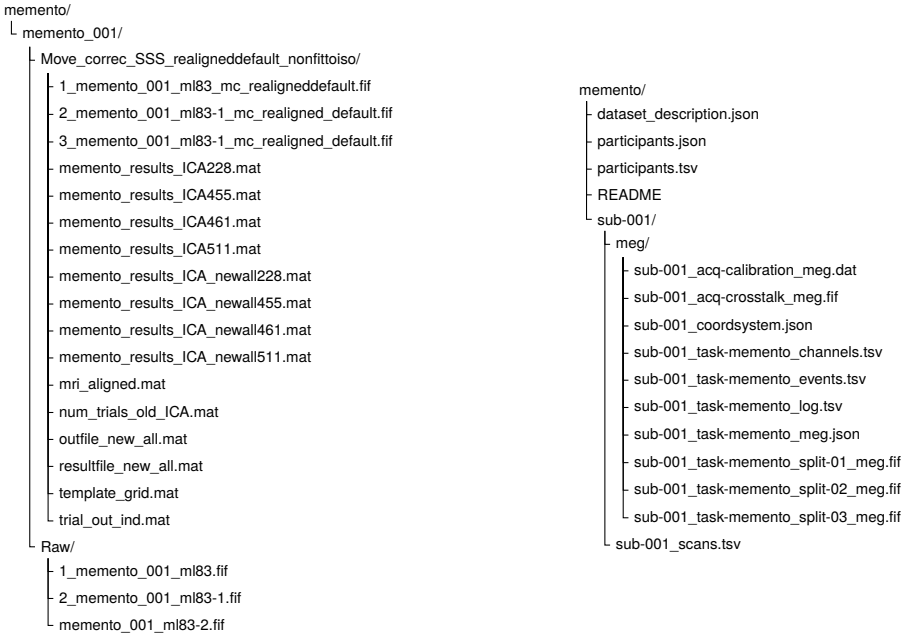


Figure 1.3: File organization in a single subject's MEG acquisition. The left side shows the file tree of the data from a project directory with an idiosyncratic organization. It includes a mix of raw data, preprocessed data, and intermediate results, and the naming scheme is inconsistent. The right side shows the data structured according to BIDS. The naming scheme is consistent, and apart from raw MEG data the directory includes metadata files from the experiment and acquisition machine that are required to understand the data without the original authors.

The storage demands of neuroimaging data are not trivial, either. BIDS regulates which file formats must be used for which data type, and focuses on open file formats for accessibility and compression to reduce disk space requirements. But neuroimaging data are nevertheless sizable (Van Horn and Toga, 2014). And a growing awareness that robust findings require sufficiently long recordings (Li et al., 2021) as well as large and representative samples (Button et al., 2013; Turner et al., 2018) leads to large-scale datasets such as the Human Connectome Project (HCP) (Van Essen et al., 2013), the Adolescent Brain Cognitive Development Study (ABCD) Study (Casey et al., 2018), or the UK Biobank (UKB) project (Matthews and Sudlow, 2015) that pose infrastructural challenges for storage, analysis, transfer and archival. Moreover, in human neuroimaging, data underlie strict data protection regulations such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States, or the General Data Protection Regulation (GDPR) in Europe. This requires compliance to variable terms of data usage agreements, and often involves idiosyncratic processes to retrieve data (Waite et al., 2022).
Processing of neuroimaging data is complex, too. It usually involves multi-stepped workflows from acquisition through analysis to archival, where each step frequently requires several different software tools (Niso, Botvinik-Nezer, et al., 2022; Poline et al., 2012). The number of "researchers' degrees of freedom" – the amount of possible combinations of methods, parameters, and tools – in neuroimaging analyses is thus large (Bowring et al., 2019). But variations in analytic choices affect numeric results and conclusions (Silberzahn et al., 2018). When Botvinik-Nezer et al. (2020) instructed 70 independent research teams to analyze the same task-based fMRI dataset with tools and methods of their choice to test the same hypotheses, conclusions were highly variable. A similar project, EEGManyPipelines (eegmanypipelines.org), is currently ongoing in the field of EEG, and expects to find even more variability. One central aspect RDM thus needs to provide in neuroimaging is precise information about how tools, data, and actors were involved in the generation of a file. This is necessary because, despite reporting guidelines for magnetic resonance imaging (MRI) (Nichols et al., 2017), MEG (Pernet et al., 2020), and EEG (Styles et al., 2021) studies, traditional publications still regularly fail to report all relevant details about recording, processing, and analysis (see, e.g., Šoškić et al., 2022). Such *digital provenance* (illustrated in Figure 1.4) is not meant to decrease the analytical variability. Instead, it captures thorough, machine-actionable processing descriptions that are necessary to investigate differences between analysis outcomes, reproduce, or reuse them as FAIR principle R2 advocates.

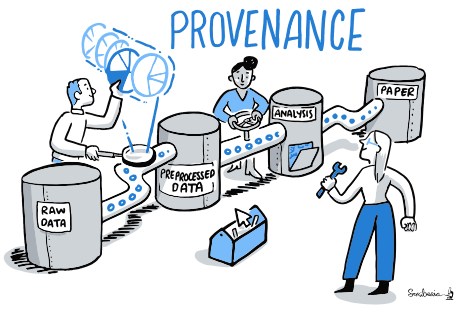


Figure 1.4: A variety of choices affect a research outcome. Digital provenance, information about how tools, data, and actors were involved in its generation, is required to retrace, understand, and trust the decisions that led to it. Good research data management yields good provenance. License: Scriberia and the Turing Way Project, CC-BY.

Overall, careful RDM is necessary to translate complex data with difficult storage requirements and sophisticated processing workflows efficiently into scientific insights in the field of neuroimaging. Given these specific complexities, solutions to these challenges often arise from the community of researchers. The following section will highlight one of several software tools that aids with the complex task of research data management: DataLad.

1.3 DataLad as a software solution for research data management challenges

The following section provides an overview of the software tool DataLad and its use for research data management. The reader is invited to refer to our original publication (Halchenko et al., 2021) for a more detailed description.

DataLad (datalad.org) is a Python-based, MIT-licensed software tool for the joint management of code, data, and their relationship. It builds up on git-annex, a versatile system for data logistics (Hess, 2010), and Git, the industry standard for distributed version control. To address the technical challenges of data management, data sharing, and digital provenance collection, it adapts principles of open-source software development and distribution to scientific workflows.
DataLad's aim is "to make data management as easy as managing code" (Halchenko et al., 2021). The latter benefits from a well-established ecosystem of tools, platforms, and processes. The distributed version control tool Git provides the ability to track changes in small-sized, text-based files, and is a de-facto standard in collaborative software development. In 2022, 93.78% of respondents in Stackoverflow's annual developer survey (survey.stackoverflow.co) reported to use it. Hosting platforms for Git repositories such as GitHub (github.com), GitLab (gitlab.com), or Gin (gin.g-node.org) have been developed to serve as facilitators for collaboration, discovery, and reuse. In the scientific community it is widely recommended to employ version control for code to foster reproducibility (e.g., Sandve et al., 2013), and it is established practice to develop and share research software and code using Git and Git hosting platforms (e.g., Bryan, 2018; Corti et al., 2019; Nord and Verbeeck, 2019; Strupler and Wilkinson, 2017). But other components of scientific projects, such as data or computing environments, are not as transparently managed or accessible as code and software. Digital research objects are usually stored in multiple different locations (Parsons et al., 2013), and their consumption is complicated by disconnected and non-interoperable hosting solutions: Different storage services use different protocols, means of authentication, or other idiosyncrasies that require custom workflows. Moreover, as will be elaborated in Chapter 3, scientific computation is not reproducible enough, because data provenance is often incomplete and rarely automatically captured. And over the course of a research project, often as part of the standard, multi-stepped processing workflow, data evolve and change just like code does: Continued acquisitions enlarge the raw dataset; transformations to – likewise evolving – community standards change file formats, dataset organization, or enrich available metadata; and continuous quality control processes can fix errors in datasets. In the case that data or other research objects were subject to change over the course of a project, there is a need to identify which exact version has been used – otherwise, the reproducibility of a result is threatened (Hardwicke et al., 2018). Yet unlike code in software development, data tend not to be as precisely identifiable because data versioning is rarely or only coarsely practiced. Last but not least, in the absence of standardized data packages, there is no uniform way to declare actionable data dependencies and derivative

relationships between inputs and outputs of a computation. Consequentially, data needs to be kept alongside to results to ensure reproducibility, even if it is hosted elsewhere already. Especially in the age of big data neuroscience (Bzdok and Yeo, 2017), downloads or storage of datasets can become infeasible (Grisham et al., 2016; Horien et al., 2021). And although projects might only draw insights from only a subset of a dataset, such as only specific modalities, tasks, or participants, a project has heavier disk space demands if the original dataset is only available as a bulk download. DataLad aims to solve these issues by providing streamlined, transparent management of code, data, computing environments, and their relationship. Its main features are 1) Version control for data of any size or type; 2) Streamlined procedures to consume, publish, and update all elements of scientific projects, with interoperability adapters to established scientific and commercial tools and services; 3) Data linkage as precisely versioned, lightweight dependencies; and 4) actionable process provenance capture for arbitrary command execution that affords automatic re-execution.

1.3.1 Technical features

DataLad Datasets for version control and modular dependency management Fundamental to DataLad's functionality is the concept of the "DataLad dataset", DataLad's central data structure. On a technical level, it is a joint Git and git-annex repository with additional metadata and features for scientific use cases added on top by DataLad.
Git excels at managing and collaborating on text files, and provides a powerful backbone to DataLad's features. Git repositories can be easily distributed as linked *clones* to suitable infrastructure to share files and their revision history. Locally, changes to files can be saved (*committed*) with detailed provenance and transferred (*pushed*) to all clones of a repository, and remote revisions can be retrieved and integrated (*pulled*).
However, Git is not designed to handle large or binary files as common in science. Git-annex overcomes this limitation and adds support to track and transfer files of arbitrary size or type without placing their content into Git. Instead, it places file content into a managed repository *annex* and only commits a lightweight reference that encodes the file's name and content identity via content hash into Git. This reference allows a decoupling of file content (handled by git-annex) and filename (tracked with Git), but retains the ability to transparently notice and track changes: If file content changes, the identity reference known to Git changes, too, and the version control system becomes aware of the modification. The same mechanism guarantees file integrity at all times and solves data privacy concerns without impacting discoverability, as potentially sensitive content can be kept private but its metadata can be distributed easily. While Git keeps a reference of the content *identity*, git-annex tracks content *availability*, and manages data transport with an extensible set of protocols and set of hosting solutions to and from a local repository annex at a granularity of individual files.
DataLad, finally, extends Git and git-annex with easy to use modularization, re-executable annotation of changes, and targeted interfaces and interoperability adapters: Modularization facilitates reusability in research workflows with large amount of files or with a heterogeneous nature, comprising different data sources or processing stages. Organizing them into a modular structure of homogeneous or smaller components enables more efficient reuse. DataLad allows to nest individual datasets via versioned linkage as lightweight dependencies, and provides seamless recursive operations across dataset boundaries by extending Git's submodule mechanism. With this, DataLad provides a "monorepo"-like user experience in datasets with arbitrarily deep nesting, and

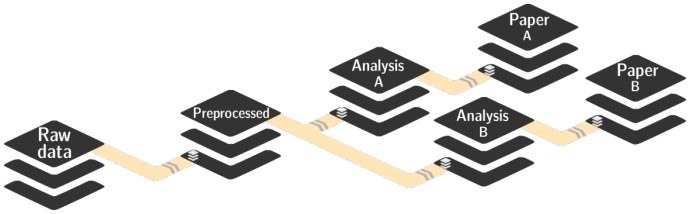


Figure 1.5: Exemplary dataset hierarchy in a research project. DataLad datasets can be linked as actionable superdataset-subdataset relationships. A DataLad dataset with raw data is linked as a dependency to preprocessed data, to an analysis, and the final paper. As DataLad datasets decouple sizable file content from file availability metadata, such links are only a fraction of the size of the data they track. This feature enables modular projects from stand-alone units that facilitate reuse.

makes it trivial to operate on individual files deep in the hierarchy or entire trees of datasets, while individual datasets can easily be reused in new contexts (see Figure 1.5).

Re-executable process provenance Re-executable annotations of changes constitute process provenance. In DataLad datasets, DataLad can wrap any command execution and automatically capture the command, the generated outputs or modification, and optionally all required input elements with a so-called `datalad run` or a `datalad containers-run` command. The former command captures the execution of any shell command, and the latter command can use software containers, tracked in DataLad datasets, for executing commands inside a specific software environment or containerized pipeline. Both commands retrieve inputs, initiate command execution, and, if it succeeds, save results together with a re-executable provenance in a structured annotation to a Git commit message (see Figure 3.4). In addition to providing reliable information about past command-line invocations, these machine-readable records make it possible to easily re-execute commands, for example to verify if a result is computationally reproducible or to apply an analog change to a different dataset state.

Streamlined data transport Interoperability adapters, finally, streamline data publication and consumption routines. Being able to efficiently retrieve and update research objects across a variety of available storage options is an important part of RDM (Borghi and Van Gulick, 2018). Git can already interact with other local or remote repositories via standard or custom network transport protocols, and git-annex readily provides access to a wide range of external data storage resources via a large set of protocols. On top of this, DataLad implements support for additional services that require custom protocols, such as the Open Science Framework (OSF) (Hanke, Poldrack, et al., 2021), and can add more fine-grained access (e.g. direct access to individual components contained in an archive hosted on cloud storage). With this approach, DataLad can help to overcome technological limitations of storage solutions, or integrate seamlessly with technology that is already in use.

1.3.2 Development principles

DataLad's development is guided by four principles (Halchenko et al., 2021):

- The only two recognized entities are Datasets and the files they comprise,
- A dataset is a Git repository with an optional annex,
- There should be as few custom procedures and data structures as possible,
- Complete decentralization, with no required central server or service, but maximum interoperability with existing third-party resources and infrastructure.

These principles ensure DataLad's open and domain agnostic nature, maximize the long-term utility of DataLad datasets, minimize users' technical debt and reduce the risk of adoption. When using DataLad datasets for digital objects, access to any resource managed with DataLad has no critical dependency on service deployments governed by central entities, and even on DataLad itself. A further characteristic is that DataLad is developed not as a singular tool, but as an extensible ecosystem of software packages. A core package provides basic functionality, and DataLad extensions, stand-alone Python packages with additional DataLad functionality, can enhance it with domain-focused or technology-specific features. This aims to keep the source code maintainable, and allows users to tune the command-suite to their specific needs.

1.3.3 User experience

On the conceptual level, a DataLad dataset is an overlay structure to version control files of any size, track and publish files in a distributed fashion, and record, publish, and execute actionable provenance of files and file transformations. On a file system, it appears like a regular directory. Users can either create DataLad datasets from scratch (`datalad create`) or install them from a variety of sources (`datalad clone`). When consuming DataLad datasets from external sources, annexed file content can be retrieved (`datalad get`) and removed (`datalad drop`) on demand if the given user has potentially necessary access rights. Generally, DataLad's commands aim to simplify standard workflows from Git, git-annex, or hosting services to make them more accessible to technical novices. For example, committing a file into Git requires a `git add` followed by a `git commit`, and annexing a file requires a `git annex add` followed by a `git commit`. A `datalad save`, on the other hand, performs either action, with automatic decision making whether the file is annexed or committed to Git. Likewise, publishing a repository to hosting sites usually requires interactions in the hosting site's web interface, but DataLad provides a set of `create-sibling-<service>` commands that spare users the need to visit the hosting service. And publication dependencies or automatic configurations can publish Git revision history and annexed file contents conjunctly, using `datalad push`. Provenance capture can be achieved by wrapping command executions in either a `datalad run` or `datalad containers-run`[1] command, and they can be re-executed using `datalad rerun` (see Figure 3.4). When working in hierarchies of datasets, any command's `--recursive` parameter applies it to downstream datasets. Despite the fact that DataLad provides its own set of commands, compatibility with all Git and git-annex functionality is retained, and users are free to choose their

[1] The `datalad containers-run` command is provided by the `datalad-container` extension package, available with standard Python package managers.

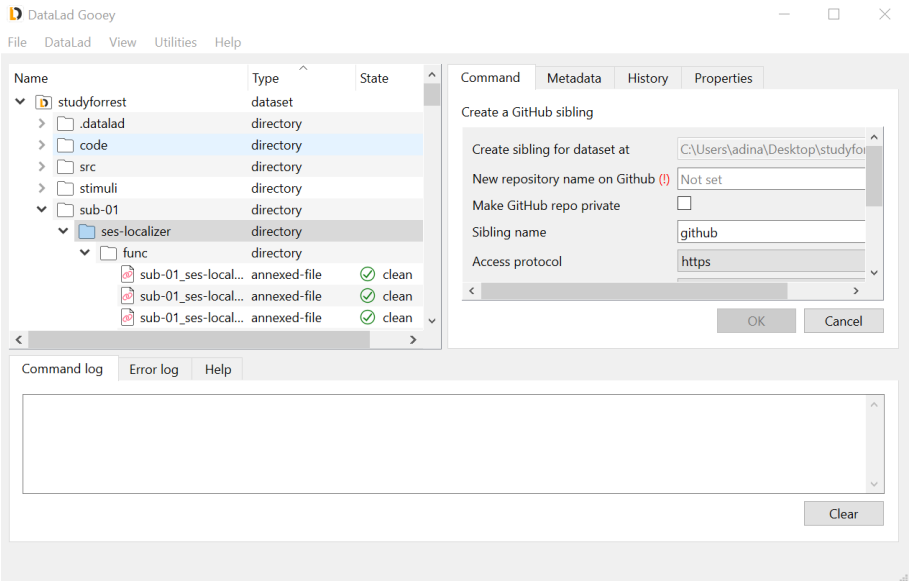


Figure 1.6: The GUI provides an overview of working with DataLad. The file tree (left) is a DataLad dataset, and its contents are annotated depending on whether they are tracked with Git, git-annex, or untracked, and whether they are modified, or if all changes are saved ("clean"). The command window (right) shows a (partial) DataLad command parametrization.

workflows. Like Git and git-annex, DataLad is primarily developed as a command line tool, but a graphical user interface (GUI) exists as well (see Figure 1.6).
With DataLad's streamlined data transport routines, users can utilize web sources, including major cloud storage providers, paths on local or remote computational infrastructure, or neuroscientific data registries such as OpenNeuro (Markiewicz et al., 2021) for file storage or collaboration. At the same time, the separation of file content and file name makes a lean, meta-data based representation independent from actual data retrieval. Thus, datasets of arbitrary scale can be cloned, exposing file names and revision history of all contained files, and provide actionable access to its content at a fraction of their total size. And in order to expose datasets for easier access, users can publish DataLad datasets to common repository hosting infrastructure as access points to install data, without actually publishing data contents to these services. These features aid with RDM tasks in scientific projects, were used extensively in upcoming data analyses. A thorough overview of features can be found in the software's documentation, which is elaborated on in Chapter 2, a comprehensive use case for reprodcubility is detailed in Chapter 3, and Chapter 4 describes its use in brain-state analyses.

1.3.4 Software adoption and relevance

As DataLad does not employ tracking code, the exact number of its users is not known. However, download statistics from the major Python package managers `pip` and `conda`, and popularity statistics from users of the Debian operating system can provide rough references. According to pypistats.org, the main DataLad Python package was downloaded from the Python Package Index 474 times per day on average in August 2023. The Python distribution Anaconda (anaconda.org) counts a total

of 380.011 downloads of the software throughout versions 0.9.3 (April 2018) and 0.19.3 (August 2023), averaging 240 downloads per day. The `popularity-contest` software is a Debian package that, if installed on a users system, reports anonymous statistics about most-used Debian packages. This data is aggregated into a popularity contest. According to it, in August 2023 between 0.08 and 0.12 percent of systems reporting statistics have a DataLad installation (see Figure 2.2b), which is considerable when projected to the total amount of Debian-based computers world-wide. All of these sources contain biases that limit conclusion to the number of users, though. Installations via Python package managers are commonly performed by individual end users, whereas installations of Debian packages can correspond to system-wide installations on multi-user systems such as high performance computing infrastructure. Likewise, upgrades of existing installations to more recent versions are included in these data, as well as temporary installations of the software in automated continuous integration test suites. Nevertheless, download statistics attest that it is an actively used tool with a user base that exceeds the circle of its developers by far. The citation count of our academic paper Halchenko et al. (2021), totaling 66 in August 2023, and active contributor community around the source repository on GitHub, currently amounting to 48 individuals with committed code contributions, confirms that. DataLad's current popularity, however, has not been established from the beginning of its development. The next chapter will highlight how scientific software like DataLad can be improved and gain relevance, and, vice versa, improve scientific practice.

2 Improving scientific practice through software usability

Better Software, Better Research

(Carole Goble)

Scientific software is an integral part of the research process, and improving the quality of research software benefits the research itself (Goble, 2014). This next chapter details how a documentation project for DataLad increased software quality, software popularity and enabled users to tackle complex RDM use cases. It refers to our original publication (Wagner, Waite, Meyer, et al., 2020).

2.1 The central role of research software in science

Scientific software, often also referred to as research software, can broadly be defined as software used for scientific purposes. As software requirements for scientific use cases can be specialized, the developers of research software often overlap with the community of scientists that uses the software (Hannay et al., 2009), and over the past decade, the term "Research Software Engineer" has been established to describe researchers that dedicate parts of their scientific work to creating and maintaining software (Hettrick, 2016). Scientific software is an integral part of research in science, engineering, and humanities. The majority of scientific work could not be done without software, thus, fittingly, the United Kingdom's Engineering and Physical Sciences Research Council describes it as a "critical infrastructure that underpins cutting edge science and engineering research"[1]. More recently, research software has been gaining academic credit it has long lacked. It is recognized as academic output in the San Francisco Declaration on Research Assessment (DORA; sfdora.org), and the Agreement on Reforming Research Assessment (CoARA; coara.eu), both of which have been signed by thousands of academic institutions worldwide. The German Research Foundation (DFG), the largest funding institution for the sciences and humanities and research in Germany, counts software as a "scientific result" in their evaluation of academic CVs. Academic journals, such as the Journal of Open Source Software (joss.theoj.org), and article types, such as Nature's "Toolbox" (www.nature.com/nature/articles?type=toolbox), have been created to support the scholarly publication and reuse of research software.
With the acknowledgment of the importance of scientific software in research, there also is an awareness that its quality, reusability, and longevity needs to be ensured and improved. As a digital research object, the FAIR principles apply to it just as to other research outputs. The DFG states that access to data and software according to the FAIR principles are of comparable importance to science as access to publications (German Research Foundation, 2020). And calls for proposals to improve the quality, usability or longevity of research software have been put forward

[1] https://www.ukri.org/what-we-offer/browse-our-areas-of-investment-and-support/research-infrastructure-theme

by major national funders, including Germany (e.g., German Research Foundation, 2022), the United Kingdom (e.g., UK Research and Innovation, 2021), and the United States (e.g., National Institutes of Health, 2002).
One set of problems from which software quality can suffer are documentation deficiencies (Mehlenbacher, 2003). The mere existence of software is insufficient to ensure its uptake and use according to best practices. To improve scientific practice, research software needs to enable and empower their users through usability and documentation. This makes documentation a major factor in the success of scientific software, and an integral part of the software development process. The next section will explore this problem and its consequences.

2.2 Documentation deficits of scientific software and their consequences

Documentation is information about a software. It fulfills different roles depending on its target audience, and the literature distinguishes several different types of documentation. Parnas (2011) categorizes documentation either as a *tutorial* or as a *reference work*. Both kinds are needed for different audiences: Whereas experienced users and contributors need reference documents to guide further development, such as the elements of an Application Programming Interface (API), new users and contributors need a basic understanding of the software tool and its intended use cases. Commonly distinguished are also *technical* and *user* documentation: The former contains information for developers and users by describing features, maintenance information, or design choices. The latter targets end users of a software product with accessible explanations how to install a tool, use its features, or step-by-step instructions. User documentation also matches the concept of *task-based* documentation, which is broken down into the activities that users will go through as they work, starting with basic tasks and continuing with more advanced tasks that become possible as users continue to work (Barker, 2003).
Research software often lacks comprehensive documentation (Pawlik, Segal, and Petre, 2012; Segal, 2007). Commonly reported reasons for this are a lack of funding, incentives, and interest by software developers (Pawlik, Segal, and Petre, 2012). Yet although it is commonly regarded as separate from the actual piece of software, software documentation is heavily tied to the quality of a software tool and the software development process. A lack of documentation hinders knowledge transfer both among users and developers, impedes maintenance, and creates a steep learning curve for new users and new developers alike (Theunissen et al., 2022). Parnas (2011) describes a vicious circle that sets in when the quality of software documentation is poor: “Reduced [documentation] quality leads to reduced [software] usage, [r]educed [software] usage leads to reductions in both resources and motivation, [r]educed resources and motivation degrade [software] quality further”.

2.3 Documentation in the DataLad project

Since the first release (0.0.1, March 2015), DataLad had technical documentation with a design overview and a reference documentation. Although any amount of documentation is better than no documentation at all, existing documentation can still be insufficient if it does not meet the needs of the target audience. Solely technical or reference documentation, for example, can

be suboptimal for novices: It may be incomplete, narrowly focused on individual commands, or assume existing knowledge readers lack (Segal, 2007; Pawlik et al., 2015), and can thereby discourage potential users or inhibit the adoption of a tool. Even though technical documentation is useful for developers, a central target audience for documentation of the DataLad ecosystem are scientists. While experts in their respective domains and methodologies, scientists may not have domain-agnostic technical skills. Task sets such as those required in RDM require a broad set of technical skills and research curricula seldom teach computing ecosystem literacy (Grisham et al., 2016). In fact, even computer science curricula often miss critical topics about the computing ecosystem. At the Massachusetts Institute of Technology (MIT), this lack famously resulted in the internationally popular, self-organized class "The missing semester of your CS education"[2]. In addition, the high usability of modern computers' and applications' front ends spares users the need to develop the same level of familiarity with their computers than previous generations of computer users had (Mehlenbacher, 2003). A considerable part of this target audience can thus be considered technical novices for which technical documentation is not ideal.
Research suggests, too, that scientists' needs for documentation go beyond reference manuals. In an analysis of user questions in support forums of scientific software packages, Swarts (2019) found that the focus in 80% of inquiries was on operations and tasks, such as a required sequence of operations to achieve a specific goal, instead of reference lists. In breaking down user questions by purpose, Swarts (2019) further found that users were most interested in a description of operations or tasks, followed by insights about the reasons behind the action. And in separating documentation types into "feature-based" (closer related to the concept of reference documentation) or "task-based", Swarts (2019) reports twice as many questions seeking explanations in software with feature-based compared to task-based documentation. Overall, this highlights that users of scientific software show a clear need beyond the documentation of individual commands, but seek to understand general usage principles and master complex combinations of features to achieve specified goals. This type of empowerment is what the DataLad handbook project aimed to achieve, and complement DataLad's existing technical documentation with in order to make the tool easier to use.

2.4 The DataLad Handbook: A user-focused and workflow-based addition to standard software documentation

The initial idea to the DataLad Handbook (handbook.datalad.org) arose at the 2019 Meeting of the Organization for Human Brain Mapping (OHBM), during the Symposium "From Open Science to Science: Shifting the status quo in data sharing, software, and publishing". Gregory Kiar's presentation "Technology and Platforms Enabling Reproducible and Open Publishing" included a recommendation and warning about DataLad: While it would be a powerful tool capable of solving many challenges, users will face difficulties learning which features existed and how they could use them (Kiar, 2019, June 11). This description had matched my own experiences, but the adoption of DataLad for the research project on brain states seemed useful. Additionally motivated by the expressed need of the scientific community for additional guidance, I initiated the DataLad Handbook project.

[2] missing.csail.mit.edu

2.4.1 Design considerations

I identified three types of stakeholders with different needs: *Researchers* need accessible educational content to understand and use the tool, *planners*, such as PIs or funders, need high-level, non-technical information in order to make informed yet efficient decisions on whether the tool fulfills their needs, and *trainers* need reliable, open teaching material. Drawing from personal user experiences with the software, I set out the Handbook project with the following goals about its contents:

- **Applicability for a broad audience**: The Handbook should showcase domain-agnostic real-world RDM applications.
- **Practical experience**: The Handbook should enable a code-along style usage, with examples presented in code that users can copy, paste, and run on their own computer. To allow a read-only style usage, too, the Handbook should also reveal what a given code execution's output would look like. For an optimal code-along or read-only experience, the code output should match the current software behavior.
- **Suitable for technical novices**: The Handbook's language should be accessible. Gradually, by explaining technical jargon and relevant tools or concepts in passing, it should provide readers with a broad set of relevant RDM skills rather than requiring prior knowledge.
- **Low barrier of entry:** The Handbook's contents should be organized in short, topical units to provide the possibility to re-read or mix and match.
- **Integrative workflows**: The Handbook's contents should build up upon each other and link back to past course content to teach how different software features interact.
- **Empowering independent users**: Instead of showcasing successful code only, it should demonstrate common errors explicitly to enable users to troubleshoot problems in their own use cases independently.

Arising from this specification analysis, the I designed the following structure (Wagner, Waite, Waite, et al., 2020):

- "Introduction": The first part of the book, covering high-level descriptions of the software and its features and detailed installation instructions for all operating systems.
- "Basics": The second part of the book, written in the form of a continuous, code-along tutorial, set in a domain-agnostic fictional storyline about an RDM application, and covering all stable software features in chapters that build up on one another.
- "Beyond Basics", covering advanced features in stand-alone chapters, added prior to the second release.
- "Usecases": The last part of the book, containing short, standalone start-to-end descriptions of real-world usecases, with concise step-by-step instructions, and references to further reading in the Basics.

To further support the design and content requirements, I set the following technical goals:

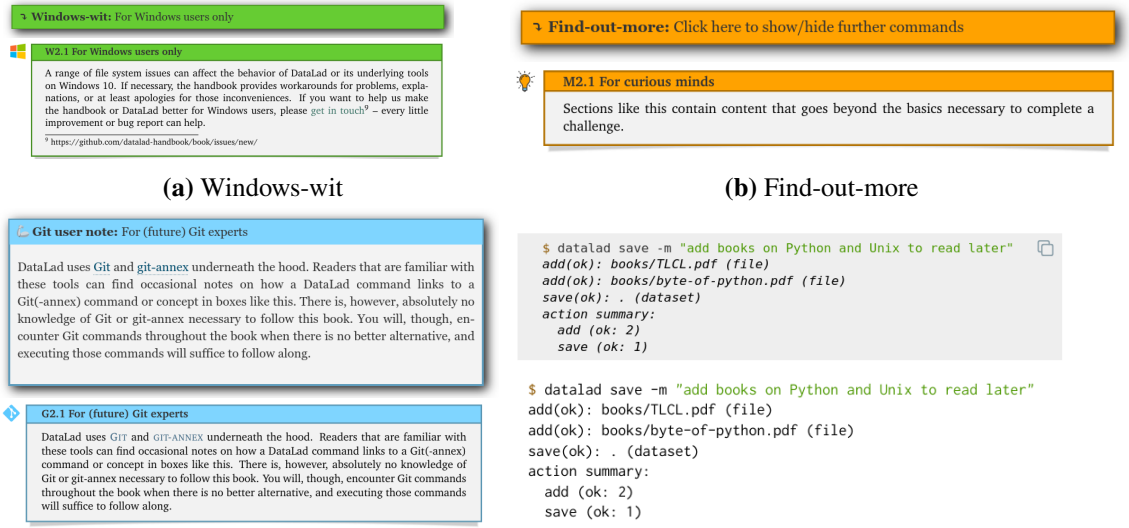
- To keep visible text short while preserving the ability to explore advanced contents, topical custom content boxes and toggle-able sections ("click to expand") should make information relevant to specific user types or optional details skippable and easily discoverable.
- To ensure functioning from a user's point of view, the workflows included in the Handbook as code need to be tested as an automated integration test suite.
- The Handbook should be available in multiple formats, at least as a web-based rendering and as a portable download to be stored offline or printed.
- The Handbook should be developed alongside the software, and semantic versioning should ensure that users of a past software version can find the corresponding version of the Handbook.

The resulting implementation of the Handbook fulfilled these requirements as follows:

2.4.2 The technical backbone

The technical backbone of the Handbook was chosen with the intent to support declared goals, and to maximize configurability, autonomy, and reusability of the project. It builds up entirely from flexible and extendable open source infrastructure: On the highest level, it uses Sphinx as a documentation generator (www.sphinx-doc.org). Sphinx transforms documents written in reStructuredText, a lightweight markup language, to a variety of output formats, among them HTML, PDF, LaTeX, or EPUB. Initially a by-product of the Python documentation, it has been adopted by the Open Source community at large; GitHub's dependency graph reports that it is used by more than 300.000 projects in August 2023[3].
Sphinx supports an extension mechanism with which additional functionality can be integrated. Leveraging this mechanism, the Handbook project extended standard Sphinx features with custom admonitions and designs, for example toggle-able boxes for optional details. This is implemented as a Python package alongside the Handbook source code, making the Handbook project a reusable and installable Sphinx extension. Figure 2.1 provides an overview of the custom-developed design features. A major functional enhancement is provided with a separate Python package, `autorunrecord`, an additional custom-made Sphinx extension that allows sequential execution of code in a specified environment, and embedding a record of the code and its output as code snippets into the documentation[4]. Instructors can further use it to automatically create scripts from selected code blocks which can then be demonstrated in a remote-controlled terminal in live-coding tutorials. Hosting for the project is provided by Read the Docs (readthedocs.org), a free software documentation hosting platform that integrates with Sphinx. Illustrations in the Handbook are based on the undraw project by Katerina Limpitsouni (undraw.co).
The ability of the documentation to sequentially execute code and record its outcomes has been used to utilize the Handbook as an integration test for the DataLad software in addition to a user guide. If new software developments in the DataLad core packages break documented workflows, a continuous integration test suite will fail, alerting developers to the fact that their changes break user workflows.

[3] github.com/sphinx-doc/sphinx/network/dependents
[4] github.com/mih/autorunrecord



(a) Windows-wit **(b)** Find-out-more

(c) Git user note **(d)** Code block

Figure 2.1: Custom admonitions and code blocks used in the DataLad Handbook. In each panel, the top image corresponds to the web version of the admonition, and the bottom image corresponds to its PDF rendering. a) Windows-wits, toggle-able in the HTML version, contain information that is only relevant for the Windows operating system. b) Find-out-more admonitions, also toggle-able in the HTML version, contain miscellaneous extra information for curious readers. c) Git user notes are colored boxes with references to the underlying tools of DataLad, intended for advanced Git users as a comparison or technical explanation. d) Code blocks show one or more commands and the resulting output, provided using the `autorunrecord` Sphinx extension. In the web version, a copy-button (top right corner) allows to copy relevant commands automatically to the clipboard. Internal annotations allow generating custom scripts from any sequence on code-blocks for live coding demonstrations.

To ensure reusability, such as the adaptation by Brooks et al. (2021), the project is released under a CC-BY-SA 4.0 license. Under its terms, all elements can be reused in original or derived form for all purposes under the condition that the original project is attributed and that derivative work is shared under an identical ("not more restrictive") license[5].

2.4.3 Content

In August 2023, the web and PDF versions of the Handbook are organized into four parts, with a total of 21 chapters across the parts Introduction, Basics, and Advanced, and 12 use cases. The "Introduction" has two different target audiences: For one, it provides *researchers* with detailed installation instructions, a basic general command line tutorial, and an overview of the handbook. Beyond this, it gives a high-level overview of the software and its capabilities to *planners*.
The "Basics" is organized into nine chapters. Following a narrative about a fictional college course

[5] creativecommons.org/licenses/by-sa/4.0

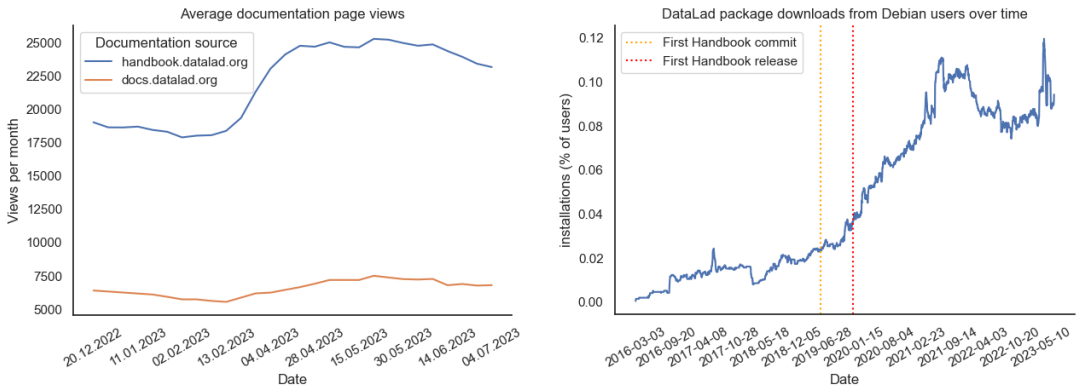
on RDM, it teaches different aspects of DataLad functionality and general research data management to *researchers* in each topical chapter. Broadly, those topics can be summarized as follows: 1) Local version control, 2) Capturing and re-executing process provenance, 3) Data integrity, 4) Collaboration and distributed version control, 5) Configuration, 6) Reproducible data analysis, 7) Computationally reproducible data analysis, 8) Data publication, and 9) Error management. The "Beyond Basics" part adds independent chapters on advanced DataLad features and workflows, big data projects, DataLad use on computational clusters, DataLad's internals, and selected DataLad extensions. The latter two parts are accompanied with code demonstrations, slides, executable notebooks, or video tutorials that *trainers* can reuse freely, intended to teach tool use, and with it also improve scientific practice. The last part, "Usecases", targets *planners* and *researchers* with short step by step instructions. They show planners what is possible, and help researchers connect their knowledge into larger workflows.

2.4.4 Project and community management

Ensuring the longevity of software projects beyond the duration of individual researcher's contracts requires community building (Koehler Leman et al., 2020). A user-driven alternative to documentation by software developers, "Documentation Crowdsourcing", has been successfully employed by the NumPy project (Pawlik, Segal, Sharp, and Petre, 2014). The Handbook project extends this concept beyond reference documentation. To achieve this, it is set up to encourage and welcome improvements by external contributors. The project is openly hosted on GitHub. Mirroring processes in larger crowd-sourced documentation projects such as the "The Turing Way handbook for reproducible, ethical and collaborative research" (The Turing Way Community, 2022), credit is given for both code-based and non-code-based contributions. Contributors are recognized in the source repository, on the DataLad Website, and as co-authors in both the printed version of the Handbook and its Zenodo releases. As of August 2023, a total of 57 contributors provided input in the form of content, bug fixes, or infrastructure improvements.

2.4.5 Impact and scope

My work on the DataLad Handbook began in June 2019, and the first release followed in January 2020. It has been under continuous development for more than four years, averaging two releases per year, and complements the DataLad ecosystem with a comprehensive user guide. Its PDF version spans more than 600 pages. Releases of the DataLad core package are coordinated with matching releases of the Handbook project, and past release versions remain accessible online.
Confirming observations from the literature (van Loggem and van der Veer, 2014), the conjunct development of user-documentation has positive effects on software quality. As the writing process involves manual software testing, I observed a higher discovery rate of software errors. My user-focused approach uncovered deficiencies of the technical documentation and API elements with suboptimal user experience. The workflow-based nature of demonstrations highlights API inconsistencies. And the integration test that the handbook constitutes catches incompatibilities between the software and common usage practice. These documentation features facilitate software development, and had a major impact on the conjoint 0.12.0 release of DataLad (Jan 2020), the first with a matching handbook release. The popularity data in Figure 2.2b confirms a marked increase in downloads from this date onward. In addition, differences in web traffic confirm that user



(a) Monthly documentation views

(b) Debian package popularity counts for DataLad

Figure 2.2: Software and documentation popularity measures. 2.2a: Documentation page views for the technical documentation (docs.datalad.org) and the user documentation (handbook.datalad.org), displayed as average views per 30 days, between December 2022 and August 2023. The DataLad Handbook consistently sees higher traffic. 2.2b: Popularity of the Debian packages `datalad` and `python3-datalad` over time, expressed in percent of Debian users that installed the package from the total amount of users submitting download statistics. Dashed lines indicate the fist commit and the first release of the Handbook, the latter marks a notable increase in software downloads. Source: Debian Popularity Contest, August 2024.

documentation is in higher demand than the technical documentation. An analysis of visits to the web version of the Handbook from December 2022 to July 2023 revealed that handbook.datalad.org averaged 22.000 total page views per 30 days. In the same time span, the technical documentation of DataLad at docs.datalad.org averaged 6600 total page views per 30 days, less than a third of the traffic of the user documentation (Figure 2.2a). In summary, the development of the DataLad Handbook had a measurable positive impact on the number of users, the popularity of the package, and the software quality. The project documents workflows that improve scientific practice, such as local and distributed version control, collaboration, data publication, and reproducible analysis. The next chapter will focus particularly on why these abilities are useful in science, and how they enable trustworthy science at the largest scale.

3 Ensuring computational reproducibility across computational environments

> You can download our code from the URL supplied. Good luck downloading the only postdoc who can get it to run, though.
>
> *(Ian Holmes, in an #overlyhonestmethods tweet)*

Partially fueled by external incentives or requirements (German Research Foundation, 2020; McKiernan et al., 2016), research curricula founded within the Open Science Movement (Munafò et al., 2017; Poldrack, Baker, et al., 2017), and a growing ecosystem of openly available infrastructure and tools (Niso, Botvinik-Nezer, et al., 2022), practices of publishing reproducibly are becoming more frequent. Widespread sharing of code and data allows researchers to verify, reuse, and improve upon past work (Borghi and Van Gulick, 2018). Grass-roots movements such as Reprohack (www.reprohack.org) or the "Ten Years Reproducibility Challenge" (rescience.github.io/ten-years) train researchers to check published studies for reproducibility. Consequently, attempts to reproduce previous studies often happen in different computational environments than those that originally created the results in question. Ensuring computational reproducibility across computational environments is, however, a difficult technical challenge. This following chapter outlines first its challenges, particularly in the field of neuroimaging, then its opportunities, and lastly an implementation to ensure computational reproducibility across computational environments.

3.1 The origins of reproducibility

Over the past decade, interest in reproducibility has been fueled by salient failures to reconfirm published results – often termed *reproducibility crises* – in numerous fields (Baker, 2016), from psychology (Open Science Collaboration, 2015), to biomedical imaging (Wagner, Maumet, et al., 2023), to artificial intelligence (Hutson, 2018), or econonmics (Camerer et al., 2016). However, proposals to increase reproducibility, transparency, and robustness of science were made independently in various disciplines long before the current trend, in some cases dating back several centuries (such as Boyle (1666), as cited in Hunter and Anstey, 2008). Even the field of *computational reproducibility* originated already more than 30 years ago in the field of seismology (Buckheit and Donoho, 1995; Claerbout and Karrenbach, 1992), despite increased usage of the term in scientific literature only from 2015 onward (see Figure 3.1). Consequently, the terminology around reproducibility has varied considerably over the years and across domains, and there is no universally agreed upon standardization of terminology in place yet (Barba, 2018). To disambiguate between several conflicting definitions of terms around reproducibility that are in active use, I will use the following definitions of these terms in this thesis:

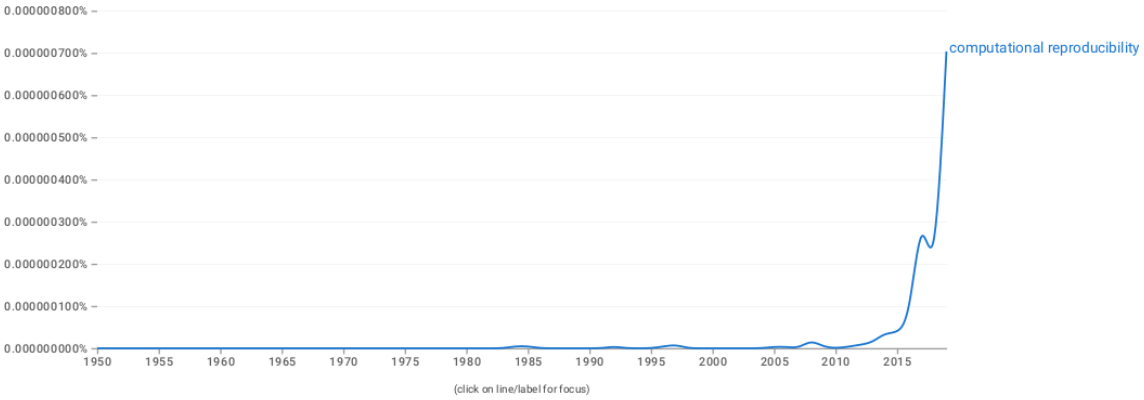


Figure 3.1: Popularity of computational reproducibility: A chart of the frequencies of the n-gram "computational reproducibility"(using yearly count, normalized be the numbers of publications in each year) in literature included in the English(2019) corpus of Google books. This graph has been created using the Google Ngram Viewer (books.google.com/ngrams, Michel et al., 2011).

Reproducibility

Following the definition of Peng et al. (2006), *reproducibility* refers to the practice of verifying a published result with the same methods and materials used by the original authors.

Replicability

Replicability, on the other hand, refers to strengthening scientific evidence when several independent researchers find similar results using "independent data, analytical methods, laboratories, and instruments" (Peng et al., 2006).

Computational Reproducibility

Computational reproducibility, finally, matches the definition put forward in the 2019 report on "Reproducibility and Replicability in Science" by the National Academies of Sciences and Medicine (2019): "We define reproducibility to mean computational reproducibility – obtaining consistent computational results using the same input data, computational steps, methods, code, and conditions of analysis".

3.1.1 "Everything matters" for computational reproducibility in neuroimaging

As this definition of computational reproducibility implies, the building blocks of research output extend to more than the files that constitute the actual research output, but also to all elements involved in its generation (Claerbout and Karrenbach, 1992). Consider different types of research output: Raw data originates from acquisitions based on – potentially ongoing – experiments, raw data transformations, or data cleaning. Processed data or results stem from computations with

analysis code or software in specific versions on particular data. And software, expressed in raw (code) or derived (transformed into executable) form, is created or used in specific computational environments, with compilers, underlying libraries, and systems in distinct versions. Consequently, these building blocks play an integral part in the genesis of research outputs, and changes in these building blocks or their composition can translate to changes in the resulting research output.
Because of their complexity, neuroimaging studies face obstacles for reproducibility. For one, the details of how code, software, or data have been used to generate a research output, such as analysis parameterization, the subset of data used as input, or sequence and invocation of employed software tools, are volatile. Shared code and data does not always suffice to reproduce a result: A study of data availability, reusability, and reproducibility demonstrated that well-described, "in principle reusable" data often does not suffice to reproduce the scientific findings of the corresponding publications due to missing process provenance metadata (Hardwicke et al., 2018). Secondly, even if methods and their sequence are well-described, precise information about the employed software tools is crucial, too. The fact that different neuroimaging analysis software can produce distinct results from the same data despite using similar conceptual methodology is well known (Bowring et al., 2019). This has been attributed to implementation differences (Palumbo et al., 2019), software errors (Eklund et al., 2016), or analytic configurations (Pauli et al., 2016). For example, in task-based fMRI, Li et al. (2021) found that the choice of output space or resolution can have a marked impact on variability between conceptually similar processing pipelines. Moreover, even with identical pipelines and data, surprising result variability can occur with minor variations in parametrization. Mueller et al. (2017) reported that the choice of resampling resolution impacts alpha inflation, and Li et al. (2021) identified the decision whether or not to include global signal regression as a major source of intra-pipeline variation. Finally, even the same analysis, with identical parametrization, software tool, and data, can result in different outcomes if it is repeated across different operating systems, or with differences in versions of a singular software tool or operating system (Glatard et al., 2015; Gronenschild et al., 2012). Computational reproducibility across computing environments thus often remains elusive unless accounted for from the very start. Therefore, in addition to "Everything matters", Kennedy et al. (2019) cued the phrase "Reproducible by Design (as opposed to reproducibility as an afterthought)" for conducting research in a way that makes computational reproducibility possible. The next section highlights a number of strategies for this.

3.2 Towards re-usable research objects

The reusability of research objects has become a distinct characteristic of scientific practice as it allows for reproduction, verification, building up upon and extending existing work, evidence synthesis, and minimizing duplicate efforts in the advancement of science (Thanos, 2017). With this, it maximizes the impact of the funding and work that resulted in the research output. Its central role in the FAIR principles (Wilkinson et al., 2016) and a variety of funding sources such as the Economic and Social Research Council (ESRC, UK), the European Research Council (ERC, EU), or the National Institutes of Health (NIH, US) are a testament to this. In the scope of the FAIR principles, reusability focuses on the ability of a human or a machine to decide if data are useful and usable in a particular context. This reusability requires trust (Bechhofer, Buchan, et al., 2013): Re-users must be able to audit the steps performed in an experiment or analysis in order to be convinced of the validity of the results or derivatives. FAIR principle R1.2, "(Meta)data are

associated with detailed provenance" (Wilkinson et al., 2016), refers to this. This principle also encodes the process provenance necessary for reproducibility. And indeed, reproducibility and trust are closely related: Where resources are not fully FAIR yet, manual reproducibility typically yields the trust that process provenance would otherwise provide. A new project, for example, commonly starts with a check if the previous foundational findings still hold.
As the FAIR principles advocate for richly curated metadata, many scientific fields or projects argue in favor of coordinated use of ontologies for metadata and brought forward efforts for ontology development and consensus building (e.g., Abrams et al., 2022; Papadiamantis et al., 2020; Wise et al., 2019). But not in all domains are the necessary metadata standards incentivized or ready to use. A few years before the publication of the FAIR principles, Bechhofer, De Roure, et al. (2010) cued the term "reusable research object" in a conceptual position paper. They describe it as what can now be seen as a precursor of a FAIR research object, specifically as a "container for a principled aggregation of resources, produced and consumed by common services and shareable *within* and *across* organisational boundaries [...that] includes not only the *data* used, and *methods* employed to produce and analyse that data, but also the *people* involved in the investigation. An association with a dataset (or service, or result collection, or instrument) is now more than just a citation or reference to that dataset (or service or result collection). The association is rather a *link* to that dataset (or service or result collection) that can be explicitly followed or dereferenced providing access to the actual resource and thus enactment of the service, query or retrieval of data, and so on." (Bechhofer, De Roure, et al., 2010). This description matches characteristics that DataLad datasets or its contents can possess. In the following, I will highlight four properties of research objects that can arise from pragmatic research data management – versioned, actionable, modular, and portable – and how these properties make them reusable even if full FAIRness can not yet be achieved (Wagner, Poline, and Hanke, 2021).

Exhaustive versioning

Bechhofer, De Roure, et al. (2010) stress that a reusable research object is an "all-encompassing" resource, including every relevant digital artifact of the project. DataLad datasets lend themselves well to this. As they can track files of any size or type, they are a suitable overlay structure to include every relevant element for a scientific project, laying the foundation to include all digital data, methods, and provenance. But beyond this, their ability to version control adds an important additional feature to exhaustiveness. The information "I generated X from data Y with software Z" is insufficient for reproducibility and trustworthiness if Y exists in multiple versions or subsets, if different releases of Z have relevant implementation differences, or if Z behaves differently depending on the environment it is used in. If digital research objects are exhaustively tracked, they can be accessed and used transparently in a uniquely identified version state. This identity registration removes ambiguity that arises if the files in question are not completely static. Therefore, the first relevant feature that DataLad provides for reproducibility and reusability is exhaustive version control for all relevant files – from data to code to software environments.

Actionable metadata

Outside of the quote cited above, Bechhofer, De Roure, et al. (2010) further denote the ability to repeat, reproduce, and validate a process as characteristics of reusable research objects: "There should be sufficient information in a Research Object for the original researcher or others to be able to repeat the study, perhaps years later. [...] [Reproducibility] can be seen as a special case of Repeatability where there is a complete set of information such that a final or intermediate result can be verified. In the process of repeating and especially in reproducing a study, we introduce the requirement for some form of comparability framework in order to ascertain whether we have indeed produced the same results. [...] provenance, and being able to audit experiments and investigations is key to the scientific method. Third parties must be able to audit the steps performed in an experiment in order to be convinced of the validity of results" Bechhofer, De Roure, et al. (2010). In common scientific practice, though, process provenance metadata how a file came to be is still often incomplete and difficult to retrace (Hardwicke et al., 2018). As it is tedious, often without an immediate benefit for curators, and rarely explicitly incentivized to retrospectively annotate research objects with process provenance (Edwards et al., 2011; San Gil et al., 2009), it should rather be captured at the time of creation, with the tools and persons that are involved in the creation of research outputs anyway (Dallas, 2016). But while early curation can increase the *comprehensiveness* of metadata, additional measures should guarantee its *validity*, as even fully described research outputs fail to be reproducible or reusable if their description or provenance contains errors (see, e.g., Manninen et al., 2017). The most pragmatic approach to validate metadata is to base subsequent processing on them. For a simple example, consider a codified parametrization of an analysis in a configuration or analysis design file (see, e.g., Jas, Larson, et al., 2018): If an analysis based on it completes successfully, it constitutes valid provenance metadata, created by an expert or automatically at the earliest possible time at no additional cost, and adds immediate benefit for curators as it captures relevant provenance and detects erroneous or missing metadata in passing. Creation and validation is easiest if it is an automatic process within the research process, and if the tools used during the creation also use the same metadata that gets eventually published alongside the final research output. Therefore, the actionable metadata DataLad can acquire from command executions are the second relevant property for reusability. Even if this metadata does not follow established community standards as required by the FAIR principles yet, it preserves knowledge that would otherwise be lost, without requiring additional training, impeding later additions, or putting additional burden on scientists – it is a byproduct of standard scientific practice. The fact that it can be re-executed automatically, and that resulting recomputations are automatically compared to previous versions by employed version control tools, provides an effortless form of validation.

Modular structure

The reusability of scientific work can improve if it is accessible in modular units that constitute unambiguous multi-use components, such as raw data, processed data, or software. In the simplest case, modularization means placing conceptually distinct content into separate files, and grouping files in individual directories to reflect more global structures. Distinct units, such as the directories "code/" and "inputs/", increase transparency if each location is associated with distinguishable content, ease flexible recombination of such components into new projects, allow continuous evolution of an individual module without impact on other components of a project, and enable location-specific access control. Though a single modular unit can not entail all relevant elements of

a scientific study or data analysis, exhaustive tracking of all elements without sacrificing modularity can be achieved by linking multiple modular units in dependency relationships. As Bechhofer, De Roure, et al. (2010) write above, associations to other research objects should be more than "just a citation or reference", but an actionable link that provides access to whichever resource it refers to. A useful metaphor that matches this description are package management systems such as conda (https://docs.conda.io) or APT (https://wiki.debian.org/Apt): A software package is a modular unit, installed with a package manager. However, packages usually depend on other software packages, which are listed as its "dependencies". During installation, package managers check if all of the linked dependencies exist on the system, and if not, install them in the required versions automatically. For scientific projects, we can conceptualize its modular units (data, software, code) as the dependencies of a given research output. When those units are tracked as datasets, dependency relationships in the form of subdatasets form actionable links with precise versions. Thus, a third feature for reusability is modularity as provided with DataLad's subdataset mechanism (see Figure 1.5). In a superdataset, a registered subdataset is known with a location and version identifier. On demand, it can be installed precisely as needed, from whichever location or service it is available from, similar to the process that Bechhofer, De Roure, et al. (2010) envisioned.

Portability

Bechhofer, De Roure, et al. (2010) further highlight the ability to share reusable research objects within and across organizational boundaries. Fittingly, the fourth and final property DataLad datasets entail for reusability is portability. The more portable a digital research object is, the easier it is to reuse it. A research object is fully portable if no adjustments are necessary for it to function the way it is intended to on different computational infrastructure – ideally even when used by a naive re-user with a different area of expertise (i.e., without domain knowledge). The more adjustment or domain knowledge is necessary, the less portable a research output becomes. As a simple example for what this would entail, consider custom scripts for an analysis. If they internally reference the file system outside of their DataLad dataset, or refer to locations with specific absolute paths, they could not run on a different computer. A re-user would need to adjust the scripts, or their system, to match the expectations implicitly encoded into the script, which impedes reusability considerably. Only if a re-user does not need to modify project files, they can be certain that they did not inadvertently break or influence the output with it.
Achieving the previous properties in a research output is work towards portability. A factor contributing to portability is self-containment such that research outputs can be used or reproduced on different computational infrastructure by other users without requiring additional elements outside of it. Exhaustive versioning and modularization pave the way to include all necessary code, data, and computational environments into a complete, self-contained structure that accompanies a research output. If a user ensures that a project can be moved across computers and remains functional without adjustment, for example by ensuring that no references to file system, operating system, or user specific idiosyncrasies are included, portability is improved further. And with actionable process provenance that can be re-executed, verification and re-use even without domain knowledge becomes possible.

Figure 3.2: Reusing research outputs implies trust, and trust should be earned through verification. DataLad datasets possess features that assist with this aim. Verification is enabled by provenance information. Provenance capture of research outcomes requires **exhaustive versioning** of all inputs, as well as process records that describe how inputs were combined and transformed to generate outputs (middle). When lightweight metadata are **actionable**, they can be used to reproduce research outputs in a different environment from precisely identified inputs by (re-)applying the recorded process (left). A data structure that affords this type of **portable** recomputation is a self-contained unit that can be reused as a **modular** input component for incremental research (right).

Although FAIR research objects are universally desirable, in practice, the necessary standards and procedures for creating FAIR (meta)data are not always already in place when research is conducted. This can turn FAIRification into a bureaucratic data governance effort, diminishing the immediately obvious benefits for the curator (Zehl et al., 2016). However, the four properties outlined above are a big step into the right direction, and essentially a byproduct of pragmatic research data management in DataLad datasets. Figure 3.2 illustrates how it yields trusted and reusable research objects, and the next section details a technical implementation and proof-of-concept analysis to create such reusable research objects as a byproduct of RDM in analyses of any scale.

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

The following section is a short overview of our original publication (Wagner, Waite, Wierzba, et al., 2022). The reader is invited to visit the published paper for further details.

Large-scale datasets pose additional challenges for FAIR research objects. Their storage and computational demands increasingly exceed common high performance computing (HPC) infrastructure, and computing procedures that are common in fields accustomed to smaller datasets such as storing multiple copies of the data become infeasible (Horien et al., 2021). As the complexity of reproducing and verifying large scale datasets growths, the trustworthiness of derivative data decreases. And as data processing results often multiply storage demands, keeping intermediate outcomes on disk is rendered increasingly prohibitive, which further impedes the possibility to retrace the origin of research outcomes. Yet for large scale datasets specifically, sharing data derivatives is the most – or sometimes the only – viable way to extend previous research (Craddock et al., 2013): It opens

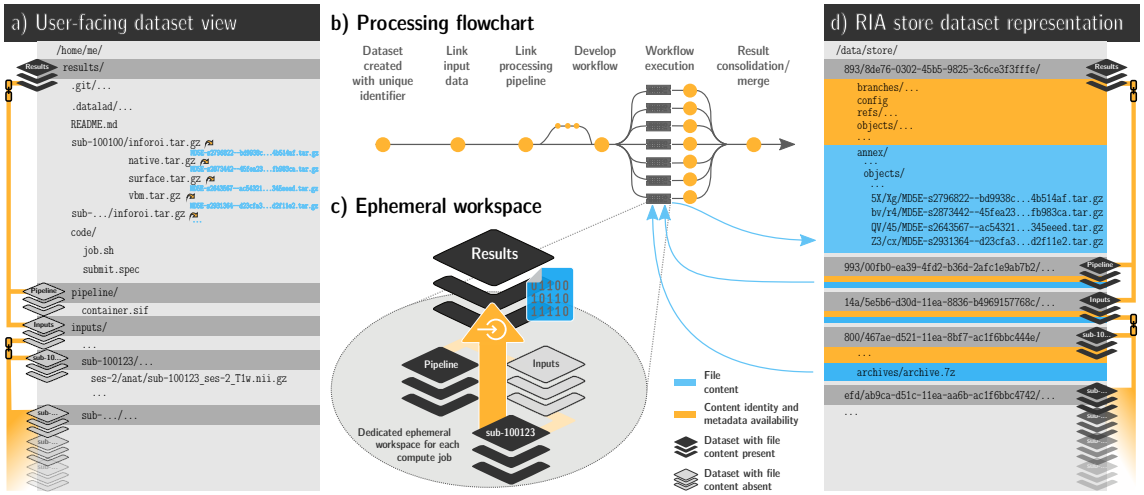


Figure 3.3: Framework overview. a) A DataLad dataset links input data and pipeline, and contains computed results with actionable process provenance and location information post-computation, allowing on-demand file retrieval or recomputation. b) Process-flowchart: First, a DataLad dataset exhaustively tracks required processing components as modules (e.g., input data, processing pipeline). Next, compute jobs are executed, if possible in parallel, capturing and validating provenance required for computation. Finally, results and provenance from parallel executions are merged. c) An ephemeral (short-lived) compute workspace for validating process provenance. The temporary, lean clone retrieves relevant subsets of data based on its parametrization and information recorded in the dataset. Upon success, process provenance and results are pushed into permanent storage (see d), and the ephemeral workspace is purged. d) The internal dataset representation in a RIA store: The store receives results and can contain input data, optionally using compressed archives or encryption during storage and transport. It is the only place where results take up permanent disk space. License: CC-BY-SA, Wagner, Waite, Wierzba, et al. (2022).

up research opportunities to scholars without access to adequate computational resources, and minimizes duplicate analysis efforts for resource-heavy, costly computations with considerable environmental impact (Portegies Zwart, 2020).

Based on DataLad, containerization software, and job scheduling systems, we build a portable, free and open source framework for scalable reproducible processing. It applies workflows from software engineering – in particular distributed development – to computational research, and empowers reusers to automatically reproduce results based on machine-actionable records of computational provenance without access to the original infrastructure. For this, it puts the aforementioned properties in practice, using DataLad datasets as a comprehensive data structure to track all elements of digital processing, perform portable computing in automatically bootstrapped ephemeral workspaces, and capture validated and re-executable process provenance records.

3.3.1 Framework overview

The framework combines distributed version control, containerization, job scheduling, and storage solutions with optional encryption and compression into a sequential analysis workflow (Figure 3.3): The start and end point of the workflow is a portable DataLad dataset that contains the exact identity and location of all data processing inputs and, eventually, re-executable processing results (Figure 3.3a). In a first step, it is assembled from all relevant processing components, most commonly input data and software containers with computational environments or pipelines (Figure 3.3b). The use of software containers, while not strictly required, is a practical method to provide portable computational environments and forgo a number of challenges that computational reproducibility otherwise poses. To harvest the advantages of modularity, processing components are placed in separate DataLad datasets and linked as subdatasets. To provide features that are necessary for processing personal or large-scale data, such as encryption (to comply with data protection regulations), compression (to save disk space), and composition into archive files (to reduce the number of files), DataLad datasets involved in the workflow are placed into so-called RIA stores (Poldrack, Wagner, et al., 2021). This "backend" representation facilitates programmatic data management and reduces storage demands by storing a dataset of any size in 25 files, with optional compression and encryption. Figure 3.3a illustrates a DataLad dataset's front-end structure when cloned in a user's workspace, and Figure 3.3d shows its RIA representation.
Based on this self-contained structure, the framework needs to conduct user-defined processing in a way that captures and validates actionable metadata. While DataLad's `containers-run` functionality provides provenance capture, simultaneous validation requires additional tweaks. To achieve it, the framework bootstraps an entire temporary analysis environment from scratch based on the information contained in the initial dataset (Figure 3.3c). In this ephemeral environment, processing can only be based on information already recorded in the dataset or provided in the execution command, which is invoked with a provenance capturing `datalad run` or `datalad containers-run`. It specifies required inputs and expected outputs by their relative path in the DataLad dataset hierarchy. If the computations succeed, two validity guarantees can be derived regarding the provenance of the computational outcomes: 1) All dataset modifications can be causally attributed to the initiated computation; 2) only declared inputs were required to produce the outcomes. This constitutes evidence that existing information is valid and sufficient to make the dataset portable. Figure 3.4 illustrates this provenance, how it is captured, and how it can be used for recomputation.

The basis for ephemeral workspaces is the ability to distribute DataLad datasets across local or remote infrastructure as lightweight, linked clones, and the process mirrors a software development routine, where changes are developed in a distributed network: An orchestration layer clones the DataLad dataset into a temporary location, uses its recorded history to infer relevant details about precessing components, and adds results on top of it. Afterwards, results and their provenance can be pushed back before the ephemeral workspace is purged. To ensure that this orchestration can scale, it needs to support parallel executions, i.e., several analyses running in ephemeral workspaces at the same time. This is complicated in Git repositories as concurrent processes can interfere with one another and simultaneous commits can cause conflicts. The use of file locking, a mechanism to control concurrent access to files by enforcing the serialization of updates, can limit the number of processes that modify specific locations on the file system and prevent read operations while a modification takes place (Xoxa et al., 2015). Beyond that, two procedural techniques, again adopted from distributed software development, make the orchestration more efficient. The first consists

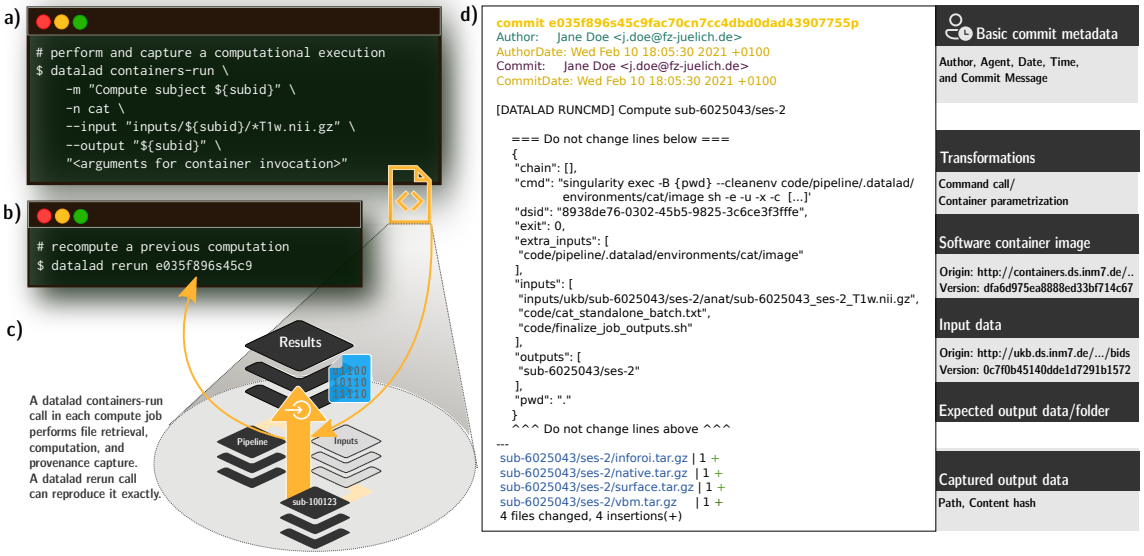


Figure 3.4: Process provenance of an individual job, its generation, and re-execution. a) A `datalad containers-run` command with a container name specification (cat), a container parametrization or command, a commit message, and an input and output data specification generates actionable process provenance as a structured record linked to a Git commit. b) To re-execute it, `datalad rerun` needs to be parameterized with a revision identifier, such as a Git tag, a "commit shasum" (`e035f896s45c9f`), or a revision range containing one or more commits with associated provenance records. c) As the core execution command (see Listing 1, line 24-30) at the center of each individual job, it performs data retrieval, container execution, and result capture, and generates the actionable provenance. d) The provenance record stored alongside computed results in the revision history. A legend (right) highlights the most important pieces of recorded provenance. It is re-executable with DataLad only, but sufficient information to repeat a computation using other means can also be inferred from the structured JSON records by other software or even humans. License: CC-BY-SA, Wagner, Waite, Wierzba, et al. (2022).

of duplicating the DataLad dataset that is used as an analysis starting point into two clones: One remains unaltered and acts as an input source from which the ephemeral clones are bootstrapped. The other one acts as the target location for results (Figure 3.3d). This separation prevents concurrent clone and push operations. Moreover, as the clone location does not receive any results, the dataset is always at the leanest state, which ensures fast cloning. The second technique involves the use of branches, independent segments of a DataLad dataset's history that allow for parallel developments based on a common starting point. Each individual ephemeral workspace saves its results and provenance on a unique branch. When several ephemeral clones push their individual histories into a single location, they thus do not conflict with one another. This process mirrors feature development in software projects where a mainline branch contains agreed upon code. Changes are build on top of the mainline code, but in branches such that concurrent developments of several contributors do not conflict, and later, a so-called `merge` can integrate a branch into another one. The clone and push orchestration is implemented as a job script that wraps the desired processing.

Listing 3.1 contains an example bash script: To set up the ephemeral workspace and push target, it receives the source dataset for cloning and result dataset for pushing as parameters (lines 5-6), clones the source into a temporary location (line 9), registers the push target (line 12), and creates a unique branch (line 14). Afterwards, analysis-specific code can run with provenance tracking, and as a last step, its results and provenance are pushed into the destination dataset (lines 34, 38, Figure 3.31b, blue arrow).

Listing 3.1 Processing orchestration in a bash script. A batch system invokes it in a temporary working directory with three parameters: a URL of a source DataLad dataset, a URL to deposit job-results at, and an identifier to select a sample for processing. The job implementation conducts three main steps: 1) `clone` a DataLad dataset with all information to bootstrap an ephemeral computing environment for each job; 2) `containers-run` a containerized pipeline with a comprehensive specification of to-be-retrieved inputs and to-be-captured outputs; 3) `push` captured outputs (concurrency-safe; line 34) and process provenance records (not concurrency-safe, protected by a file lock; line 38) to a permanent storage location. Jobs are independent and can run in parallel. Potential job configurations can be supplied using batch system specific means (e.g., `DSLOCKFILE` and `JOBID` environment variables). Processing can be adjusted with the container invocation.

```
1  #!/bin/bash
2  # fail on any issue, show commands
3  set -e -u -x
4  # name arguments for readability
5  dssource="$1"
6  pushgitremote="$2"
7  subid="$3"
8  # obtain the source dataset
9  datalad clone "${dssource}" ds
10 cd ds
11 # register location for result deposition
12 git remote add outputstore "$pushgitremote"
13 # create job-specific branch for results
14 git checkout -b "job-$JOBID"
15 # START OF APPLICATION-SPECIFIC CODE
16 # pull down input data manually,
17 # only needed for wildcard-based file
18 # selection in the next command
19 datalad get -n "inputs/ukb/${subid}"
20 # datalad containers-run executes
21 # the "CAT" pipeline. Specified
22 # inputs are auto-obtained, specified
23 # outputs are saved with provenance
24 datalad containers-run \
25 -m "Compute subject ${subid}" \
26 -n cat \
27 --explicit \
28 --output "${subid}" \
29 --input "inputs/ukb/${subid}/*T1w.nii.gz"
30 "<container invokation arguments>"
31 # END OF APPLICATION-SPECIFIC CODE
32 # push result file content to the
33 # configured "storage-remote"
34 datalad push --to storage-remote
35 # push branch with provenance records
36 # needs a global lock to prevent
37 # write conflicts
38 flock "$DSLOCKFILE" git push outputstore
39 # log entry to mark non-error exit
40 echo SUCCESS
41
```

One script execution equals one processing in one ephemeral workspace. By splitting the computation into parts, operations can be parallelized. A common example is one processing pipeline per participant, i.e., the parallelized execution of a processing pipeline on different, independent parts of input data. Listing 3.1 exemplifies this: A specific participant identifier, provided as a parameter (line 7), determines the subset of input data a processing pipeline runs on (Figure 3.3c). Precisely how to split a specific computation is dependent on its nature, but as parallelization often corresponds to the granularity at which a recomputation will be possible in our framework, relevant considerations are, for example: "What is the smallest unit for which a recomputation is desirable?", or "For which unit size is a recomputation still feasible on commodity hardware?" (Wagner, Waite, Wierzba, et al., 2022). Once a fitting granularity is determined, a job scheduling system can assist in deploying thousands of computations in parallel. To automate this process, the framework supports

two major job schedulers, HTCondor (Thain et al., 2005) and SLURM (Yoo et al., 2003), natively. Their configuration is highly infrastructure-specific, and must determine an optimum balance of resource demands, such as run time, memory and storage requirements, in order to achieve optimal throughput. Importantly, the final process provenance neither contains the orchestration of ephemeral workspaces, nor the scheduling layer. For a original computation, these layers can thus contain necessary platform-specific tuning without impacting the portability of the final research object. This means DataLad, a software container tool, and a basic UNIX shell environment are the only requirements for recomputing captured outputs, and recomputation on systems with different batch scheduling software is possible by providing alternative job specifications, without changes to the pipeline implementation.

If the necessary processing elements exist in suitable formats (e.g., DataLad datasets with processing components, scripts stripped of platform idiosyncrasies, and so forth), the above setup can be created automatically with an openly shared bootstrapping script[1]. After setup, two things are left to do for a user: Start the processing, and, once the results from all ephemeral workspaces are aggregated in branches in the target dataset, consolidate them the mainline branch (Figure 3.3b, "Result consolidation"). In the simplest case, all compute jobs produced non-intersecting outputs, i.e., no single file was written to by more than one compute job. In this case, all branches can be merged at once using a so-called octopus-merge.

The outcome of this consolidation process is a self-contained DataLad dataset, with valid, machine-actionable provenance information for every single result file of the performed data processing. As such, it is a modular unit of data, suitable as input for further processing and analysis. And it is a practical example of how the four properties – exhaustively versioned, actionable, modular, and portable – translate to trustworthy, reusable research outputs. Overall, our framework is a general-purpose solution that is compatible with any data that can be represented as files of any size, and any computation that can be performed via a command line call. It is built on a collection of portable, free and open-source tools that can be deployed without special privileges or administrative coordination on standard high-throughput computing (HTC)/HPC infrastructure, or personal computing equipment. An open tutorial and result showcase is publicly available at github.com/psychoinformatics-de/fairly-big-processing-workflow-tutorial. To test its capabilities further, we conducted a proof-of-concept analysis on one of the largest neuroscientific datasets to date.

3.3.2 Proof-of-concept analysis

In a proof-of-concept analysis, we applied the framework to run a containerized pipeline for voxel-based morphometry (VBM) (Ashburner and Friston, 2000) from the Computational Anatomy Toolbox (CAT) (Gaser and Dahnke, n.d.) on data from the UKB project (Matthews and Sudlow, 2015, comprising 76 TB in 43 million files under strict usage constraints). In doing so, we assessed the framework's scalability and if it can be used across different infrastructures, investigated result variability between two recomputations of the pipeline, and, in order to demonstrate that the framework can capture and re-execute complete process provenance, we also recomputed individual results on a personal laptop. An overview of the analysis is shown in Figure 3.5, and the setup steps are publicly available as a bootstrap script[2].

[1] www.github.com/psychoinformatics-de/fairly-big-processing-workflow.

[2] https://github.com/psychoinformatics-de/fairly-big-processing-workflow/blob/main/bootstrap_ukb_cat.sh

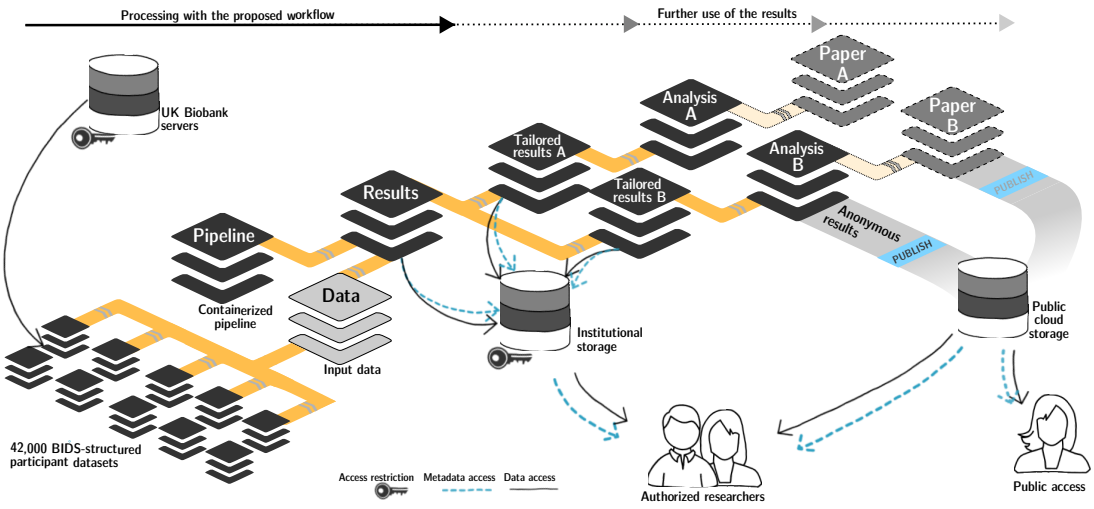


Figure 3.5: Overview of DataLad dataset linkage through processing and reuse. Any DataLad dataset may comprise other DataLad datasets as subdatasets via lightweight but actionable and versioned links. This connects a dataset to the content and provenance of a different modular unit of data, such as the outcomes of a the preceding processing step. The genesis of an analysis output (Analysis A/B) based on intermediate processing outcomes (Tailored results A/B) can thus be traced back all the way to the original raw data. Access control and storage choices are independent across individual components in this network of linked data modules. Aggregated data and analysis results can be shared with larger audiences or publicly on a variety of platforms, while raw and derived data may underlie particular access restrictions, or require particular hosting solutions due to their size. License: CC-BY-SA, Wagner, Waite, Wierzba, et al. (2022).

Workflow setup

First, using the DataLad extension `datalad-ukbiobank` (Hanke, Waite, et al., 2022), we retrieved MRI data into one DataLad dataset per participant and restructured it to BIDS (Gorgolewski, Auer, et al., 2016), yielding 42 715 datasets in total. A single superdataset ("Data" in Figure 3.5) tracks them using about 40 MB of space. It is installable within seconds to retrieve any registered file content in the DataLad dataset hierarchy on demand.

To set up a processing pipeline, we built a Singularity container for the Computational Anatomy Toolbox (Gaser and Dahnke, n.d., CAT, version: CAT12.7-RC2, r1720), which is an extension to the Statistical Parametric Mapping software (SPM; version: SPM12, r7771; www.fil.ion.ucl.ac.uk/spm/software)[3]. From it, we chose CAT's default segmentation of structural T1-weighted images using geodesic shooting (Ashburner and Friston, 2011), including calculation of total gray matter (GM), white matter (WM), and total intracranial volume (TIV), as well as extraction of regional GM estimates from several brain parcellations. We then linked these two inputs as subdatasets to the processing dataset ("Results" in Figure 3.5), and added two custom code files. First, a batch script with CAT's processing steps and parameterization, which bundled up all relevant analysis steps into

[3] A detailed description and full recipe of the container together with instructions on how to build and use it is publicly available at github.com/m-wierzba/cat-container.

single command. This script runs per image, and constitutes the smallest unit for recomputation. And second, a utility script to post-process all relevant outputs with a focus to reduce the number of files and artificial variations between recomputations. For this, it stripped results of timestamps and other non-deterministic log file content, and used the `tar` utility to bundle VBM results into four reproducibly organized archives according to envisioned consumption scenarios (see the files in Figure 3.3a). These measures were taken to obtain a meaningful estimate of intra-pipeline variability across recomputations, and to keep the number of files in the resulting dataset in a manageable range.

To test the frameworks portability and estimate result variability across computations, we set the analysis up to run on two systems, a HPC cluster and a HTC cluster. The two systems used SLURM or HTCondor for job scheduling, respectively, and each system imposed different resource constraints. The HPC system, a modular supercomputer with abundant disk space and compute power (Krause and Thörnig, 2018), imposed an inode quota, a limit on the total number of files, of 4.4 million – less than the total number of files of the raw dataset. The HTC cluster, in contrast, was constrained on storage capacity, preventing the existence of more than one copy of the raw dataset, and limiting the size of derivatives that could be stored. The framework was thus setup identically across the employed systems apart from the implementation of job submission, which accounted for platform specific idiosyncrasies such as these restrictions. Irrespective of system, one compute job per participant was generated. This compute job serially processed all available anatomical images (0, 1, or 2) for a given participant. For each image, a dedicated provenance record was captured, yielding a total of 41 180 records. The platform-specific job submission scripts adjusted the job load to the available disk space or inode resources when disk space or inode availability were insufficient for the full dataset by scheduling only as many jobs in parallel as there were resources to handle extracted inputs.

Workflow execution and consolidation

On the HPC system we completed data processing for the one-hour-per-image pipeline within 10.5 hours, using 25 dedicated compute nodes, each executing 125 jobs in parallel on RAM disks with GNU Parallel (Tange et al., 2011). On the HTC system in turn, computations were scheduled dynamically across several weeks using HTCondor. On both systems, recorded outputs and provenance records amounted to a total of 995.6 GB of computed derivatives in 163 212 files. After completion, results were consolidated by merging result branches, yielding a collection of different VBM-related measures for all images in the sample, represented in archives, and annotated with re-executable provenance records in a DataLad dataset.

This DataLad dataset was then subsampled into smaller "special purpose" datasets tuned for specific research questions, containing extracted and optionally aggregated subsets of the results for easier consumption and faster access. For this, the main result DataLad dataset became an input to a new tailored DataLad dataset via nesting ("Tailored results A/B" in Figure 3.5), which extracted and transformed required files with provenance tracking using `datalad run`. If underlying large-scale computation is redone or extended, the subsampled datasets can be updated by re-applying this transformation with a `datalad rerun`. And throughout the dataset hierarchy – using the encoded, machine-actionable provenance information – a single result can be traced to the precise files they were generated from in a transparent and reproducible manner. The direct computational output of the proof-of-concept framework execution is therefore not a final result, but an intermediate

representation optimized for storage and handling. As more tailored views for concrete use cases can be flexibly and reproducibly created, we achieve a compromise between the desires of a data consumer and the demands of the storage infrastructure and operators for such large scale datasets.

Intra-pipeline variability

Pipeline replication can play an important role for reproducible research as a means of exploring analytic variation and assessing the robustness of findings (Li et al., 2021), and our framework is particularly convenient for this application. In order to investigate intra-pipeline variability, the result consolidation described above was first performed separately on each computational infrastructure. Afterwards, the two complete sets of results were integrated in the same dataset, as two different branches, to estimate result variability. With the exception of execution time, the number of jobs, proportion of successful jobs, and size and structure of the results were identical between the two systems[4]. As content identity is precisely captured, bit-identical recomputations are easy to identify. One type of the created output archives, surface and thickness projections, never yielded bit-identical results across computations. But among the other three types of archives (modulated gray matter density and partial volume estimates in template space, atlas projections and partial volumes in individual space, and regional volume and thickness estimates of several atlases/parcellations), more than 50% of all output files were bit-identical across the two computations. A closer investigation of non-identical results revealed that outcome variability was largely attributable to minor numerical differences. We illustrated the amount of dissimilarity by computing the mean squared error (MSE) over recomputations for a range of key VBM estimates. They can be found in Table 3.1. We also correlated VBM estimate distributions across recomputations for different brain parcellations included in the CAT toolbox output. The lowest observed correlation were Pearson's $\rho > 0.998$ for the Destrieux 2009 surface parcellation (Destrieux et al., 2010) for all brain regions. Finally, we derived quality control metrics for anatomical images from the computed results (Dahnke et al., 2013, 2015) and correlated them across results. They exhibit $\rho > 0.99999$ for computation and recomputation.

measure	μ	MSE
total surface area	1891	0.315
cerebro-spinal fluid	365	0.052
total intracranial volume	1508	0.052
white matter	519	0.004
gray matter	621	0.001

Table 3.1: Mean and mean squared error of results across computations

[4]The difference in computational performance can be seen in a visualization of provenance information at youtube.com/watch?v=UsW6xN2f2jc. This visualization was awarded second place in the category videos & animations in the 2021 OHBM Brain Art Competition.

Infrastructure-agnostic reproducibility

The potential for full computational reproducibility not only permits structured investigations of result variability, but also increases the trustworthiness of the research process per se. To confirm computational reproducibility for a consumer, we performed an automatic recomputation of individual results on a personal laptop with much lower storage and compute capacity than a cluster, and without job scheduling software. This type of spot-checking results resembles the scenario of an interested reader or reviewer of a scientific publication with access to (parts of) the data, but no access to adequate large-scale computing resources. The recomputation solely relied on the local availability of the Singularity container technology, but was otherwise fully automatic. Its success confirmed that platform idiosyncrasies were confined to the outer job scheduling layer only, and that the resulting research object was portable across infrastructures.

3.3.3 Summary

Our framework constitutes a technical solution that brings together a range of previous work in the area of research data management, reproducibility, and software development. We have tested its validity and usefulness with concrete applications in the field of neuroimaging, and yielded a reusable research outcome that feeds into numerous data analysis applications. The work outlined in this thesis so far thus culminated in the creation of a flexible and scalable framework to create reusable research objects. This work also laid out the foundations for the reproducibility and portability required by the distributed nature of the upcoming MEG analyses. But the luxury of applying standardized analysis or preprocessing pipelines to data is not a given in all research projects. Thus, the next and final chapter of this thesis will apply the work so far for exploratory projects rooted in the topic and methodology of investigating brain states in working memory.

4 Reproducible brain state analyses

> If we knew what it was we were doing, it would not be called research, would it?
>
> *(Albert Einstein)*

The previous chapter focused on the technical challenges of reproducibility management, especially when applications need to scale to large samples, but it also highlighted readily available, pragmatic strategies to ensure portability and reusability. A different challenge arises when the to be undertaken analyses are not entirely mapped out from the start, and a research project is exploratory in nature. This upcoming chapter describes the central data analysis project in this thesis, studying working memory in sequential decision making. The setup of the exploratory analyses I conducted in this project used best practices and tools from Chapter 1.2.2, and followed the strategies and the framework outlined in Chapter 3. Corresponding project directories leave a transparent digital provenance trace for future reuse, and analyses are portable across infrastructures. At the time of writing, the work presented in this chapter is being prepared as a research article.

4.1 Project outline

Drawing insights from the partial overview of the literature in Chapter 1, methods to study working memory have shifted from traditional evoked potentials to computationally intensive processing. The field has progressed from studying spiking activity in single cells to an implicit consensus that the neural representation of working memory during maintenance is high-dimensional or embedded in a high-dimensional space, time-varying, and spatiotemporal distributed. But despite the wealth of theories and methodological approaches, the field has not yet converged at a full understanding of working memory. Recent reviews have begun to point out where findings and standard views fall short to yet explain the mechanisms underpinning working memory (Nobre, 2022), or highlight the inconsistencies in results across studies (Pavlov and Kotchoubey, 2022). While studies traditionally focused on the involvement of prefrontal areas due to its involvement in executive control (e.g., Funahashi et al., 1989; Fuster and Alexander, 1971; Miller et al., 1996), many other brain areas have been shown to represent working memory items, too. In a recent review, Sreenivasan and D'Esposito (2019) summarized how single-cell measurements, MEEG, and fMRI acquisitions have found increased activity during working memory maintenance throughout the cortex and even some subcortical areas. D'Esposito (2007) also described working memory as an emergent property of functional interactions prefrontal areas and the rest of the brain as opposed to being localized to a single brain region. And while an often proposed mechanism for the coordination of temporal and spatial population codes are brain oscillations in specific frequency bands Roux and Uhlhaas (e.g., 2014), a recent systematic review by Pavlov and Kotchoubey (2022) highlighted a prominent lack of agreement across results in experimental studies that reported observations of brain oscillations in working memory tasks.

In light of variable and even conflicting findings, Nobre (2022) called for sustained open-mindedness and creativity in researching working memory in a recent reflective piece. In this exploratory spirit, I conducted the simulations and analyses that will be described in this chapter. They build up on previous analysis that were conducted on this dataset by Kaiser, Gruendler, et al. (2018), who studied the temporal dynamics of working memory maintenance with this dataset, in particular how decision-relevant stimulus features are represented in the delay phase. My extensions of their analyses connect to the works of Murray et al. (2017) and Machens et al. (2010): Murray et al. (2017) found neural subspaces within dynamic population-level activity that showed stable coding for working memory items. They used PCA to construct low-dimensional subspaces ($k = 1$ or $k = 2$, respectively) in electro-physiological recordings from non-human primates, and projected neural recordings into the subspaces such that the principal components became the axes of a new, lower-dimensional state space. In these subspaces, stable population codes and above-chance decoding accuracies for working memory items in two different working memory tasks arose. Machens et al. (2010) similarly used PCA on high-dimensional electro-physiological data to create a new, lower-dimensional subspace ($k = 6$) based on identified components, and interpreted the resulting axes in relation to the experimental decision making task. In the following, they identified components representing time ("when") and components representing working memory content ("what"). Taken together, these studies thus employed dimensionality reduction methods to perform statistical decomposition in a high dimensional "native neural space" to project the axes of a task space into subspaces of the native neural space and assigned meaning to them (see Figure 1.1b). Yet contrary to Machens et al. (2010) and Murray et al. (2017), the upcoming analyses do not confine the neural signal to a specific brain area such as the PFC, but attempt to find neural signatures of decision-relevant working memory items across the cortex. Central to the analyses is also the attempt to use a functional alignment method more commonly used in fMRI for dimensionality reduction.

Such an exploratory neuroscientific project faces several challenges. As outlined in Chapter 1, analyses are often multi-stepped. But repeating the same processing step in all of many different analyses is computationally inefficient. Furthermore, a large number of analyses makes it difficult to retain an intuitive structure. When several different analyses are conducted in the same directory, researchers unfamiliar with the project struggle to associate inputs, code, software, and results. But when different analyses are broken into distinct projects, the disk space demands of the underlying neural data usually multiply. The distributed nature of the project across several institutions also demanded uncompromising portability. Common exploratory approaches such as interactive computing or notebooks typically do not fulfill this requirement due to their short-lived and environment-dependent nature. To overcome these challenges, research data management principles from the previous chapter were applied. All analyses were set up as portable DataLad datasets. The project evolution was captured in a linked hierarchy of datasets, starting with a BIDS-standardized raw dataset, continuing with a preprocessed derivative dataset, and spreading into multiple analyses applications based on a common starting point (Figure 4.1). Analyses were codified into infrastructure-agnostic scripts, and their execution was provenance-tracked to allow for re-computations over the course of the project. Each individual component in the Figure 4.1 below constitutes a self-described entry point to further analyses at a later point in time.

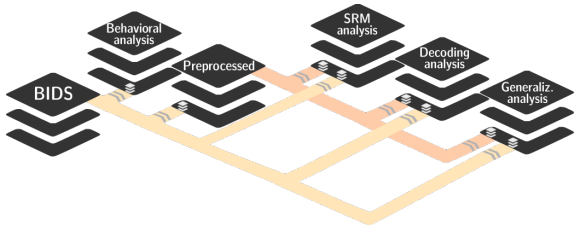


Figure 4.1: Project outline as linked DataLad datasets for provenance capture, disk usage reduction, and computational efficiency. A BIDS-structured raw dataset is the anchor of all analyses. A common preprocessing pipeline is the basis of three major MEG analyses. If raw or preprocessed data changes, analysis results can be recomputed automatically.

4.2 Magnetoencephalograpy (MEG)

The plurality of analyses in the upcoming sections warrants a more detailed overview of the basic properties, strengths, and weaknesses of data stemming from magnetoencephalography acquisitions. MEG is a neuroimaging method to record neural activity from magnetic flux, the total magnetic field passing though a given area. It is measured on the surface of the head with sensitive detectors, so-called Superconducting Quantum Interference Devices (SQUID) sensors. The electric currents that give rise to these magnetic fields stem from synchronous inhibitory or excitatory postsynaptic potentials in parallel pyramidal cells in the cortex, mostly those cells in the walls of the cortical sulci. As the potential differences between soma and axons of neurons form magnetic dipoles, and these cells are perpendicular to the cortical sheet of the gray matter and thus similarly oriented, measurable magnetic fields emerge at the sculp when aggregated across a neuronal population. The resulting cortical magnetic fields are on the order of 100-500 femtotesla (fT) (Hari and Puce, 2017). Compared to the environmental noise, this is a very weak magnetic field. Therefore, in addition to active and passive shielding from magnetic or electronic inferences, liquid helium in the MEG device cools the SQUID sensors to -269°C and allows them to be superconductive. In this state, a superconducting coil in the SQUID can amplify the magnetic flux, which is picked up by a nearby pickup coil, also called flux transformer. Three main types of flux transformers exist: magnetometer, axial gradiometer, and planar gradiometer, each with a different sensitivity profile. An MEG device can contain different types of flux transformers and the total number of flux transformers is the amount of channels of the MEG system (Puce and Hämäläinen, 2017). The analysis of the recorded data then consists of interpreting the resulting topographies given the employed flux transformer. Typically, MEG devices contain around 150-300 sensors arranged in a helmet-shaped array that cover the whole human scalp at a frequency of 1000-5000Hz. The resulting temporal resolution is high compared to neuroimaging methods such as fMRI or positron emission tomography (PET). This makes it a highly suitable modality to study cognitive processes that occur in the range of milliseconds. The spatial resolution is, however, inferior to the spatial resolution that can be achieved with fMRI. This is aggravated further by the superposition principle of the magnetic field, stating that fields arising from several currents are linear sums of fields generated by each single current. Thus, the measured signal at the sculp is a mix of multiple magnetic fields, stemming from different sources within and outside the brain that are difficult to disentangle (Hari

and Puce, 2017). Data are said to be in "sensor space" when read out from the sensors, and in "source space" when its cortical sources have been estimated. All analyses conducted on MEG data in this chapter were carried out in sensor space.

4.3 Study overview

The data basis of the upcoming analyses stems from a study that investigated how decision-relevant information is retained in working memory using a decision making task with concurrent MEG acquisition. The data were acquired as part of the SFB 779 project B16N ("offline value representations in sequential decision making") at the Otto-von-Guericke University Magdeburg by Kaiser, Gründler, and Jocham (2016), and have not been previously published. Ethics approval was obtained from the institutional review board of the medical faculty of the University Magdeburg under application number 101/15.

4.3.1 Participants

$N = 22$ right-handed, healthy participants with normal or corrected-to-normal vision were recruited at the Center for Behavioral Brain Sciences and on the University Magdeburg campus. Their mean age was 26 years, and 10 participants were male. Handedness was assessed with the Edinburgh Handedness Inventory (Oldfield, 1971). Participants gave their informed consent to participate in the study, and received a base monetary compensation in the order of 8€ per hour with a performance-dependent bonus of up to 3€.

4.3.2 Experimental design

The experiment consisted of 510 trials, grouped into 5 blocks with a variable break in between. Each trial required a decision between one of two stimulus options, presented as gabor patches on the left and right side of the screen (Figure 4.3b). Importantly, stimulus options were presented in succession, with a delay period through which the decision-relevant properties of the first stimulus had to be retained in working memory. The number of stripes in the gabor patch encoded the reward magnitude (either 0.5, 1, 2, or 4 points), and the angle of stripes encoded reward probability (either 10%, 20%, 40%, or 80%). The more stripes, the higher the reward, and the larger the angle, the higher the probability. Participants learned these associations in a tutorial prior to the experiment (Figure 4.3a), and Figure 4.2 provides an overview of stimuli. Stimulus combinations and sequences were selected to fulfill the following requirements:

1. To prevent subjects from forming a decision already in the delay period (Curtis and Lee, 2010), the left stimulus never contained the best or worst combination of properties (see Figure 4.2).
2. In the first option, all magnitude and probability options are shown equally often.
3. No more than three repetitions of the same probability and magnitude combination occur in the left stimulus.
4. First and second stimulus are never identical.

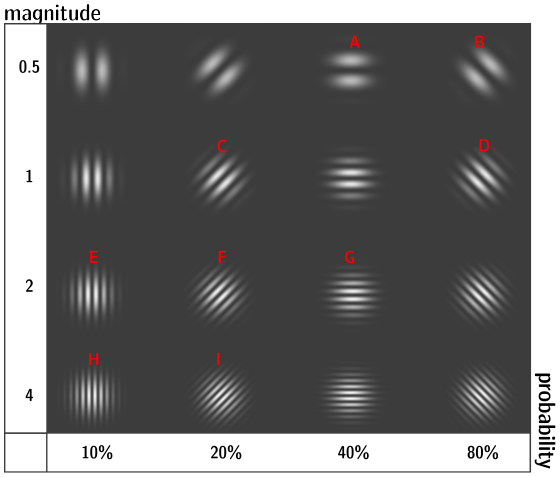
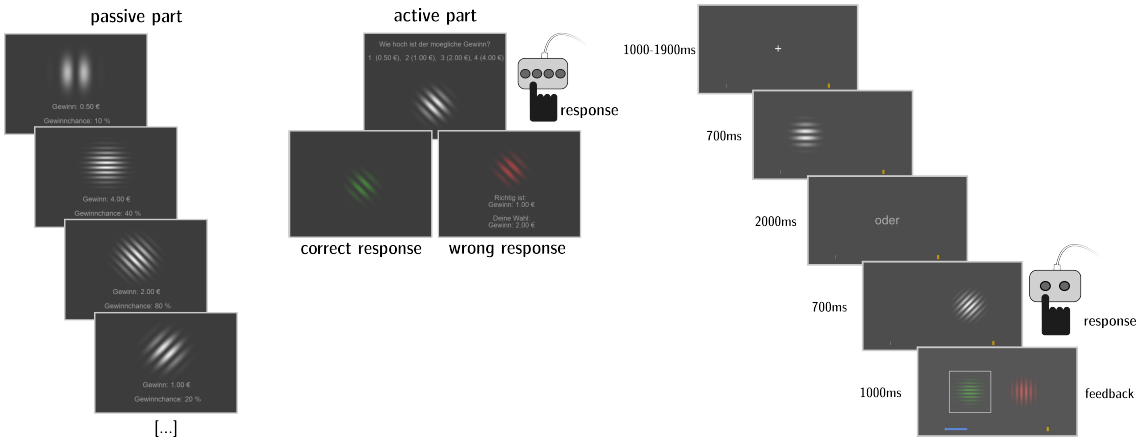


Figure 4.2: Stimulus overview: Gabor patches varied in the number of stripes and their angle, encoding magnitude and probability, respectively. While all presented stimuli were learned in the tutorial, red letters indicate the selection of nine stimulus types used for the left option in the actual experiment. Trials without this annotation were only used intermittent as the right stimulus option to balance the overall expected value between the left and right stimulus option over the course of the experiment.

5. The expected values of the left and right stimulus were balanced over the course of the experiment such that left and right options yielded the same reward on average.

Based on these requirements, a fixed reward schedule was generated, and all subjects underwent the same sequence of stimulus combinations. In approximately half of the trials one of the two stimulus options was clearly favorable, and both its properties (magnitude and probability) were either equal to or greater than that of the other option. These trials consequently required no integration of stimulus properties to assess which option's value is higher. In a subset of those trials, called "no-brainer" in all following analyses (roughly 10% of all trials), both magnitude and probability were strictly greater in one of the two options, posing the easiest decision form. In standard trials, however, stimulus properties were differentially advantageous for the two options.
Each trial (Figure 4.3b) started with a fixation cross (1000-1900ms, jittered), followed by the first stimulus on the left side of the screen (700ms), a 2000ms delay period through which a central "or" was presented, the second stimulus option on the right side (700ms), and a feedback screen. Once the second stimulus option was displayed, participants chose the left or right option via a button press with the right or left index finger, respectively. The feedback screen showed both options side by side with a frame around the chosen option, and revealed which option(s) had been rewarded via color coding (red: unrewarded, green: rewarded). If a decision was not made within 5 seconds, the trial was aborted and participants saw a message to respond faster. A progress bar at the bottom of the screen tracked gains over time, resulting in a bonus payment whenever it hit a gold target line. Participants were instructed to maximize their gains and to respond as fast as possible. Stimulus presentation was controlled by Psychtoolbox (Kleiner et al., 2007) running on Matlab 2012b (The Mathworks Company, Natick, MA). All stimuli were presented on a grey background with a contrast optimized for the MEG recording chamber. In total, the experiment lasted approximately 60 minutes.



(a) Schematic overview of the tutorial. **(b)** Schematic overview of a single trial.

Figure 4.3: Schematic overview of the tutorial (a) and a single trial in the experiment (b). The tutorial was split in an active and a passive part. In the passive part, participants were presented with a stimulus and how its properties translated to reward magnitude and probability. All possible magnitude and probability combinations were presented twice, and participants controlled the pace. The active part then tested participants' knowledge by presenting a stimulus without the annotation. Based on a text prompt, participants had to report its associated magnitude or probability via button press. They received feedback with a green colored stimulus for a correct response, or, for an incorrect response, a red colored stimulus together with a report of their response versus the true response.

4.3.3 MEG acquisition

Following a metal test, MEG data were acquired on an Elekta Neuromag TRIUX System with internal helium recycler and 306 sensors (204 planar gradiometers and 102 magnetometers) in a magnetically shielded room. Participants were instructed to take a comfortable seating position and sit as still as possible. The experiment was presented on a screen in a distance of one meter from the sitting participants via a projector with a refresh rate of 60 Hz located outside the MEG recording chamber. Additional sensors captured confounding biological signals: Lateral and vertical eye movements and blinks were captured using electrooculography (EOG) surface electrodes on the tori supra- and infraorbitalis and next to the external canthi. Heartbeat artifacts were recorded with an electrocardiogram (ECG). Head position indicator coils captured the participants' head movements. Behavioral responses were registered using an MEG compatible keyboard. The neural data was recorded at a sampling rate of 1000Hz and active internal shielding (IAS).

4.4 Data preparation

The code I wrote for data preparation, preprocessing and analysis is available as the Python package `pymento_meg` from GitHub[1] and the Python package index PyPi.org. Underlying software packages are listed in Table 4.1, and the software environment required for all analyses is also available as a Singularity image that can be found alongside the source code.

Software	Version	Citation
`autoreject`	0.4.2	Jas, Engemann, et al. (2017)
`brainiak`	custom fork based on v.0.11	Brainiak contributors (2016)
`imbalanced-learn`	0.10.1	Lemaitre et al. (2017)
`matplotlib`	3.7.1	Hunter (2007)
`meegkit`	0.1.3	Barascud et al. (2022)
`mne-bids`	0.12	Appelhoff et al. (2019)
`mne-python`	1.4.0	Gramfort et al. (2013)
`pandas`	2.0.2	The pandas development team (n.d.)
`scikit-learn`	1.0	Pedregosa et al. (2011)
`scipy`	1.10.1	Virtanen et al. (2020)
`seaborn`	0.12.2	Waskom (2021)

Table 4.1: Overview of internally employed software packages, their versions, and citations.

Prior to preprocessing, raw data were restructured to BIDS format (v1.4.0) following the MEG extension of BIDS (Niso, Gorgolewski, et al., 2018). Several properties of the project made this challenging. As the acquisition was done in a different institution several years back by Kaiser, Gründler, and Jocham (2016) and the data had moved locations, crucial metadata was missing from the project files, either because it was lost or not recorded in the first place. Some information, such as metadata files from the acquisition machine, was acquired post-hoc by emailing the facility in Magdeburg. Other information, such as some participants' ages or raw data from the handedness acquisitions could not be found. Furthermore, as MEG data was acquired on an Elekta Neuromag machine, previous "raw" data former analyses were based on was preprocessed in-scanner with Neuromag's propietary MaxFilter™ for motion correction and denoising. Data that is preprocessed with such a proprietary tool is not an ideal data analysis basis as it can not be easily recomputed without access to the original acquisition machine, which would make a fundamental processing step nontransparent. However, as only preprocessed files were used previously, it went unnoticed that one of the pristine raw MEG data files was not copied over from the acquisition machine to the project archive. With the help of the original authors and a contact at the acquisition facility in Magdeburg it was possible to restore this file from internal archives. Finally, as the experiment was Matlab-based, behavioral log files were written to proprietary .mat files. During the transformation to BIDS, these log files were transformed into the open TSV format (see Figure 1.3).

[1] https://github.com/adswa/pymento_meg

4.5 Preprocessing

An overview of preprocessing steps is shown in Figure 4.4. The following paragraphs outline individual steps in detail.

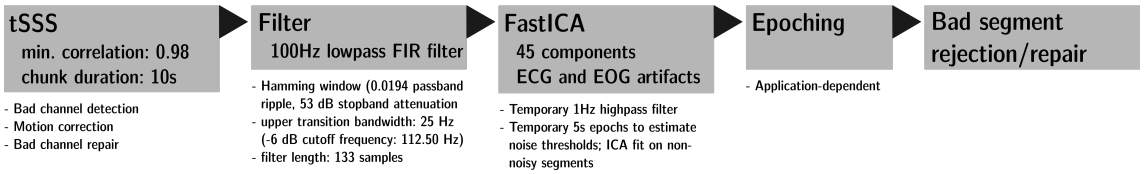


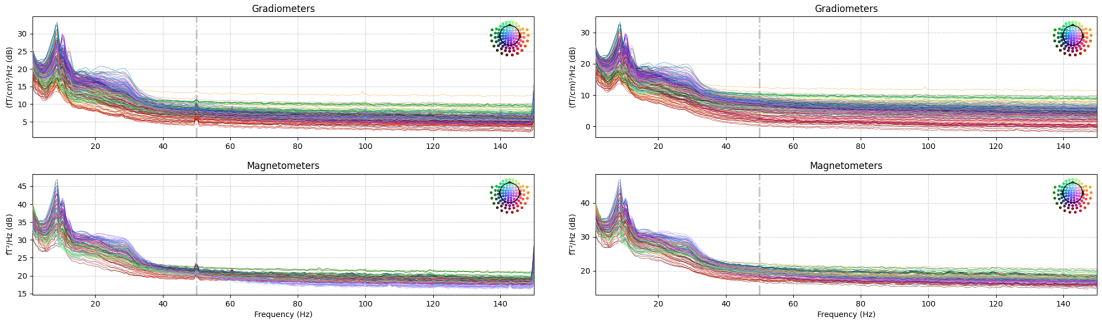
Figure 4.4: Overview of preprocessing steps and their details.

Signal Space Separation As the first step of preprocessing, the spatiotemporal extension of the Signal Space Separation (SSS) method (Taulu and Kajola, 2005), *spatiotemporal Signal Space Separation (tSSS)* (Taulu and Simola, 2006), was applied, as is common for recordings on Neuromag MEG systems with active internal shielding. SSS and tSSS remove strong interference from external noise sources and sources within the body itself from the MEG signal. The methods can be applied on whole-scalp multichannel data when the precise sensor calibrations are known, and were initially developed as the proprietary MaxFilter^{TM} algorithm by Elekta Neuromag. To this end, Neuromag systems provide a cross-talk compensation and fine calibration file which reduces interference between their co-located magnetometer and paired gradiometer sensor units and encodes site-specific information about sensor orientation and calibration, respectively. Based on the Maxwell equations and the geometry of the sensor array, SSS decomposes MEG signals (ϕ) into elementary magnetic fields from sources within the sensor helmet (the *internal* subspace with the signal of interest) and into an orthogonal set for fields arising from sources outside (the *external* subspace with interference sources) (Taulu and Kajola, 2005). To this end, it transforms the $N = 306$-dimensional signal vector into a lower-dimensional, linearly independent subspace that spans all measurable signals, the "SSS basis" S. Its dimensionality is dependent on two user-defined truncation values L_{in} and L_{out}, which correspond to the highest possible frequencies in the internal and external subspace. For Elekta systems with $N = 306$ channels, optimal values are $L_{in} = 8$ and $L_{out} = 3$ (Taulu, Simola, and Kajola, 2005), resulting in $n = ((L_{in}+1)^2-1)+((L_{out}+1)^2-1) = 95$ dimensions (80 internal, 15 external) (Taulu and Kajola, 2005). With $n \ll N$, ϕ can be uniquely decomposed into internal and external subspaces S_{in} and S_{out}, which contain the biomagnetic signal and arbitrary external interference, respectively.

$$\phi = S_x = [S_{in} S_{out}] \begin{bmatrix} x_{in} \\ x_{out} \end{bmatrix} = \phi_{in} + \phi_{out} \tag{4.1}$$

As the internal and external subspaces are provably linearly independent, brain signals are then reconstructed by retaining only sources inside the helmet, thus excluding external interferences.

$$\begin{aligned} \hat{x} &= \begin{bmatrix} \hat{x}_{in} \\ \hat{x}_{out} \end{bmatrix} = S^{\dagger}\phi \\ \hat{\phi}_{in} &= S_{in}\hat{x}_{in} \end{aligned} \tag{4.2}$$



(a) Power spectral density before ZAPline filtering **(b)** Power spectral density after ZAPline filtering

Figure 4.5: Power spectral density of all MEG channels from a single subject before (4.5a) and after (4.5b) ZAPLine filtering. Two spikes at 50Hz (power-line frequency) and 60Hz (likely an artifact of the stimulus presentation) are markedly reduced afterwards.

According to Taulu and Simola (2006), SSS can separate brain signals from sources >0.5m away, suppressing external interference by a factor >100. The spatiotemporal extension tSSS can further detect inferences from closer sources such as stimulators or pacemakers. These are estimated based on the fact that their strength typically exceeds that of sensor noise and they thus, unlike brain signal, leak into both the internal and external part of the SSS reconstruction. After detecting components with a high temporal correlation between the external and internal subspaces, tSSS removes close-by artifacts by projecting the components common to the internal and external subspace out of the internal subspace. In the absence of nearby artifacts, tSSS reduces to SSS (Taulu and Hari, 2009). tSSS was implemented using `mne-python`'s open source implementation `maxwell_filter()` with a chunk duration of 10 seconds and a correlation threshold of at least 0.98. Prior to tSSS, bad channels were detected and annotated automatically in order to prevent bad channel noise from spreading. To compensate for head movements, measurements from the head position indicator coils were used to estimate subject motion, and motion correction was then performed as part of the tSSS procedure: As the signal representation in the SSS basis is device independent, internal data can simply be transformed to a sensor array corresponding to the average head position. Similarly, formerly bad channels have also been effectively repaired by the procedure. After tSSS, gradiometers and magnetometers contain highly similar information and have an altered inter-channel correlation structure because they were reconstructed from a common 80-dimensional subspace (Garcés et al., 2017).

ZAPLine filtering After tSSS, visual inspection revealed that minor spectral artifacts remained in the data, among them power line noise at 50Hz and a spectral peak at 60Hz, likely originating from the presentation screen's refresh rate. ZAPline filtering (de Cheveigné, 2020) was performed to remove them. Figures 4.5a and 4.5b show power spectral density plots of the signal before and after applying ZAPline filters.

Filtering Next, data were first low-pass filtered with a 100Hz lowpass FIR filter to constrain it into a frequency range of interest. The filter properties are reported in Figure 4.4 and a visualization is depicted in Figure 4.6.

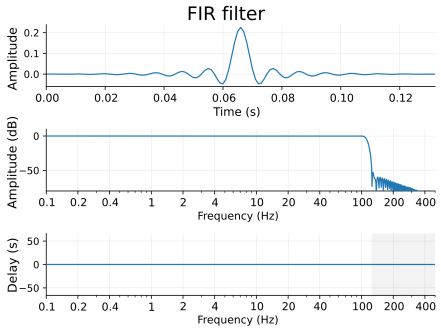


Figure 4.6: Properties of the 100Hz Low-pass filter applied to the data.

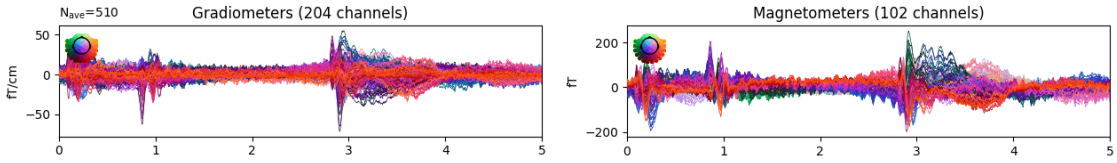


Figure 4.7: Neural recordings from a single exemplary subject. Shown is the average of cleaned epochs with a length of 5 seconds from left stimulus onset, corresponding to a trial from first visual stimulation until 1600ms after the offset of the second stimulus.

Independent component analysis As eye movements, eye blinks, heart beats, and facial muscle contractions survive tSSS, independent component analysis (ICA) was used to detect and remove further artifacts. As ICA is sensitive to low-frequency drifts (Winkler et al., 2015), the data were first processed with a temporary 1Hz highpass filter. ICA can further be sensitive to bad segments in the recording. Therefore, data were temporarily epoched into 5 second splits from the onset of the fixation cross. `autoreject` (Jas, Engemann, et al., 2017), an algorithm to reject and repair bad trials as well as to estimate bad sensors on a per-trial basis, was then used on the first 200 of these epochs to estimate the noise level and compute rejection thresholds. Afterwards, FastICA (Hyvarinen, 1999) was used to decompose the signal of all epochs below these rejection thresholds into 45 independent components. For most subjects, components corresponding to ECG activity were identified using cross-trial phase statistics (Dammers et al., 2008) with an automatically computed threshold of 0.16 for the Kuiper statistic, and EOG related components were found using Pearson correlation. For one subject, however, the ECG channel was flat, and components were manually selected. Afterwards, the ICA solution was applied to the continuous recording and detected ECG and EOG components were zeroed out.

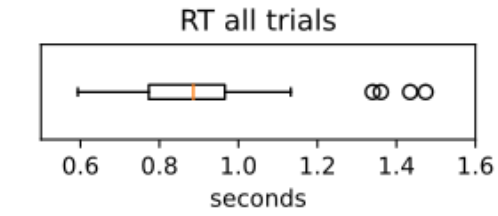
Bad segment rejection and repair As a final step, the continous recording was chunked into epochs of varying length depending on analysis, and `autoreject` was used to detect and repair or drop bad epochs. Figure 4.7 shows the average of cleaned, 5 second epochs starting at the presentation of the first stimulus option for a single subject.

4.6 Analysis prerequisites

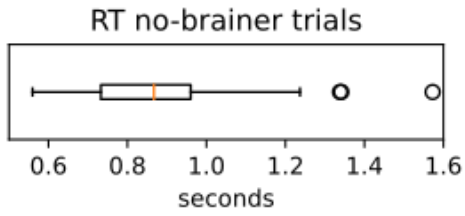
Many of the upcoming analyses are rooted in machine-learning methodology. This section introduces relevant concepts, and summarizes common elements across analysis. I set up the machine-learning based analyses either purely in the standard library *scikit-learn* (4.7, behavioral analysis), using functionality from *mne-python* that builds up on scikit-learn for MEEG applications (4.10, temporal generalization analysis), or by implementing scikit-learn compatible custom estimators and scorers myself (4.9, decoding analyses).
Even if upcoming analyses extend scikit-learn's capabilities, the framework's terminology remains central to describe the analysis setup in depth: The behavioral and decoding analyses I conducted are classification problems, belonging to the family of supervised learning. In these analyses, the goal is to predict a discrete target variable y from an input data matrix X with n observations, for example, if a participant performs a left or right response (target) based on the stimulus properties (features) in each trial. In scikit-learn's terms, an *estimator* is what performs this prediction. A *scorer* is a method that evaluates an *estimator*'s predictions on a given dataset, such as computing the accuracy from predicted and actual target labels. A *transformer* is an object that alters the input data, for example by cleaning, reducing, or expanding it, or by generating features from it. In supervised learning, data an *estimator* is trained on are labeled with the actual value of target variable y, but this data can not be reused to test the trained estimators' performance. Fundamental to evaluating machine-learning applications are thus *train-test* data splits. These partition the available data into a set used to *train* the *estimator*, and a set yet unknown to the *estimator*, used to *test* how well its predictions generalize to unseen data. The accuracy of such a model is calculated from the mismatch between predictions and actual labels in the test set, which is evaluated with a *scorer*. I set this up within a k-fold cross-validation framework, which repeats model fitting and predictions with *test* and *train* splits over k different partitions of data such that each partition k is used as a test set once. k was usually set to a value between 5 and 10 depending on the analysis. Compared to other common forms of cross-validation in neuroimaging such as leave-one-out cross-validation, this balances the trade-off between the need for sufficient training data to reach a good fit, and the need for sufficient testing data to decrease the variance of estimated accuracy for stable estimates (Varoquaux et al., 2017). The final model evaluation is based on an average of the prediction performance in each fold, reducing the variance of the model performance estimate. In classification problems with an unequal amount of target classes I used a stratified k-fold cross-validation paradigm, which creates splits such that the relative distribution of class labels in its split matches that in the full dataset approximately. If testing data leaks into the training data, performance estimates get overly optimistic as the model is partially trained on the data it will be evaluated on. To avoid this common pitfall, all machine-learning analyses were set up as a so-called *pipelines*, sequences of *transformers* or *estimators* that chain processing steps while internally ensuring that *test* and *train* data are kept strictly separate. Additionally, *pipelines* ensure consistent preprocessing by applying *transformers* to *training* and *test* data. A *transformer* common to *pipelines* in all analyses was a `StandardScaler()`, transforming data to have zero mean and unit variance. The final *estimator* common to all *pipelines* was a logistic regression classifier with L2 regularization and a `liblinear` solver. This was chosen to match the choice of classifier by Kaiser, Gruendler, et al. (2018). Internally, the `liblinear` solver uses a coordinate descent (CD) algorithm, which extends binary classifications problems to a multinomial case by decomposing the optimization problem into several one-vs-rest problems such that multi-target classifications become possible (scikit-learn, 2023).

4.7 Behavioral analysis

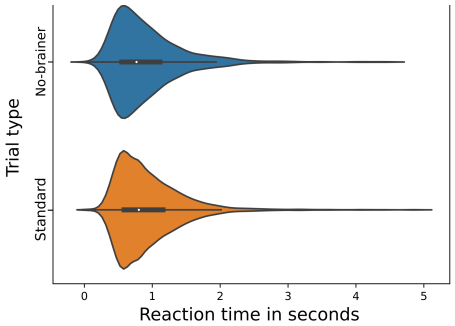
As a first step, I conducted several analyses on behavioral data to check if participants' behavior matched the task demands and to inform further processing. For the former, I conducted analyses regarding participants' performance. One measure to evaluate performance in the experiment is the gain that participants achieved at the end of the last trial. It is the sum of rewards accumulated over all trials, and should be higher the better participants judged stimulus values. On average, participants gained $G = 349.5$ points in the experiment (range = $[300.5, 393]$; top boxplot in Figure 4.9a). To assess whether this exceeds chance level performance, I conducted a simulation of $N = 10\,000$ experiments with random choice behavior (bottom boxplot in Figure 4.9a), confirming that all participants performed significantly better than chance at the 95% confidence level (dotted line in Figure 4.9a). A different measure of performance is reaction time. The average reaction time pooled across participants was $\mu_{RT} = 0.93$s. Contrary to the expectation that no-brainer trials would pose easier decisions, the difference in average reaction times for no-brainer trials compared to standard trials was negligent ($\mu_{RTnobrainer} = 0.91$s compared to $\mu_{RTstandard} = 0.94$s; Figure 4.8c). An inspection of average reaction times per subject revealed a small subset of slow participants with average reaction times exceeding 1.3s (Figures 4.8a, 4.8b).



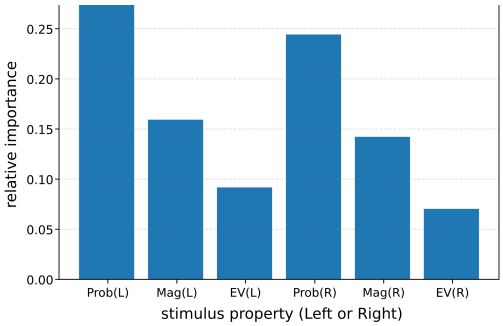
(a) Average reaction time (subject-wise)



(b) Average no-brainer reaction time (subject-wise)



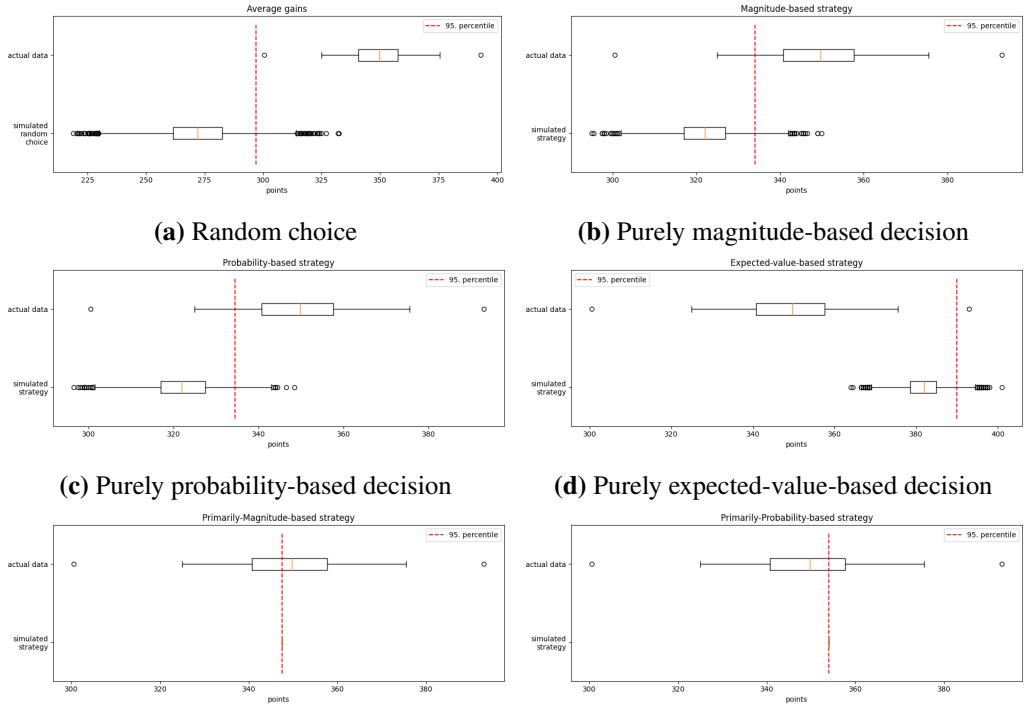
(c) Distribution of reaction times



(d) Parameter influence on choice

Figure 4.8: Results from the behavioral analysis. a) and b) show average reaction times over trials for all subjects. c) depicts individual reaction times across trials and subjects for standard trials (orange) and trials an no-brainer trials. d) shows the relative influence of stimulus properties on choice (left option) as computed across a 10-fold cross-validated logistic regression.

To inform further processing, I investigated which stimulus properties influenced participants' behavior. The stimulus properties relevant for the eventual behavioral choice in each trial were reward magnitude (stripe frequency), reward probability (stripe orientation), and their combination (expected value). I assessed their relative influence on choice by two means: First, by simulating different strategies by which participants could use these properties, and comparing them against the actual gains. I created five different strategies: Pure probability- or magnitude-based strategies (choice fell on which ever stimulus had a higher value, and reduced to random choice where values were equal), primarily probability- or magnitude-based strategies (choice fell on which ever stimulus had a higher value, and in cases where values were equal, it fell on which ever stimulus had a higher value on the other property, akin to a sequential evaluation process), and an expected-value-based strategy (choice fell on the stimulus with the higher expected value, and reduced to random choice where expected values were equal). These simulations revealed that strategies based solely on either magnitude or probability yield outcomes that are predominantly worse than participants' actual gains, bearing a single outlier. The strategy following expected value was the best, yielding predominantly higher gains than participants' actual gains, bearing one outlier whose performance exceeded even that of 95% of simulated experiments with the expected value strategy. The two sequential strategies yielded a single, deterministic outcome that was close to the group average.
The second mean to assess which stimulus properties influenced choice was a stratified 10-fold cross-validated logistic regression analysis of left and right stimulus characteristics on choice for each subject. The model included six predictors (magnitude and probability of the left and right option, and expected value calculated from demeaned magnitude and demeaned probability) and an intercept. The analysis was constructed as a classification analysis following the description in Section 4.6. Thus, the model estimated beta coefficients on a training set, and tested their predictive accuracy in a test set. For each subject, I calculated the average accuracy over folds as a measure of model quality, and normalized beta coefficients within range $[0, 1]$ to assess their relative influence on choice behavior (Figure 4.8d). The average model accuracy was 0.83 (std = 0.06, range = $[0.68, 0.93]$). The Spearman rank correlation between experiment performance (total gain) and model accuracy was $r = 0.53$ ($p = 0.012$), indicating that well-detectable influence of stimulus properties on choice was associated with higher gains in the experiment. Across participants, reward probability had the highest relative influence on choice behavior, followed by reward magnitude, and the influence of left stimulus properties was slightly higher than that of right stimulus properties. On the level of individual subjects this pattern held in all but one participant for whom magnitude had a higher influence than probability, and right stimulus probabilities had a higher influence than left stimulus probabilities.
Overall, these behavioral results confirm that participants acted according to task demands. They further provide evidence that a majority of participants acted according to the strategy modeling decision as a sequential evaluation of probability and then magnitude (Figure 4.9f). The counter-intuitive finding that participants' reaction time was not reduced in trials that did not require integration across stimulus properties could suggests that participants did not integrate stimulus properties in any trial type, but used the same strategy in no-brainer and standard trials.



(a) Random choice

(b) Purely magnitude-based decision

(c) Purely probability-based decision

(d) Purely expected-value-based decision

(e) Primarily magnitude-based decision

(f) Primarily probability-based decision

Figure 4.9: Experiment gains compared to different behavioral strategies. Participants' actual gains in the experiment are always plotted in the top boxplot. The bottom boxplot is a simulation of the experiment (N = 10 000) with different behavioral strategies. The red dotted line denotes the 95. percentile in the simulation data to allow for statistical comparisons at the 95% confidence level. a) depicts simulated random choice behavior. Even the outlier at the bottom end of the actual results exceeds the results of more than 95% of simulated experiments, indicating that all participants performed better than chance. b), c) and d) depict strategies that are purely based on higher magnitude, probability, or expected value, respectively, and reduce to random choice if the feature is equal across options. e) and f) depict deterministic decision strategies, based on sequentially evaluating first magnitude and then probability, or vice versa. As no random element is involved, the average gain is identical over experiment repetitions.

4.8 Shared response modeling

As outlined in section 4.5, tSSS decomposes the neural signal from the 306 dimensional sensor space into an 80-dimensional subspace, the SSS basis. The signal of interest in MEG data can thus be expressed in fewer dimensions than the D = 306 channels of the MEG machine. A growing body of literature finds lower-dimensional structure in high-dimensional datasets. In the following analyses, I therefore sought to find out whether I can find meaningful low-dimensional structure in the high-dimensional dataset by means of the shared response model (SRM). SRM (Chen et al., 2015) is a method that combines functional alignment, a family of methods prominently established in fMRI research to align functional topographies instead of inter-individually distinct anatomical

topographies (Haxby, Guntupalli, Connolly, et al., 2011), and dimensionality reduction. A central underlying assumption of functional alignment methods can be summarized as follows: When subjects experience the same events, their brain activity patterns may differ anatomically, but they should correspond to similar cognitive processes. SRM was proposed as a method to functionally align the neural responses of several participants as measured with fMRI. For this, it models each subject i's response to temporally synchronized stimuli $X_i \in \mathbb{R}^{s \times t}$ as an orthogonal subject-specific base $W_i \in \mathbb{R}^{s \times k}$ and a k-dimensional shared response $S \in \mathbb{R}^{k \times t}$ such that

$$\begin{aligned} & min_{w_i,s} \sum_i \|X_i - W_i S\|_F^2 \\ & s.t. W_i^T W_i = I_k \end{aligned} \tag{4.3}$$

where t is the number of time points, s is the number of sensors, k is a hyper-parameter denoting dimensionality, and $\|\cdot\|_F$ denotes the Frobenius norm. The orthonormal constraint $W_i^T W_i = I_k$ yields the computational advantages of robustness and preserving temporal geometry (Chen et al., 2015). Unlike, for example, PCA, the reduced dimensionality does not only emerge from the data of a single brain, but taking the activation patterns of several participants into account. It is a latent factor model, and rather than searching for direct correspondences across subjects, an initial dimensionality reduction is used to identify latent factors that are shared across subjects and support the observed measurements. One latent feature k is a functional topography, and SRM models the neural signal as a linear combination of subject-specific functional topographies (see Figure 4.10). SRM thus identifies common activity patterns across neural responses, and provides a method to transform the original high-dimensional signal into a lower dimensional shared latent component space. For the MEG dataset at hand, the dimensionality is transformed as follows: Where the original data in sensor space is a 306 sensors × number of samples matrix, the shared response space is a k features × number of samples matrix, and the subject specific weights that map between the shared space and each subject's idiosyncratic sensor space is an orthogonal matrix of dimensionality 306 sensors × k features. All upcoming analyses used the probabilistic SRM implementation of the `brainiak` library (Brainiak contributors, 2016).

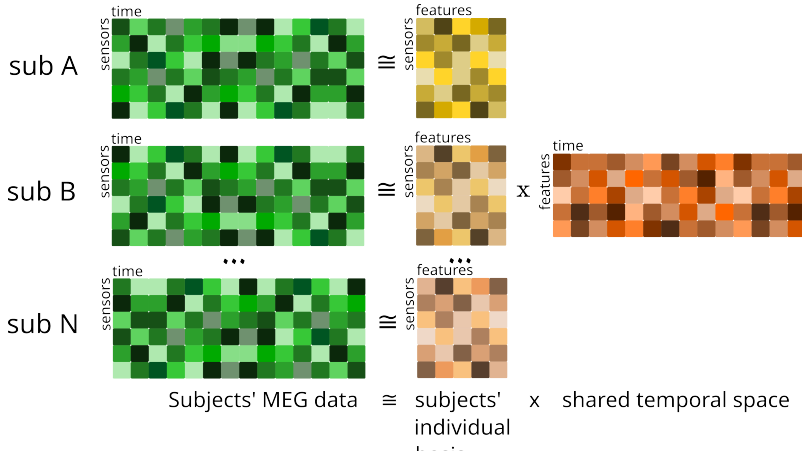
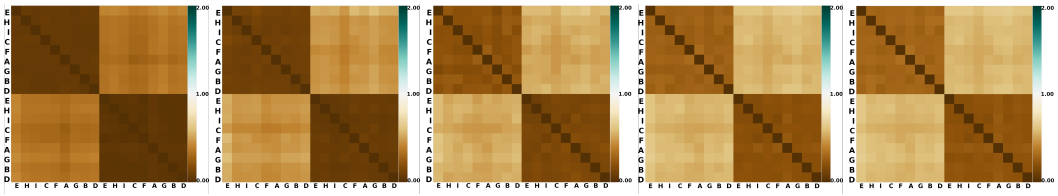


Figure 4.10: SRM overview: Sensor-by-time matrices are decomposed into a orthogonal subject specific basis matrix (sensor-by-features), a common shared feature matrix (features-by-time), and a subject-specific error.

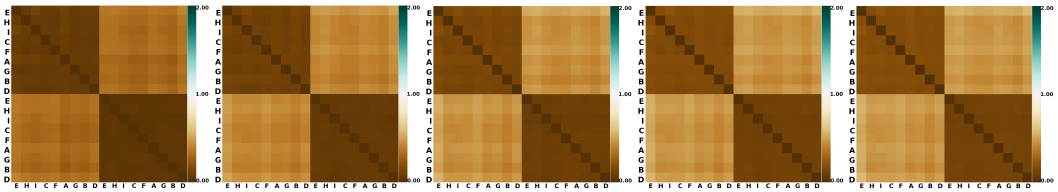
After an SRM is fit, several procedures become possible. Mapping one participant's data to a different participant's functional topography, denoising neural data by transforming the shared response via the subject-specific basis back to the individual participant's original topography, or inferring idiosyncratic signal components in individual participants' data by subtracting the shared response are applications that have been successfully demonstrated, albeit usually with fMRI (Bazeille et al., 2021; Chen et al., 2015; Kalyani et al., 2023). Most commonly, the fMRI acquisitions used for SRM and other functional alignment methods stem from feature-rich, naturalistic stimulation such as movie watching tasks (Haxby, Guntupalli, Nastase, and Feilong, 2020). This has the advantage that it generates a variety of cortical patterns with which these models can better generalize to unseen stimuli or novel tasks. While the experimental paradigm at hand likely generates fewer cognitive patterns than naturalistic protocols, the fact that it is measured with MEG opens up flexible opportunities for data analysis. One major advantage is the high temporal frequency of MEG data. Typical recommendations of minimal acquisition lengths are in the range of 5 to 15 minutes for fMRI data (Häusler, 2023) due to its long repetition time of 1-2s. At a sampling frequency of 1kHz, MEG acquires the same amount of samples in a fraction of recording time. The amount of samples measured in a single MEG experiment's trial can contain as many samples as an entire fMRI experiment. This opens up opportunities to train an SRM based on trial time courses of a few seconds.



(a) $k = 5$ **(b)** $k = 10$ **(c)** $k = 20$ **(d)** $k = 40$ **(e)** $k = 80$

Figure 4.11: Shared spaces of stimulus presentation, visualized as the correlation distance between the trajectories of functional topographies k in the shared space between stimulus types. Darker brown values denote stronger relationships, lighter values denote independence, and darker blue values would denote anticorrelation. Stimulus types are split into nine left and nine right options, sorted by increasing probability.

One possible angle to analyze the experiment data with the aim of finding lower-dimensional structure is to create a shared response room with shared activity over specific parts of a trial. A comparison of the resulting trajectories of latent components in the lower-dimensional shared space could constitute a mean to differentiate neural representations between different trial types. Pursuing this line of thought, I conducted an analysis to investigate whether the latent functional topographies of different stimulus types differ. Given that the nine main magnitude and probability combinations (letters A-H in Figure 4.2) contained information to inform participants' decision process and thus should be maintained throughout the delay, I hypothesized that lower-dimensional representations might relate to these properties, for example as an abstract representation of the size of a potential reward magnitude. To investigate this, I created training data as follows: Within each subject, I shuffled epochs to forgo sequence effects and split them into a training and a test set, balanced with regard to the number of stimulus types, and z-scored the data within sensors for each epoch. I then aggregated epochs in which the same stimulus was shown into average time series, separately for the left and right stimulus presentation. Importantly, these time series were

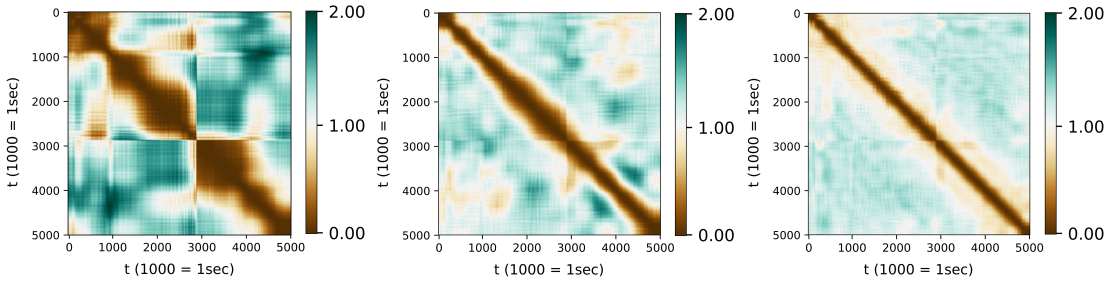


(a) $k = 5$ **(b)** $k = 10$ **(c)** $k = 20$ **(d)** $k = 40$ **(e)** $k = 80$

Figure 4.12: Similarity of time courses from different stimulus options in shared space. Test data is transformed into the shared space using subject-specific basis of the SRM, averaged across subjects. As correlation distances do not follow a normal distribution, individual distance matrices were Fisher-Z-transformed prior to averaging and transformed back to correlation distance afterwards.

limited to the 700ms of visual presentation of the option. This confined the shared response model to find shared functional topographies in each options visual presentation, which in turn would allow to compare the shared space of each different stimulus. With nine common stimuli as the first and second option, respectively, this generated 18 different averaged time series per subject in training and test data. After fitting the shared response model, I retrieved the subject-specific base matrices to transform the unseen test data into the shared space. These spaces constitute a $k \times 700$ samples matrix, i.e., k trajectories over a time span of 700ms, for each stimulus type. To uncover differentially similar functional topographies across trial types, I then computed the correlation distance between each trial type's shared components, i.e., the similarity between the shared response during one type of trial to all other trial types, for both the left and the right stimulus option. Correlation distance can take values in range $[0, 2]$, where 0 indicates perfect positive correlation between vectors, 1 indicates independence, and 2 indicates perfect negative correlation. As I had no prior assumptions about the dimensionality of the shared space, I repeated the analysis for different values of k. Figure 4.11 visualizes the resulting shared spaces for values of k between 5 and 80. Figure 4.12 visualizes test data transformed into this shared space across subjects. The distance matrix is split into left and right options, which are in turn ordered by increasing probability. The first two rows and columns in the matrix thus correspond to options with 10% probability, the next three to options with 20% probability, and so forth. Instead of revealing a grouping of stimulus types by characteristics such as their probability or magnitude, however, the most marked differences lay between the sides on which the stimulus was presented. The functional topographies in participant's neural activity were more similar for different stimuli presented on the left than the same stimulus presented on the left and right side, and the more fine-grained distances between stimulus types did not follow a pattern determined by the values of the reward magnitude or probability.

One potential reason for this is that the potentially subtle differences between stimulus types were not prominent enough to be reflected strongly in the shared spaces, and signals corresponding to more general cognition, such as generic visual processing, dominated instead. To investigate whether features corresponding to the trial structure in general are preserved during SRM, I used the shared response space generated over the entire trial course to visualize the time point by time point similarity of the trial in shared space. This analysis mirrored the previous approach, but epochs were not split by stimulus type and pooled instead, and the correlation distance was not calculated between time series, but between k features per time point. The result is visualized



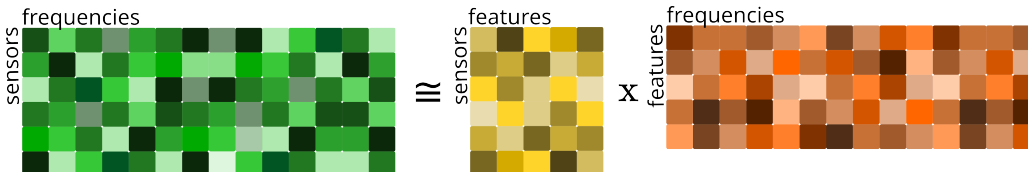
(a) k=5 **(b)** k=10 **(c)** k=20

Figure 4.13: Time point by time point correlation distances in shared spaces built from the entire trial, across epochs with different stimulus types, for different values of k. Notable events in the trial, such as the stimulus presentation (0-700ms, 2700-3400ms) or the delay phase (700-2700ms) emerge as patterns.

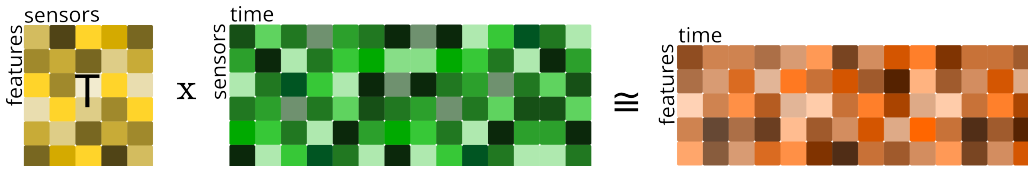
for three different values of k in Figure 4.13. The trial structure, in particular marked differences between the first stimulus, the delay phase, and the second stimulus, are visible for shared spaces of all dimensionalities. With growing dimensionality k, similarities and dissimilarities decrease, which can be expected given that larger dimensionality values extend the vectors between which correlation distance is calculated, and can also be observed with increasing dimensionality in the previous shared spaces. While individual stimulus-type differences did not emerge in the shared space, the SRM was nevertheless able to capture trial-course induced differences.

4.8.1 Shared response modeling in spectral space

In further analyses, I investigated delay and decision periods of the trial. In comparison to the trial phases with visual stimulation, neural activity in these periods should be driven less by external visual stimuli, but rather by internal cognitive processes related to working memory maintenance, motor preparation, or decision making. The distribution of reaction times in the task (see Figure 4.7), however, revealed different processing speeds, evident in intra- and inter-individual variation in response time. Therefore, even if participants showed similar cognitive processes during the trial that SRM could detect, their temporal characteristics likely vary considerably across trials and subjects. Thus, given the explicit requirement of temporal synchronicity for functional alignment and the high temporal resolution of MEG, this could pose a challenge. To use the shared response model nevertheless, I utilized the flexibility with which the SRM method allows transformations between idiosyncratic and shared neural spaces, and trained the model with input data in a space without timing information. For this, I drew inspiration from spectral analysis.
Spectral analysis consists of deconstructing a time domain signal of a given length into its constituent oscillatory components using Fourier analysis. MEG signal consists of oscillations with an amplitude, a frequency, and a phase. Frequency, usually measured in hertz (Hz), is the number of times a specified event occurs within a specified time interval, while amplitude is the height, force or power of the wave, and power is the squared amplitude. Phase involves the relationship between two or more signals that share the same frequency and describes the relationship between the position of the amplitude crests and troughs of two waveforms, measured in distance, time, or degrees. If the peaks of two signals with the same frequency are in exact alignment at the same time, they are said to be in phase and out of phase otherwise. If the exact timing or duration of



(a) Fitting the shared response model on spectral data



(b) Transforming time resolved data into the shared space

Figure 4.14: Spectral shared response modeling: A shared response model decomposes spectral neural data (4.14a, green) into a shared spectral space (features by frequencies; orange) and subject-specific basis (sensors by features; yellow). Because the subject specific bases transform sensor data independent on whether it is time resolved or spectral, they can be used to transform time resolved data (4.14b; green) into time courses of the shared response spaces features. Capital "T" denotes a matrix transpose.

shared activity in question is subject to intra- and inter-individual variation, it is not necessarily phase-locked. However, a spectral decomposition of this signal results in power spectrum, with the frequency domain of the oscillations on the x axis, and the amplitude on the y axis. Typically, these transformations are used in time-frequency analyses to investigate the progression of MEG power in different frequency bands of interest over time (Hari and Puce, 2017). In the upcoming analyses, however, I used them to express signal as a function of frequency rather than time, and used this transformation from a time-resolved representation of data to a representation in the frequency domain to find shared signals with different temporal signatures. Importantly, the model bases of an SRM assign weights to sensors regardless of whether they contribute to a shared signal in time-resolved or spectral space. Therefore, I can fit a shared response model on spectral data to find shared frequency components, but then use the model bases to transform time-resolved data (see Figure 4.14). This not only allows the visualization of shared components as a time series, but also easier interpretation of components in the context of the experimental paradigm. To evaluate the feasibility of this novel method, I first conducted a simulation study.

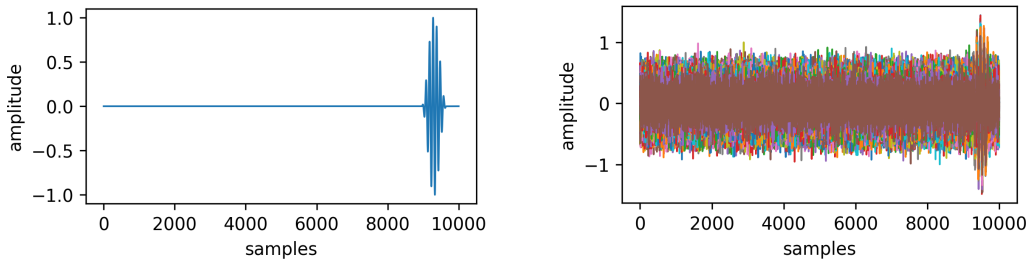
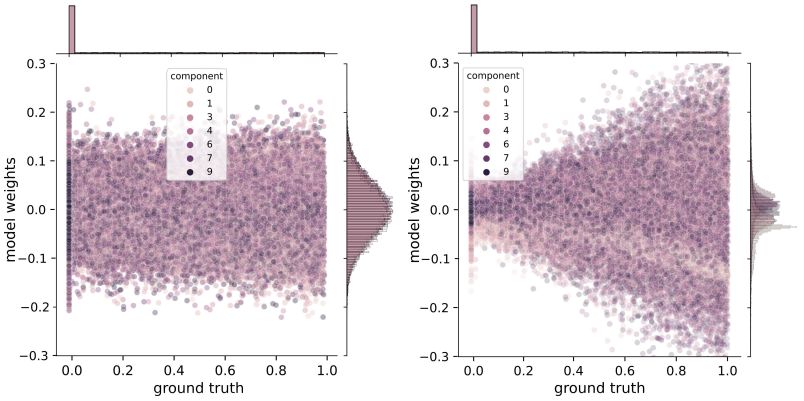


Figure 4.15: Artificial signal (10 000 samples) with a 1000 sample spanning 10Hz main frequency component (left), and partially embedded into 306 sensors with Gaussian noise (right). 20 percent of sensors contain the main signal in varying amounts, mirroring how brain signals are only present in certain sensors and decrease in strength over distance.

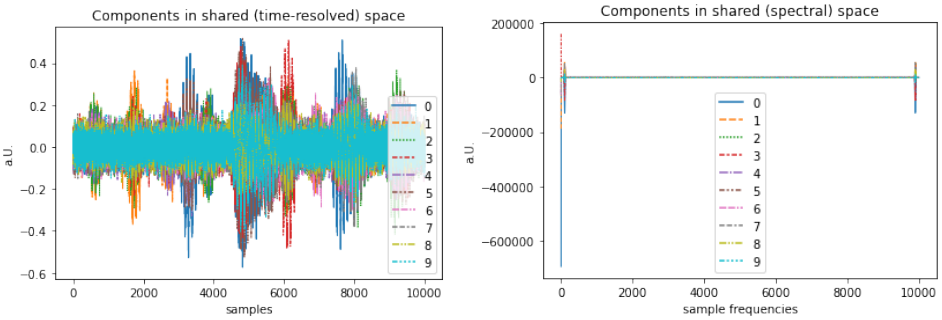
4.8.2 Simulation study

To simulate MEG data, I generated a "ground truth" signal with a given frequency and wave form, and embedded it partially into 306 artificial sensors with Gaussian noise (Figure 4.15). To mirror how brain signals are only present in certain sensors and with varying strengths, only a fixed amount of sensors received the signal. For each sensor with a signal, a random weight is drawn that determines the strength with which the signal is scaled. This is repeated for $N = 30$ artificial subjects, but with a different random phase offset in the signal for each to simulate data from several subjects with a common frequency component that occurs at different points in time. If the SRM is successful, the shared response should reflect the ground truth signal despite phase offsets, and the subject-specific model bases should reflect the magnitudes of the weights for each sensor.
Using this artificial data as the basis for shared response modeling, I fitted a probabilistic SRM with $k = 10$ features to recover the signal as a shared component - either on time resolved data, or after transforming the data into its frequency spectrum.
When fit on time resolved data with phase shifts, the resulting shared space consists of components that differentially picked up signals from one or more participants, but represent it in a time series of repeated or overlapping signals. This makes an interpretation of individual components difficult (Figure 4.17a). The sensor weights that the model estimates for each component also do not show a clear association with the true weights used in the generation of the artificial data (Figure 4.16a). In other words, with phase shifts between individual simulations' signals, different components of the shared space capture several participants' signals, but no *general* ground truth signal. However, transforming the signal into its frequency components, the spectral space, removes timing information, and with it, the phase offsets. While the components in spectral space are not easy to interpret or differentiate (see Figure 4.17b), the scatterplot reveals that certain components' weights show a clear association to the weights used for model generation 4.16b.

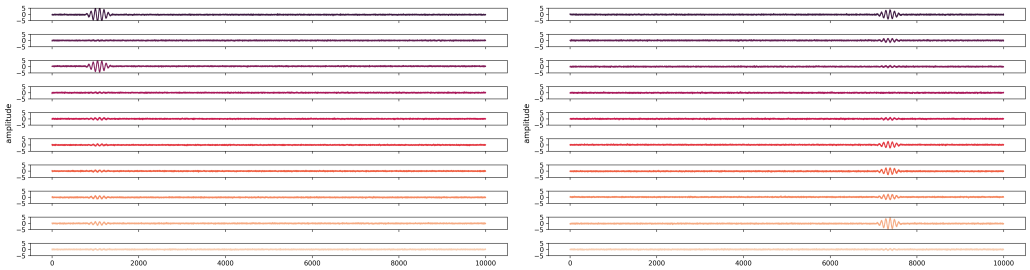


(a) SRM on time-resolved data **(b)** SRM on spectral data

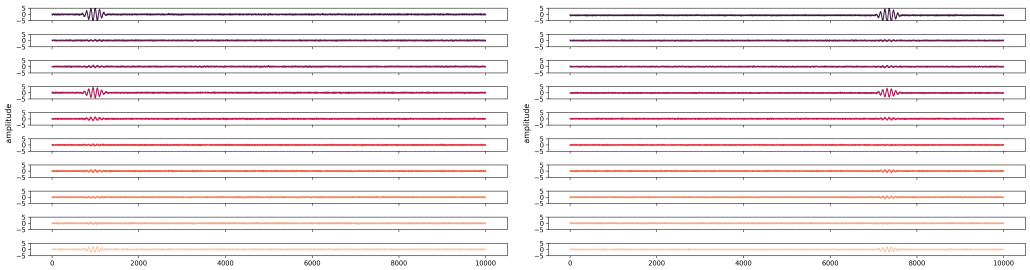
Figure 4.16: Recovered versus true model weights from an SRM fitted on time resolved (left) or spectrally transformed data with phase shifts (right): Without spectral transformation, the relationship between model weights and ground truth weights appears mostly random. When fit on spectrally transformed data, the relationship between model weights and ground truth weights clearly captures an association for some of the $k = 10$ components, indicating successful recovery of the original weights.



(a) Shared space from time-resolved data **(b)** Shared space from spectral data



(c) Shared components from 4.17a (visualized via transformation, for two "subjects")



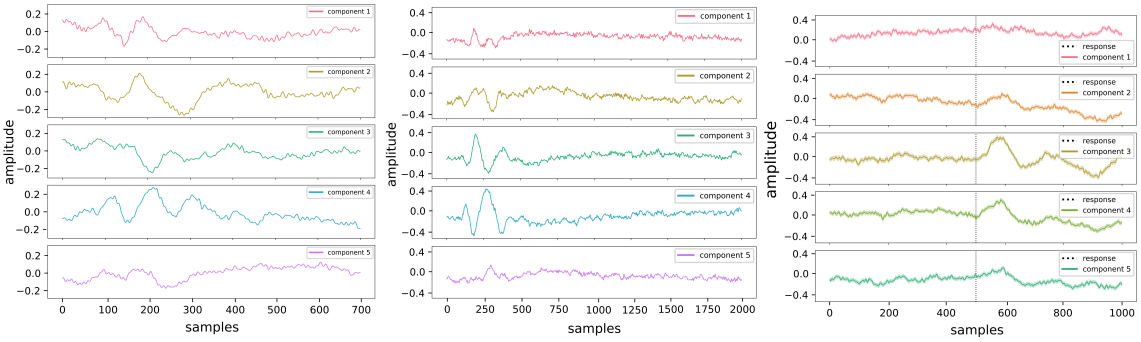
(d) Shared components from 4.17b (visualized via transformation, for two "subjects")

Figure 4.17: Properties of a shared time-resolved or spectral space. a) visualizes the shared space of a model fit on time-resolved data: Gaussian noise is reduced and signal of interest is retained, but signals from several subjects emerge as consecutive or overlapping activity. Using individual subject's weight matrices and raw data to visualize individual model components reveals that the signal of interest is not represented in the same components across two subjects (c). In comparison, b) shows the shared spectral space, and d) its transformed components. The components in shared space are spectral and difficult to interpret. But the strength with which components capture the signal of interest is consistent in the transformations with time-resolved data.

This becomes apparent when shared components are visualized as time series by using the subject-specific weight matrices of the SRM and individual subject's time-resolved data. A few components consistently represent the original signal well across subjects, laying the basis for identifying shared signal in phase-shifted data and interpreting the shared components resulting from it.

4.8.3 Visualizing shared components

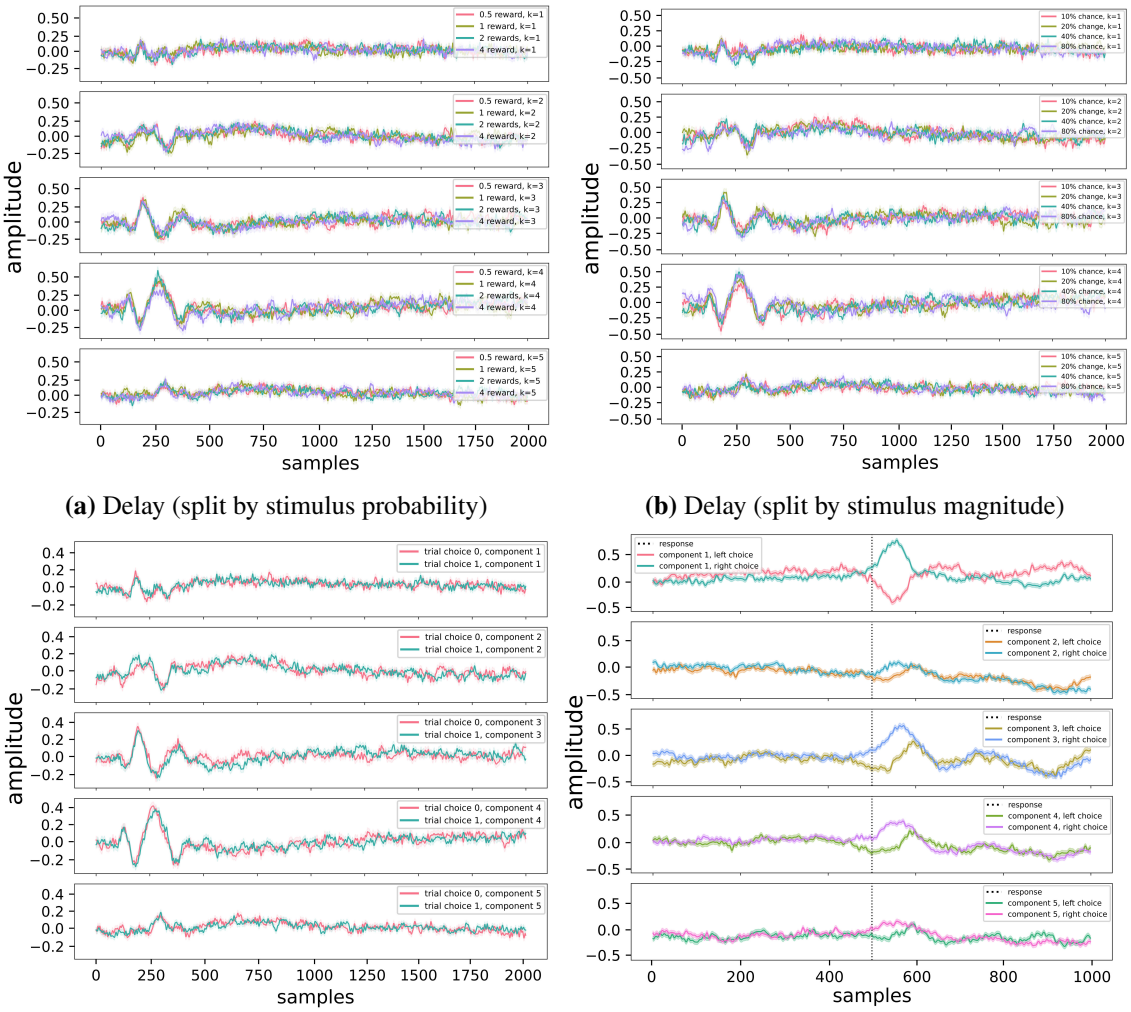
After the simulation study demonstrated feasibility of the method, I subsequently applied it to the neural MEG recordings. The underlying aim was to find shared components that represented decision-related or -relevant properties. Unlike the previous SRM analyses, models were trained with data from all participants. Using the subject-specific bases, the time-resolved test data was then transformed in the shared space while retaining the temporal resolution (4.14b) to visualize the components that the shared response model found. The resulting components are thus time series. I fit different models for different time slices of a trial, namely the first stimulus presentation, the delay phase, and the decision and feedback phase. A subset of these visualizations are shown to illustrate the results for shared response models fit with $k = 5$ in Figure 4.18.



(a) First stimulus presentation **(b)** Delay period **(c)** Decision and feedback

Figure 4.18: Time courses of shared components for different trial phases, as obtained by fitting a SRM with $k = 5$ features on spectrally transformed data. Each panel in each figure results from transforming left-out time-resolved epochs with the subject-specific base matrix for one component in the shared space and averaging the resulting transformations. Thus, each panel visualizes the time course of a functional topography identified with spectral SRM. Depicted are three different trial phases, the first stimulus presentation, the delay phase, and the decision phase. In c), the dashed line indicates the motor response.

One possibility is that the modulation of these functionality topographies encodes different decision-related properties. To investigate whether shared components differentially related to trial-specific attributes, unseen test data was transformed into the shared space, but visualized separately for trials with specific attributes. For example, trials in which the first stimulus option had different probability values were contrasted with each other. I hypothesized that this could uncover shared components encoding stimulus properties, modulated via their amplitude. To investigate this, I reused the models for visualizations of test data in various attribute splits. A subset of these visualizations is shown in Figure 4.19. They revealed that marked modulation differences only existed for splits between decisions, and those were only seen in shared spaces of the decision phase (Figure 4.19d).



(a) Delay (split by stimulus probability)

(b) Delay (split by stimulus magnitude)

(c) First stimulus presentation (split by choice)

(d) Decision (split by choice)

Figure 4.19: Time courses of shared components, split by trial attributes. Shown are splits by the probability (a) and magnitude (b) of the first stimulus option during the delay phase, and splits by left or right choice during the presentation of the first option (c) and decision phase (d). In d), the dashed line indicates the motor response.

4.9 Decoding

As previous analyses were unable to directly identify neural subspaces representing stimulus features, I turned to a multivariate method in the form of a temporal decoding analysis. This method involves fitting a predictive model on each time instant within a trial, and evaluating the model performance at the same instant on new epochs (King, Gramfort, et al., 2018). A temporal decoding analysis for the stimulus properties probability, magnitude, and expected value of the first stimulus option has already been performed by Kaiser, Gruendler, et al. (2018) in source space. However, stimulus properties had above-chance decoding accuracies only during the stimulus presentation and a few hundred milliseconds into the delay period. To investigate if dimensionality reduction methods including SRM improved the decodability of stimulus properties in the delay period, I implemented

a flexible temporal decoding pipeline that was able to employ different forms of dimensionality reduction internally. This analysis would, if successful, provide evidence for a lower dimensional subspace that represented stimulus properties. With this approach, I followed the works of Murray et al. (2017), who was able to decode stable stimulus representations in a lower-dimensional subspace, in their case by dimensionality reduction via PCA.
I prepared the analysis as several independent decoding pipelines, and compared their performance subsequently. Similar to the temporal generalization analysis, computations were first performed on the subject-level, and then aggregated into group-level visualizations. To compare an SRM- or spectral-SRM-based dimensionality reduction approach with alternative methods that are more commonly used in MEG analysis, I also included PCA, which has been shown to improve decoding results (see e.g., Grootswagers et al., 2017) and was the method of choice for Murray et al. (2017). A number of methods commonly employed to improve decoding accuracy such as sliding windows and trial averaging (see also Table 4.2) were included as internal, configurable steps in the pipeline. The inclusion of dimensionality reduction methods into the temporal decoding analysis posed technical implementation challenges because varying requirements for the shape of features X within the pipeline made pre-existing functionality in `mne-python` and `scikit-learn` incompatible. While `mne-python`'s `SlidingEstimator` for temporal decoding handles the three-dimensional shape of MEG data (trials × channels × trial-length) well, it can't incorporate internal transformations via SRM as this method requires two-dimensional features (sensor × samples). Additionally, it limits the available scoring methods to those that have a one-dimensional shape, which prevents scoring with confusion matrices. This is unfortunate as confusion matrices are highly useful in decoding analysis: They allow to visualize prediction errors split by labels and uncover which targets drive successful decoding, while other metrics, including accuracies, can be computed from it easily nevertheless. Scikit-learn, on the other hand, expects two-dimensional data, and its pipelines are not natively able to work with multi trial MEG epochs. Some steps in the pipeline further required conjoint transformations of features and targets, such as trial averaging transformations, which is not supported by `scikit-learn` transformers and pipelines. I solved these issues with a range of custom implementations. I used an `imbalanced-learn` pipeline (Lemaitre et al., 2017), which enabled me to write a custom function sampler to transform features X and targets y jointly within a `scikit-learn`-compatible pipeline. I reimplemented `mne-python`'s `SlidingEstimator` for temporal decoding, and implemented custom scorers that were able to return confusion matrices for all targets of interest to work with it. In order to include dimensionality reduction methods, I implemented three custom transformers, one for PCA, one for SRM, and one for spectral SRM. The SRM-based transformers followed the procedures to fit and apply a (spectral) SRM from the previous analysis: A virtual subject was generated by concatenating data from each target, and a configurable number of these virtual subjects were created to fit the shared response model on. As before, the model allowed to limit the training of the model to a particular time frame of interest, such as only the first visual stimulus presentation. The resulting basis matrices in the model were averaged and transformed into neurophysiologically interpretable weights following Haufe et al. (2014). The PCA transformer was based on an existing PCA transformer in `scikit-learn`, but adjusted to work with three-dimensional multi-trial MEG data. To make it comparable to the SRM transformers, I further adjusted the PCA transformer to be train-able on user-defined time frames from the trial, too. Utilizing the fact that only the delay phase was relevant for this analysis, I used data from later parts of the trial to optimize parameters of the pipeline with a grid search. A number of configurable parameters were included, and are summarized in Table 4.2.

Parameter	Justification	Values	Choice
Sliding window	Sliding windows integrate neural signals over time or space, and are commonly used to improve the signal to noise ratio as well (Hari and Puce, 2017). This parameter control which form of sliding window was used.	temporal sliding window, spatiotemporal sliding window	temporal sliding window
Trial averaging	To improve the signal to noise ratio, it is common to average trials of the same trial type (Hari and Puce, 2017).	None, 5, 10	10 (None in spectral SRM)
Bootstrapping	If several trials are averaged to reduce the signal to noise ratio, this parameter offsets the loss of trials and controls how many averaged trials are drawn in total	minimum, maximum, 500	maximum
k (SRM, PCA)	This parameter determines the dimensionality of the subspace.	2, 5, 10, 25, 80	25
N training series (SRM)	This parameter determines the amount of training data with which a shared response model will be build.	10, 20, 50	50

Table 4.2: Overview of configurable pipeline parameters and the values explored in a grid search.

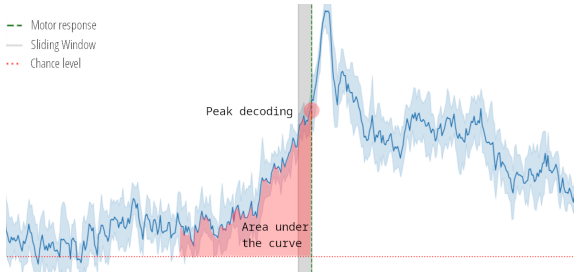
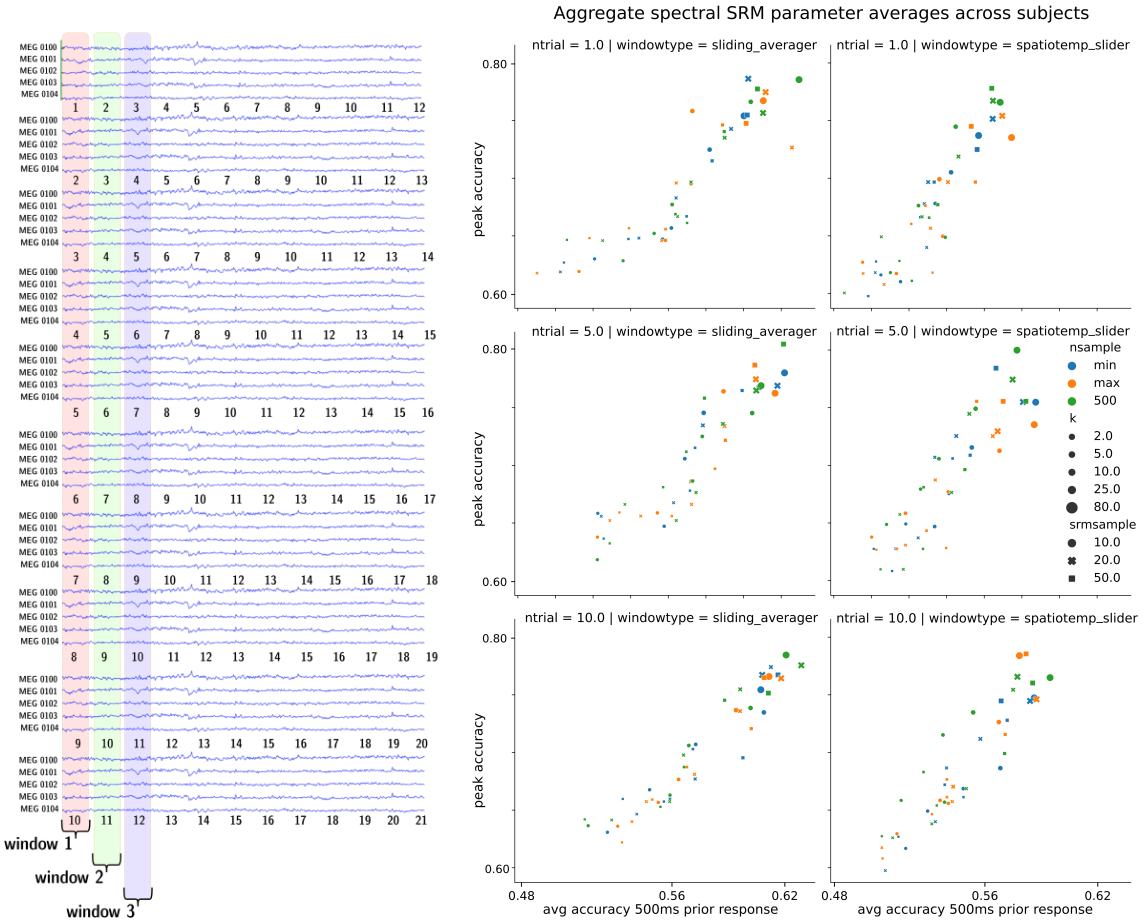


Figure 4.20: Two measures served for model evaluation in a decision-centered subset of data that was not used otherwise in this analysis: The average accuracy 500ms prior to the motor response, and the peak accuracy in this time frame.

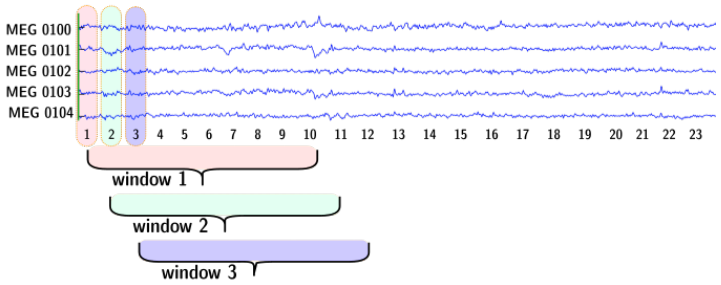
Some of these parameters bear further explanation, in particular the bootstrapping extension to trial averaging, and the different forms of sliding windows. I implemented two separate sliding windows: One was a temporal sliding window, integrating signal in the time dimension by averaging the signal within a sensor over multiple time points. An alternative sliding window was spatiotemporal. It integrates information over both space and time in a pattern inspired by multivariate pattern analysis, by extending the signal vector at one time point with the signals of the following time points. Thus, the data entering the decoders for this sliding window were pooled over multiple time points, so as to exploit not only information that is encoded in spatial activation patterns, but also information that is encoded in their temporal structure, similar to Muhle-Karbe et al. (2021). Both types of sliding windows were used with a window size of 10 samples, and are illustrated in Figures 4.21c and 4.21a.

The bootstrapping extension to trial averaging was created to offset the reduction in available trials in the model when several trials are averaged. This was implemented as follows: During trial averaging, all available trials were pooled according to their target label. Out of this pool, N trials were drawn with replacement and averaged. This process was repeated as often as the bootstrapping parameter specified. The values that were evaluated in the grid search were 500 averaged trials (an amount exceeding the available number of trials in each target label), "minimum" (an amount determined by the smallest number of trials with the same label), and "maximum" (an amount determined by the largest number of trials with the same label). As trial averaging from a random draw of trials effectively created new artificial trials, this method practically increased the amount of available data in the pipeline.
To keep the computational complexity and processing time of the grid search manageable, a limited set of values for each parameter was included. An overview of them is given in Table 4.2. The regularization parameter C for the logistic regression classifier was left at its default value as it does not have a strong influence in L2-regularized models (Varoquaux et al., 2017). A grid search was then ran for each dimensionality reduction method separately. The underlying data for this was neural activity from the motor response of the trial. I based the evaluation on two accuracy measures (peak accuracy and average accuracy), visualized in Figure 4.20. The parameter optimizations yielded similar results for the parameters common to decoding pipelines with and without dimensionality reduction. Trial averaging improved average and peak accuracies, and this trend was increased with larger bootstrapping samples (ACC_{peak} = .66 vs .67 vs .67 and ACC_{avg} = .53 vs .54 vs .54 for no trial averaging, averaging 5 trials, and averaging 10 trials). The decoding accuracy was higher when sliding windows were used, and the temporal sliding window performed slightly better on both measures (ACC_{peak} = .68 vs .67 vs .66 and ACC_{avg} = .55 vs .53 vs .52 for the temporal, spatiotemporal, and no sliding window respectively). For all pipelines with dimensionality reduction, larger values of k yielded better performance (ACC_{peak} = .65 vs .66 vs .67 vs .68 vs .70 and ACC_{avg} = .52 vs .53 vs .54 vs. .55 vs. .56 for the values of 2, 5, 10, 25, and 80 respectively). For SRM-based dimensionality reduction methods, increasing the amount of data with which the SRM model was trained did not change performance measures. Figure 4.21b shows an exemplary visualization of the grid search analysis for one dimensionality reduction method.

The temporal decoding analysis of interest was set up in a 5-fold cross validation. To keep computational time manageable and reduce noise levels, data were downsampled to 200Hz prior to decoding. The parametrization of configurable pipeline steps can be found in Table 4.2. All dimensionality reduction models were trained only on signal during the presentation of the first option in an attempt to constrain the extracted components to neural signatures present during stimulus encoding. Although the highest dimensionality value $k = 80$ was most successful in the grid search, this is the dimensionality of the SSS basis which is already known to contain all measurable signal. To investigate whether subspaces of interest can be found in fewer dimensions, I used the next lower value of k, resulting in $k = 25$.
The results revealed a similar temporal decoding pattern as the one found by Kaiser, Gruendler, et al. (2018), and this pattern was similar across dimensionality reduction methods: Initial above chance accuracies during the encoding period at stimulus presentation that reduced to chance level within 1000ms into the delay period and were reinstated during the second stimulus presentation. Probability yielded the highest decoding accuracies overall, and all results depicted in Figures 4.22 and 4.23 stem from pipelines with this property as the target. Compared to no dimensionality reduction and dimensionality reduction with SRM, the pipelines employing PCA and spectral SRM transformers yielded marginally reduced accuracies. Overall, none of the dimensionality

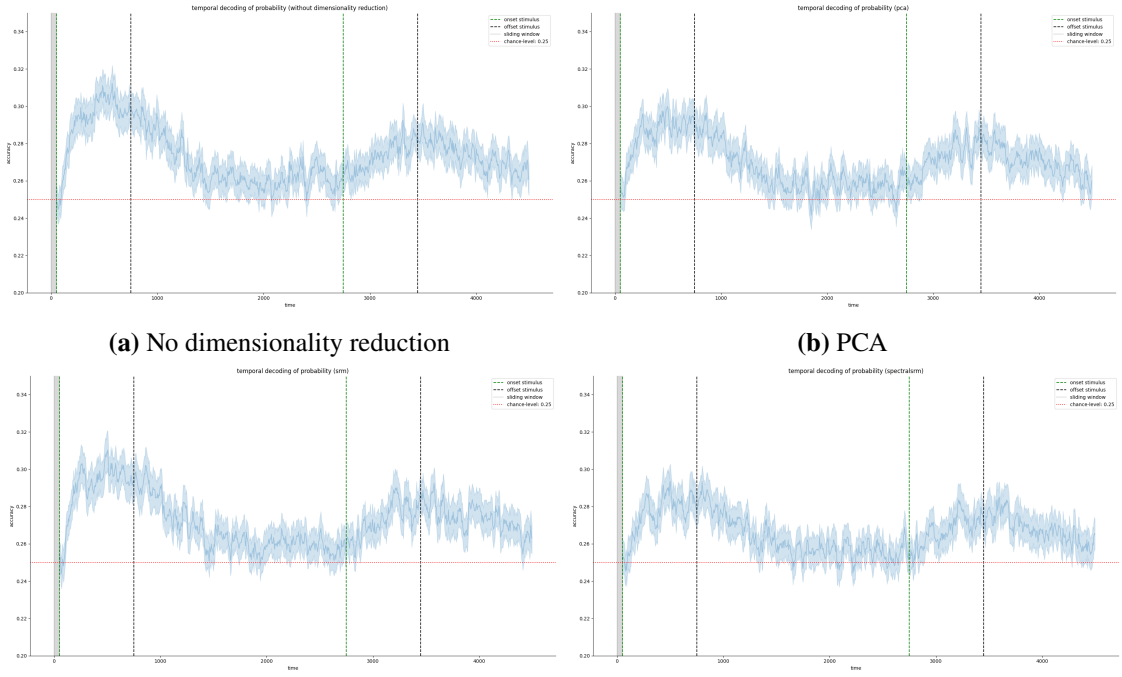


(a) Spatiotemporal sliding window **(b)** Exemplary optimization results from spectral SRM pipeline



(c) Temporal sliding window

Figure 4.21: Schematic of different sliding window types. 4.21c depicts a temporal sliding window, integrating the signal across samples via averaging. 4.21a depicts a spatial sliding window. All samples in the sliding window are concatenated such that the one resulting sample per sliding window includes the spatiotemporal configuration of all sensors in the window. 4.21b visualizes the results of tuning various pipeline parameters on decoding for decoding with a spectral SRM transformer.



(a) No dimensionality reduction

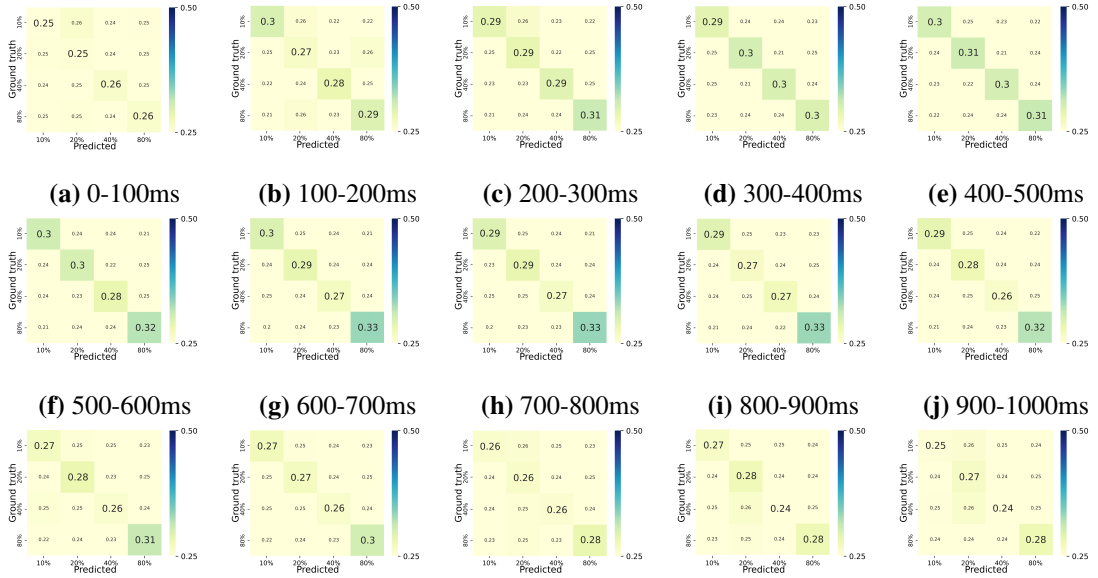
(b) PCA

(c) SRM

(d) spectral SRM

Figure 4.22: Decoding results for the probability (angle) of the first stimulus option. Grey shading visualizes the length of the temporal sliding window, and vertical lines visualize trial structure. The horizontal dashed line indicates chance level decoding. Blue shading depicts variability across folds in the cross-validation.

reduction methods revealed stable neural codes throughout the delay, and contrary to expectations they also did not improve decoding accuracy. I also investigated a possible pattern of decoding errors by means of confusion matrices. For this, I computed confusion matrices in 100ms time slices to visualize dominantly decoded labels across the trial structure (Figure 4.23). There was no prominent confusion pattern over the trial course, but the confusion matrices made it evident that the highest probability was the one category that retained above chance decoding accuracies in the delay phase.



(a) 0-100ms **(b)** 100-200ms **(c)** 200-300ms **(d)** 300-400ms **(e)** 400-500ms

(f) 500-600ms **(g)** 600-700ms **(h)** 700-800ms **(i)** 800-900ms **(j)** 900-1000ms

(k) 1000-1100ms **(l)** 1100-1200ms **(m)** 1200-1300ms **(n)** 1300-1400ms **(o)** 1400-1500ms

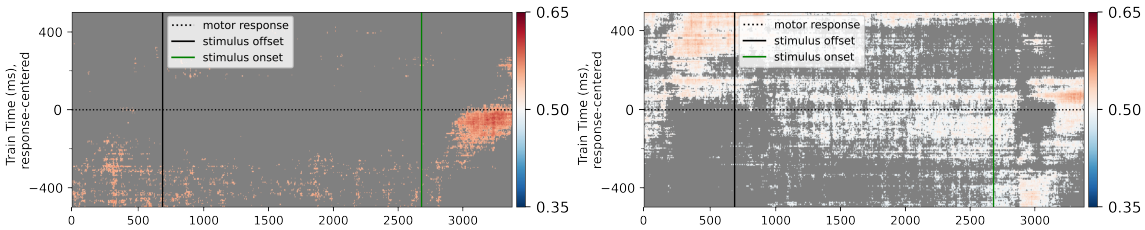
Figure 4.23: Confusion matrices depicting decoding results (accuracy) for the probability (angle) of the first stimulus option in 100ms time slices. During the stimulus presentation (0-700ms), increased decoding accuracy emerges for all probability labels. In the delay, above chance decoding fades and is only achieved for the highest probability. The confusion matrices depicted here stem from a pipeline without dimensionality reduction, but the pattern emerges across dimensionality reduction methods, too.

Overall, these results replicated the findings of Kaiser, Gruendler, et al. (2018), and did neither yield extended decodability of stimulus features in the delay nor higher decodability in general. This holds true not only for the rather experimental technique of applying an SRM, but also for the more established PCA. It is worthwhile to bear in mind that all dimensionality reduction methods placed particular importance on the time of the encoding during first stimulus presentation: The SRM or PCA models with which data were dimensionality-reduced prior to being given to the logistic regression estimator were trained explicitly only on data from the first 700ms of the trial. Neither method, however, was able to encode lower-dimensional features of this training phase that matched activity in the delay phase after roughly one second.

4.10 Temporal generalization

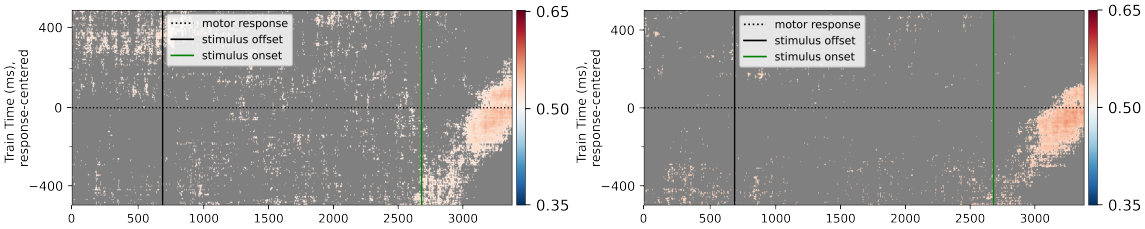
After I did not find stable stimulus properties in the delay period, I returned to the results of the behavioral analysis. Extending the behavioral logistic regression classification and strategy simulations, I performed several within-subject temporal generalization analyses to find decision-relevant representations in other trial properties than stimulus features. King and Dehaene (2014) formalized this method as follows: Several classifiers are trained on different time slices of training data each, and tested on all available times in the test data. Off-diagonal spreading of decoding accuracies indicates increasingly stable coding, whereas sole decodability in the diagonal of the

matrix is a sign for evolving neural codes that do not generalize well to other time points.
A temporal generalization analysis was already performed by Kaiser, Gruendler, et al. (2018). In their work, a classifier was trained to decode the magnitude, probability, or expected value of the first stimulus option at certain time points throughout the entire trial, and tested on all other time points. With this, they assessed the signal stability of the neural codes for these stimulus properties. Their results mirror the temporal decoding results, and showed stable above-chance decoding accuracy of stimulus properties mostly during the stimulus presentation. In the delay period, stable coding decreased sharply, and even time-point-to-time-point decodability, the diagonal of the matrix, vanished, most severely for magnitude and expected value. It was thus mostly during the visual presentation of the stimulus and shortly afterwards that value properties – or their associated visual features – had stable or dynamic neural codes that classifiers could pick up.
Extending these analyses based on my previous explorations, I investigated whether the neural representation during the delay period could correspond to a decision signal rather than to concrete property values. This falls in line with work by Hunt et al. (2013) who investigated whether the necessary comparison of options in decision making tasks occurs in the "space of abstract goods" or the "space of alternative actions". Whereas they found stronger value-related representations in a decision making task in which stimulus options where shown side-by-side and at the same time, they found a stronger action-related representation when the same decision making task was presented sequentially, similar to the experiment used in this dataset. To investigate this, I utilized the temporal generalization method's flexibility in the choice of training and testing time points. In the training phase, I let classifiers learn a decision signal from time points in the trial centered around the actual motor response. In the testing phase, I then used these classifiers to find a decision signal earlier in the trial. If those signals could be found during the delay period, they would indicate that a preparatory decision state is a part of working memory maintenance in this experiment.
For this, I split the MEG data into separate epochs: One set of epochs centered around the motor response with a length of 1000ms. And one set of epochs locked to the first stimulus presentation, extending 3.4 seconds into the trial, until the end of the second stimulus presentation. The first set of epochs was used to train multiple logistic regression estimators on MEG data in sensor space ($X = 306 \times n\_samples$ features per trial) to classify the target y, left or right choice in each trial. The second set of epochs was used as test data, to see when and how well a neural decision signal emerges over the trial course. While the estimators were always trained on the entire training data, I manipulated the set of features in the test set to investigate the emergence of a decision representation based on specific trial characteristics, given that distinct stimulus properties and values likely contribute differentially to a stronger representation. For each analysis, I ran a permutation test with $N = 10\,000$ repetitions with shuffled targets to identify clusters of significant decoding accuracy. To evaluate findings on the group level, I averaged all subjects' temporal generalization results, and bootstrapped a sample of each participants' permutation maps to estimate cluster reaching statistical significance following the approach proposed by Stelzer et al. (2013).
Overall, I investigated the following manipulations: First, I decoded the decision representation for trials with high versus medium versus low values in magnitude or probability in the first stimulus (Figure 4.24). The underlying hypothesis for this analysis was if participants focus primarily on one of these properties as the behavioral analysis indicated, a high (or low) value in the first option might evoke a neural representation of a left (or right) choice already earlier in the trial, and allow an investigation if this neural code remains stable across the delay. While these analyses revealed better decodability of eventual choice for high and low values compared to less informative medium values (only visible in raw accuracies, masked in Figure 4.24), statistically significant clusters only emerged at the end of the trial where training time points and testing time points began to overlap.



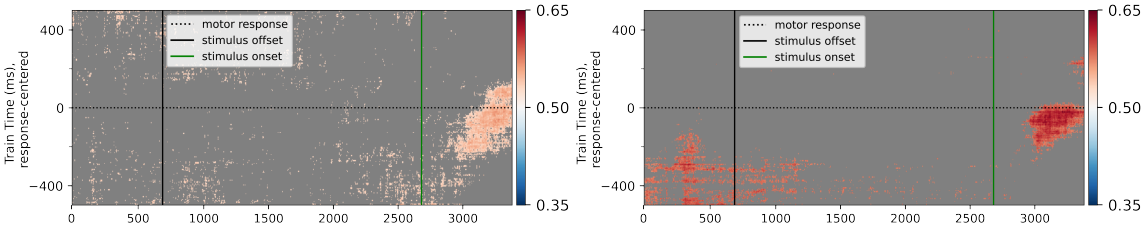
(a) High magnitude trials (4 points)

(b) High probability trials (80% chance)



(c) Medium magnitude trials

(d) Medium probability trials



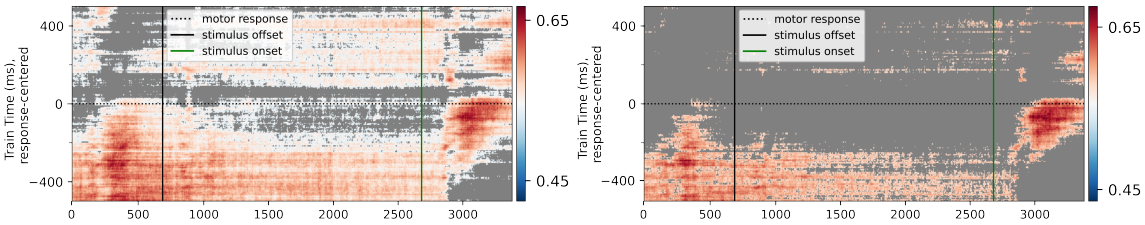
(e) Low magnitude trials (0.5 points)

(f) Low probability trials (10% chance)

Figure 4.24: Temporal generalization results of decoding eventual choice (right option), overlaid with a permutation mask. The y-axes depict training time points, with 0 (horizontal line) being the motor response, negative values denoting trial samples prior to the motor response, and positive values denoting trial samples after the motor response. The x-axis depicts test data, covering the onset of the first stimulus until the offset of the second stimulus. Vertical lines illustrate trial structure. Only those parts of the plot that are not masked yielded accuracies that were higher than 95% of accuracies obtained in N =10 000 permutations with shuffled training labels.

A statistically significant representation of the eventual choice in the delay was thus not found. However, this could have been a consequence of experiment design decisions: Stimulus properties for the first option were explicitly constrained to never contain the best or worst combination of properties to prevent a decision forming already in the delay period (Curtis and Lee, 2010). A promising amount of probability or magnitude in the first option would regularly be beaten by a better combination in the second option and vice versa, influencing the eventual decision that became the target variable, and compromising a previously formed preparatory decision signal. To circumvent this restriction, I adjusted the target definition in the test data. Instead of decoding the actual choice at the end of the trial, I decoded a hypothetical target corresponding to the behavioral strategies from Figures 4.9b, 4.9c, 4.9e, and 4.9f: If the first stimulus' magnitude (or probability, respectively) was high, I assumed the prepared choice would be left, and vice versa. While this

analysis would not decode the actual “choice”, it would decode a preparation signal expected given the presumed decision strategy and available information already at the start of the trial. The results of this analysis reveal a stronger decodable cluster, in which neural codes prior to the motor response generalize to trial parts during the first stimulus presentation and into the delay phase for both magnitude and probability (Figure 4.25).



(a) Low magnitude trials (0.5 points)

(b) Low probability trials (10% chance)

Figure 4.25: Temporal generalization results of decoding hypothetical targets. Figure and analysis are similar to 4.24, but the decoding target (right option) was hypothetical, derived from whether the stimulus property in question was high or low in the first stimulus option, mirroring the behavioral strategies of evaluating stimulus properties preferentially or sequentially.

Finally, I repeated this analysis taking into account the relative influence of both stimulus properties for each individual participant. For this, I used the beta coefficients of the behavioral analysis to estimate the subjective choice C after the first stimulus presentation according to the formula

$$C = LMag * \beta_{LMag} + LProb * \beta_{LProb} + LEV * \beta_{LEV} \tag{4.4}$$

where $LMag$, and $LProb$ are the min-max scaled magnitude and probability values of the left stimulus option per trial, LEV is the expected value calculated from demeaned magnitude and probability, similarly min-max scaled, and all β values are beta coefficients obtained from the logistic regression analysis on behavioral data. With this formula, the opposite ends of the value range of C denote a tendency for the left and right stimulus option, respectively. From this value range, I extracted the lowest and highest quartile as test data trials, assigning estimated choices as target labels. Compared to the previous analyses which modeled hypothetical choice based on only one stimulus property, a decision signal generalizing to these estimated targets would indicate an integration of stimulus properties. The resulting decoding pattern mirrored that in Figure 4.24, revealing no clusters in the first stimulus presentation or delay period.

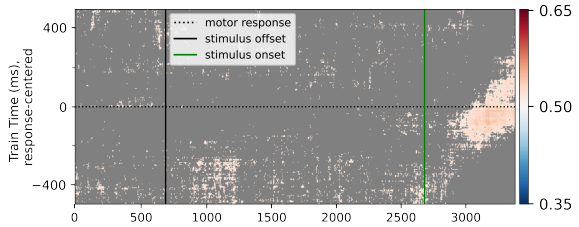


Figure 4.26: Temporal generalization results of decoding estimated targets, computed from integrating stimulus properties. Figure and analysis are similar to 4.24, but the presumed choice that served as the decoding target (right option) was derived from each participants' individual parameter influence on decision.

4.11 Discussion

The analysis set presented here consisted of a variety of methods. While some analyses were conducted to test the applicability of approaches or justify them, such as the initial SRM applications or the simulation study conducted for spectral SRM, the underlying focus of all analyses was to uncover the neural representations of trial-related features in working memory, using the delay period in a decision making task.
Evidence for a major hypothesis of persistent neural representations of stimulus features (magnitude, probability, or expected value) in working memory could not be found. Dimensionality-reduction, in particular methods based on the shared response model, a spectral variation of the shared response model, and PCA, were unsuccessful in finding a lower-dimensional space in which such features were represented in a stable and persistent manner. With this, my temporal decoding analyses confirmed the temporal dynamics uncovered by Kaiser, Gruendler, et al. (2018) with adjusted or novel methodology: Individual stimulus features had above-chance decoding accuracy only a few hundred milliseconds into the delay phase. This means, however, that unlike Murray et al. (2017), the analyses were not able to find lower-dimensional neural state spaces in which stimulus characteristics are persistently decodable, even with a similar PCA-based approach. And unlike Machens et al. (2010), the resulting components that spanned the lower-dimensional space, although they clearly captured trial-related information, were not easily interpretable regarding stimulus features. There are multiple possible reasons for this. One may be that the shared response model and PCA transformer identified subspaces that are not relevant to the chosen targets' decoding. In other words, the lower-dimensional space they identified may not have been the "right" one. The initial SRM analyses on data during stimulus presentation resulted in lower-dimensional spaces that clearly represented a difference between the first and second stimulus, but they did not represent differences between stimulus types well. These spaces thus may have better captured a left versus right difference in visual presentation than stimulus identity. Likewise, the shared components in the spectral SRM analysis only showed a marked distinction for left versus right decisions, but not for the fine-grained stimulus feature value differences, such as different levels of magnitude. Thus, data from the decision phase seemingly provided a good basis to identify a subspace in which motor responses were well represented, but such clear distinctions did not emerge in data from the delay phase for stimulus characteristics. In those cases, other neural processes than those corresponding to stimulus feature maintenance have potentially been more prominent and thus over-represented.

Left versus right visual stimulation or left versus right motor responses should yield lateralized neural signals in visual and motor cortices, which are presumably more distinct than neural signals emerging during the evaluation of different stimulus value levels. Especially since whole-brain data was the input of all analyses, a variety of other cognitive processes were likely reflected in the neural signal and thus the resulting shared spaces. Both Machens et al. (2010) and Murray et al. (2017) had confined their analyses to prefrontal areas, and used electro-physiological recordings which are spatially much more precise than the overlaid neural signals in sensor-space MEG data, and may have confined their data better to neural signals of interest. Other differences in analysis or data preparation approaches could play a role, too. In Murray et al. (2017), lower-dimensional mnemonic subspaces were constructed from averaged delay activity over stimulus conditions and time. Thus, their stable subspaces did not contain precise timing information, which may have removed potentially relevant dynamics from the underlying data. In contrast, the subspaces built in this work were optimized to retain precise timing information. On the other hand, the particularly high sampling rate of the data presented here may have also introduced higher levels of noise, which could have further overshadowed task-relevant stimulus representations. Likewise, given the potential that shared components captured other or many neural processes at once, the interpretations of the identified components may not be as straight-forward as initially envisioned. The visualization approaches in the initial SRM analysis and spectral SRM variation for example focused on finding simple amplitude-based differences between stimulus types in the shared space, either via correlation distance between pairs of shared spaces for individual stimulus types, or by transforming data into shared spaces based on stimulus feature values. This approach would have failed to find more complex representations, for example based on interactions between several components. Yet another alternative worthwhile of consideration is that methods identifying subspaces based on "sharedness" are suboptimal for the neural process in question. Working memory is known to have idiosyncratic components. Lee and Geng (2017) for example found idiosyncratic representations of a search cue both across participants but also within participants. Approaches focused on common representational structures do not capture such participant- or trial-specific neural codes well. Another reason may be that coding of decision-relevant information in working memory is indeed dynamic and activity silent, and thus impossible to decode stably or without a reactivation as performed by Wolff et al. (2017). However, this interpretation is challenging in light of the fact that there is above-chance decoding accuracy during the first half of the delay. The results of Kaiser, Gruendler, et al. (2018) further showed that above-chance decoding of stimulus characteristics vanishes even in temporal generalization analyses, yet if this information is required in order to form a decision a decision without such information is difficult to explain. Therefore, a plausible reason why stimulus features could not be decoded through the entire delay may be that it is not stimulus characteristics that are encoded throughout the trial but an alternative representation, such as decision preparation. This matches the works of Hunt et al. (2013), who hypothesized that in sequential decision making trials, participants may form an evaluation in an action-related space.

Some support for this comes from the joint results of the behavioral simulations and the temporal generalization analyses. Prior temporal generalization analyses from Kaiser, Gruendler, et al. (2018) already found that the neural representation of the stimulus features probability, magnitude, and expected value decay during the delay similarly as in the temporal decoding analyses. In my extension of their analyses, I thus focused on an alternative representation of decision-relevant information – not in terms of value, but in terms of an upcoming decision or motor response. These temporal generalization analyses indicated that a neural representation of such a decision-related signal persists throughout the delay period. A representation matching neural codes a few hundred

milliseconds prior the motor response was found throughout the delay for "presumed decisions", i.e., decisions that a participant would be likely to prepare given the information they had after seeing the first stimulus. Above-chance decoding in the delay phase only emerged from presumed decisions calculated separately for magnitude and probability values, but not when integrated into an idiosyncratic stimulus value with the subject-specific weights from the logistic regression analysis. This is interesting in two aspects: It represents a neural signal that matches a behavioral strategy based on sequential evaluation of stimulus features, and it is a signal more similar to the neural codes that emerge in the preparation of a decision than to the neural codes that emerge during visual stimulation. With regard to conclusions about the exact behavioral strategy, the temporal generalization results alone are inconclusive. Unlike the logistic regression classification in behavioral analysis, the temporal generalization analysis followed the sequential nature of a trial in which participants learned of stimulus properties consecutively. This bears the interesting opportunity to observe how the neural representation evolves. However, it can not differentiate whether participants went on to do a sequential evaluation of the second stimulus property (strategies 4.9c, 4.9b) or focused on a single property. The behavioral simulation results, however, can fill this gap by providing sufficient confidence for a sequential evaluation strategy: When comparing simulation results with actual gains, strategies based on a sequential evaluation of stimulus features magnitude and probability rather than an integration into expected values or a focus on a singular stimulus features were closest to participants' actual performance. The lack of a representation of presumed choice from an integrated signal in the temporal generalization analysis can be seen as further evidence against an immediate integration of stimulus properties. Thus, given that a decision-related neural code was better represented during the maintenance phase than absolute stimulus property values, this result would support a decision-representation in working memory rather than stimulus features. But while it would explain the lack of stimulus feature representations in other analyses, this interpretation is complicated by the fact that the experiment design was explicitly focused on uncovering stimulus feature representations only. The reward schedule was created to explicitly prevent a decision signal from forming early by disallowing particularly informative, i.e., particularly attractive or unattractive, stimuli from the first stimulus option. Thus, the experimental design is arguably inept to test the hypothesis whether a decision signal emerges thoroughly. Consequently, these results could only be obtained by working around explicit design decisions in the experiment. The inclusion of trials that would allow confident decisions already after the first stimulus presentation could constitute an insightful addition to the experiment that would allow to test this hypothesis further.

Finally, it has to be noted explicitly that the analyses in this chapter were purely exploratory. Their validity needs to be tested on independent data, especially since the plurality of analyses that have been conducted on this dataset by now – both by me and others – defies attempts for meaningful corrections for multiple comparisons. It is well known that data analyses in neuroscience can be overfit to a particular dataset (Hosseini et al., 2020). If relevant results from this dataset however would generalize in an independent dataset, their evidence would be strengthened. In particular, this relates to the finding that stimulus feature representations can not be found in the delay, and that decision-related signals are instead being maintained. The provenance-tracked and transparent nature in which my analyses were conducted and openly shared code provide a useful starting point for such work.

Despite the only preliminary insights on working maintenance in this exploratory project, my work has nevertheless contributed valuable reusable research outcomes in several aspects. For one, the work I put into the preparation of the dataset has made further analyses much more feasible, regardless of whether they are performed by researchers acquainted to the dataset or not, and

regardless of whether the various actors involved in the acquisition, processing, and preparation of the data are available for questions. Open code and provenance tracked analysis executions contribute to the reusability of these intermediate research outcomes further. As SRM has so far rarely been employed in MEG data in the published literature, this project also provided ample initial evidence that the method could be useful for this modality. While the SRM analyses did not confirm prior hypothesis, all studies showed that the method retained major experiment-induced variation, such as the trial structure in the MEG data, or the signal of interest in the simulation study. In the temporal decoding analysis, pipelines with an SRM also did not fare worse than pipelines without it. And the simulation study demonstrated that the method, regardless of whether it is applied with the novel spectral transformation we tested or not, can markedly reduce random noise in the data. Thus, a potential and promising application for SRM in MEG might be signal cleaning. Similar observations were made by Xie et al. (2021) when they used SRM with intracranial stereotactic EEG recordings from epilepsy patients.

At the same time, the analyses conducted here give way to further research ideas beyond generalizing results to new datasets. One commonality of all analyses was the use of all available channels, and thus whole-brain data. This was partially motivated by the fact that a variety of brain regions were found to be involved in aspects of working memory maintenance Sreenivasan and D'Esposito (2019), and a restriction to specific sensor-arrays or brain regions could have missed contributions from left out areas. However, it is common practice in MEG research to confine the number of sensors, either to specific regions, or to specific flux transformers (Garcés et al., 2017) to limit the amount of task-irrelevant noise, and it has been shown that reducing the number of channels in brain-computer-interface applications can improve decodability (Roy et al., 2020). Future work can investigate whether the SRM also yields more promising results when its input data stems from predefined cortical areas or sensor arrays, such as only prefrontal locations.

Under the assumption that neural processes of working memory maintenance are more idiosyncratic, a different approach to improve the ratio of interesting signal to overall signal could employ the SRM method to identify uninteresting shared signals. It has been demonstrated that the data after SSS represents the most dominating magnetic field patterns in MEG recordings (Garcés et al., 2017). Likewise, SRM components capture dominant signals shared across participants. Given the possibility that other neural processes may create more prominent signals, or that important idiosyncratic processes could be discarded as noise, future studies could attempt to identify these dominant signals via SRM, and subtract them from individual participants' neural signal to either remove dominating sources of uninteresting shared signals, or investigate these idiosyncratic signals in particular.

And finally, this project has been an acid test for the research data management strategies outlined in previous chapters, in a project with significant evolution in the foundational data representations, ambitious demands on portability of its building blocks, and without the luxury of predefined analysis pipelines. Following the project outline in Figure 4.1, Figure 4.27 is an example of the digital provenance that is attached to each reported analysis in this work, and stems from a portable DataLad dataset of one analysis. Such provenance makes the files in these projects self-descriptive, and in many cases, directly re-computable. This successful application of the research data management strategies and tools validates their utility for scientific conduct in exploratory data analyses.

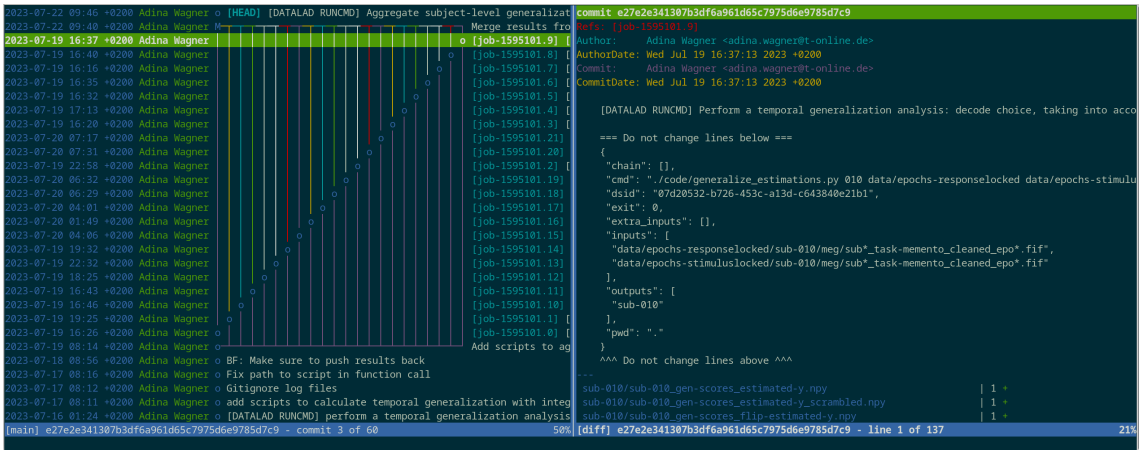


Figure 4.27: An example revision history of one of the DataLad datasets in this project as visualized with the software tool `tig`. Shown is an overview of saved updates on the left, with a human-readable description, an author, and a date, as well as a detailed overview of one such updates on the right. The detailed overview reveals additional information such as a longer description, a list of resulting or modified files (only partially shown), and also a re-executable DataLad run-record (see Figure 3.4). By means of the commit identifier, the temporal generalization analysis of one subject described in this provenance record can thus be recomputed by the original authors or unrelated others with access to the project.

5 Conclusion

> If I have seen further it is by standing on the shoulders of Giants
>
> *(Isaac Newton)*

The original aim of this work was to find novel insights about brain state transitions in a delayed decision making task from previously unpublished data by connecting and building up on prior work by other researchers. However, the context in which this project was conducted made research data management and research software engineering more central than originally planned. Viewed in conjunction, this thesis has been a testament to the foundational importance that organizational and technical aspects of scientific conduct play in research endeavors.

Chapter 1 laid out an overview of the study of working memory maintenance, in particular novel methods that viewed neural signals as trajectories through high-dimensional state spaces. It further introduced preexisting tools and solutions for managing research data and projects. The FAIR principles and the BIDS standard are established elements of good RDM, and DataLad a promising software tool for data versioning, transport, and digital provenance capture. In conjunction, they enable researchers to create scientific outputs that are portable and reusable as stand-alone, well-described units, without reliance on the original authors – and thus, a fitting tool set to conduct the brain-state analysis in Chapter 4.

However, Chapter 1 also shed light on DataLad's shortcomings and deficiencies. Thus, my work on the DataLad Handbook, outlined in Chapter 2 and rooted in the area of research software engineering, improved some of these with documentation and workflows. This constituted not only a beneficial foundation for my own DataLad-supported analysis in Chapter 4, but also led to lasting resources for the general public and improved software usability. The Handbook's popularity and its association with increased relevance of the software tool yielded evidence for the importance of user-focused documentation. Although this work is not a traditional academic paper, it can, similar to research software, published findings, or open code impact scientific conduct positively.

Continuing in a similar spirit, Chapter 3 focused on two central aspects in research data management, specifically reusability and reproducibility, and showed how they connected to each other: Reproducibility is a basis for reusability, and both are the outcome of research data management that becomes increasingly more FAIR. But Chapter 3 also outlined the practical difficulties of creating reusable research outputs that arise despite the existence of the FAIR principles because full FAIRness can not always be achieved. To address them, I conceived a set of four pragmatic research data management strategies that can make research objects more reusable, even when they are not yet fully FAIR. While our work recognizes the FAIR principles, it makes a pragmatic compromise between the aspirational ideal state, and the continuously developing research and meta data landscape in computational sciences. In passing, as a byproduct of RDM, its outcomes go a long way towards FAIR, and by applying the four properties of exhaustively versioned, actionable, modular and portable to research projects, become re-usable by default. Chapter 3 then concluded with a computational framework that put these strategies into practice, and highlighted our proof of principle work to test its applicability and scalability on a neuroscientific dataset of the largest scale.

Our framework brought together a range of work that the previous chapters outlined already. At the center of our framework lies thoughtful research data and reproducibility management, not as an afterthought, but – given the technical challenges computational reproducibility poses – as a sturdy and necessary basis. In the spirit of reusable and FAIR digital research objects, it creates outcomes that lend themselves to reuse naturally. Especially in large-scale computations, current results will be the inputs of future projects, and their built-in reusability provides the necessary trust for this. Chapter 1 also established DataLad as a software tool to deliver this research data management. Its development principles and features in the spirit of decentralized software development translate into utility for our framework. For one, it is able to connect a range of established services or and open source tools to ease adoption and maximize flexibility. And although the workflows and concepts our framework draws from are not based in the domain of reproducible data processing, they lend themselves well for this application, and what made distributed software development productive, fast, and reliable, now helps to do the same for data analysis. Fundamental to the success and usability of DataLad and its use in this framework is its documentation, as argued in Chapter 2. On a high level, the framework constitutes yet another use case, only possible because the tools' features are combined into something greater than the sum of its parts. Many years of refining user-centered workflows and software usability formed the basis for this. The available documentation allows interested readers to learn beyond the publication, up to a point where they can extend it to their own use cases. A testament to this, and to the utility of the framework is its adoption into dedicated new tools already in an early stage of its development. Heunis (2023) makes use of it for reproducible metadata aggregation, it plays a role in the CuBIDS packages for the reproducible curation of BIDS datasets (Covitz et al., 2022), and a dedicated software tool, BABS, has been created to improve the usability of the workflow even more (Zhao et al., 2023).
Chapter 4 then put the technical and organizational work of previous chapters into action. The idiosyncratic structure of the *memento* dataset has posed a significant challenge to identify relevant files, distinguish processing states, understand the dataset as a whole, and base processing on it. The conversion to the BIDS standard for MEG was an indispensable prerequisite to working with this data. The spatio-temporally distributed nature of the project, spread over three institutions and two generations of researchers working with the dataset, placed reusability and portability demands on my work that the previous chapters laid the foundations to. I was unable to find the delay representation of stimulus properties which the experiment had originally set out to find. However, I was able to extend previous analyses with novel ideas, methods that are yet unconventional in MEG, and interesting exploratory findings that can spark further research. With the RDM principles I theorized in Chapter 3, documented in Chapter 2, helped to software engineer in Chapter 1, and applied in Chapter 4, future re-users can obtain or recompute my results, or investigate differences between re-computations after adjusting pipelines or parametrization with novel research ideas. The intermediate research outcomes of this project thus are a valuable stepping stone for further research.

Bibliography

Abrams, M. B., Bjaalie, J. G., Das, S., Egan, G. F., Ghosh, S. S., Goscinski, W. J., Grethe, J. S., Kotaleski, J. H., Ho, E. T. W., Kennedy, D. N., et al. (2022). A standards organization for open and fair neuroscience: The international neuroinformatics coordinating facility. *Neuroinformatics*, *20*(1), 25–36 (cit. on p. 38).

Appelhoff, S., Sanderson, M., Brooks, T. L., Vliet, M. van, Quentin, R., Holdgraf, C., Chaumon, M., Mikulan, E., Tavabi, K., Höchenberger, R., Welke, D., Brunner, C., Rockhill, A. P., Larson, E., Gramfort, A., & Jas, M. (2019). Mne-bids: Organizing electrophysiological data into the bids format and facilitating their analysis. *Journal of Open Source Software*, *4*(44), 1896. https://doi.org/10.21105/joss.01896 (cit. on p. 57).

Ashburner, J., & Friston, K. J. (2000). Voxel-based morphometry—the methods. *Neuroimage*, *11*(6), 805–821 (cit. on p. 46).

Ashburner, J., & Friston, K. J. (2011). Diffeomorphic registration using geodesic shooting and gauss–newton optimisation. *NeuroImage*, *55*(3), 954–967 (cit. on p. 47).

Attwell, D., & Laughlin, S. B. (2001). An energy budget for signaling in the grey matter of the brain. *Journal of Cerebral Blood Flow & Metabolism*, *21*(10), 1133–1145 (cit. on p. 13).

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature*, *533*(7604) (cit. on p. 35).

Barascud, N., Alcocer, P., Roujansky, P., Tresols, J. J. T., Szul, M., Pals, M., Quentin, R., Royen, C., Ferraro, G., & ludovicdmt. (2022). *Nbara/python-meegkit: V0.1.3* (Version v0.1.3). Zenodo. https://doi.org/10.5281/zenodo.7319631 (cit. on p. 57).

Barba, L. A. (2018). Terminologies for reproducible research. (Cit. on p. 35).

Barker, T. T. (2003). *Writing software documentation*. NY: Longman, New York. (Cit. on p. 28).

Bazeille, T., Dupre, E., Richard, H., Poline, J.-B., & Thirion, B. (2021). An empirical evaluation of functional alignment using inter-subject decoding. *NeuroImage*, *245*, 118683 (cit. on p. 66).

Bechhofer, S., Buchan, I., De Roure, D., Missier, P., Ainsworth, J., Bhagat, J., Couch, P., Cruickshank, D., Delderfield, M., Dunlop, I., et al. (2013). Why linked data is not enough for scientists. *Future Generation Computer Systems*, *29*(2), 599–611 (cit. on p. 37).

Bechhofer, S., De Roure, D., Gamble, M., Goble, C., & Buchan, I. (2010). Research objects: Towards exchange and reuse of digital knowledge. *Nature Precedings* (cit. on pp. 16, 38–40).

Borghi, J. A., & Van Gulick, A. E. (2018). Data management and sharing in neuroimaging: Practices and perceptions of mri researchers. *PloS one*, *13*(7), e0200562 (cit. on pp. 22, 35).

Botvinik-Nezer, R., Holzmeister, F., Camerer, C. F., Dreber, A., Huber, J., Johannesson, M., Kirchler, M., Iwanir, R., Mumford, J. A., Adcock, R. A., et al. (2020). Variability in the analysis of a single neuroimaging dataset by many teams. *Nature*, *582*(7810), 84–88 (cit. on p. 19).

Bowring, A., Maumet, C., & Nichols, T. E. (2019). Exploring the impact of analysis software on task fmri results. *Human brain mapping*, *40*(11), 3362–3384 (cit. on pp. 19, 37).

Brainiak contributors. (2016). Brain imaging analysis kit. (Cit. on pp. 57, 65).

Brooks, P. P., McDevitt, E. A., Mennen, A. C., Testerman, M., Kim, N. Y., Visconti di Oleggio Castello, M., & Nastase, S. A. (2021). *Princeton handbook for reproducible neuroimaging* (Version v0.2.0). Zenodo. https://doi.org/10.5281/zenodo.4317623 (cit. on p. 32).

Bryan, J. (2018). Excuse me, do you have a moment to talk about version control? *The American Statistician*, *72*(1), 20–27 (cit. on p. 20).

Buckheit, J. B., & Donoho, D. L. (1995). *Wavelab and reproducible research*. Springer. (Cit. on p. 35).

Buonomano, D. V., & Maass, W. (2009). State-dependent computations: Spatiotemporal processing in cortical networks. *Nature Reviews Neuroscience*, *10*(2), 113–125 (cit. on pp. 14, 15).

Button, K. S., Ioannidis, J. P., Mokrysz, C., Nosek, B. A., Flint, J., Robinson, E. S., & Munafò, M. R. (2013). Power failure: Why small sample size undermines the reliability of neuroscience. *Nature reviews neuroscience*, *14*(5), 365–376 (cit. on p. 19).

Bzdok, D., & Yeo, B. T. (2017). Inference in the age of big data: Future perspectives on neuroscience. *Neuroimage*, *155*, 549–564 (cit. on p. 21).

Camerer, C. F., Dreber, A., Forsell, E., Ho, T.-H., Huber, J., Johannesson, M., Kirchler, M., Almenberg, J., Altmejd, A., Chan, T., et al. (2016). Evaluating replicability of laboratory experiments in economics. *Science*, *351*(6280), 1433–1436 (cit. on p. 35).

Casey, B. J., Cannonier, T., Conley, M. I., Cohen, A. O., Barch, D. M., Heitzeg, M. M., Soules, M. E., Teslovich, T., Dellarco, D. V., Garavan, H., et al. (2018). The adolescent brain cognitive development (abcd) study: Imaging acquisition across 21 sites. *Developmental cognitive neuroscience*, *32*, 43–54 (cit. on p. 19).

Chen, P.-H. (, Chen, J., Yeshurun, Y., Hasson, U., Haxby, J., & Ramadge, P. J. (2015). A reduced-dimension fmri shared response model. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 28). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2015/file/b3967a0e938dc2a6340e258630febd5a-Paper.pdf (cit. on pp. 64–66).

Cheveigné, A. de. (2020). Zapline: A simple and effective method to remove power line artifacts. *NeuroImage*, *207*, 116356 (cit. on p. 59).

Claerbout, J. F., & Karrenbach, M. (1992). Electronic documents give reproducible research a new meaning. In *Seg technical program expanded abstracts 1992* (pp. 601–604). Society of Exploration Geophysicists. (Cit. on pp. 35, 36).

Corti, L., Van den Eynden, V., Bishop, L., & Woollard, M. (2019). *Managing and sharing research data: A guide to good practice*. Sage. (Cit. on p. 20).

Courtney, S. M., Ungerleider, L. G., Keil, K., & Haxby, J. V. (1997). Transient and sustained activity in a distributed neural system for human working memory. *Nature*, *386*(6625), 608–611 (cit. on p. 12).

Covitz, S., Tapera, T. M., Adebimpe, A., Alexander-Bloch, A. F., Bertolero, M. A., Feczko, E., Franco, A. R., Gur, R. E., Gur, R. C., Hendrickson, T., et al. (2022). Curation of bids (cubids): A workflow and software package for streamlining reproducible curation of large bids datasets. *NeuroImage*, *263*, 119609 (cit. on p. 90).

Craddock, C., Benhajali, Y., Chu, C., Chouinard, F., Evans, A., Jakab, A., Khundrakpam, B. S., Lewis, J. D., Li, Q., Milham, M., et al. (2013). The neuro bureau preprocessing initiative: Open sharing of preprocessed neuroimaging data and derivatives. *Frontiers in Neuroinformatics*, *7*, 27 (cit. on p. 41).

Csomós, G., Vida, Z. V., & Lengyel, B. (2020). Exploring the changing geographical pattern of international scientific collaborations through the prism of cities. *PloS one*, *15*(11), e0242468 (cit. on p. 12).

Cunningham, J. P., & Yu, B. M. (2014). Dimensionality reduction for large-scale neural recordings. *Nature neuroscience*, *17*(11), 1500–1509 (cit. on p. 15).

Curtis, C. E., & Lee, D. (2010). Beyond working memory: The role of persistent activity in decision making. *Trends in cognitive sciences*, *14*(5), 216–222 (cit. on pp. 13, 54, 81).

Dahnke, R., Ziegler, G., Grosskreutz, J., & Gaser, C. (2013). Retrospective Quality Assurance of MR Images. https://doi.org/10.13140/RG.2.2.25494.91200 (cit. on p. 49).

Dahnke, R., Ziegler, G., Grosskreutz, J., & Gaser, C. (2015). Quality Assurance in Structural MRI. https://doi.org/10.13140/RG.2.2.16267.44321 (cit. on p. 49).

Dallas, C. (2016). Digital curation beyond the "wild frontier": A pragmatic approach. *Archival Science*, *16*(4), 421–457 (cit. on p. 39).

Dammers, J., Schiek, M., Boers, F., Silex, C., Zvyagintsev, M., Pietrzyk, U., & Mathiak, K. (2008). Integration of amplitude and phase statistics for complete artifact removal in independent components of neuromagnetic recordings. *IEEE transactions on biomedical engineering*, *55*(10), 2353–2362 (cit. on p. 60).

D'Esposito, M. (2007). From cognitive to neural models of working memory. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *362*(1481), 761–772 (cit. on p. 51).

Destrieux, C., Fischl, B., Dale, A., & Halgren, E. (2010). Automatic parcellation of human cortical gyri and sulci using standard anatomical nomenclature. *Neuroimage*, *53*(1), 1–15 (cit. on p. 49).

Duncan, J. (2001). An adaptive coding model of neural function in prefrontal cortex. *Nature reviews neuroscience*, *2*(11), 820–829 (cit. on p. 13).

Edwards, P. N., Mayernik, M. S., Batcheller, A. L., Bowker, G. C., & Borgman, C. L. (2011). Science friction: Data, metadata, and collaboration. *Social studies of science*, *41*(5), 667–690 (cit. on p. 39).

Eklund, A., Nichols, T. E., & Knutsson, H. (2016). Cluster failure: Why fmri inferences for spatial extent have inflated false-positive rates. *Proceedings of the national academy of sciences*, *113*(28), 7900–7905 (cit. on p. 37).

Esteban, O., Markiewicz, C. J., Blair, R. W., Moodie, C. A., Isik, A. I., Erramuzpe, A., Kent, J. D., Goncalves, M., DuPre, E., Snyder, M., et al. (2019). Fmriprep: A robust preprocessing pipeline for functional mri. *Nature methods*, *16*(1), 111–116 (cit. on p. 18).

European Commission and Directorate-General for Research and Innovation. (2019). *Cost-benefit analysis for fair research data: Cost of not having fair research data*. Publications Office. https://doi.org/doi/10.2777/02999 (cit. on p. 17).

Ferguson, A. R., Nielson, J. L., Cragin, M. H., Bandrowski, A. E., & Martone, M. E. (2014). Big data from small data: Data-sharing in the'long tail'of neuroscience. *Nature neuroscience*, *17*(11), 1442–1447 (cit. on p. 12).

Fiebig, F., & Lansner, A. (2017). A spiking working memory model based on hebbian short-term potentiation. *Journal of Neuroscience*, *37*(1), 83–96 (cit. on p. 13).

Finn, E. S., Scheinost, D., Finn, D. M., Shen, X., Papademetris, X., & Constable, R. T. (2017). Can brain state be manipulated to emphasize individual differences in functional connectivity? *Neuroimage*, *160*, 140–151 (cit. on p. 12).

Funahashi, S., Bruce, C. J., & Goldman-Rakic, P. S. (1989). Mnemonic coding of visual space in the monkey's dorsolateral prefrontal cortex. *Journal of neurophysiology*, *61*(2), 331–349 (cit. on pp. 12, 51).

Fuster, J. M., & Alexander, G. E. (1971). Neuron activity related to short-term memory. *Science*, *173*(3997), 652–654 (cit. on pp. 13, 51).

Garcés, P., López-Sanz, D., Maestú, F., & Pereda, E. (2017). Choice of magnetometers and gradiometers after signal space separation. *Sensors*, *17*(12), 2926 (cit. on pp. 59, 86).
Gaser, C., & Dahnke, R. (n.d.). Computational anatomy toolbox (cat). (Cit. on pp. 46, 47).
German Research Foundation. (2020). Digitaler wandel in den wissenschaften. https://doi.org/10.5281/zenodo.4191345 (cit. on pp. 17, 27, 35).
German Research Foundation. (2022). Call for proposals: Research software – quality assured and re-usable. (Cit. on p. 28).
Glatard, T., Lewis, L. B., Ferreira da Silva, R., Adalat, R., Beck, N., Lepage, C., Rioux, P., Rousseau, M.-E., Sherif, T., Deelman, E., et al. (2015). Reproducibility of neuroimaging analyses across operating systems. *Frontiers in neuroinformatics*, *9*, 12 (cit. on p. 37).
Goble, C. (2014). Better software, better research. *IEEE Internet Computing*, *18*(5), 4–8 (cit. on p. 27).
Goldman-Rakic, P. S. (1995). Cellular basis of working memory. *Neuron*, *14*(3), 477–485 (cit. on p. 12).
Gorgolewski, K. J., Alfaro-Almagro, F., Auer, T., Bellec, P., Capotă, M., Chakravarty, M. M., Churchill, N. W., Cohen, A. L., Craddock, C. R., Devenyi, G. A., et al. (2017). Bids apps: Improving ease of use, accessibility, and reproducibility of neuroimaging data analysis methods. *PLoS computational biology*, *13*(3), e1005209 (cit. on p. 18).
Gorgolewski, K. J., Auer, T., Calhoun, V. D., Craddock, C. R., Das, S., Duff, E. P., Flandin, G., Ghosh, S. S., Glatard, T., Halchenko, Y. O., et al. (2016). The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. *Scientific data*, *3*(1), 1–9 (cit. on pp. 18, 47).
Gorgolewski, K. J., & Poldrack, R. A. (2016). A practical guide for improving transparency and reproducibility in neuroimaging research. *PLoS biology*, *14*(7), e1002506 (cit. on p. 12).
Gramfort, A., Luessi, M., Larson, E., Engemann, D. A., Strohmeier, D., Brodbeck, C., Goj, R., Jas, M., Brooks, T., Parkkonen, L., & Hämäläinen, M. S. (2013). MEG and EEG Data Analysis with MNE-Python. *Frontiers in Neuroscience*, *7*(267), 1–13. https://doi.org/10.3389/fnins.2013.00267 (cit. on p. 57).
Grisham, W., Lom, B., Lanyon, L., & L. Ramos, R. (2016). Proposed training to meet challenges of large-scale data in neuroscience. *Frontiers in Neuroinformatics*, *10*, 28 (cit. on pp. 12, 21, 29).
Gronenschild, E. H., Habets, P., Jacobs, H. I., Mengelers, R., Rozendaal, N., Van Os, J., & Marcelis, M. (2012). The effects of freesurfer version, workstation type, and macintosh operating system version on anatomical volume and cortical thickness measurements. *PloS one*, *7*(6), e38234 (cit. on p. 37).
Grootswagers, T., Wardle, S. G., & Carlson, T. A. (2017). Decoding dynamic brain patterns from evoked responses: A tutorial on multivariate pattern analysis applied to time series neuroimaging data. *Journal of cognitive neuroscience*, *29*(4), 677–697 (cit. on p. 74).
Halchenko, Y. O., Meyer, K., Poldrack, B., Solanky, D. S., Wagner, A. S., Gors, J., MacFarlane, D., Pustina, D., Sochat, V., Ghosh, S. S., Mönch, C., Markiewicz, C. J., Waite, L., Shlyakhter, I., Vega, A. de la, Hayashi, S., Häusler, C. O., Poline, J.-B., Kadelka, T., . . . Hanke, M. (2021). Datalad: Distributed system for joint management of code, data, and their relationship. *Journal of Open Source Software*, *6*(63), 3262. https://doi.org/10.21105/joss.03262 (cit. on pp. 20, 23, 25).
Hanke, M., Poldrack, B., Wagner, A. S., Huijser, D., Sahoo, M., Ashish K. and Boos, Steinkamp, N., Simon R. and Guenther, & Appelhoff, S. (2021). *Datalad/datalad-osf: Cleanup* (Version 0.2.3). Zenodo. https://doi.org/10.5281/zenodo.4572455 (cit. on p. 22).

Hanke, M., Waite, L. K., Poline, J.-B., & Hutton, A. (2022). *datalad/datalad-ukbiobank: 0.3.5(.1) (Nov 06, 2022) – modernized* (Version 0.3.5). Zenodo. https://doi.org/10.5281/zenodo.7296550 (cit. on p. 47).

Hannay, J. E., MacLeod, C., Singer, J., Langtangen, H. P., Pfahl, D., & Wilson, G. (2009). How do scientists develop and use scientific software? *2009 ICSE workshop on software engineering for computational science and engineering*, 1–8 (cit. on p. 27).

Hardwicke, T. E., Mathur, M. B., MacDonald, K., Nilsonne, G., Banks, G. C., Kidwell, M. C., Hofelich Mohr, A., Clayton, E., Yoon, E. J., Henry Tessler, M., et al. (2018). Data availability, reusability, and analytic reproducibility: Evaluating the impact of a mandatory open data policy at the journal cognition. *Royal Society open science*, *5*(8), 180448 (cit. on pp. 20, 37, 39).

Hari, R., & Puce, A. (2017). *Meg-eeg primer*. Oxford University Press. https://doi.org/10.1093/med/9780190497774.001.0001 (cit. on pp. 53, 69, 75).

Haufe, S., Meinecke, F., Görgen, K., Dähne, S., Haynes, J.-D., Blankertz, B., & Bießmann, F. (2014). On the interpretation of weight vectors of linear models in multivariate neuroimaging. *Neuroimage*, *87*, 96–110 (cit. on p. 74).

Häusler, C. O. (2023). *Exploring naturalistic stimuli as an alternative to a traditional functional localizer* [Doctoral dissertation, Heinrich Heine Universtität Düsseldorf]. (Cit. on p. 66).

Haxby, J. V., Guntupalli, J. S., Connolly, A. C., Halchenko, Y. O., Conroy, B. R., Gobbini, M. I., Hanke, M., & Ramadge, P. J. (2011). A common, high-dimensional model of the representational space in human ventral temporal cortex. *Neuron*, *72*(2), 404–416 (cit. on pp. 15, 65).

Haxby, J. V., Guntupalli, J. S., Nastase, S. A., & Feilong, M. (2020). Hyperalignment: Modeling shared information encoded in idiosyncratic cortical topographies. *elife*, *9*, e56601 (cit. on p. 66).

Hess, J. (2010). git-annex. (Cit. on p. 20).

Hettrick, S. (2016). A not-so-brief history of research software engineers. https://www.software.ac.uk/blog/2016-08-17-not-so-brief-history-research-software-engineers-0 (cit. on p. 27).

Heunis, S. (2023). *Datalad/datalad-catalog: V0.2.1b - pre-release* (Version v0.2.1b). Zenodo. https://doi.org/10.5281/zenodo.7967553 (cit. on p. 90).

Horien, C., Noble, S., Greene, A. S., Lee, K., Barron, D. S., Gao, S., O'Connor, D., Salehi, M., Dadashkarimi, J., Shen, X., et al. (2021). A hitchhiker's guide to working with large, open-source neuroimaging datasets. *Nature human behaviour*, *5*(2), 185–193 (cit. on pp. 21, 41).

Hosseini, M., Powell, M., Collins, J., Callahan-Flintoft, C., Jones, W., Bowman, H., & Wyble, B. (2020). I tried a bunch of things: The dangers of unexpected overfitting in classification of brain data. *Neuroscience & Biobehavioral Reviews*, *119*, 456–467 (cit. on p. 85).

Hunt, L. T., Woolrich, M. W., Rushworth, M. F., & Behrens, T. E. (2013). Trial-type dependent frames of reference for value comparison. *PLoS computational biology*, *9*(9), e1003225 (cit. on pp. 80, 84).

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55 (cit. on p. 57).

Hunter, M., & Anstey, P. (2008). Robert boyle's 'designe about natural history'. *Early Science and Medicine*, *13*(2), 83–126. https://doi.org/https://doi.org/10.1163/157338208X263435 (cit. on p. 35).

Hutson, M. (2018). Artificial intelligence faces reproducibility crisis. (Cit. on p. 35).

Hyvarinen, A. (1999). Fast and robust fixed-point algorithms for independent component analysis. *IEEE transactions on Neural Networks*, *10*(3), 626–634 (cit. on p. 60).
Jas, M., Engemann, D. A., Bekhti, Y., Raimondo, F., & Gramfort, A. (2017). Autoreject: Automated artifact rejection for meg and eeg data. *NeuroImage*, *159*, 417–429 (cit. on pp. 57, 60).
Jas, M., Larson, E., Engemann, D. A., Leppäkangas, J., Taulu, S., Hämäläinen, M., & Gramfort, A. (2018). A reproducible meg/eeg group study with the mne software: Recommendations, quality assessments, and good practices. *Frontiers in neuroscience*, *12*, 530 (cit. on p. 39).
Jazayeri, M., & Ostojic, S. (2021). Interpreting neural computations by examining intrinsic and embedding dimensionality of neural activity. *Current opinion in neurobiology*, *70*, 113–120 (cit. on p. 15).
Kaiser, L. F., Gruendler, T. O. J., Hinrichs, H., & Jocham, G. (2018). Temporal and spatial dynamics of working memory based decision-making. (Cit. on pp. 12, 52, 61, 73, 76, 79, 80, 83, 84).
Kaiser, L. F., Gründler, T., & Jocham, G. (2016). Memento dataset (acquired within crc 779, project b16n "offline value representations in sequential decision making"). (Cit. on pp. 11, 54, 57).
Kalyani, A., Contier, O., Klemm, L., Azañon, E., Schreiber, S., Speck, O., Reichert, C., & Kuehn, E. (2023). Reduced dimension stimulus decoding and column-based modeling reveal architectural differences of primary somatosensory finger maps between younger and older adults. *bioRxiv*, 2023–01 (cit. on p. 66).
Kennedy, D. N., Abraham, S. A., Bates, J. F., Crowley, A., Ghosh, S., Gillespie, T., Goncalves, M., Grethe, J. S., Halchenko, Y. O., Hanke, M., et al. (2019). Everything matters: The repronim perspective on reproducible neuroimaging. *Frontiers in neuroinformatics*, 1 (cit. on p. 37).
Kiar, G. (2019, June 11). Technology and platforms enabling reproducible and open publishing. (Cit. on p. 29).
King, J.-R., & Dehaene, S. (2014). Characterizing the dynamics of mental representations: The temporal generalization method. *Trends in cognitive sciences*, *18*(4), 203–210 (cit. on pp. 13, 79).
King, J.-R., Gramfort, A., et al. (2018). Encoding and decoding neuronal dynamics: Methodological framework to uncover the algorithms of cognition (cit. on p. 73).
Kleiner, M., Brainard, D., & Pelli, D. (2007). What's new in psychtoolbox-3? (Cit. on p. 55).
Koehler Leman, J., Weitzner, B. D., Renfrew, P. D., Lewis, S. M., Moretti, R., Watkins, A. M., Mulligan, V. K., Lyskov, S., Adolf-Bryfogle, J., Labonte, J. W., et al. (2020). Better together: Elements of successful scientific software development in a distributed collaborative community. *PLoS computational biology*, *16*(5), e1007507 (cit. on p. 33).
Krause, D., & Thörnig, P. (2018). Jureca: Modular supercomputer at jülich supercomputing centre. *Journal of large-scale research facilities JLSRF*, *4*, A132–A132 (cit. on p. 48).
LaRocque, J. J., Lewis-Peacock, J. A., Drysdale, A. T., Oberauer, K., & Postle, B. R. (2013). Decoding attended information in short-term memory: An eeg study. *Journal of cognitive neuroscience*, *25*(1), 127–142 (cit. on p. 13).
Lee, J., & Geng, J. J. (2017). Idiosyncratic patterns of representational similarity in prefrontal cortex predict attentional performance. *Journal of Neuroscience*, *37*(5), 1257–1268 (cit. on p. 84).
Lee, S.-H., & Dan, Y. (2012). Neuromodulation of brain states. *neuron*, *76*(1), 209–222 (cit. on p. 12).
Lemaitre, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, *18*(17), 1–5. http://jmlr.org/papers/v18/16-365 (cit. on pp. 57, 74).
Lewis-Peacock, J. A., Drysdale, A. T., & Postle, B. R. (2015). Neural evidence for the flexible control of mental representations. *Cerebral Cortex*, *25*(10), 3303–3313 (cit. on p. 13).

Li, X., Ai, L., Giavasis, S., Jin, H., Feczko, E., Xu, T., Clucas, J., Franco, A., Solon Heinsfeld, A., Adebimpe, A., et al. (2021). Moving beyond processing and analysis-related variation in neuroscience. *BioRxiv*, 2021–12 (cit. on pp. 19, 37, 49).

Loggem, B. van, & Veer, G. C. van der. (2014). A documentation-centred approach to software design, development and deployment. In A. Ebert, G. C. van der Veer, G. Domik, N. D. Gershon, & I. Scheler (Eds.), *Building bridges: Hci, visualization, and non-formal modeling* (pp. 188–200). Springer Berlin Heidelberg. (Cit. on p. 33).

Machens, C. K., Romo, R., & Brody, C. D. (2010). Functional, but not anatomical, separation of "what" and "when" in prefrontal cortex. *Journal of Neuroscience*, *30*(1), 350–360 (cit. on pp. 15, 52, 83, 84).

Manninen, T., Havela, R., & Linne, M.-L. (2017). Reproducibility and comparability of computational models for astrocyte calcium excitability. *Frontiers in neuroinformatics*, *11*, 11 (cit. on p. 39).

Markiewicz, C. J., Gorgolewski, K. J., Feingold, F., Blair, R., Halchenko, Y. O., Miller, E., Hardcastle, N., Wexler, J., Esteban, O., Goncavles, M., et al. (2021). The openneuro resource for sharing of neuroscience data. *Elife*, *10*, e71774 (cit. on pp. 18, 24).

Matthews, P. M., & Sudlow, C. (2015). The uk biobank. (Cit. on pp. 19, 46).

Mazor, O., & Laurent, G. (2005). Transient dynamics versus fixed points in odor representations by locust antennal lobe projection neurons. *Neuron*, *48*(4), 661–673 (cit. on p. 15).

McKiernan, E. C., Bourne, P. E., Brown, C. T., Buck, S., Kenall, A., Lin, J., McDougall, D., Nosek, B. A., Ram, K., Soderberg, C. K., et al. (2016). How open science helps researchers succeed. *elife*, *5*, e16800 (cit. on p. 35).

Mehlenbacher, B. (2003). Documentation: Not yet implemented, but coming soon. *The HCI handbook: Fundamentals, evolving technologies, and emerging applications*, 527–543 (cit. on pp. 28, 29).

Meyers, E. M. (2018). Dynamic population coding and its relationship to working memory. *Journal of neurophysiology*, *120*(5), 2260–2268 (cit. on p. 14).

Meyers, E. M., Freedman, D. J., Kreiman, G., Miller, E. K., & Poggio, T. (2008). Dynamic population coding of category information in inferior temporal and prefrontal cortex. *Journal of neurophysiology*, *100*(3), 1407–1419 (cit. on p. 13).

Michel, J.-B., Shen, Y. K., Aiden, A. P., Veres, A., Gray, M. K., Team, G. B., Pickett, J. P., Hoiberg, D., Clancy, D., Norvig, P., et al. (2011). Quantitative analysis of culture using millions of digitized books. *science*, *331*(6014), 176–182 (cit. on p. 36).

Miller, E. K., Erickson, C. A., & Desimone, R. (1996). Neural mechanisms of visual working memory in prefrontal cortex of the macaque. *Journal of neuroscience*, *16*(16), 5154–5167 (cit. on pp. 13, 51).

Mongillo, G., Barak, O., & Tsodyks, M. (2008). Synaptic theory of working memory. *Science*, *319*(5869), 1543–1546 (cit. on p. 13).

Mons, B. (2018). *Data stewardship for open science: Implementing fair principles*. CRC Press. (Cit. on pp. 12, 16).

Mueller, K., Lepsien, J., Möller, H. E., & Lohmann, G. (2017). Commentary: Cluster failure: Why fmri inferences for spatial extent have inflated false-positive rates. *Frontiers in human neuroscience*, *11*, 345 (cit. on p. 37).

Muhle-Karbe, P. S., Myers, N. E., & Stokes, M. G. (2021). A hierarchy of functional states in working memory. *Journal of Neuroscience*, *41*(20), 4461–4475 (cit. on pp. 15, 75).

Munafò, M. R., Nosek, B. A., Bishop, D. V., Button, K. S., Chambers, C. D., Percie du Sert, N., Simonsohn, U., Wagenmakers, E.-J., Ware, J. J., & Ioannidis, J. (2017). A manifesto for reproducible science. *Nature human behaviour*, *1*(1), 1–9 (cit. on p. 35).

Murray, J. D., Bernacchia, A., Roy, N. A., Constantinidis, C., Romo, R., & Wang, X.-J. (2017). Stable population coding for working memory coexists with heterogeneous neural dynamics in prefrontal cortex. *Proceedings of the National Academy of Sciences*, *114*(2), 394–399 (cit. on pp. 15, 52, 74, 83, 84).

National Academies of Sciences, E., & Medicine. (2019). Reproducibility and replicability in science. https://doi.org/https://doi.org/10.17226/25303 (cit. on p. 36).

National Institutes of Health. (2002). Program announcement: Continued development and maintenance of bioinformatics and computational biology software. (Cit. on p. 28).

Nationale Forschungsdaten Infrastruktur. (2022). Nfdi-positionspapier zum direkten recht eines datenzugangs für die öffentliche forschung. https://www.nfdi.de/wp-content/uploads/2023/01/NFDI-Positionspapier-zum-Datenzugangsrecht-der-Forschung.pdf (cit. on p. 17).

Nichols, T. E., Das, S., Eickhoff, S. B., Evans, A. C., Glatard, T., Hanke, M., Kriegeskorte, N., Milham, M. P., Poldrack, R. A., Poline, J.-B., et al. (2017). Best practices in data analysis and sharing in neuroimaging using mri. *Nature neuroscience*, *20*(3), 299–303 (cit. on p. 19).

Niso, G., Botvinik-Nezer, R., Appelhoff, S., De La Vega, A., Esteban, O., Etzel, J. A., Finc, K., Ganz, M., Gau, R., Halchenko, Y. O., Herholz, P., Karakuzu, A., Keator, D. B., Markiewicz, C. J., Maumet, C., Pernet, C. R., Pestilli, F., Queder, N., Schmitt, T., ... Rieger, J. W. (2022). Open and reproducible neuroimaging: From study inception to publication. *NeuroImage*, *263*, 119623. https://doi.org/https://doi.org/10.1016/j.neuroimage.2022.119623 (cit. on pp. 19, 35).

Niso, G., Gorgolewski, K. J., Bock, E., Brooks, T. L., Flandin, G., Gramfort, A., Henson, R. N., Jas, M., Litvak, V., T Moreau, J., et al. (2018). Meg-bids, the brain imaging data structure extended to magnetoencephalography. *Scientific data*, *5*(1), 1–5 (cit. on p. 57).

Niso, G., Rogers, C., Moreau, J. T., Chen, L.-Y., Madjar, C., Das, S., Bock, E., Tadel, F., Evans, A. C., Jolicoeur, P., et al. (2016). Omega: The open meg archive. *Neuroimage*, *124*, 1182–1187 (cit. on p. 12).

Nobre, A. C. (2022). Opening questions in visual working memory. *Journal of Cognitive Neuroscience*, *35*(1), 49–59 (cit. on pp. 51, 52).

Nobre, A. C., & Stokes, M. G. (2011). Attention and short-term memory: Crossroads. (cit. on p. 13).

Nord, M., & Verbeeck, J. (2019). Towards reproducible and transparent science of (big) electron microscopy data using version control. *Microscopy and Microanalysis*, *25*(S2), 232–233 (cit. on p. 20).

Oldfield, R. C. (1971). The assessment and analysis of handedness: The edinburgh inventory. *Neuropsychologia*, *9*(1), 97–113 (cit. on p. 54).

Open Science Collaboration. (2015). Estimating the reproducibility of psychological science. *Science*, *349*(6251), aac4716 (cit. on p. 35).

Palumbo, L., Bosco, P., Fantacci, M., Ferrari, E., Oliva, P., Spera, G., & Retico, A. (2019). Evaluation of the intra-and inter-method agreement of brain mri segmentation software packages: A comparison between spm12 and freesurfer v6. 0. *Physica Medica*, *64*, 261–272 (cit. on p. 37).

Papadiamantis, A. G., Klaessig, F. C., Exner, T. E., Hofer, S., Hofstaetter, N., Himly, M., Williams, M. A., Doganis, P., Hoover, M. D., Afantitis, A., et al. (2020). Metadata stewardship in nanosafety research: Community-driven organisation of metadata schemas to support fair nanoscience data. *Nanomaterials*, *10*(10), 2033 (cit. on p. 38).

Parnas, D. L. (2011). Precise documentation: The key to better software. In S. Nanz (Ed.), *The future of software engineering* (pp. 125–148). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-15187-3_8 (cit. on p. 28).

Parsons, T., Grimshaw, S., & Williamson, L. (2013). Research data management survey: Report. (Cit. on p. 20).

Pauli, R., Bowring, A., Reynolds, R., Chen, G., Nichols, T. E., & Maumet, C. (2016). Exploring fmri results space: 31 variants of an fmri analysis in afni, fsl, and spm. *Frontiers in neuroinformatics*, *10*, 24 (cit. on p. 37).

Pavlov, Y. G., & Kotchoubey, B. (2022). Oscillatory brain activity and maintenance of verbal and visual working memory: A systematic review. *Psychophysiology*, *59*(5), e13735 (cit. on p. 51).

Pawlik, A., Segal, J., & Petre, M. (2012). Documentation practices in scientific software development. *2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering (CHASE)*, 113–119 (cit. on p. 28).

Pawlik, A., Segal, J., Sharp, H., & Petre, M. (2014). Crowdsourcing scientific software documentation: A case study of the numpy documentation project. *Computing in Science & Engineering*, *17*(1), 28–36 (cit. on p. 33).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*, 2825–2830 (cit. on p. 57).

Peng, R. D., Dominici, F., & Zeger, S. L. (2006). Reproducible epidemiologic research. *American Journal of Epidemiology*, *163*(9), 783–789. https://doi.org/10.1093/aje/kwj093 (cit. on p. 36).

Pernet, C., Garrido, M. I., Gramfort, A., Maurits, N., Michel, C. M., Pang, E., Salmelin, R., Schoffelen, J. M., Valdes-Sosa, P. A., & Puce, A. (2020). Issues and recommendations from the ohbm cobidas meeg committee for reproducible eeg and meg research. *Nature neuroscience*, *23*(12), 1473–1483 (cit. on p. 19).

Pfurtscheller, G., Scherer, R., Müller-Putz, G., & Lopes da Silva, F. (2008). Short-lived brain state after cued motor imagery in naive subjects. *European Journal of Neuroscience*, *28*(7), 1419–1426 (cit. on p. 12).

Poldrack, B., Wagner, A. S., Q., W. A., Waite, L. K., & Hanke, M. (2021). A model implementation of a scalable data store for scientific computing with datalad [version 1; not peer reviewed]. https://doi.org/https://doi.org/10.7490/f1000research.1118494.1 (cit. on p. 43).

Poldrack, R. A., Baker, C. I., Durnez, J., Gorgolewski, K. J., Matthews, P. M., Munafò, M. R., Nichols, T. E., Poline, J.-B., Vul, E., & Yarkoni, T. (2017). Scanning the horizon: Towards transparent and reproducible neuroimaging research. *Nature reviews neuroscience*, *18*(2), 115–126 (cit. on p. 35).

Poline, J.-B., Breeze, J., Ghosh, S., Gorgolewski, K., Halchenko, Y., Hanke, M., Helmer, K., Marcus, D., Poldrack, R., Schwartz, Y., Ashburner, J., & Kennedy, D. (2012). Data sharing in neuroimaging research. *Frontiers in Neuroinformatics*, *6*. https://doi.org/10.3389/fninf.2012.00009 (cit. on p. 19).

Portegies Zwart, S. (2020). The ecological impact of high-performance computing in astrophysics. *Nature Astronomy*, *4*(9), 819–822 (cit. on p. 42).

Puce, A., & Hämäläinen, M. S. (2017). A review of issues related to data acquisition and analysis in eeg/meg studies. *Brain sciences*, *7*(6), 58 (cit. on pp. 12, 53).

Rainer, G., & Miller, E. K. (2000). Neural ensemble states in prefrontal cortex identified using a hidden markov model with a modified em algorithm. *Neurocomputing*, *32*, 961–966 (cit. on p. 15).

Rigotti, M., Barak, O., Warden, M. R., Wang, X.-J., Daw, N. D., Miller, E. K., & Fusi, S. (2013). The importance of mixed selectivity in complex cognitive tasks. *Nature*, *497*(7451), 585–590 (cit. on p. 13).

Roux, F., & Uhlhaas, P. J. (2014). Working memory and neural oscillations: Alpha–gamma versus theta–gamma codes for distinct wm information? *Trends in cognitive sciences*, *18*(1), 16–25 (cit. on p. 51).

Roy, S., Rathee, D., Chowdhury, A., McCreadie, K., & Prasad, G. (2020). Assessing impact of channel selection on decoding of motor and cognitive imagery from meg data. *Journal of Neural Engineering*, *17*(5), 056037 (cit. on p. 86).

San Gil, I., Baker, K., Campbell, J., Denny, E. G., Vanderbilt, K., Riordan, B., Koskela, R., Downing, J., Grabner, S., Melendez, E., et al. (2009). The long-term ecological research community metadata standardisation project: A progress report. *International Journal of Metadata, Semantics and Ontologies*, *4*(3), 141–153 (cit. on p. 39).

Sandve, G. K., Nekrutenko, A., Taylor, J., & Hovig, E. (2013). Ten simple rules for reproducible computational research. *PLoS computational biology*, *9*(10), e1003285 (cit. on p. 20).

Santhanam, G., Yu, B. M., Gilja, V., Ryu, S. I., Afshar, A., Sahani, M., & Shenoy, K. V. (2009). Factor-analysis methods for higher-performance neural prostheses. *Journal of neurophysiology*, *102*(2), 1315–1330 (cit. on p. 15).

scikit-learn. (2023). Scikit-learn software documentation version 1.3.0 [Last accessed 28 July 2023]. https://scikit-learn.org/dev/modules/linear_model.html#logistic-regression (cit. on p. 61).

Segal, J. (2007). Some problems of professional end user developers. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*, 111–118 (cit. on p. 28).

Seidemann, E., Meilijson, I., Abeles, M., Bergman, H., & Vaadia, E. (1996). Simultaneously recorded single units in the frontal cortex go through sequences of discrete and stable states in monkeys performing a delayed localization task. *The Journal of Neuroscience*, *16*(2), 752 (cit. on p. 15).

Silberzahn, R., Uhlmann, E. L., Martin, D. P., Anselmi, P., Aust, F., Awtrey, E., Bahník, Š., Bai, F., Bannard, C., Bonnier, E., Carlsson, R., Cheung, F., Christensen, G., Clay, R., Craig, M. A., Dalla, A., Rosa, Dam, L., Evans, M. H., Flores Cervantes, I., ... Nosek, B. A. (2018). Many analysts, one data set: Making transparent how variations in analytic choices affect results. *Advances in Methods and Practices in Psychological Science*, *1*(3), 337–356. https://doi.org/10.1177/2515245917747646 (cit. on p. 19).

Šoškić, A., Jovanović, V., Styles, S. J., Kappenman, E. S., & Ković, V. (2022). How to do better n400 studies: Reproducibility, consistency and adherence to research standards in the existing literature. *Neuropsychology Review*, *32*(3), 577–600 (cit. on p. 19).

Sreenivasan, K. K., Curtis, C. E., & D'Esposito, M. (2014). Revisiting the role of persistent neural activity during working memory. *Trends in cognitive sciences*, *18*(2), 82–89 (cit. on p. 14).

Sreenivasan, K. K., & D'Esposito, M. (2019). The what, where and how of delay activity. *Nature reviews neuroscience*, *20*(8), 466–481 (cit. on pp. 11, 51, 86).

Stelzer, J., Chen, Y., & Turner, R. (2013). Statistical inference and multiple testing correction in classification-based multi-voxel pattern analysis (mvpa): Random permutations and cluster size control. *Neuroimage*, *65*, 69–82 (cit. on p. 80).

Stokes, M. G. (2015). 'activity-silent'working memory in prefrontal cortex: A dynamic coding framework. *Trends in cognitive sciences*, *19*(7), 394–405 (cit. on p. 13).

Stokes, M. G., Kusunoki, M., Sigala, N., Nili, H., Gaffan, D., & Duncan, J. (2013). Dynamic coding for cognitive control in prefrontal cortex. *Neuron*, *78*(2), 364–375 (cit. on p. 13).

Strupler, N., & Wilkinson, T. C. (2017). Reproducibility in the field: Transparency, version control and collaboration on the project panormos survey. *Open Archaeology*, *3*(1), 279–304 (cit. on p. 20).

Styles, S. J., Ković, V., Ke, H., & Šoškić, A. (2021). Towards artem-is: Design guidelines for evidence-based eeg methodology reporting tools. *NeuroImage*, *245*, 118721 (cit. on p. 19).

Swarts, J. (2019). Open-source software in the sciences: The challenge of user support. *Journal of business and technical communication*, *33*(1), 60–90 (cit. on p. 29).

Tange, O., et al. (2011). Gnu parallel-the command-line power tool. *The USENIX Magazine*, *36*(1), 42–47 (cit. on p. 48).

Taulu, S., Simola, J., & Kajola, M. (2005). Applications of the signal space separation method. *IEEE Transactions on Signal Processing*, *53*(9), 3359–3372. https://doi.org/10.1109/TSP.2005.853302 (cit. on p. 58).

Taulu, S., & Hari, R. (2009). Removal of magnetoencephalographic artifacts with temporal signal-space separation: Demonstration with single-trial auditory-evoked responses. *Human brain mapping*, *30*(5), 1524–1534 (cit. on p. 59).

Taulu, S., & Kajola, M. (2005). Presentation of electromagnetic multichannel data: The signal space separation method. *Journal of Applied Physics*, *97*(12), 124905 (cit. on p. 58).

Taulu, S., & Simola, J. (2006). Spatiotemporal signal space separation method for rejecting nearby interference in meg measurements. *Physics in Medicine & Biology*, *51*(7), 1759 (cit. on pp. 58, 59).

Thain, D., Tannenbaum, T., & Livny, M. (2005). Distributed computing in practice: The condor experience. *Concurrency and computation: practice and experience*, *17*(2-4), 323–356 (cit. on p. 46).

Thanos, C. (2017). Research data reusability: Conceptual foundations, barriers and enabling technologies. *Publications*, *5*(1), 2 (cit. on p. 37).

The pandas development team. (n.d.). *pandas-dev/pandas: Pandas*. https://github.com/pandas-dev/pandas (cit. on p. 57).

The Turing Way Community. (2022). *The Turing Way: A handbook for reproducible, ethical and collaborative research* (Version 1.0.2). Zenodo. https://doi.org/10.5281/zenodo.7625728 (cit. on p. 33).

Theunissen, T., Heesch, U., & Avgeriou, P. (2022). A mapping study on documentation in continuous software development. *Information and Software Technology*, *142*, 106733. https://doi.org/10.1016/j.infsof.2021.106733 (cit. on p. 28).

Trübutschek, D., Marti, S., Ojeda, A., King, J.-R., Mi, Y., Tsodyks, M., & Dehaene, S. (2017). A theory of working memory without consciousness or sustained activity. *elife*, *6*, e23871 (cit. on p. 13).

Turner, B. O., Paul, E. J., Miller, M. B., & Barbey, A. K. (2018). Small sample sizes reduce the replicability of task-based fmri studies. *Communications Biology*, *1*(1), 62 (cit. on p. 19).

UK Research and Innovation. (2021). Funding opportunity: Software for research communities. (Cit. on p. 28).

Van Essen, D. C., Smith, S. M., Barch, D. M., Behrens, T. E., Yacoub, E., Ugurbil, K., Consortium, W.-M. H., et al. (2013). The wu-minn human connectome project: An overview. *Neuroimage*, *80*, 62–79 (cit. on p. 19).

Van Horn, J. D., & Toga, A. W. (2014). Human neuroimaging as a "big data" science. *Brain imaging and behavior*, *8*, 323–331 (cit. on p. 19).

Varoquaux, G., Raamana, P. R., Engemann, D. A., Hoyos-Idrobo, A., Schwartz, Y., & Thirion, B. (2017). Assessing and tuning brain decoders: Cross-validation, caveats, and guidelines [Individual Subject Prediction]. *NeuroImage*, *145*, 166–179. https://doi.org/https://doi.org/10.1016/j.neuroimage.2016.10.038 (cit. on pp. 61, 76).

Vidaurre, D., Quinn, A. J., Baker, A. P., Dupret, D., Tejero-Cantero, A., & Woolrich, M. W. (2016). Spectrally resolved fast transient brain states in electrophysiological data. *Neuroimage*, *126*, 81–95 (cit. on p. 15).

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., . . . SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2 (cit. on p. 57).

Wagner, A. S., Poline, J.-B., & Hanke, M. (2021). A pragmatic approach to reusable research outputs (poster) [version 1; not peer reviewed]. https://doi.org/https://doi.org/10.7490/f1000research.1118575.1 (cit. on p. 38).

Wagner, A. S., Waite, L. K., Meyer, K., Heckner, M. K., Kadelka, T., Reuter, N., Waite, A. Q., Poldrack, B., Markiewicz, C. J., Halchenko, Y. O., et al. (2020). The datalad handbook. *Zenodo*, *10*. https://doi.org/10.5281/zenodo.3608611 (cit. on p. 27).

Wagner, A. S., Waite, L. K., Waite, A. Q., Reuter, N., Poldrack, B., Poline, J.-B., Kadelka, T., Markiewicz, C. J., Vavra, P., Paas, L. K., Herholz, P., Mochalski, L. N., Kraljevic, N., Heckner, Y. O., Marisa K. and Halchenko, & Hanke, M. (2020). The DataLad Handbook: A user-focused and workflow- based addition to standard software documentation. https://doi.org/10.5281/zenodo.7906718 (cit. on p. 30).

Wagner, A. S., Waite, L. K., Wierzba, M., Hoffstaedter, F., Waite, A. Q., Poldrack, B., Eickhoff, S. B., & Hanke, M. (2022). Fairly big: A framework for computationally reproducible processing of large-scale data. *Scientific data*, *9*(1), 80 (cit. on pp. 41, 42, 44, 45, 47).

Wagner, A. S., Maumet, C., Ganz, M., & Praag, C. G. van. (2023). 10 years of reproducibility in biomedical research: How can we achieve generalizability and fairness? *20th IEEE International Symposium on Biomedical Imaging (ISBI 2023)* (cit. on p. 35).

Waite, L. K., Waite, A. Q., & Hanke, M. (2022). A data steward walks into a bar: Experiences managing open and restricted brain imaging data. http://hdl.handle.net/2128/31449 (cit. on p. 19).

Wallis, J. D., Anderson, K. C., & Miller, E. K. (2001). Single neurons in prefrontal cortex encode abstract rules. *Nature*, *411*(6840), 953–956 (cit. on p. 13).

Waskom, M. L. (2021). Seaborn: Statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. https://doi.org/10.21105/joss.03021 (cit. on p. 57).

White, I. M., & Wise, S. P. (1999). Rule-dependent neuronal activity in the prefrontal cortex. *Experimental brain research*, *126*, 315–335 (cit. on p. 13).

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., Silva Santos, L. B. da, Bourne, P. E., et al. (2016). The fair guiding principles for scientific data management and stewardship. *Scientific data*, *3*(1), 1–9 (cit. on pp. 16, 17, 37, 38).

Winkler, I., Debener, S., Müller, K.-R., & Tangermann, M. (2015). On the influence of high-pass filtering on ica-based artifact reduction in eeg-erp. *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 4101–4105. https://doi.org/10.1109/EMBC.2015.7319296 (cit. on p. 60).

Wise, J., Barron, A. G. de, Splendiani, A., Balali-Mood, B., Vasant, D., Little, E., Mellino, G., Harrow, I., Smith, I., Taubert, J., et al. (2019). Implementation and relevance of fair data principles in biopharmaceutical r&d. *Drug discovery today*, *24*(4), 933–938 (cit. on p. 38).

Wolff, M. J., Jochim, J., Akyürek, E. G., & Stokes, M. G. (2017). Dynamic hidden states underlying working-memory-guided behavior. *Nature neuroscience*, *20*(6), 864–871 (cit. on pp. 13, 84).

Xie, T., Cheong, J. H., Manning, J. R., Brandt, A. M., Aronson, J. P., Jobst, B. C., Bujarski, K. A., & Chang, L. J. (2021). Minimal functional alignment of ventromedial prefrontal cortex intracranial eeg signals during naturalistic viewing. *bioRxiv*, 2021–05 (cit. on p. 86).

Xoxa, N., Mehilli, A., Tafa, I., & Fejzaj, J. (2015). Implementations of file locking mechanisms, linux and windows. *2015 12th International Conference on Information Technology-New Generations*, 756–757 (cit. on p. 43).

Yoo, A. B., Jette, M. A., & Grondona, M. (2003). Slurm: Simple linux utility for resource management. *Job Scheduling Strategies for Parallel Processing: 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003. Revised Paper 9*, 44–60 (cit. on p. 46).

Zehl, L., Jaillet, F., Stoewer, A., Grewe, J., Sobolev, A., Wachtler, T., Brochier, T. G., Riehle, A., Denker, M., & Grün, S. (2016). Handling metadata in a neurophysiology laboratory. *Frontiers in neuroinformatics*, *10*, 26 (cit. on p. 41).

Zhao, C., Jarecka, D., Covitz, S., Chen, Y., Eickhoff, S. B., Fair, D. A., Franco, A. R., Halchenko, Y. O., Hendrickson, T. J., Hoffstaedter, F., et al. (2023). A reproducible and generalizable software workflow for analysis of large-scale neuroimaging data collections using bids apps. *bioRxiv*, 2023–08 (cit. on p. 90).

Eidesstattliche Erklärung

Ich versichere an Eides Statt, dass die Dissertation von mir selbständig und ohne unzulässige fremde Hilfe unter Beachtung der „Grundsätze zur Sicherung guter wissenschaftlicher Praxis an der Heinrich-Heine-Universität Düsseldorf“ erstellt worden ist.

Düsseldorf, 25.10.2023

Erklärung über bisherige Promotionsversuche

Ich erkläre hiermit, dass die vorliegende Dissertation keiner anderen Fakultät vorgelegt wurde, und es sich um meinen ersten Promotionsversuch handelt.

Düsseldorf, 25.10.2023

A Curriculum Vitae

Personalien

Name	Adina Svenja Wagner geboren am 09.04.1994 in Wolfsburg ledig, deutsch

Schulbildung

2012	Abitur, Phoenix Gymnasium Vorsfelde. Abschlussnote: 1.1

Studium

10/12 - 04/16	Technische Universität Braunschweig, BSc Psychologie. Abschlussnote: 1.2
	Thema der Bachelorarbeit: "Disentangling Loneliness: A Prospective Study on Differential Effects of Loneliness Measures on Health in Adults in their Second Half of Life."
10/16 - 02/19	Otto von Guericke Universität Magdeburg, MSc Psychologie - Klinische Neurowissenschaften. Abschlussnote: 1.1
	Thema der Masterarbeit: "Catching the eye - Investigating the functional neuroanatomy of visuospatial attention with fMRI and eyegaze recordings during natural stimulation."

Promotion	Heinrich Heine Universität Düsseldorf, Institut für experimentelle Psychologie
04/19 - heute	Forschungszentrum Jülich, Institut für Neurowissenschaften und Medizin: Gehirn und Verhalten (INM-7)

October 16, 2023

B Publications

The following publications of mine formed the basis of this thesis.

Halchenko, Y. O., Meyer, K., Poldrack, B., Solanky, D. S., Wagner, A. S., Gors, J., MacFarlane, D., Pustina, D., Sochat, V., Ghosh, S. S., Mönch, C., Markiewicz, C. J., Waite, L., Shlyakhter, I., Vega, A. de la, Hayashi, S., Häusler, C. O., Poline, J.-B., Kadelka, T., . . . Hanke, M. (2021). Datalad: Distributed system for joint management of code, data, and their relationship. *Journal of Open Source Software*, *6*(63), 3262. https://doi.org/10.21105/joss.03262 (cit. on p. 110).

Wagner, A. S., Waite, L. K., Meyer, K., Heckner, M. K., Kadelka, T., Reuter, N., Waite, A. Q., Poldrack, B., Markiewicz, C. J., Halchenko, Y. O., et al. (2020). The datalad handbook. *Zenodo*, *10*. https://doi.org/10.5281/zenodo.3608611

Wagner, A. S., Waite, L. K., Wierzba, M., Hoffstaedter, F., Waite, A. Q., Poldrack, B., Eickhoff, S. B., & Hanke, M. (2022). Fairly big: A framework for computationally reproducible processing of large-scale data. *Scientific data*, *9*(1), 80 (cit. on p. 110).

My contributions to these works are as follows:

1) Wagner A.S, Waite L.K., Meyer, K., Heckner, M.K., Kadelka, T., Reuter, N., Waite, A.Q., Poldrack, B., Markiewicz, C.J., Halchenko, Y.O., Vavra, P., Chormai, P., Poline, J-B., Paas, L-K., Herholz, P., Mochalski, L.N., Kraljevic, N., Wiersch, L., Hutton, A. & Hanke, M. (2020). The DataLad Handbook. Zenodo, 10. https://doi.org/10.5281/zenodo.3608611.

I conceived the conceptual structure and educational concept, co-created the technical backbone, coordinated development efforts and community management, and wrote most of its contents. I have also further refined the contents and its layout, and published it as a physical book (ISBN: 979-8-85-703797-3).

2) Halchenko, Y.O., Meyer, K., Poldrack, B., Solanky, S., Wagner, A.S, Gors, J., MacFarlane, D., Pustina, D., Sochat, V., Gosh, S.S., Mönch, C., Markiewicz, C.J., Waite, L.K., Shlyakhter, I., de la Vega, A., Hayashi, S., Häusler, C.O., Poline, J-B., Kadelka, T., Skyten, K., Jareka, D., Kennedy, D., Strauss, T., Cieslak, M., Ioanas, H.I., Vavra, P., Schneider, R., Pflüger, M., Haxby, J.V., Eickhoff, S.B., & Hanke, M. (2021). DataLad: Distributed system for joint management of code, data, and their relationship. Journal of Open Source Software, 6(63). 3262. https://doi.org/10.21105/joss.03262

I contributed to the conceptualization, writing, and editing of the manuscript, and have been a regular contributor to the underlying software since 2019. With the exception of the first and last author, the order of authors was determined by the amount of committed code contributions.

3) Wagner, A.S., Waite, L.K., Wierzba, M., Hoffstaedter, F., Waite, A.Q., Poldrack, B., Eickhoff, S.B. & Hanke, M. (2022). Fairly big: A framework for computationally reproducible processing of large-scale data. Scientific data, 9(1), 80

I co-conceived the setup of the framework, co-piloted and documented an earlier version of the framework with a smaller dataset, co-implemented the open workflow, wrote the first draft of the manuscript, and contributed to the conceptualization, writing, and editing of the manuscript.

The original publications from Halchenko et al. (2021) and Wagner, Waite, Wierzba, et al. (2022) are included hereafter.

scientific data

Check for updates

OPEN

ARTICLE

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

Adina S. Wagner[1,4]✉, Laura K. Waite[1,4], Małgorzata Wierzba[1,2,4], Felix Hoffstaedter[1], Alexander Q. Waite[1], Benjamin Poldrack[1], Simon B. Eickhoff[1,3] & Michael Hanke[1,3]

Large-scale datasets present unique opportunities to perform scientific investigations with unprecedented breadth. However, they also pose considerable challenges for the findability, accessibility, interoperability, and reusability (FAIR) of research outcomes due to infrastructure limitations, data usage constraints, or software license restrictions. Here we introduce a DataLad-based, domain-agnostic framework suitable for reproducible data processing in compliance with open science mandates. The framework attempts to minimize platform idiosyncrasies and performance-related complexities. It affords the capture of machine-actionable computational provenance records that can be used to retrace and verify the origins of research outcomes, as well as be re-executed independent of the original computing infrastructure. We demonstrate the framework's performance using two showcases: one highlighting data sharing and transparency (using the studyforrest.org dataset) and another highlighting scalability (using the largest public brain imaging dataset available: the UK Biobank dataset).

Introduction

The amount of data available to researchers has steadily grown, but over the past decade, a focus on diverse, representative samples has resulted in datasets of unprecedented size. The Wind Integration National Dataset (WIND) Toolkit[1], CERN data (opendata.cern.ch), or NASA Earth data (earthdata.nasa.gov) are only some of the prominent examples of large, openly shared datasets across scientific disciplines. This development is accompanied by a growing awareness of the importance to make the data more findable, accessible, interoperable, and reusable (FAIR)[2], and increasing availability of research standards and tools that facilitate data sharing and management[3].

Though large-scale datasets present unique research opportunities, they also constitute immense challenges. Storage and computational demands strain the capabilities of even well-endowed research institutions' high-performance compute (HPC) infrastructure — rendering the analysis of these datasets unaffordable using methods common in fields accustomed to smaller datasets (e.g. multiple copies of the data, computationally inefficient processing). With the growing complexity of handling large scale datasets, the trustworthiness of derivative data can be at stake as large-scale computations are more difficult to reproduce, comprehend, and verify. Yet especially in the case of large scale datasets, sharing and reusing data derivatives emerges as the most — or sometimes the only — viable way to extend previous work[4]. It minimizes duplicate efforts to perform resource-heavy, costly computations that also have considerable environmental impact[5], and it can open up research on large data to scholars who do not have access to adequate computational resources. In such contexts, data should thus not only be as FAIR as possible, but also handled in a sustainable manner that places data sharing and reuse as a priority.

The challenges of big data are particularly relevant to the life sciences, such as neuroscience or genetics, where datasets scale to millions of files, hundreds of terabytes[6,7], acquired from tens of thousands of participants. Well known examples, such as the Human Connectome Project[8], the Adolescent Brain Cognitive Development

[1]Institute of Neuroscience and Medicine, Brain & Behaviour (INM-7), Research Center Jülich, 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, Düsseldorf, Germany. [4]These authors contributed equally: Adina S. Wagner, Laura K. Waite, Małgorzata Wierzba. ✉e-mail: adina.wagner@t-online.de

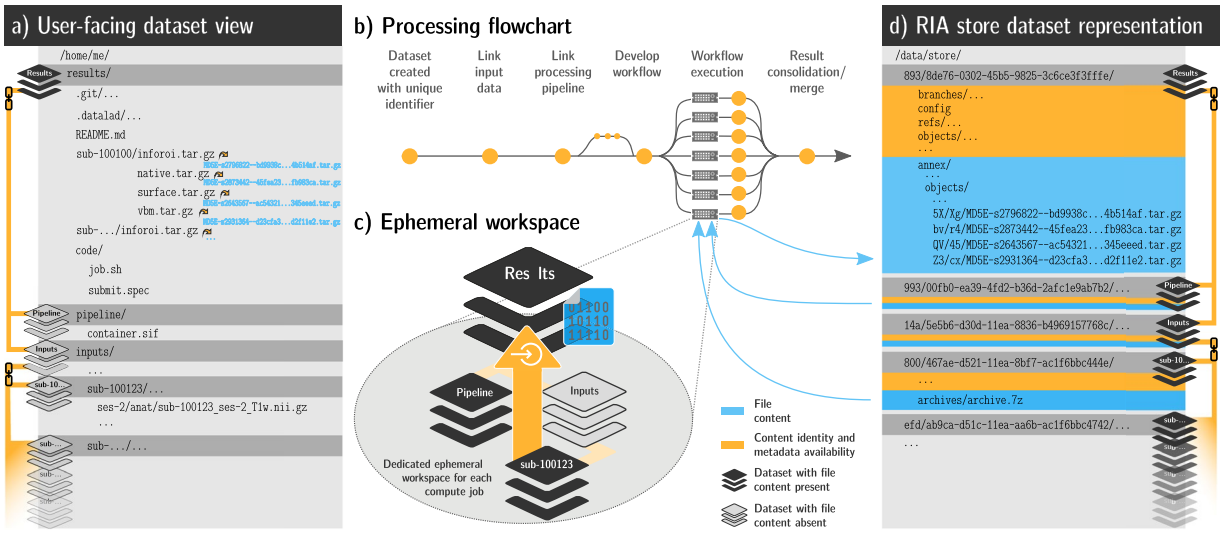


Fig. 1 Schematic overview of the processing framework. (**a**) The user-facing representation of the results on a file system after completed processing: A lean DataLad dataset that tracks the computed results, links input data and pipeline, and contains actionable process provenance and location information, allowing on-demand file retrieval or recomputation. Depicted files are from the UK Biobank showcase. (**b**) Process-flowchart: First, a DataLad dataset links required processing components (e.g., input data, processing pipeline, additional scripts). Next, compute jobs are executed, if possible in parallel. Afterwards, results and provenance are aggregated (merged). (**c**) An ephemeral (short-lived) compute workspace: Each compute job creates a temporary, lean *clone*, which *retrieves* only relevant subsets of data, and *captures* the processing execution as provenance. After completion, results and provenance are *pushed* into permanent storage (see d), and the ephemeral workspace is purged. (**d**) The internal dataset representation in a RIA store: The store receives results and can contain input data, optionally using compressed archives (for reduced disk space/inode consumption) or encryption during storage and transport. It is the only place where results take up permanent disk space. If inputs are available from other infrastructure (external, web-accessible servers, cloud infrastructure), jobs can obtain them from registered sources, removing the need for duplicate storage of input data.

Study (ABCD)[9], or the UK Biobank (UKB) project[10], contain diverse data ranging from brain imaging to genetics to clinical and non-clinical measures.

In addition, computational processing of biomedical datasets is rarely fully transparent. Often, datasets contain personal data, which imposes usage constraints and prohibits the open distribution of data. Thus, handling these datasets can only be as open as the responsible use of sensitive data permits. Moreover, common processing pipelines possess considerable analytical flexibility, and many tools commonly used in biomedical research rely on proprietary software[11], which cannot be easily shared or accessed by others. This threatens the reproducibility of results[12], and their *digital provenance* — information about how tools, data, and actors were involved in the generation of a file — is often incomplete. As data processing results often multiply storage demands, the just-keep-everything data management approach is rendered increasingly prohibitive. This fact further impedes the possibility to retrace and verify the origin and provenance of research outcomes fully and transparently[13], and hence limits the trustworthiness of the research process and its outcomes[2].

Here, we present a portable, free and open source framework — built on DataLad[14] and containerization software[15] — to reproducibly process large-scale datasets. It empowers independent consumers to verify or reproduce the results based on *machine-actionable* (i.e., machine-readable, automatically re-executable) records of computational provenance, in an infrastructure-agnostic fashion. The framework capitalizes on established technology, used in conjunction with workflows from software development and workload management. Two use cases demonstrate different framework features and its scalability: 1) an application of a MATLAB-based, containerized, neuroimaging processing pipeline on big data from the UKB project[16] (comprising 76 TB in 43 million files under strict usage constraints), and 2) a showcase implementation with openly available processing pipeline and data that illustrates the framework's potential for transparent sharing and reuse of reproducible derivatives. While one can apply the framework by following the description in this work, a bootstrapping script for each use case is provided that — given input dataset and processing pipeline — performs the necessary setup from scratch.

Results

The proposed framework employs a range of software tools for data, code, and computation management to apply workflows from software engineering — in particular distributed development — to computational research. Specifically, it orchestrates arbitrary data processing via a lean network of interconnected, but self-sufficient workspaces while optimizing for portability, scalability, and automatic computational reproducibility.

To achieve this, our framework combines a range of open source software tools — distributed version control systems, containerization software, job scheduling tools, and storage solutions with optional encryption and compression — into a sequential workflow. A complete, schematic overview is depicted in Fig. 1 and basic DataLad concepts are summarized in Box 1. Three key features of this data management solution are central to the framework:

- Comprehensive data structure to track all elements of digital processing
- Computation in automatically bootstrapped ephemeral workspaces
- Process provenance capture in machine-actionable records

Box 1 Main concepts about the design and function of the framework, DataLad, and its underlying technical components. DataLad, integral to the processing framework, is a domain agnostic data management solution based on Git (git-scm.com) and git-annex[18]. It provides standard interfaces for arbitrary data transport methods, comprehensive process provenance capture for computational reproducibility, and the means to apply proven workflows from collaborative software development to the domain of data processing. More information on DataLad is available at datalad.org and handbook.datalad.org[40]

DataLad concepts

DataLad dataset DataLad's core data structure is the *dataset*. On a technical level, it is a joint Git/git-annex[18] repository. Conceptually, it is an overlay data structure that is particularly suited to address data integration challenges. It enables users to version control files of any size or type, track and transport files in a distributed network of dataset *clones*, as well as record and re-execute actionable process provenance on the genesis of file content. DataLad datasets have the ability to retrieve or drop registered, remote file content on demand with single file granularity. This is possible based on a lean record of file identity and file availability (checksum and URLs) irrespective of the true file size. A user does not need to be aware of the actual download source of a file's content, as precise file identity is automatically verified regardless of a particular retrieval method, and the specification of redundant sources is supported. These technical features enable the implementation of infrastructure-agnostic data retrieval and deposition logic in user code.

A clone (Git concept) is a copy of a DataLad dataset that is linked to its origin dataset and its history. The clones are lightweight and can typically be obtained within seconds, as they are primarily comprised of file identity and availability records. DataLad enables synchronization of content between clones and, hence, the propagation of updates.

A branch (Git concept) is an independent segment of a DataLad dataset's history. It enables the separation of parallel developments based on a common starting point. Branches can encompass arbitrarily different modifications of a dataset. In a typical collaborative development or parallel processing routine, changes are initially introduced in branches and are later consolidated by merging them into a mainline branch.

Nesting A DataLad dataset can also contain other DataLad datasets. Analog to file content, this linkage is implemented using a lightweight dataset identity and availability record (based on Git's submodules). This nesting enables flexible (re-)use of datasets in a different context. For example, it allows for the composition of a project directory from precisely versioned, modular units that unambiguously link all inputs of a project to its outcomes. Nesting offers actionable dataset linkage at virtually no disk space cost, while providing the same on-demand retrieval and deposition convenience as for file content operations because DataLad can work with a hierarchy of nested datasets as if they are a single monolithic repository. When a DataLad dataset B is nested inside DataLad dataset A, we also refer to A as the *superdataset* and to B as a *subdataset*. A superdataset can link any number of subdatasets, and datasets can simultaneously be both super- and subdataset.

RIA store A file-system based store for DataLad datasets with minimal server-side software requirements (in particular no DataLad, no git-annex, and Git only for specific optional features)[25]. These stores offer inode minimization (using indexed 7-zip archives). A dataset of arbitrary size and number of files can be hosted while consuming fewer than 25 inodes, while nevertheless offering random read access to individual files at a low and constant latency independent of the actual archive size. Combined with optional file content encryption and compression, RIA ("Remote Indexed Archive") stores are particularly suited for staging large-scale, sensitive data to process on HPC resources.

DataLad extension The core DataLad software is extensible via independently developed Python packages. We developed a custom extension, `datalad-ukbiobank`[41] (http://docs.datalad.org/projects/ukbiobank-docs.datalad.org/projects/ukbiobank), to use the UK Biobank (UKB) as a data source for reproducible research. This extension equips DataLad with a set of commands to obtain, monitor, and restructure the UKB imaging data release. UKB data are tracked in DataLad datasets that can be updated whenever the UKB updates or adjusts its offerings. Using a multi-branch approach, the DataLad datasets provide a BIDS-structured representation in addition to the UKB-native data organization, without storage duplication and with full provenance capture of the BIDS transformation. We also employed the `datalad-container`[56] extension, which integrates container-based command execution with DataLad's process provenance capture capabilities (see docs.datalad.org/projects/container for more information).

Comprehensive data structure to track all elements of digital processing. All files involved in processing are contained in DataLad datasets, a Git-repository-based data representation that streamlines data management, sharing, and reuse[17]. In our framework, such datasets have a common representation (a regular directory tree) familiar to users, and also have a storage representation (a RIA store) that facilitates programmatic data management and reduces storage demands (Fig. 1a,d).

DataLad datasets can version control files regardless of file size, and can link other DataLad datasets at precise versions in modular *superdataset-subdataset* relationships. Based on this feature, all processing components, such as data, code, and computational environments in the form of software container images, can be uniquely and transparently identified with single file granularity across a hierarchy of linked DataLad datasets. Unlike purely Git-based tracking, version control and file identification are based on a cryptographic hash of the file content, a feature provided by the software git-annex[18]. More precisely, each file's content is translated into a checksum, and this checksum is saved (*committed*) as a file content identifier into the *revision history*—a detailed record of all changes in a DataLad dataset, including their date, time, and author. Exemplary shortened identifiers can be found in Fig. 1. This checksum is irreversible, i.e., one cannot infer the file content based on the identifier, but one can verify the content of files that are present on disk. Because file content is not stored in the revision history, the potential to leak sensitive information is significantly reduced, while the data representation still allows for thorough tracking and content verification.

Computation in automatically bootstrapped ephemeral workspaces. DataLad datasets can be distributed across local or remote infrastructure as lightweight, linked clones. They share their origin dataset's revision history and can extend it. File content transport across this network is possible via versatile transport logistics that allow for local or remote data hosting. This can enable data transports on systems with too little available disk space for multiple copies, allow redundant storage to be configured, interoperate with hosting services to publish results, or reconfigure data access when remote hosting locations change—without needing to alter the data representation in the dataset.

With these technical features, how and where data are stored (e.g., local, encrypted storage; remote, cloud-based hosting) becomes orthogonal to how and where computations are performed (e.g., on-site compute cluster; remote cloud-computing service). This allows our framework to bootstrap *ephemeral* (short-lived) workspaces for individual computational jobs, retrieve only relevant processing elements (e.g., subsets of input data), and extend the DataLad datasets' revision history with their results and process provenance (Fig. 1c). This, in turn, opens the possibility for parallel *and* version controlled analysis progression, using a distributed network of temporary clones. Results and revision histories can be merged to form a full processing history, in a similar way to how code is collaboratively developed with distributed version control tools[19]. Importantly, DataLad itself is not a workflow engine, but can be employed for individual nodes and segments of a processing graph defined by other solutions like HTCondor DAGMan[20], or snakemake[21].

Process provenance capture in machine-actionable records. Process provenance—how code and commands created results from input data in a particular computational environment—of any processing routine can be captured and stored in machine-readable, automatically re-executable records (Fig. 2). These records are created by a datalad run command for the execution of a shell command, or a container invocation by datalad containers-run. Users need to supply the command, a software environment, input data, and optionally which results should be saved as parameters. DataLad's execution wrappers retrieve inputs, initiate command execution, and save results together with a provenance record. Through the use of ephemeral workspaces during provenance capture, the validity and completeness of provenance records is automatically tested: only declared inputs are retrieved, only declared outputs are saved and deposited on permanent storage.

A resulting process provenance record is identified with one unique, hash-based identifier in the revision history, and can subsequently be used by authorized actors to automatically retrieve required components and re-execute the processing, irrespective of whether the original compute infrastructure is available[22]. This potential for full computational reproducibility of arbitrary processing steps not only increases the trustworthiness of the research process per se, but permits structured investigations of result variability, and furthermore provides the means to rerun any analysis on new data or with updated analysis components.

Showcases. These DataLad features offer great flexibility for transparently conducting reproducible, high-performance data processing in a wide variety of computational environments. In two concrete showcases we next highlight 1) the scalability of this approach, and 2) complete transparency and reproducibility of this data processing method, when combined with open data and open source tools.

Use case: large-scale medical imaging data processing. To demonstrate the framework's scalability, we conducted containerized analyses on one of the largest brain imaging datasets, the UKB imaging data. The strain that this dataset places on computational hardware is considerable both in terms of disk space usage (i.e., the amount of data that a hard drive can store) and inode usage (i.e., the number of files that a file system can index). To show how the framework can mitigate hardware limitations, we processed the dataset on two different infrastructures, an HPC system with inode constraints that preclude storage of the full number of files, and a high throughput computing (HTC) system with disk space limitations that preclude data duplication. In doing so, we assessed if the framework can be used across different infrastructures, investigated result variability between two recomputations of the pipeline, and probed the framework's features under distribution restrictions of both the data and the MATLAB-based software component. Finally, in order to demonstrate that the framework can capture and re-execute complete process provenance, we also recomputed individual results on a personal laptop.

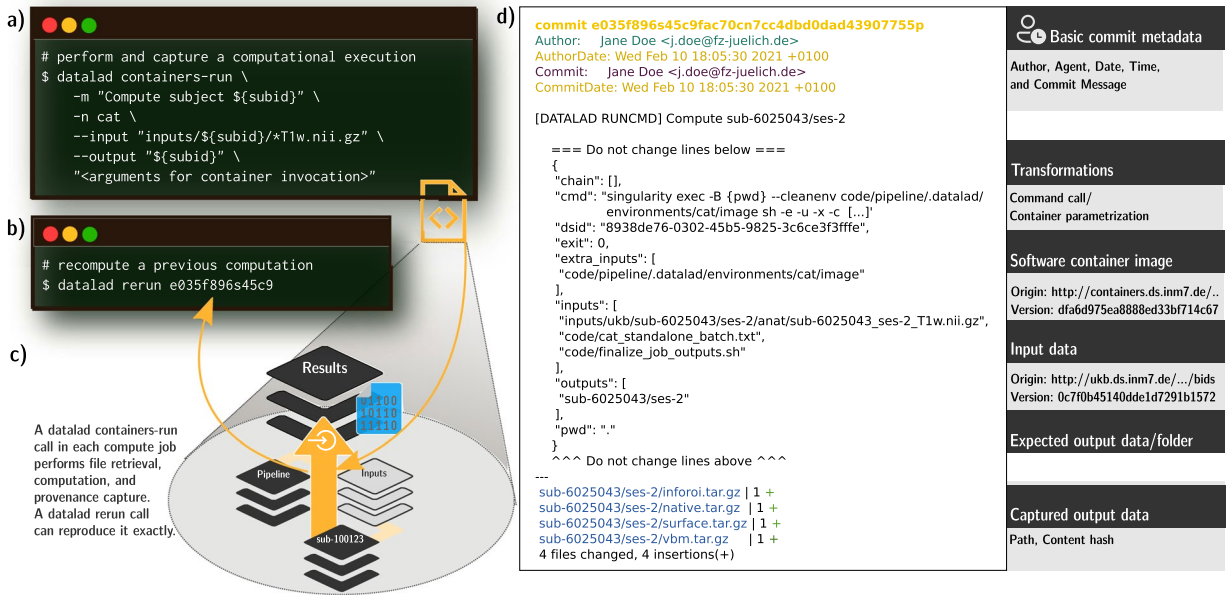


Fig. 2 Process provenance of an individual job, its generation, and re-execution. (**a**) Actionable process provenance is generated with a `datalad containers-run command`. This example contains a container name specification (`cat`), a container parametrization or command, a commit message, and an input and output data specification. The provenance is stored as a structured, JSON-formatted record linked to a Git commit. (**b**) To re-execute a process, the `datalad rerun` command only needs to be parameterized with a revision identifier, such as a Git tag, a "commit shasum" (`e035f896s45c9fa[...]` in this example), or a revision range containing one or more commits with associated provenance records. (**c**) The `datalad containers-run` call is at the center of each individual job. As the core execution command (see Listing 1, line 33–39), it performs data retrieval, container execution, and result capture, and generates the actionable provenance that a subsequent `datalad rerun` command (**b**) can re-execute. With complete provenance, a re-execution is supported on the original hardware, or on different infrastructure. (**d**) The machine-readable, re-executable provenance record stored alongside computed results in the revision history. A legend (right) highlights the most important pieces of recorded provenance. While *automatic* re-execution requires the tool DataLad, sufficient information to repeat a computation using other means can also be inferred from the structured JSON records by other software or even humans. This information forms the basis for standardized provenance reporting, for example using the PROV data model[54].

As a first step, we prepared input data and computational pipeline. We created a Singularity container with a pipeline to perform voxel-based morphometry (VBM)[23] on individual T1-weighted MRI images based on the Computational Anatomy Toolbox (CAT)[24]. We stored UKB data in archives in a DataLad RIA store[25] (Fig. 1d) to mitigate disk-space and inode limitations on the two different systems. The store comprised 42,715 BIDS-structured[26] DataLad datasets, one per study participant, that were jointly tracked by a single additional superdataset (UKB-BIDS; "Data" in Fig. 3). In total, the domain-agnostic data representation in the store hosted 76 TB of version-controlled data with 43 million individually accessible files while consuming less than 940k inodes.

Next, we assembled a single DataLad dataset to capture all processing inputs and outputs ("Results" in Fig. 3). It initially tracked: 1) the UKB-BIDS DataLad dataset; 2) a DataLad dataset providing the containerized CAT pipeline; 3) the compute job implementation responsible for bootstrapping a temporary workspace, performing a parameterized execution of the pipeline, capturing its outputs, and depositing results in the RIA store (see Listing 1 for a simplified version); and 4) the job scheduling specifications for SLURM[27] (used on the HPC system) and HTCondor[20] (used on the HTC system). Despite the total size of all tracked components, the pre-execution state of this dataset was extremely lean, as only availability (a URL) and joint identity (single checksum) information on the linked datasets is stored, and all other information is contained in the linked datasets themselves. This also implies that the DataLad dataset tracking the computational outputs is *not* automatically encumbered with sensitive information, even though it precisely identifies the medical imaging input data.

The compute job implementation minimized the number of output files using tar archives to reduce the strain on the technical infrastructure, and removed undesired sources of result variability (time stamps, file order differences in archives, etc.) to allow comparisons between recomputations. Later, these archives were partially extracted into tailored result datasets for easy consumption (see Methods, "(Re)use"). To maximize practical reproducibility of computational outcomes, a compute job implementation does not reference any system-specifics, such as absolute paths, or programs and services not tracked and provided by the DataLad dataset itself. This means that any system with DataLad installed, the ability to execute Singularity container images, and a basic UNIX shell environment is capable of recomputing captured outputs. Any performance-related

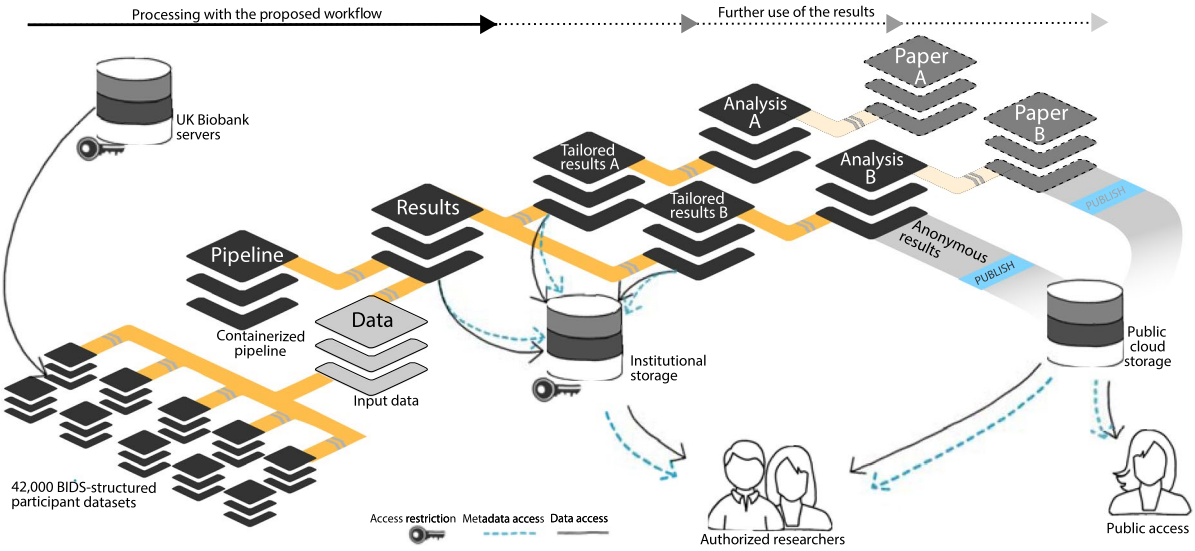


Fig. 3 Overview of DataLad dataset linkage through processing and reuse. Any DataLad dataset may comprise other DataLad datasets as *subdatasets* via lightweight but actionable and versioned links. This connects a dataset to the content and provenance of a different modular unit of data, such as the outcomes of a the preceding processing step. The genesis of an analysis output (*Analysis A/B*) based on intermediate processing outcomes (*Tailored results A/B*) can thus be traced back all the way to the original raw data. Access control and storage choices are independent across individual components in this network of linked data modules. Aggregated data and analysis results can be shared with larger audiences or publicly on a variety of platforms, while raw and derived data may underlie particular access restrictions, or require particular hosting solutions due to their size.

adaptations to the particular systems used for our computations were strictly limited to the job scheduling layer, which is clearly separated from the processing pipeline. Computation and recomputation on systems with different batch scheduling software is then possible by providing alternative job specifications, without changes to the pipeline implementation.

We performed processing on the HPC and HTC infrastructure starting from the exact same dataset version state, but with job orchestration tuned to the respective job scheduling system (a visualization of the different processing speeds can be found at www.youtube.com/watch?v=UsW6xN2f2jc). Provenance for each execution of the CAT pipeline on an individual image was captured in a dedicated commit, and recorded on a participant-specific Git branch. Recorded outputs and provenance records were pushed to the RIA store on job completion, yielding a total of 995.6 GB of computed derivatives in 163,212 files. The second computation added matching commits and branches to the DataLad dataset that enabled straightforward comparison and visualization of results using standard Git tools and workflows. To confirm the practicality of computational reproducibility solely based on the captured computational provenance information, we performed automatic recomputation of individual results on a consumer-grade, personal laptop without job scheduling. This type of spot-checking results resembles the scenario of an interested reader or reviewer of a scientific publication with access to (parts of) the data, but no access to adequate large-scale computing resources.

With the exception of execution time, the number of jobs, proportion of successful jobs, and size and structure of the results were identical between the two systems. Specifically, with the exception of one output flavor (projections of computed estimates to the cortical surface) more than 50% of all output files were identical across the two computations. Outcome variability for non-identical results was largely attributable to minor numerical differences, as illustrated by the mean squared error (MSE) over recomputations for a range of key VBM estimates: total surface area ($\mu = 1891$, $MSE = 0.315$), cerebro-spinal fluid ($\mu = 365$, $MSE = 0.052$), total intracranial volume ($\mu = 1508$, $MSE = 0.052$), white matter ($\mu = 519$, $MSE = 0.004$), and gray matter ($\mu = 621$, $MSE = 0.001$). We also computed correlations over different brain parcellations included in the CAT toolbox. The lowest observed correlation across recomputations for VBM estimate distribution across different brain parcellations were Pearson's $\rho > 0.998$ for the Destrieux 2009 surface parcellation[28] for all brain regions. Quality control metrics depicted in Fig. 4 exhibit $\rho > 0.99999$ for computation and recomputation.

The complete implementation of this showcase cannot be shared due to imposed data usage and software license restrictions. However, we provide a bootstrapping script that implements all required setup steps at https://github.com/psychoinformatics-de/fairly-big-processing-workflowgithub.com, and share a detailed description and full recipe of the container together with instructions on how to build and use it at https://github.com/m-wierzba/cat-containergithub.com/m-wierzba/cat-container.

Use case: Open tutorial. As strict software license restrictions and data usage agreements prevent fully open sharing of computed results and a public demonstration of their process provenance records, we set up an open tutorial analysis using free and open source fmriprep software[29] and open data from the studyforrest.

org project[30]. We confirmed that process provenance was sufficient to enable automatic recomputations on an HTC system, a personal work station running Debian, and a Mac, and published the resulting DataLad dataset to public GitHub (https://github.com/psychoinformatics-de/fairly-big-processing-workflow-tutorial) and Gin (https://gin.g-node.org/adswa/processing-workflow-tutorial) repositories. This demonstration allows for in-depth inspection, retrieval (datalad get) of any and all data processing inputs and outputs, as well as automatic recomputation (datalad rerun) of all captured results.

Discussion

The proposed framework aims to make the results of any processing as open and reusable as the given limits of individual components allow. It streamlines computation, re-computation, and sharing with appropriate audiences for datasets and on compute infrastructure of any size. To this end, proven procedures from software development and a set of open source software tools are assembled into a scalable and portable framework with a variety of features: The basis for transparency is laid with version control for all involved files, including software environments. Distributed data transport and storage logistics offer flexibility to adapt to particular computing infrastructure. Reproducible results are enabled via comprehensive capture of machine-actionable process provenance records, capitalizing on portable containerized environments. Combining distributed computing with ephemeral workspaces that resemble workflows from collaborative software development yields efficient processing, and ensures the validity of provenance information.

The framework shares features and goals with a number of related systems, some of which we want to highlight in order to illustrate how the proposed workflow and its main building blocks relate to other solutions. The proposed *Pan-Neuro*[31] is an alternative solution for neuroscientific computing on large-scale data, but is more geared towards interactive processing, and represents a centralized, cloud-based platform for computing and data hosting. *IPFS*, the InterPlanetary File System (https://ipfs.io), is a distributed system for data transport that employs an approach to content addressing that is based on cryptographic hashes of file content. This concept is identical to the one employed by git-annex, except for differences in the hash or key composition details. Consequently, these systems are interoperable, and the proposed framework could directly employ IPFS-based data sources via the git-annex integration (https://git-annex.branchable.com/special_remotes/ipfs/). *DVC*[32] is a version control system and workflow manager also built on Git. It employs distributed version control for individual large files or collections of files, and captures provenance of language-agnostic machine-learning pipelines that connect multiple steps of building an ML model into a directed acyclic graph (DAG). Major differences lie in DVC's specific focus on machine learning models in workflow management, and less emphasis on portability and reproducible environments. *Snakemake*[21] is a feature-rich and domain-agnostic workflow engine, with support for including software environments in the form of conda-environments or software containers, portable workflows that allow execution on remote resources such as cloud services or batch systems on compute clusters, and provenance capture. DataLad can enhance snakemake workflows by retrieving versioned input data files (http://docs.datalad.org/projects/mihextras/generated/man/datalad-x-snakemake.html), and snakemake-based compute job orchestration could be employed as an alternative to the custom implementations for SLURM and HTCondor used in the work presented here. *Parsl*[33] is a Python-based parallel scripting library that is designed to enable compositional programming for a variety of scientific use cases. It features workflow management, a mix-and-match style portability with provider interfaces to configure resource-specific requirements across tools or infrastructures, and data management that can perform data transfers to and from resources via several protocols. Similar to snakemake, it provides an alternative workflow definition and orchestration solution, and could employ the provenance capture approach proposed here. Unlike the framework proposed here, Parsl does not build on a version-controlled core data structure, but on a specification of interconnected apps and the data flow between them. However, like snakemake files, these specifications could be tracked within DataLad datasets in order to combine the capabilities of these systems. *Apache Spark*[34] and *Dask*[35] are both feature-rich, distributed computing solutions targeting different software ecosystems. Unlike the proposed framework or the aforementioned solutions they require the deployment of dedicated services across a distributed computing resource, and can perform large-scale computations via internal parallelization, where a comparable compute-node local provenance capture step is not performed or possibly even meaningful.

Overall, our framework is a general-purpose solution that is compatible with any data that can be represented as files of any size, and any computation that can be performed via a command line call. It is built on a collection of portable, free and open-source tools that can be deployed without special privileges or administrative coordination on standard HTC/HPC infrastructure, or personal computing equipment.

While it is an explicit aim for the framework to yield FAIR outputs, this aspirational goal is not fully reached. Metadata used and produced by the framework does not conform to explicit annotation standards. Instead, it encodes essential metadata, such as author, date, time, and description in locations that are provided by the version control system Git. Other metadata are put into plain-text, key-value data structures that conform to no particular formal ontology or vocabulary. This shortcoming limits the findability and accessibility of its outputs severely. Questions like "which outcomes were computed with a specific version of a particular software?" cannot be reasonably answered without additional standardization and annotation effort.

That being said, the main contribution of the proposed computational approach is the association of process provenance with captured outcomes (FAIR R1.2), with precise linkage of any data inputs within and across individual datasets (FAIR I3), using unique, content-hash based identifiers for all components (FAIR F1). These metadata are tracked in a dedicated overlay data structure that can ensure their accessibility, even when the underlying data are no longer available or a particular entity has no permission to access them (FAIR A2).

What the framework provides today is a technical system that, despite its ignorance regarding formal metadata standards affords practical, automatic recomputation of arbitrary data processing results. This

ability dramatically elevates the starting point for future FAIRification efforts of computational outcomes. Reproducibility can be programmatically verified, thereby providing a confirmation of the comprehensiveness of data and essential process metadata encoded in a DataLad dataset. Subsequent annotations of precisely versioned data, or tracked computational environments can retroactivily boost findability and accessibility of outcomes. For example, an added annotation of the composition of an employed containerized pipeline can help answer the question posed above. Neither metadata format nor terminology are constrained by the proposed framework. Importantly, the ability to recompute outcomes provides a strong incentive for researchers to produce computational outcomes with verifiably complete (meta)data. This is an important half-step towards a FAIRer future that boosts the availability of research outputs that can receive continuous updates to co-evolve with further developments of metadata standards and requirements of future metadata-driven applications. To this end, a DataLad dataset can also be exported to different formats used in frameworks with a similar aim, such as `BagIt`[36].

The use of containerized software environments plays a key role in the proposed framework. They represent the most practical solution to portable computational environments today. However, their long-term, universal accessibility is not guaranteed. Even today the `singularity` software does not support all major operating systems. Ten years ago, the popular `docker` software did not yet exist, and it is unclear whether its container images will be executable in ten years from now. Providing the build instructions for a container image, rather than (or in addition to) the readily executable image, may improve the longevity of their accessibility, and also mitigate the problem of license-imposed sharing restriction. However, it is not guaranteed that executing a recipe twice results in identical software containers. *Reproducible builds*, the practice of creating identical container images from a recipe[37], for example, require the specification and availability of software and system libraries at precise versions. For the same reason of long-term accessibility, it will also be necessary to incorporate DataLad's own idiosyncratic provenance metadata into such a comprehensive provenance report — then matching a format and standardization desirable for a particular scope or application. A promising effort towards portability and longevity of container technology is the Open Container Initiative (OCI; https://opencontainers.org), which aims to create open, vendor-neutral, and portable industry standards around containers formats and runtimes.

Based on process provenance and version control, structured analyses of variability between (re)computations on the same or different infrastructure are facilitated[13,38]. Bit-identical recomputation of a result are trivially verifiable. The comprehensive capture of input data version, computational environments and process parameterization enable deep inspection of other sources of result variability. Building on this foundation, more standardized process descriptions[39] and reproducible computational environments[37] can further enhance these types of analyses. Nevertheless, computational results that do not reproduce exactly are a challenge for content checksum based version control systems like Git. If irreproducibility is solely caused by issues of numerical precision and reproducibility of basic floating point operations, it may be possible to nevertheless achieve bitwise identical result by reducing the precision of stored outcomes to empirically meaningful levels of detail, like we did here for the aggregated brain structure scores in the UKB use case. However, in general data type specific and research focus specific implementations of "identity" operators would be required, and while Git offers means for their integration (so-called diff drivers), there are no generally applicable off-the-shelf solutions available for this problem.

The approach to reproducible computation proposed here is applicable to a wide range of use cases. Different datasets, different processing pipelines, and different containerization technologies, such as Docker, can be employed by simply replacing the respective components, and utilizing features already built into DataLad. These possibilities are illustrated in the extensive DataLad Handbook[40] at http://www.handbook.datalad.org. Combining this framework with more capable workflow engines for defining and orchestrating compute jobs and their interdependencies in the future, would open up the possibility for implementing other types of data processing that go beyond the parallel execution of mutually independent compute jobs that make up the use case illustrated in this work. Such an implementation would need to fulfill three general requirements that were implemented here: 1) The execution of a compute job must take place in a workspace that is defined by a recorded state in a DataLad dataset. 2) Execution via Datalad captures process provenance in a new, incremental dataset state. 3) Advanced dataset states after any parallel processing stage are consolidated into a merged mainline. When these conditions are met, the structure of a processing DAG would be reflected in the version-control history of a DataLad dataset hierarchy that comprises all inputs and outputs of a computational project, with each recorded step being associated with machine-actionable provenance records that enable deep inspection of large-scale computing outcomes.

The presented showcases provide two concrete examples for the adoption of the proposed framework that deal with typical obstacles for transparent, reproducible science. The UKB's data size exceeds the capacity of most infrastructure. We demonstrated the scalability of our framework by processing these data on systems with hardware limitations that would typically render even storage of inputs and outputs difficult or impossible. As the proposed framework enables selective recomputation even on commodity hardware, consumers can investigate results without having to rely on the original authors, and without access to the original computational infrastructure. Even though the raw data may be too large to allow users a complete recomputation, the process provenance entails a trail of processing steps that permits automatic recomputation of individual results. A one-time computation on larger infrastructure can thus build a verifiable, trustworthy foundation for numerous subsequent analyses by other researchers.

Finally, above and beyond everything else, the framework makes research as open as desired. The medical imaging showcase featured a processing pipeline based on proprietary software and pseudonomized personal data under usage constraints. Data and computational environment are not publicly shareable. But if data usage agreements and software licensing permit, as it is the case in the second showcase, processing results can be

shared publicly that are independently and automatically reproducible by any interested party. This level of transparency dramatically improves the accessibility of scientific outcomes.

Methods

The proposed framework aids the reproducible execution of a containerized pipeline on input data, by associating computational outcomes with machine-actionable provenance records in a version control system. We illustrate the technical details of this process with two use cases that differ in scale as well as data access and processing requirements, but follow a common pattern in general setup and composition.

Framework setup. The technical nature of the framework components, in particular its foundation, the version control software Git, enables distributed computational workflows that utilize and extend established procedures from collaborative software development to data processing. The framework is bootstrapped in two steps that could be performed by a tailored shell script for a particular application.

Self-contained processing specification as a DataLad dataset. The first step is the creation of a new DataLad dataset that will eventually track the processing results (dataset labeled "Results" in Fig. 3). Input data, images of containerized pipelines, or custom code are added to this dataset. While the use of software containers to provide processing pipelines is not strictly required, they are a practical method to provide stable and portable computational environments. Because such containers can be stored in image files, they can be tracked and precisely versioned like any other component of a DataLad dataset. The datalad-container extension provides a convenience interface for registering containers and for executing commands in such environments.

All processing components, such as processing-specific, customized scripts and applications or data, can be added directly to the dataset as individual files. More typically, however, individual processing components, for example input data or containerized pipelines, are placed in separate DataLad datasets and linked as subdatasets (Fig. 3). This more modular structure enables (re)use of independently maintained components, while strictly separating access modalities to each of them. In this way, access-restricted input data does not impair sharing of less sensitive outcomes and the versioned link between superdataset and subdatasets guarantees precise identification of processing components, regardless of whether a particular dataset consumer has access to a given component.

The resulting dataset is the entry point to a self-contained directory structure, potentially comprising other nested DataLad datasets, that jointly define identity and location of all data processing inputs in the exact form needed for a particular computation.

Environment and performance optimized orchestration. The second step is the preparation of the computational environment and processing orchestration. This relates to *what* is computed as well as *how* it is computed. The compute job orchestration, the *how-to-compute*, could be as simple as direct, sequential executions of required processing steps in a shell script for-loop. However, large-scale computations typically require some form of parallelization. The compute job orchestration is thus likely to be implemented using the job scheduling system of a given compute infrastructure. As such, *how-to-compute* is highly infrastructure-specific, and must determine an optimum balance of resource demands, such as run time, memory and storage requirements, in order to achieve optimal throughput.

The *what-to-compute*, the computational instructions, pipelines, or scripts, need to be independent computational units that can be executed in parallel. A common example is the parallelized execution of a processing pipeline on different, independent parts of input data. As parallelization often corresponds to the granularity at which a recomputation will be possible in our framework, relevant considerations are, for example: "What is the smallest unit for which a recomputation is desirable?", or "For which unit size is a recomputation still feasible on commodity hardware?". To ensure reproducibility for an audience that does not have access to the original infrastructure, *what-to-compute* needs to be infrastructure-agnostic, without references to system-specifics such as absolute paths, or programs and services not tracked and provided by the DataLad dataset itself. Then, computation and recomputation of *what-to-compute* are possible on different systems, with any potential adjustments only relating to the job orchestration layer in *how-to-compute*.

Execution and result consolidation workflow. After the two preparatory steps are completed the actual data processing can be executed by submitting the compute jobs to the job scheduling system. Each compute job will clone the DataLad dataset with the processing specification to a temporary location, bootstrap an ephemeral workspace that is populated with all inputs required for the given job with a job-specific parameterization, execute the desired computing pipeline, and capture a precise provenance record of this execution, comprising all inputs, parameters and generated outcomes. Lastly, it pushes this provenance metadata and result file content to permanent storage. This workflow resembles a standard distributed development workflow in software projects (obtain a development snapshot, implement a new feature, and integrate the contribution with the mainline development and other simultaneously executed developments) but applies it to processing of data of any size. Specific details of this workflow are outlined in sequential order in the following paragraphs. Where applicable, they annotate and rationalize the generic compute job implementation in Listing 1.

Dataset clone source and update push target can be separated in an initial setup step to improve performance. When all compute jobs deposit their outcomes at the same DataLad dataset location that later compute jobs also clone from, version history in this dataset accumulates and progressively slows the bootstrapping of work environments of compute jobs, because more information needs to be transferred. Moreover, result deposition in a DataLad dataset is a write operation that must be protected against concurrent read and write access for technical reasons, and hence introduces a throughput bottleneck. Both problems are addressed by placing

an additional clone of the pre-computation state of the processing specification dataset in a RIA store before job submission (Fig. 1d). This clone is used for result deposition only (Listing 1, lines 17 and 49). Dataset clones performed by jobs are done from the original location that is never updated, hence also never grows. In order to avoid unintentional modifications during long computations, the dataset clone source for jobs may not be the dataset location used for preparation (Fig. 1a), but yet another, separate clone in a different RIA store. The clone source and push target locations are provided as parameters to compute jobs (Listing 1, lines 5-6). All dataset locations are not confined to exist on the same hardware as long as they are accessible via supported data transport mechanisms over the network.

Job-specific ephemeral workspaces are the centerpiece of the computation, and the location where the actual data processing takes place (Fig. 1c). Critically, these workspaces are bootstrapped using information from the specification DataLad dataset only. This is achieved by cloning this dataset into the workspace first (Listing 1, line 11), and subsequently performing all operations in the context of the clone. After computation and result deposition the clone and the entire workspace are purged. This ensures that all information required to perform a computation is encoded in this portable specification, that it is actionable enough to create a suitable computing environment, and that all desired outcomes are properly registered with the DataLad dataset to achieve deposition on permanent storage.

Containerized execution and provenance capture happens within the ephemeral workspace on a uniquely identified branch per job (Fig. 1b, "workflow execution"; Listing 1, line 21). Prior computation, the state of this branch is identical for all jobs. It comprehensively and precisely identifies all processing inputs, and links them to author identities, time stamps, and human readable descriptions encoded in the Git revision history of the dataset (Fig. 2d, top).

Based on this initial state, a computational pipeline is then executed, and all relevant computational outcomes are saved to the DataLad dataset to form an updated state (Listing 1, line 33–39). For this execution, all required input files are specified by their relative path in the DataLad dataset (potentially pointing into linked subdatasets). Importantly, only these job-specific inputs will be transferred to the compute job's environment. Likewise to be saved outcomes are selected by providing path specifications. Given the execution of the computation in an isolated, ephemeral workspace that is unique for each individual job, two guarantees can be derived regarding the provenance of the computational outcomes: 1) All dataset modifications can be causally attributed to the initiated computation; 2) only declared inputs were required to produce the outcomes.

DataLad commands like run (for command line execution) or containers-run (for execution in containerized environments) yield machine-readable provenance records that express what command was executed, with which exact parameters, based on which inputs, to generate a set of output files (Fig. 2d). Such a record is embedded in the Git commit message of the newly saved dataset state as structured data. The record itself is lean and free of explicit version information for individual inputs, because the dataset state as a whole jointly identifies all versions of all dataset components, such that individual versions are readily retrievable on a (later) inspection of this state.

The captured provenance record is machine-actionable. Using the dataset and information in the provenance record in a dataset state's commit message, the DataLad command rerun can reobtain necessary inputs and run the exact same command again, given availability of data and environments. Importantly, this re-execution does not strictly depend on the original compute infrastructure, but benefits from DataLad's ability to retrieve file content from multiple redundant locations.

Result deposition takes place after successful completion of each job. The file content of computational outcomes, along with their provenance, are pushed to permanent storage (Fig. 1b, blue arrow). Two different components of result deposition have to be distinguished.

Transfer of file content (Listing 1, line 44) is an operation that is independent across compute jobs, and can be performed concurrently. This enables simultaneous transfer of (large) files. Importantly, only file content blobs (i.e., git-annex keys) are transferred at this point.

Additionally, critical metadata must be deposited too. All essential metadata is encoded in the new dataset state commit, recorded on the job-specific Git branch. Consequently, it is deposited using a git push call (Listing 1, line 49). This push operation is not concurrency-safe, hence must be protected by a global lock that ensures only one push is performed at a time across all compute jobs (using the tool flock). Therefore this step represents a central bottleneck that can influence computational throughput. However, when file-content is only tracked by checksum with git-annex, the changes encoded in the Git branch are metadata only, and a transfer is typically fast.

After successful completion of all computations, the DataLad dataset on permanent storage holds the provenance records of all results in separate job-specific branches, and the content of all output files in a single git-annex object tree.

Result consolidation is the final workflow step. After processing, the result DataLad dataset contains as many branches as successfully completed jobs. These branches must be consolidated into a new state of the mainline branch that jointly represents the outcomes of a individual computations (Fig. 1b, "result consolidation/merge").

How exactly this merging operation must be conducted depends on the nature of the changes. In the simplest case, all compute jobs produced non-intersecting outputs, i.e., no single file was written to by more than one compute job. In this case, all branches can be merged at once using a so-called *octopus-merge*:

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

Depending on the number of result branches, it may be necessary to merge branches in batches to circumvent operating system or shell limits regarding a maximum command line length. If computational outcomes are not independent across jobs (i.e., order of computation/modification matters), a merge strategy has to be

employed that appropriately acknowledges such dependencies. If the deposition dataset is hosted in a RIA store (as suggested above for performance reasons) this operation is performed in a temporary clone.

As a final step, valid metadata on output file content availability must be generated. File content resides in the result dataset at the deposition site already, but the required metadata was not pushed to its internal git-annex branch from all compute jobs, in order to avoid a consolidation bottleneck. Instead, these metadata are generated only now, by probing the availability of the required file content blobs for all files present in the mainline branch after merging all compute job branches.

```
# discover/confirm result file availability
git annex fsck --fast -f output -storage
```

```
1 #!/bin/bash
2 # fail on any issue, show commands
3 set -e -u -x
4 # name arguments for readability
5 dssource="$1"
6 pushgitremote="$2"
7 subid="$3"
8
9 # obtain the analysis dataset, which
10 # also tracks the required inputs
11 datalad clone "${dssource}" ds
12 cd ds
13
14 # register location for result
15 # deposition, separate from the input
16 # source for performance reasons only
17 git remote add outputstore
  ↪ "$pushgitremote"
18
19 # all job results will be put into
20 # a job-specific, dedicated branch
21 git checkout -b "job-$JOBID"
22
23 # START OF APPLICATION-SPECIFIC CODE
24 # pull down input data manually,
25 # only needed for wildcard-based file
26 # selection in the next command
27 datalad get -n "inputs/ukb/${subid}"
28 # datalad containers-run executes
29 # the "cat" computational pipeline.
30 # specified inputs are auto-obtained,
31 # specified outputs are saved with
32 # provenance record
33 datalad containers-run \
34   -m "Compute subject ${subid}" \
35   -n cat \
36   --explicit \
37   --output "${subid}" \
38   --input
  ↪ "inputs/ukb/${subid}/*T1w.nii.gz"
39   "<container invokation arguments>"
40 # END OF APPLICATION-SPECIFIC CODE
41
42 # push result file content to the
43 # configured "storage-remote"
44 datalad push --to storage-remote
45
46 # push branch with provenance records
47 # needs a global lock to prevent
48 # write conflicts
49 flock "$DSLOCKFILE" git push
  ↪ outputstore
50
51 # log entry to mark non-error exit
52 echo SUCCESS
```

Listing 1 Complete compute job implementation as a bash script. A batch system invokes the job-script in a temporary working directory with three parameters: a URL of a DataLad dataset tracking all code and input data, a URL to deposit job-results at, and an identifier to select a sample for processing. Apart from performance-related optimizations, the job implementation conducts three main steps: 1) `clone` a DataLad dataset with all information to bootstrap an ephemeral computing environment for each job; 2) `containers-run` a containerized pipeline with a comprehensive specification of to-be-retrieved inputs and to-be-captured outputs; 3) push captured outputs and process provenance records to a permanent storage location. Preparation, computation, provenance record creation, and file content deposition on permanent storage are fully independent across jobs, and are executed in parallel. Only the `git push` of the provenance record to a central repository must be protected against concurrent write-access for technical reasons. Additional job parametrization (`DSLOCKFILE` and `JOBID` environment variables) are defined at job-submission using batch system specific means. The job script can be adjusted to a different processing pipeline by replacing the container invocation (see `APPLICATION-SPECIFIC CODE` markers).

```
# push consolidated provenance records and file availability # metadata
to permanent storage
datalad push --data nothing
```

The `git-annex fsck` command probes the configured output-storage site whether it possesses a given annex key (i.e., a file content blob corresponding to a particular checksum), and generates an appropriate availability metadata record. The final datalad push command (Listing 1, line 44) transferred these verified metadata records to permanent storage.

The outcome of this consolidation process is a self-contained DataLad dataset, with valid, machine-actionable provenance information for every single result file of the performed data processing. As such, it is a modular unit of data, suitable as input for further processing and analysis. It translates the advantages of comprehensive and precise linkage of all its components across any number of other data modules to any consumer.

Balance of reproducibility and performance. Taken together the described approach to reproducible, large-scale computation implements a three layer strategy. From bottom to top, these layers feature different

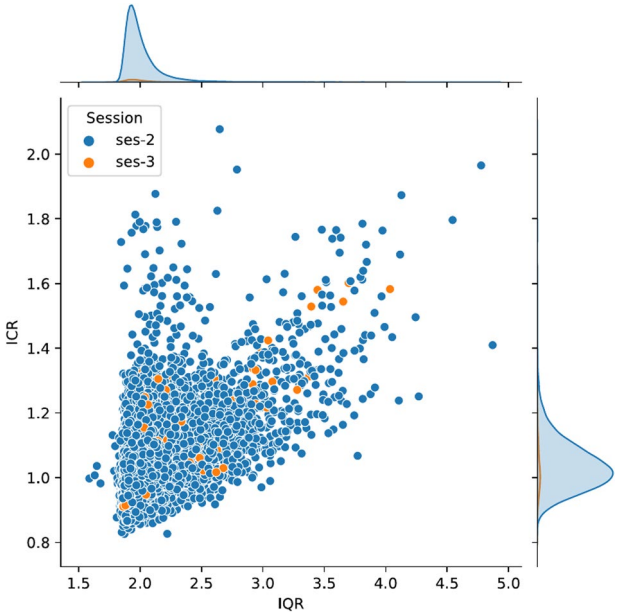


Fig. 4 Distribution of quality assurance measures, derived for 41,180 unprocessed T1-weighted images from the UKB dataset. The quality measures were obtained retrospectively based on the preprocessing methods[45,55]. For both measures, lower values correspond to better quality. Abbreviations: IQR - image quality rating, a weighted composite score based on noise, inhomogeneity, and image resolution (0.5–1.5 = "perfect", 1.5–2.5 = "good", 2.5–3.5 = "average", 3.5–4.5 = "poor", 4.5–5.5 = "critical", >5.5 = "unacceptable"); ICR - inhomogeneity contrast ratio, estimated as the standard deviation within the white matter segment of the intensity scaled image.

trade-offs regarding portability/reproducibility vs. flexibility for performance adaptations to particular computational environments: The lowest layer is the (`containers-`)run command, comprising an environment specification and instructions to compute the desired outcomes from inputs in this environment. Using suitable technologies, such as computational containers, this layer offers a maximum of portability, but also a minimum of flexibility, as this exact environment must be provided in order to reproduce results. Consequently, the proposed framework captures process provenance at this layer (Fig. 2). The middle layer describes how a self-contained, ephemeral workspace can be generated that is suitable for executing the specification of the previous layer (Listing 1). Here, general infrastructural choices can be made. For example, a limitation to a POSIX-compatible environment that is common for HPC/HTC systems, or the granularity with which provenance records are captured (and therefore the granularity at which reproducibility is supported). This layer plays a key role in ensuring that process provenance records are valid and complete. The topmost layer is concerned with maximizing performance on a particular infrastructure via tailored job orchestration, and composition. This layer is poorly portable as it references infrastructure-specific elements, such as job scheduling systems, absolute paths, user names or resource identifiers. While the implementation of all three layers should be provided within the DataLad dataset for a computational project, only the lowest layer is strictly required for reproducing results.

Software requirements. The software and their minimum version requirements for executing the framework are `datalad` v0.14.2, `Git` v2.24, `git-annex` v8.202003, and `datalad-container` v1.1.2 (not required for recomputation). Optional requirements are job scheduling systems as well as containerization software (e.g., `Singularity` v2.6.).

In principle, the framework could also be used without a software container. But despite their problems, containers represent the contemporary optimum for encapsulating computational environments that can be shared and reused across different systems. Here we have used Singularity[15], one of the most widely used container solutions for both single- and multi-user environments, suitable for HPC/HTC architectures. This choice limits the target platform on which a provenance-based recomputation can be attempted, and for example rules out the Windows operating system for which this software is not available. Other technologies, such as Docker, offer a different set of supported environments.

UK Biobank computing use case. To demonstrate the framework's scalability and its ability to create reusable derivatives for subsequent analyses, we applied it to data from the brain imaging component of the UKB project[16]. We performed a containerized analysis for voxel-based morphometry (VBM)[23] based on the Computational Anatomy Toolbox[24], a common method for anatomical brain imaging data. This choice of data and processing pipeline posed particular challenges for openness, transparency, and reproducibility. The UKB imaging project is one of the largest studies of this kind. The data are under strict usage constraints to ensure the responsible use of participants' personal data. Moreover, the chosen processing pipeline is based on MATLAB, at present still the

most prevalent programming environment in biomedical research[11], enforcing rigid redistribution limits due to its proprietary, closed-source license. The setup steps were implemented in a bootstrap script available at https://github.com/psychoinformatics-de/fairly-big-processing-workflow/blob/main/bootstrap_ukb_cat.sh.

Self-contained processing specification as a DataLad dataset. The UKB provides imaging data in ZIP archives, with one archive containing all files for a single modality of a single participant in one format. Direct downloads via versioned perma-URLs are not possible, but `ukbfetch`, a custom binary-only downloader application, must be used.

We implemented `datalad-ukbiobank`[41], a `DataLad extension` (see Box 1) that aids retrieval, indexing, and versioning of UKB data offerings in the form of DataLad datasets. Such a dataset represents data in three variants (using dedicated Git branches): the downloaded ZIP files, extracted ZIP file context using UKB-native filenames, and an alternative data organization following the BIDS standard.

Using `datalad-ukbiobank`, we retrieved MRI bulk data for all participants in NIFTI format. Each participant's data were represented as an individual DataLad dataset, yielding 42,715 datasets in total. The BIDS-structured branches of all these datasets were jointly tracked by a single UKB superdataset ("Data" in Fig. 3). This UKB superdataset is installable within seconds. On a filesystem, it takes up about 40 MB of space, but can retrieve any of the registered file content in the entire DataLad dataset hierarchy, comprising 76 TB across 43 million files, on demand.

As processing pipeline, we chose CAT's default segmentation of structural T1-weighted images using geodesic shooting[42], including calculation of total gray matter (GM), white matter (WM), and intracranial volume (TIV), as well as extraction of regional GM estimates from several brain parcellations. To this end, we built a Singularity container for the MATLAB-based Computational Anatomy Toolbox (CAT; version: CAT12.7-RC2, r1720)[24], which is an extension to the Statistical Parametric Mapping software (SPM; version: SPM12, r7771; www.fil.ion.ucl.ac.uk/spm/software). As MATLAB requires a commercial, non-transferable license, we used a compiled version of the CAT toolbox provided by the authors, which does not require the availability of a MATLAB license at runtime. Due to software license restrictions (the MATLAB Compiler Runtime in the container is subject to the MATLAB Runtime license), we cannot redistribute the container image, but we share a detailed description and full recipe of the container together with instructions on how to build and use it at https://github.com/m-wierzba/cat-container.

We added two custom code files to the dataset. First, a batch script, with a comprehensive specification and parameterization of all processing steps to be performed by CAT in an input image. This script allowed us to bundle up all relevant analysis steps into single command that also defines the smallest unit for recomputation. Second, a utility script to post-process all relevant outputs (≈30 individual files) into four `tar` archives per computation in order to minimize disk space usage and number of resulting files. Controlling the total number of output files was important due to the amount of computational outcomes to be tracked in this particular result dataset. Only four files per computation translate to more than 160,000 files in total. Such large datasets require substantial file system operations, even when only a subset of file content is retrieved for a particular use case.

The resulting `tar` archives are organized according to envisioned consumption scenarios (vbm containing modulated gray matter density and partial volume estimates in template space, `native` with atlas projections and partial volumes in individual space, `surface` with surface projection and thickness, and `inforoi` containing regional volume and thickness estimates of several atlases/parcellations; Fig. 1a). `tar` was parameterized to create archived with a normalized file order, creation time, and file permissions in order to not introduce artificial variation between recomputations. Likewise, all result files were carefully stripped of timestamps and other non-deterministic log file content. The resulting *reproducible* tarballs allow to attribute file content variability across re-computations to actual result variability.

Environment and performance optimized orchestration. Data were processed on an HPC cluster and a high-throughput computing (HTC) cluster, each imposing a different set of resource constraints. The HPC system is a modular supercomputer with 1,872 nodes, currently among the 500 fastest compute infrastructures in the world[43]. While available disk space was abundant, storage was constrained by an inode quota of 4.4 million files – less than the total number of files of the raw dataset. In contrast, the HTC cluster is a mid-sized computational cluster with 31 nodes with only about 400 TB storage capacity, preventing the existence of more than one copy of the raw dataset, and limiting the size of derivatives that could be stored.

To reduce the disk space and inode demands, all DataLad datasets were stored in a RIA store. In this "backend" representation (Fig. 1d), a single participant dataset encompasses 25 inodes and about 4 GB of disk space. When cloned into a workspace (Fig. 1a), it expands to several hundreds of files. In total, the employed RIA store hosts 42,715 datasets comprising the full UKB data, and consumes 75.6 TB of disk space with less than 940k inodes.

The ability to extract subsets of otherwise compressed inputs only when needed in ephemeral workspaces allowed us to adjust the parallel job load to the available resources. This enabled computations when disk space or inode availability were insufficient for the full dataset. With this setup, we were able to complete data processing for a one-hour-per-image pipeline on the HPC system within 10.5 hours, using 25 dedicated compute nodes, each executing 125 jobs in parallel on RAM disks with GNU Parallel[44]. On the HTC system in turn, HTCondor scheduled jobs dynamically across several weeks for available compute slots in an otherwise busy system used for unrelated computations by other researchers.

In order to validate different aspects of reproducibility all data processing was performed twice, once on each computing platform, and also a third time for a small subset of the data on a personal laptop. For the two main computing platforms dedicated job submission scripts were implemented for SLURM and HTCondor

respectively. In contrast, the partial recomputation on a laptop solely relied on the local availability of the Singularity container technology, but was otherwise fully automatic, based on the captured provenance record, to confirm practical reproducibility for an independent consumer.

Because of the large number of participants in the dataset and the aim to be able to rerun the data processing on a future, even larger release, one compute job per participant was generated. A compute job serially processed either two images, only a single image, or none, depending on the actual data availability. A dedicated provenance record was captured for each pipeline execution on an individual input image, yielding a total of 41,180 records.

Execution and result consolidation workflow. Data processing was executed based on two variants of the same DataLad dataset, each containing a common computational environment, and the same input data, but a different, optimized job submission implementation. Result consolidation was first performed separately on each computational infrastructure, following the steps described in the framework setup. Lastly, the two complete sets of computational outcomes were integrated in the same dataset, as two different branches, for comparison.

Result verification. As prior manual data inspection was infeasible due to the amount of data, we included basic checks to ensure availability of T1-weighted images during processing. Subsequent quality control analyses were derived from the computed results. Figure 4 shows the distribution of quality control metrics for T1-weighted images[45] across the sample. In addition, we assessed result replicability between recomputations by comparing binary identity of result files between analysis repetitions. To better estimate the amount of dissimilarity between recomputations, we also calculated mean squared error (MSE) over recomputations for a range of key VBM estimates, and the correlations between the brain atlasses included in the CAT toolbox.

(Re)use. After successful completion, results comprise a collection of different VBM-related measures for all images in the sample, represented in archives. For easier consumption, and as researchers are rarely interested in the full set of measures, the output DataLad dataset was subsampled into smaller "special purpose" datasets. These datasets contained a subset of the results in extracted, and optionally aggregated form, tailored to different research questions, for easier and faster access. This process relied on registering the main result DataLad dataset into a new tailored DataLad dataset via nesting ("Tailored results A/B" in Fig. 3), extracting and transforming the required files with provenance-tracking by `datalad run`, i.e., the same mechanism that captured provenance for the initial computation. This approach yields a transparently generated data view that can be updated by re-applying this transformation in case of changed inputs via the `datalad rerun` command.

As a concrete example we generated a DataLad dataset with tissue volume statistics for regions of interests in each parcellation and for all participants. We implemented a script that extracted aggregated noise-to-contrast-ratio, inhomogeneity-to-contrast-ratio, image quality rating, total intracranial volume, total gray matter volume, total white matter volume, total cerebral spinal fluid volume, total white matter hyperintensities volume, and total surface area into one CSV file per brain parcellation. Importantly, we limited the numerical representation of the scores in these tables to an empirically meaningful precision, thereby helping to suppress the undesirable impact of technical side-effects of non-deterministic algorithm implementations and floating point operations on the effective reproducibility of results for any practical purpose. These results are a fraction of the size and number of files of the total results, but sufficient for investigating VBM-related research questions. Using the encoded, machine-actionable provenance information, each result can be traced to the precise files they were generated from in a transparent and reproducible manner.

The direct computational output of the workflow on the UKB sample is therefore not a final result, but an intermediate representation optimized for storage and handling. More tailored views for concrete use cases can be optimized for access convenience. With this, we achieve a compromise between the desires of a data consumer and the demands of the storage infrastructure and operators.

Open tutorial. As license restrictions prevent open sharing of data and container image used in the UKB showcase, we implemented the processing framework for an additional use case, for which all components can be publicly shared in readily usable form. The resulting, fully populated DataLad dataset is publicly available at https://github.com/psychoinformatics-de/fairly-big-processing-workflow-tutorial. It can serve as a functional reference implementation that affords reproducibility based on machine-actionable provenance records. All setup steps were implemented in a bootstrap script available at https://github.com/psychoinformatics-de/fairly-big-processing-workflow/blob/main/bootstrap_forrest_fmriprep.sh

Self-contained processing specification as a DataLad dataset. As input data we employed a dataset with structural brain imaging data for 20 individuals[46] from the studyforrest.org project[30], linked as a subdataset at inputs/data. This is a BIDS-structured dataset published under the permissive PDDL license. It is publicly available as a DataLad dataset at https://github.com/psychoinformatics-de/studyforrest-data-structural.

For data processing we use fMRIprep's structural preprocessing pipeline[29] (version v20.2.0) that is freely available as a Singularity container in the DataLad dataset of the public Repronim container registry https://github.com/repronim/containers. With this pipeline, each T1-weighted MRI scan was corrected for intensity non-uniformity using `N4BiasFieldCorrection` v2.1.0[47] and skull-stripped using `antsBrainExtraction.sh` v2.1.0 (using the OASIS template). Spatial normalization to the ICBM 152 Nonlinear Asymmetrical template version 2009c[48] was performed through nonlinear registration with the `antsRegistration` tool of ANTs v2.1.0[49], using brain-extracted versions of both T1w volume and template. Brain tissue segmentation of CSF, WM, and GM was performed on the brain-extracted T1w using `FAST`[50] (FSL v5.0.9).

Environment and performance optimized orchestration. As both foundational DataLad datasets for input data and pipeline are available from public sources, their file content did not need to be stored on local infrastructure at all. Instead, the processing specification superdataset linked the two components with their GitHub URL, and individual compute jobs retrieved relevant input data from their associated web sources directly.

An example HTCondor-based job-scheduling setup for the HTC infrastructure used in the Open Tutorial showcase is included in the shared resources.

Execution and result consolidation workflow. For demonstration purposes the same execution workflow as for the UKB showcase was used. However, due to the small number of compute jobs, and the long individual runtime of each job, implementation details like the separation of clone sources and push targets, or the distinction of result file transfer and provenance metadata deposition only has negligible performance impact.

Data availability

Data from the UK Biobank project were obtained from a third party, UK Biobank, upon application. Interested parties can apply for data from UK Biobank directly, at http://www.ukbiobank.ac.uk.

Structural data from the Studyforrest project[46] (https://doi.org/10.12751/g-node.zdwr8e) are available at https://www.github.com/psychoinformatics-de/studyforrest-data-structural. The studyforrest derivatives computed by the tutorial workflow[51] (https://doi.org/10.5281/zenodo.6019794) are publicly available from https://www.github.com/psychoinformatics-de/fairly-big-processing-workflow-tutorial.

Code availability

All scripts used to process the data[52] are publicly available at https://www.github.com/psychoinformatics-de/fairly-big-processing-workflow (https://doi.org/10.5281/zenodo.6019782). The recipe used to build the CAT Singularity container[53] (https://doi.org/10.5281/zenodo.6021002) is publicly available at https://www.github.com/m-wierzba/cat-container.

Received: 10 November 2021; Accepted: 11 February 2022;
Published online: 11 March 2022

References

1. Draxl, C., Clifton, A., Hodge, B.-M. & McCaa, J. The Wind Integration National Dataset (WIND) Toolkit. *Applied Energy* **151**, 355–366 (2015).
2. Wilkinson, M. D. *et al*. The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data* **3**, 160018 (2016).
3. Wiener, M., Sommer, F., Ives, Z., Poldrack, R. & Litt, B. Enabling an Open Data Ecosystem for the Neurosciences. *Neuron* **92**, 617–621 (2016).
4. Craddock, C. *et al*. The Neuro Bureau Preprocessing Initiative: open sharing of preprocessed neuroimaging data and derivatives. *Front. Neuroinform*. (2013).
5. Portegies Zwart, S. The ecological impact of high-performance computing in astrophysics. *Nature Astronomy* **4**, 819–822 (2020).
6. Bzdok, D. & Yeo, B. T. T. Inference in the age of big data: Future perspectives on neuroscience. *NeuroImage* **155**, 549–564 (2017).
7. Horien, C. *et al*. A hitchhiker's guide to working with large, open-source neuroimaging datasets. *Nature Human Behaviour* **5**, 185–193 (2021).
8. Van Essen, D. C. *et al*. The WU-Minn Human Connectome Project: An Overview. *NeuroImage* **80**, 62–79 (2013).
9. Casey, B. *et al*. The adolescent brain cognitive development (abcd) study: imaging acquisition across 21 sites. *Developmental cognitive neuroscience* **32**, 43–54 (2018).
10. Matthews, P. M. & Sudlow, C. The UK Biobank. *Brain* **138**, 3463–3465 (2015).
11. Poldrack, R. A., Gorgolewski, K. J. & Varoquaux, G. Computational and Informatic Advances for Reproducible Data Analysis in Neuroimaging. *Annual Review of Biomedical Data Science* **2**, 119–138 (2019).
12. Botvinik-Nezer, R. *et al*. Variability in the analysis of a single neuroimaging dataset by many teams. *Nature* **582**, 84–88 (2020).
13. Kennedy, D. N. *et al*. Everything Matters: The ReproNim Perspective on Reproducible Neuroimaging. *Frontiers in Neuroinformatics* **13** (2019).
14. Halchenko, Y. O. *et al*. Datalad: distributed system for joint management of code, data, and their relationship. *Journal of Open Source Software* **6**, 3262 (2021).
15. Kurtzer, G. M., Sochat, V. & Bauer, M. W. Singularity: Scientific containers for mobility of compute. *PLOS ONE* **12**, e0177459 (2017).
16. Miller, K. L. *et al*. Multimodal population brain imaging in the uk biobank prospective epidemiological study. *Nature neuroscience* **19**, 1523–1536 (2016).
17. Hanke, M. *et al*. In defense of decentralized research data management. *Neuroforum* **27**, 17–25 https://www.degruyter.com/document/doi/10.1515/nf-2020-0037/html. Publisher: De Gruyter Section: Neuroforum. (2021).
18. Hess, J. git-annex. https://git-annex.branchable.com/.
19. Bryan, J. Excuse Me, Do You Have a Moment to Talk About Version Control? *The American Statistician* **72**, 20–27 (2018).
20. Thain, D., Tannenbaum, T. & Livny, M. Distributed computing in practice: the Condor experience. *Concurrency and Computation: Practice and Experience* **17**, 323–356 (2005).
21. Köster, J. & Rahmann, S. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics* **28**, 2520–2522 (2012).
22. De Smedt, K., Koureas, D. & Wittenburg, P. Fair digital objects for science: From data pieces to actionable knowledge units. *Publications* **8** (2020).
23. Ashburner, J. & Friston, K. J. Voxel-Based Morphometry—The Methods. *NeuroImage* **11**, 805–821 (2000).
24. Gaser, C. & Dahnke, R. Computational Anatomy Toolbox (CAT). http://www.neuro.uni-jena.de/cat/.
25. Poldrack, B., Wagner, A., Waite, A., Waite, L. & Hanke, M. A model implementation of a scalable data store for scientific computing with DataLad. *F1000Research* **10** (2021).
26. BIDS-contributors. The Brain Imaging Data Structure (BIDS) Specification. *Zenodo* https://doi.org/10.5281/zenodo.4085321 (2020).
27. Jette, M. A., Yoo, A. B. & Grondona, M. SLURM: Simple linux utility for resource management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, 44–60 (Springer-Verlag, 2002).
28. Destrieux, C., Fischl, B., Dale, A. & Halgren, E. Automatic parcellation of human cortical gyri and sulci using standard anatomical nomenclature. *NeuroImage* **53**, 1–15 (2010).
29. Esteban, O. *et al*. fMRIPrep: a robust preprocessing pipeline for functional MRI. *Zenodo* https://zenodo.org/record/4252786#.YBlHdiUo8UE (2020).

30. Hanke, M. *et al.* A high-resolution 7-Tesla fMRI dataset from complex natural stimulation with an audio movie. *Sci. Data* **1**, 140003 (2014).
31. Rokem, A., Dichter, B., Holdgraf, C. & Ghosh, S. S. Pan-neuro: Interactive computing at scale with BRAIN datasets. *OSF Preprints* (2021).
32. Kuprieiev, R. *et al.* Dvc: Data version control - git for data & models. *Zenodo*. https://doi.org/10.5281/zenodo.5562238 (2021)
33. Babuji, Y. *et al.* Parsl: Pervasive parallel programming in python. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '19, 25–36 (Association for Computing Machinery, New York, NY, USA, 2019).
34. Zaharia, M. *et al.* Apache spark: A unified engine for big data processing. *Commun. ACM* **59**, 56–65 (2016).
35. Rocklin, M. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference*, vol. 130, 136 (Citeseer, 2015).
36. Madduri, R. *et al.* Reproducible big data science: A case study in continuous fairness. *PloS one* **14**, e0213013 (2019).
37. Nüst, D. *et al.* Ten simple rules for writing dockerfiles for reproducible data science. *PLoS Comput Biol* **16**, e1008316 (2020).
38. Glatard, T. *et al.* Reproducibility of neuroimaging analyses across operating systems. *Frontiers in Neuroinformatics* **9** (2015).
39. Glatard, T. *et al.* Boutiques: a flexible framework to integrate command-line applications in computing platforms. *GigaScience* **7**, giy016 (2018).
40. Wagner, A. S. *et al.* The DataLad Handbook. *Zenodo* https://zenodo.org/record/3905791#.X_Xm5yUo8UE (2020)
41. Hanke, M., Waite, L. K., Poline, J.-B. & Hutton, A. datalad/datalad-ukbiobank: drop fix. *Zenodo* https://zenodo.org/record/4773629 (2021).
42. Ashburner, J. & Friston, K. J. Diffeomorphic registration using geodesic shooting and gauss–newton optimisation. *NeuroImage* **55**, 954–967 (2011).
43. Krause, D. & Thörnig, P. JURECA: Modular supercomputer at Jülich Supercomputing Centre. *Journal of large-scale research facilities JLSRF* **4** (2018).
44. Tange, O. Gnu parallel-the command-line power tool. *The USENIX Magazine* **36**, 42–47 (2011).
45. Dahnke, R., Ziegler, G., Grosskreutz, J. & Gaser, C. Quality Assurance in Structural MRI. http://rgdoi.net/10.13140/RG.2.2.16267.44321 (2015).
46. Hanke, M., Wagner, A. S., Waite, L. K. & Mönch, C. Studyforrest structural mri scans. *Gnode* https://doi.org/10.12751/g-node.zdwr8e (2022).
47. Tustison, N. J. *et al.* N4ITK: Improved N3 Bias Correction. *IEEE Transactions on Medical Imaging* **29**, 1310–1320 (2010).
48. Fonov, V., Evans, A., McKinstry, R., Almli, C. & Collins, D. Unbiased nonlinear average age-appropriate brain templates from birth to adulthood. *NeuroImage* **47**, S102 (2009).
49. Avants, B. B., Epstein, C. L., Grossman, M. & Gee, J. C. Symmetric diffeomorphic image registration with cross-correlation: Evaluating automated labeling of elderly and neurodegenerative brain. *Medical Image Analysis* **12**, 26–41 (2008).
50. Zhang, Y., Brady, M. & Smith, S. Segmentation of brain MR images through a hidden Markov random field model and the expectation-maximization algorithm. *IEEE Transactions on Medical Imaging* **20**, 45–57 (2001).
51. Wagner, A., Wierzba, M. & Hanke, M. psychoinformatics-de/fairly-big-processing-workflow-tutorial: Publication. *Zenodo* https://doi.org/10.5281/zenodo.6019794 (2022).
52. Wagner, A., Felix, H. & Wagner, A. psychoinformatics-de/fairly-big-processing-workflow: Publication. *Zenodo* https://doi.org/10.5281/zenodo.6019782 (2022).
53. Wierzba, M. FelixH. m-wierzba/cat-container: Publication. *Zenodo* https://doi.org/10.5281/zenodo.6021002 (2022).
54. Belhajjame, K. *et al.* PROV-DM: The PROV data model. *W3C Recommendation* **14**, 15–16 (2013).
55. Dahnke, R., Ziegler, G., Grosskreutz, J. & Gaser, C. Retrospective Quality Assurance of MR Images (2013).
56. Meyer, K., Hanke, M., Halchenko, Y., Poldrack, B. & Wagner, A. datalad/datalad-container 1.1.2. *Zenodo* https://doi.org/10.5281/zenodo.4445141 (2021).

Acknowledgements

The authors wish to thank Timo Dickscheid, Michał Szczepanik and Stephan Heunis for their feedback on earlier versions of this manuscript. This work was supported by European Union's Horizon 2020 research and innovation programme under grant agreements Human Brain Project (SGA3, H2020-EU.3.1.5.3, grant no. 945539) and VirtualBrainCloud (H2020-EU.3.1.5.3, grant no. 826421). The development of the DataLad software was supported by grants from the US National Science Foundation (NSF 1912266, 1429999) and the German Federal Ministry of Education and Research (BMBF 01GQ1905, 01GQ1411). MW was supported by ETIUDA grant received from the National Science Centre, Poland (2018/28/T/HS6/00507). This research has been conducted using the UK Biobank Resource under application number 41655.

Author contributions

A.S.W., A.Q.W, B.P., F.H., L.K.W., M.H., and M.W. conceived the setup of the framework. A.S.W., L.K.W, and M.H. piloted and documented an earlier version of this framework with a smaller dataset. B.P., L.K.W, and M.H. wrote the software to download and structure UK Biobank to BIDS. F. H., M.H., and M.W. containerized the CAT processing toolbox. M. H. implemented the HTCondor setup and bootstrapping. F. H. implemented the SLURM setup and conducted QC analysis on the results. A.S.W., and M.W. implemented the tutorial on studyforrest.org data. A.S.W. wrote the first draft of the manuscript. A.S.W., A.Q.W., B.P., F.H., L.K.W., M.H., M.W., and S.E. contributed to the conceptualization, writing, and editing of the manuscript. All authors read and approved the final draft.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to A.S.W.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

The Journal of Open Source Software

DataLad: distributed system for joint management of code, data, and their relationship

Yaroslav O. Halchenko*[1], Kyle Meyer[1], Benjamin Poldrack[2], Debanjum Singh Solanky[1], Adina S. Wagner[2], Jason Gors[1], Dave MacFarlane[3], Dorian Pustina[4], Vanessa Sochat[5], Satrajit S. Ghosh[6], Christian Mönch[2], Christopher J. Markiewicz[7], Laura Waite[2], Ilya Shlyakhter[8], Alejandro de la Vega[9], Soichi Hayashi[10], Christian Olaf Häusler[2, 11], Jean-Baptiste Poline[12], Tobias Kadelka[2], Kusti Skytén[13], Dorota Jarecka[6], David Kennedy[14], Ted Strauss[15], Matt Cieslak[16], Peter Vavra[17], Horea-Ioan Ioanas[18], Robin Schneider[19], Mika Pflüger[20], James V. Haxby[1], Simon B. Eickhoff[2, 11], and Michael Hanke†[2, 11]

1 Center for Open Neuroscience, Department of Psychological and Brain Sciences, Dartmouth College, Hanover, NH, USA **2** Institute of Neuroscience and Medicine, Brain & Behaviour (INM-7), Research Center Jülich, Jülich, Germany **3** McGill Center for Integrative Neuroscience, Montreal, Canada **4** CHDI Management/CHDI Foundation, Princeton, NJ, USA **5** Lawrence Livermore National Lab, Livermore, CA, USA **6** Massachusetts Institute of Technology, Cambridge, MA, USA **7** Stanford University, Stanford, CA, USA **8** Quest Diagnostics, Marlborough, MA, USA **9** The University of Austin at Austin, Austin, TX, USA **10** Indiana University, Bloomington, IN, USA **11** Institute of Systems Neuroscience, Medical Faculty, Heinrich Heine University Düsseldorf, Düsseldorf, Germany **12** Faculty of Medicine and Health Sciences, McConnell Brain Imaging Center, McGill University, Montreal, Canada **13** University of Oslo, Oslo, Norway **14** University of Massachusetts Medical School, Worcester, MA, USA **15** Montreal Neurological Institute, McGill University, Montreal, Canada **16** University of Pennsylvania, Philadelphia, PA **17** Department of Biological Psychology, Otto-von-Guericke-University Magdeburg, Magdeburg, Germany **18** Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, USA **19** Independent Developer, Germany **20** Potsdam Institute for Climate Impact Research (PIK) e. V., Potsdam, Germany

DOI: 10.21105/joss.03262

Software

- Review
- Repository
- Archive

Editor: Ariel Rokem

Reviewers:

- @szorowi1
- @jkanche

Submitted: 03 May 2021
Published: 01 July 2021

Summary

DataLad is a Python-based tool for the joint management of code, data, and their relationship, built on top of a versatile system for data logistics (git-annex) and the most popular distributed version control system (Git). It adapts principles of open-source software development and distribution to address the technical challenges of data management, data sharing, and digital provenance collection across the life cycle of digital objects. DataLad aims to make data management as easy as managing code. It streamlines procedures to consume, publish, and update data, for data of any size or type, and to link them as precisely versioned, lightweight dependencies. DataLad helps to make science more reproducible and FAIR (Wilkinson et al., 2016). It can capture complete and actionable process provenance of data transformations to enable automatic re-computation. The DataLad project (datalad.org) delivers a completely open, pioneering platform for flexible decentralized research data management (RDM) (Hanke, Pestilli, et al., 2021). It features a Python and a command-line interface, an extensible architecture, and does not depend on any centralized services but facilitates interoperability with a plurality of existing tools and services. In order to maximize its utility and target

*Contributed equally
†Contributed equally

audience, DataLad is available for all major operating systems, and can be integrated into established workflows and environments with minimal friction.

Statement of Need

Code, data and computing environments are core components of scientific projects. While the collaborative development and use of research software and code is streamlined with established procedures and infrastructures, such as software distributions, distributed version control systems, and social coding portals like GitHub, other components of scientific projects are not as transparently managed or accessible. Data consumption is complicated by disconnected data portals that require a large variety of different data access and authentication methods. Compared with code in software development, data tend not to be as precisely identified because data versioning is rarely or only coarsely practiced. Scientific computation is not reproducible enough, because data provenance, the information of how a digital file came to be, is often incomplete and rarely automatically captured. Last but not least, in the absence of standardized data *packages*, there is no uniform way to declare actionable data dependencies and derivative relationships between inputs and outputs of a computation. DataLad aims to solve these issues by providing streamlined, transparent management of code, data, computing environments, and their relationship. It provides targeted interfaces and interoperability adapters to established scientific and commercial tools and services to set up unobstructed, unified access to all elements of scientific projects. This unique set of features enables workflows that are particularly suited for reproducible science, such as actionable process provenance capture for arbitrary command execution that affords automatic re-execution. To this end, it builds on and extends two established tools for version control and transport logistics, Git and git-annex.

Why Git and git-annex?

Git is the most popular version control system for software development[1]. It is a distributed content management system, specifically tuned towards managing and collaborating on text files, and excels at making all committed content reliably and efficiently available to all clones of a repository. At the same time, Git is not designed to efficiently handle large (e.g., over a gigabyte) or binary files (see, e.g., Kenlon, 2016). This makes it hard or impossible to use Git directly for distributed data storage with tailored access to individual files. Git-annex takes advantage of Git's ability to efficiently manage textual information to overcome this limitation. File content handled by git-annex is placed into a managed repository annex, which avoids committing the file content directly to Git. Instead, git-annex commits a compact reference, typically derived from the checksum of a file's content, that enables identification and association of a file name with the content. Using these identifiers, git-annex tracks content availability across all repository clones and external resources such as URLs pointing to individual files on the web. Upon user request, git-annex automatically manages data transport to and from a local repository annex at a granularity of individual files. With this simple approach, git-annex enables separate and optimized implementations for identification and transport of arbitrarily large files, using an extensible set of protocols, while retaining the distributed nature and compatibility with versatile workflows for versioning and collaboration provided by Git.

[1] https://en.wikipedia.org/wiki/Git#Adoption

What does DataLad add to Git and git-annex?

Easy to use modularization. Research workflows impose additional demands for an efficient research data management platform besides version control and data transport. Many research datasets contain millions of files, but a large number of files precludes managing such a dataset in a single Git repository, even if the total storage demand is not huge. Partitioning such datasets into smaller, linked components (e.g., one subdataset per sample in a dataset comprising thousands) allows for scalable management. Research datasets and projects can also be heterogeneous, comprising different data sources or evolving data across different processing stages, and with different pace. Beyond scalability, modularization into homogeneous components also enables efficient reuse of a selected subset of datasets and for recording a derivative relationship between datasets. Git's *submodule* mechanism provides a way to nest individual repositories via unambiguously versioned linkage, but Git operations must still be performed within each individual repository. To achieve modularity without impeding usability, DataLad simplifies working with the resulting hierarchies of Git repositories via recursive operations across dataset boundaries. With this, DataLad provides a "monorepo"-like user experience in datasets with arbitrarily deep nesting, and makes it trivial to operate on individual files deep in the hierarchy or entire trees of datasets. A testament of this is datasets.datalad.org, created as the project's initial goal to provide a data distribution with unified access to already available public data archives in neuroscience, such as crcns.org and openfmri.org. It is curated by the DataLad team and provides, at the time of publication, streamlined access to over 260 TBs of data across over 5,000 subdatasets from a wide range of projects and dozens of archives in a fully modularized way.

Re-executable annotation of changes. Digital provenance is crucial for the trustworthiness and reproducibility of a research result, and contributes to the reusability aspect of the FAIR principles (Wilkinson et al., 2016). Knowing which code and data were used is essential, but, for changes that are programmatically introduced, how a command or script was invoked is another key piece of information to capture. One approach is to include this information in the Git commit message that accompanies a change, but doing so manually is tedious and error prone. To solve this, DataLad supports executing a command and automatically generating a commit message that includes a structured record with comprehensive details on the invocation. In addition to providing reliable information about past command-line invocations, these machine-readable records make it possible to easily re-execute commands (e.g., to verify if a result is computationally reproducible or to apply an analog change to a different dataset state).

Targeted interfaces and interoperability adapters. Interoperability with scientific or commercial computing and storage services allows researchers to integrate data management routines into their established workflows with minimal friction. Git can already interact with other local or remote repositories via standard or custom network transport protocols. DataLad implements support for additional services that require custom protocols, such as the Open Science Framework (OSF) (Hanke, Poldrack, et al., 2021). Git-annex readily provides access to a wide range of external data storage resources via a large set of protocols. DataLad builds on this support and adds, for example, more fine-grained access (e.g. direct access to individual components contained in an archive hosted on cloud storage) or specialized services, such as XNAT (www.xnat.org). Efficient and seamless access to scientific data is implemented using the *special remote* protocol provided by git-annex (Hess, 2013), through which external tools, like DataLad, can provide custom transport functionality transparently to a user. With this approach, DataLad and other projects can jointly facilitate access to an ever-growing collection of resources (Hess, 2011) and overcome technological limitations of storage solutions, like file size or inode limits.

Metadata management. Metadata are essential for scientific discovery, as they are routinely used to complete all data analyses. Metadata is the core concept behind Git and git-annex functioning: Git records and uses metadata about each change (author, date, description,

original state, etc) for each commit. Git-annex manages metadata about content availability and allows to associate additional arbitrary key-value pairs to any annexed content. Files managed by git and git-annex can in turn be of standardized file formats comprised of data with rich metadata records. Moreover, entire repositories might conform to a standard (e.g., BIDS (Gorgolewski et al., 2016)) or provide a standardized dataset descriptor (e.g., Frictionless data package). To facilitate metadata availability and utility, DataLad provides an extensible framework for metadata extraction and aggregation. Metadata for each file (contained in the file or recorded by git and git-annex) or associated with the entire dataset can be extracted into a collection of machine-readable (JSON) records and aggregated across all contained sub-datasets. Such simple mechanism makes it possible to provide immediate access to metadata about all contained data within a larger super-dataset (such as datasets.datalad.org).

Overview of DataLad and its ecosystem

Design principles

Besides the free software nature and open processes of the DataLad project, the development of DataLad is guided by four principles to ensure its open and domain agnostic nature, to maximize the long-term utility of its datasets and to minimize users' technical debt:

- Datasets and the files they comprise are the only two recognized entities
- A dataset is a Git repository with an *optional* annex
- Minimization of custom procedures and data structures
- Complete decentralization, with no required central server or service, but maximum interoperability with existing 3rd-party resources and infrastructure

In conjunction, these principles aim to reduce the risk of adoption for DataLad users. They foster the resilience of an ecosystem using DataLad datasets as a standard package format for any digital objects by avoiding any critical dependency on service deployments governed by central entities, and even on DataLad itself, for access to any resources managed with DataLad.

DataLad core

The `datalad` Python package provides both a Python library and a command line tool which expose core DataLad functionality to fulfill a wide range of decentralized RDM use cases for any domain. All DataLad commands operate on *DataLad datasets*. On a technical level, these datasets are Git repositories with additional metadata. On a conceptual level, they constitute an overlay structure that allows to version control files of any size, track and publish files in a distributed fashion, and record, publish, and execute actionable provenance of files and file transformations. Figure 1 summarizes key commands and concepts for local or distributed data and provenance management.

The Journal of Open Source Software

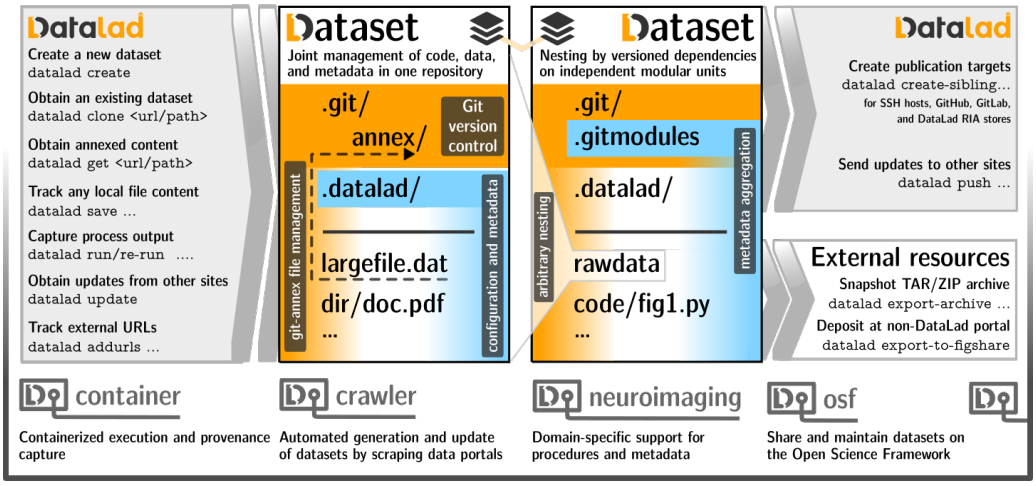


Figure 1: Schematic overview of a dataset, datasets nesting, and selected commands for content and dataset management. A more comprehensive cheatsheet is provided in the DataLad handbook (Wagner, 2020).

DataLad's features can be flexibly integrated into standard scientific workflows. For example, by using the concept of dataset nesting to modularize the evolution of a research project, DataLad can fulfill the YODA principles for reproducible science (YODA Team, 2021), and, with this simple paradigm, facilitate efficient access, composition, scalability, reuse, sharing, and reproducibility of results (see Figure 2). With core commands that aim to simplify operation of the underlying tools, DataLad makes RDM workflows more accessible to novices and experts alike. Importantly, compatibility with all Git/git-annex functionality is retained.

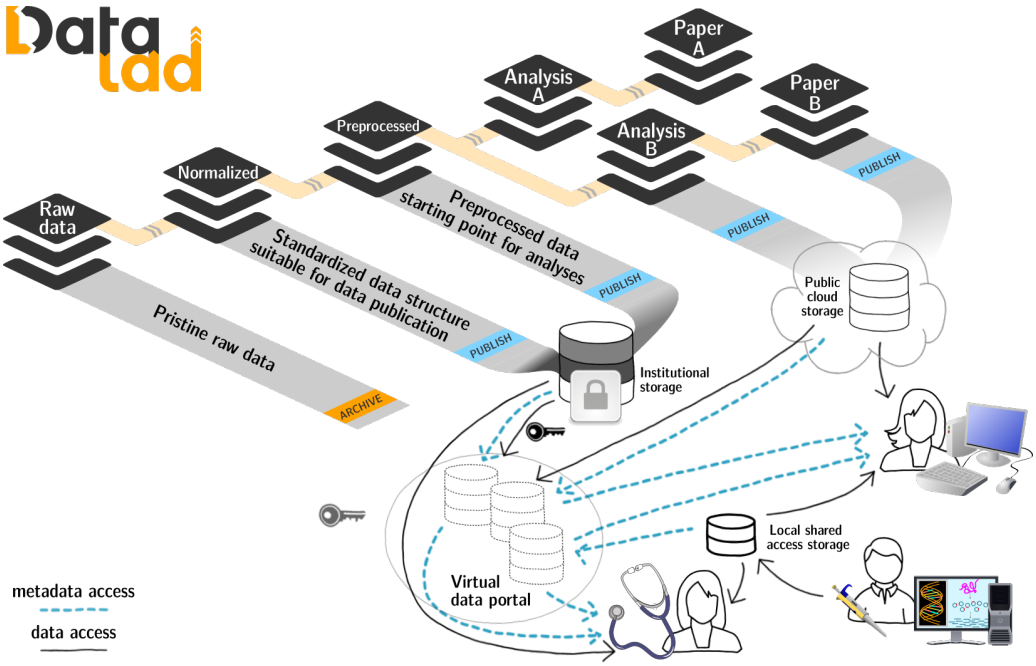


Figure 2: DataLad datasets are reusable modular components, which can be nested to establish a complete provenance trail all the way from a publication to the original data. Various access schemes to datasets and data are provided, and further extensibility is a key architectural property.

Extensions

Like Git and git-annex, DataLad core is a generic tool that is not specifically tuned to particular data types or use cases. It offers a robust foundation to build more specialized solutions on top of. *DataLad extensions*, stand-alone Python packages with additional DataLad functionality, extend DataLad with domain-focused or technology-specific features. A dedicated datalad-extension-template repository provides a starting point for creating new DataLad extensions. Some established extensions include:

- datalad-container (Meyer et al., 2021) to simplify management and use of Docker and Singularity containers typically containing complete computational environments
- datalad-crawler (Halchenko et al., 2021) to automate creation and updates of DataLad datasets from external resources
- datalad-neuroimaging (Hanke et al., 2020) to provide neuroimaging-specific procedures and metadata extractors
- datalad-osf (Hanke, Poldrack, et al., 2021) to collaborate using DataLad through the Open Science Framework (OSF)
- datalad-ukbiobank (Hanke, Waite, et al., 2021) obtain and BIDS-normalize imaging data releases of the UKBiobank

The same mechanism of extensions is used for rapid development of new functionality to later be moved into the core tool (e.g., datalad-metalad). The datalad-extensions repository provides a list of extensions and continuous integration testing of their released versions against released and development versions of the DataLad core.

External uses and integrations

DataLad can be used as an independent tool to access and manage data (see e.g. Wittkuhn & Schuck (2021), Gautheron et al. (2021), Gautheron (2021)) or as a core technology behind another tool or a larger platform (e.g. Far et al. (2021)). TemplateFlow (Ciric et al., 2021) uses DataLad for the management of neuroimaging templates. OpenNeuro uses DataLad for data logistics with data deposition to a public S3 bucket. CONP-PCNO adopts aforementioned features for modular composition and nesting to deliver a rich collection of datasets with public or restricted access to data. ReproMan integrates with DataLad to provide version control and data logistics. www.datalad.org/integrations.html provides a more complete list of DataLad usage and integration with other projects, and Hanke, Pestilli, et al. (2021) provides a systematic depiction of DataLad as a system for decentralized RDM used by a number of projects.

Documentation

Developer-focused technical documentation at docs.datalad.org, with detailed descriptions of the command line and Python interfaces, is automatically generated from the DataLad core repository. A comprehensive handbook (Wagner et al., 2021a) provides user-oriented documentation with an introduction to research data management, and numerous use case descriptions for novice and advanced users of all backgrounds (Wagner et al., 2021b).

Installation

The handbook provides installation instructions for all major operating systems. DataLad releases are distributed through PyPI, Debian, NeuroDebian, brew, and conda-forge. The

datalad-installer (also available from PyPI) streamlines the installation of DataLad and its dependencies, in particular git-annex, across a range of deployment scenarios, such as continuous integration systems, or high-performance computing (HPC) environments.

Development

DataLad has been developed openly in a public repository (github.com/datalad/datalad) since its inception in 2013. At the time of this publication, the repository amassed over 13.5k commits, 2.5k merged PRs, and 2.3k closed (+700 open) issues from over 30 contributors. Issue tracker, labels, milestones, and pull requests are used to coordinate development. The development process of DataLad is not isolated from its foundational building blocks. For every new feature or bug fix the most appropriate software layer is determined to maximize the size of the benefitting user base and, importantly, also the associated developer audience. This strategy aims to achieve a robust integration with the larger open source software ecosystem, and simultaneously minimize the total technical debt carried solely by the DataLad development team. Consequently, DataLad development is tightly connected to and involves frequent communication with the git-annex project and its main developer Joey Hess (Hess & DataLad Team, 2016). To guarantee robust operation across various deployments, DataLad heavily utilizes continuous integration platforms (Appveyor, GitHub actions, and Travis CI) for testing DataLad core, building and testing git-annex (in a dedicated github.com/datalad/git-annex), and integration testing with DataLad extensions (datalad-extensions).

Contributions

DataLad is free and open source software and encourages unconstrained use and reuse in any context. Therefore, DataLad is released under DFSG- and OSI-compliant MIT/Expat license. License terms for reused components in the code-base are provided in the COPYING file. The project aims to promote contributions rather than detached developments in forks and anyone is highly welcome to contribute to DataLad in any form under these terms. Technical and procedural guidelines for such contributions can be found in the CONTRIBUTING.md file shipped within DataLad's source repository. Contributors are acknowledged on the project website, and also credited in the form of co-authorship in the Zenodo-based archival of software releases. All co-authors of this paper as well as the contributors acknowledged below have added to the project with code- or non-code-based contributions, and we thank past, present, and future contributors of this community for their involvement and work.

Conflicts of interest

There are no conflicts to declare.

Acknowledgements

We express our gratitude to Joey Hess for the development and maintenance of git-annex, and for years of productive collaboration with the DataLad team. We would like to extend our gratitude to Joey Zhou, Matteo Visconti di Oleggio Castello, John T. Wodder II, Satya Ortiz-Gagné, Jörg Stadler, Andrew Connolly, John Lee, Nolan Nichols, Elizabeth DuPre, Cécile Madjar, Gergana Alteva, Timo Dickscheid, Alex Waite for notable contributions to the codebase, bug reports, recommendations, and promotion of DataLad.

DataLad development was made possible thanks to support by NSF 1429999, 1912266 (PI: Halchenko) and BMBF 01GQ1411, 01GQ1905 (PI: Hanke) through the CRCNS program.

It received significant contributions from ReproNim 1P41EB019936-01A1 (PI: Kennedy) and DANDI 5R24MH117295-02 (PIs: Ghosh, Halchenko) NIH projects. It also received contributions from the Canadian Open Neuroscience Platform and the NeuroHub (Co-PI: Poline) projects thanks in part to funding from a Brain Canada Platform Support Grant Competition Award in addition to funds and in-kind support from sponsor organizations, and from the Canada First Research Excellence Fund, awarded through the Healthy Brains, Healthy Lives initiative at McGill University, and the Brain Canada Foundation with support from Health Canada. This development was supported by the European Regional Development Fund (Project: Center for Behavioral Brain Sciences Magdeburg, Imaging Platform, PI: Hanke), the European Union's Horizon 2020 research and innovation programme under grant agreements Human Brain Project (SGA3, H2020-EU.3.1.5.3, grant no. 945539; Co-Investigators: Eickhoff, Hanke), and Virtual Brain Cloud (H2020-EU.3.1.5.3, grant no. 826421; PI: Eickhoff), the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grants SFB 1451 (431549029; Co-PIs: Eickhoff, Hanke) and IRTG 2150 (269953372; Co-PIs: Eickhoff, Hanke).

References

Ciric, R., Lorenz, R., Thompson, W., Goncalves, M., MacNicol, E., Markiewicz, C., Halchenko, Y., Ghosh, S., Gorgolewski, K., Poldrack, R., & Esteban, O. (2021). *TemplateFlow: A community archive of imaging templates and atlases for improved consistency in neuroimaging*. https://doi.org/10.21203/rs.3.rs-264855/v1

Far, M. S., Stolz, M., Fischer, J. M., Eickhoff, S. B., & Dukart, J. (2021). JTrack: A digital biomarker platform for remote monitoring in neuropsychiatric and psychiatric diseases. *CoRR*, *abs/2101.10091*. https://arxiv.org/abs/2101.10091

Gautheron, L. (2021). *The LAAC superdataset: Datasets of infant day-long recordings*. https://github.com/LAAC-LSCP/datasets

Gautheron, L., Rochat, N., & Cristia, A. (2021). *Managing, storing, and sharing long-form recordings and their annotations*. https://doi.org/10.31234/osf.io/w8trm

Gorgolewski, K. J., Auer, T., Calhoun, V. D., Craddock, R. C., Das, S., Duff, E. P., Flandin, G., Ghosh, S. S., Glatard, T., Halchenko, Y. O., Handwerker, D. A., Hanke, M., Keator, D., Li, X., Michael, Z., Maumet, C., Nichols, B. N., Nichols, T. E., Pellman, J., … Poldrack, R. A. (2016). The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. *Scientific Data*, *3*(1). https://doi.org/10.1038/sdata.2016.44

Halchenko, Y., Hanke, M., Meyer, K., Olson, T., Chaselgrove, & Poldrack, B. (2021). *datalad-crawler: DataLad extension for crawling external resources*. Zenodo. https://doi.org/10.5281/ZENODO.2558512

Hanke, M., Halchenko, Y., Poldrack, B., & Meyer, K. (2020). *datalad-neuroimaging: DataLad extension for neuroimaging*. Zenodo. https://doi.org/10.5281/ZENODO.3874225

Hanke, M., Pestilli, F., Wagner, A. S., Markiewicz, C. J., Poline, J.-B., & Halchenko, Y. O. (2021). In defense of decentralized research data management. *Neuroforum*, *27*(1). https://doi.org/10.1515/nf-2020-0037

Hanke, M., Poldrack, B., Wagner, A. S., Huijser, D., Sahoo, A. K., Boos, M., Steinkamp, S. R., Guenther, N., & Appelhoff, S. (2021). *datalad-osf: DataLad extension for integration with OSF.io*. Zenodo. https://doi.org/10.5281/ZENODO.3900277

Hanke, M., Waite, L. K., Poline, J.-B., & Hutton, A. (2021). *DataLad extension for working with the UKbiobank* (Version 0.3.3) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.4773629