REPRODUCIBILITY OF BIG DATA PROCESSING PIPELINES IN NEUROIMAGING **From Tristan:** I FIND THIS TITLE TOO GENERAL

Mohammad Ali Salari

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Doctor of Philosophy
Concordia University
Montréal, Québec, Canada

December 2021

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Mr. Mohammad Ali Salari**

Entitled: **Reproducibility of Big Data Processing Pipelines in Neuroimaging From Tristan: I find this title too general**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Ph.D.)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

_____ Chair

_____ External Examiner

_____ Examiner

_____ Examiner

_____ Examiner

_____ Supervisor

Approved _____
Chair of Department or Graduate Program Director

_____ 20 _____ _____

Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Reproducibility of Big Data Processing Pipelines in Neuroimaging **From Tristan:** I find this title too general

Mohammad Ali Salari, Ph.D.
Concordia University, 2022

From Tristan: Needs editing The changes of computational infrastructure, including operating system, software version, and hardware architecture, introduce variability in neuroimaging analyses that could affect the reproducibility of the scientific conclusions. This is probably due to the creation, propagation, and amplification of numerical instabilities in analysis pipelines. In this regard, it is critical to identify numerical instabilities to make experiments computationally reproducible. In this thesis, we characterize the numerical stability of commonly-used complex pipelines in the context of neuroimaging analysis across the operating systems and provide accessible tools for developers and researchers to evaluate their pipelines and findings. First, we present Spot tool that identifies the processes from which differences originate and the path along which they propagate in the pipelines. In the next step, to study the numerical instabilities more comprehensively, we introduce controlled numerical perturbations to the floating-point computations using the Monte-Carlo Arithmetic method. We propose an interposition technique to model the effect of operating system updates on analysis pipelines using the Monte-Carlo arithmetic. Finally, leveraging the interposition technique, we compare numerical variability with tool variability in an fMRI analysis. All the methods implemented in this thesis can be used to facilitate further investigations toward stabilizing pipelines.

Acknowledgments

First and foremost, I would like to thank my research supervisor Tristan Glatard for all the invaluable support, understanding, guidance, and encouragement he has given me over the past years. Without his assistance and dedicated involvement in every step throughout the process, the success of this thesis would not be possible. I would also like to thank my research team members, Gregory Kiar and Yohan Chatelain, for their generous advice and ongoing contributions to my work. Special thanks to all the BIN Lab members for a cherished time spent together in the lab and social settings, with a special mention to Valerie and Martin; you guys have been wonderful labmates. Getting through my dissertation required more than academic support, and I have many people to thank for listening to and, at times, having to tolerate me over the past years. My appreciation also goes out to my family and friends for their unconditional love, encouragement, and support throughout my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reproducibility is regarded as a fundamental concept in the scientific community. Research findings are expected to be reproducible so that their authenticity and reliability can be evaluated. The goal of my research is to investigate the reproducibility of analysis across different computing environments. In particular, we are mostly interested in neuroimaging as a case study. We aim to present techniques to evaluate the numerical instability of analysis across different computing environments instead of masking the reproducibility problem by fixing parameters.

In this chapter, we summarize the main definitions and principles relevant to reproducibility. We describe the context of the current "reproducibility crisis" acknowledged in several scientific disciplines. Multiple studies have shown that research findings could not be reproduced by independent researchers, or even by the original researchers themselves. We discuss the main causes for this lack of reproducibility, focusing on the computational aspects. Additionally, we describe different kind of analysis of neuroimaging data and their implemented software which are considered through this thesis.

1.1 Reproducibility Definitions

There are different definitions for the terms reproducibility, repeatability, and replicability, which leads to confusion since the same words are used for different concepts. Here we present different terminologies found in the literature and summarized in [105] (see Table 1).

According to Peng's definition [102], reproducibility is defined as the ability to regenerate the same results as the original findings when the experiment is reanalyzed given exactly the same analytic methods and data. Reproducibility ensures that independent scientists can

reproduce the same results using the same data and procedure as published in the original publication. On the contrary, replicability is defined as the ability to obtain similar results as published in the original study when the experiment is reimplemented using independent data and analytic methods. Replicability confirms scientific claims and ensures that independent investigators can produce consistent results, using new data and methods. Peng introduced the idea of reproducibility spectrum based on his definition of reproducibility, which defines a minimum standard to evaluate the authenticity of scientific claims. In this spectrum, according to what data and sources are available, a full replication or no replication of a study can be achieved. The same definitions of reproducibility and replicability are also used by Schwab et al. [112].

Donoho et al. [32] defined reproducible computational research as a process where "all details of computations such as code and data are made conveniently available to others". The authors associate reproducible research with open science, including open code and data. They observe that reproducibility can be achieved by publishing the experimental resources over the Internet, which facilitates versioning, testing, discovery and access to the research materials.

In addition, Goodman et al. [47] renamed Peng's reproducibility and replicability as methods reproducibility and results reproducibility respectively, and adopted a new terminology called inferential reproducibility. From Goodman's terminology, exactly the same data and procedure are reanalyzed in methods reproducibility. Result reproducibility is equivalent to Peng's replicability terminology which is defined as getting almost the same results compared to the original study from an independent replication of a study. Also, inferential reproducibility is defined as getting the same conclusions from either a reanalysis of the original study or an independent replication of a study with different data and analysis procedures.

Furthermore, the Association for Computing Machinery (ACM) [3] proposes three different categories of repeatability, replicability, and reproducibility. Repeatability is defined as repeating computation by the same experimental setup including operator team, operating conditions, location, and measuring system. Similar to replicability, repeatability uses identical experimental conditions except performer team. This means that an independent group can achieve the same results through the same experimental parameters. Additionally, reproducibility is measured by performing different experimental setups via different teams independently. It should be noted that reproducibility and replicability are used inversely compared to Peng's definitions.

Table 1: Overview of definitions.

Schwab et al.$_{(2000)}$	Donoho et al.$_{(2009)}$	Peng$_{(2011)}$	ACM$_{(2016)}$	Goodman et al.$_{(2016)}$
Reproducibility Replicability	Open code and data	Reproducibility spectrum	Repeatability Replicability Reproducibility	Method reproducibility Results reproducibility Inferential reproducibility

In addition, numerical reproducibility is defined as the ability to regenerate bit for bit identical results from multiple runs [56]. Two files will be considered numerically reproducible if they have identical binary contents. Binary comparison is calculated by comparing checksums. It must be pointed out that a computation might be reproducible based on Peng's definition, but not be numerically reproducible. For instance, small numerical errors created during the pipeline execution may hamper numerical reproducibility, but be negligible in the final results.

Reproducibility, as the cornerstone of scientific research, guarantees the reliability of results. A reproducible study provides a context in which one can get results consistent with the original work. In addition, it not only saves a great deal of time, but also enables others to use existing works as a part of their experiments [105]. In our work, we follow Peng's definition of reproducibility unless we directly refer to numerical reproducibility. We seek to identify why such reproducibility may not be ensured, focusing particularly on computational aspects.

1.2 Reproducibility Crisis

Recently, scientists began to realize that the results of many scientific experiments were neither replicable nor reproducible. This realization is termed the reproducibility crisis. In this section, we provide an overview of evidence for the reproducibility crisis, which has raised important concerns in the scientific community.

Ioannidis [57] introduces an important framework to demonstrate the probability that research findings are false, and the propagation of valid findings in a given research field. He defined biased research as "the combination of various design, data, analysis, and presentation factors that tend to produce research findings when they should not be produced". Consequently, biased research, focused on an individual discovery rather than on broader evidences, decreases the chance of true findings. He concluded that "most of the research claims are less likely to be true than false for most fields and research designs". The author

argued that the probability of true findings is highly dependent on the number of similar studies in a scientific field, the number of researchers/teams involved in the study, and the flexibility of analytic models, definitions, and outcomes. For example, the smaller the studies conducted in a scientific field, the less likely the research findings are to be true.

To highlight the importance of scientific reproducibility, the survey in [8] collected data from 1500 scientists among different disciplines mostly from biology, medicine, and engineering. This report found that 70% of the scientists polled could not replicate another scientist's findings, and even 50% failed to reproduce their own results. Moreover, this study listed some of the main reasons that lead to irreproducibility of analysis such as poor statistics, the pressure to publish and then selective analysis. With this, over 50% of the scientists polled believed that there was a significant crisis.

Furthermore, some studies underline the reproducibility issues of current analysis methods in neuroimaging [67, 94, 34]. For example, to evaluate reproducibility of a group of functional MRI (fMRI) analyses, the study in [34] collected resting-state fMRI data from 499 healthy controls. Using this dataset, they found that the most common software packages for fMRI analysis (SPM, FSL, AFNI) can result in a high degree of false positives, up to 70% compared with the expected 5%. These results question the validity of some 40,000 fMRI studies and may have a large impact on the interpretation of neuroimaging results. All these evidences show a significant crisis in reproducibility of experiments that should be taken into consideration in scientific communities.

1.3 Main Computational Causes of Irreproducibility

The main barrier to reproducibility in many cases is that the analysis program, data, and analytic methods are no longer available. Addressing this problem requires the development of a culture of reproducibility among scientific community, which enables the third party to reproduce the same experiment [102, 114].

From the computational point of view, reasons such as the lack of details of the computational environments can contribute to irreproducibility of research results. Analyses need sufficient information on code, software, hardware, and implementation details to be computationally reproducible. In addition, capturing such information is complicated, particularly in domains where results rely on a sequence of complex analyses such as neuroimaging pipelines. To overcome this complexity, a mechanism called provenance capturing is designed to encompass all dependency information of the computational analysis such as input/output

data, processing steps, and detail of computing environments.

Furthermore, the variety of computational infrastructures including workstation types, parallelization methods, operating systems, and analysis packages are known to influence reproducibility because of the creation of small numerical errors [51, 31, 46]. For instance, we will explain further that different order of summation operation of floating-point numbers can lead to creation of small numerical differences. The propagation and amplification of these tiny errors by analysis pipelines may cause reproducibility issues. We will discuss in more details the effect of each one of these factors in Chapter 2.

1.4 Analyzing Neuroimaging Data

There are many different kinds of imaging techniques to acquire brain image data. The most common techniques are structural magnetic resonance imaging (sMRI), functional magnetic resonance imaging (fMRI) and diffusion magnetic resonance imaging (dMRI).

Structural neuroimaging deals with the anatomical structure of the brain and helps diagnose brain injury and certain diseases such as tumor and stroke. The main software packages used for sMRI are CIVET [4], FreeSurfer [39], and FSL (FMRIB Software Library) [63]. As opposed to structural imaging, functional imaging is used to measure brain function based on specific tasks completed by subjects such as listening to sounds, reading, or small movements. Functional imaging identifies the areas of the brain that are involved with responding to the tasks. In addition to task-based fMRI, an explicit task may not be performed to identify the functional activity of brain in a resting-state condition (RS-fMRI). The main software packages that implement fMRI processing are SPM (Statistical Parametric Mapping) [2], FSL(FMRIB Software Library) [63], and AFNI (Analysis of Functional NeuroImages) [23]. Diffusion imaging is another kind of MRI analysis that measures the anatomical connectivity between regions, and the main toolboxes are DIPY (Diffusion Imaging in Python) [40], MRtrix [117], and FSL.

Depending on the analysis type, several steps can be involved in a neuroimaging study. Generally, the analysis procedure can be divided into pre-processing and statistical steps. Pre-processing steps are taken to prepare data for the statistical analysis and are common between all analysis modalities, including brain extraction to separate the brain tissues from the other parts, or brain alignment which aligns a brain extracted image with a reference image such as the one produced by MNI (Montreal Neuroimaging Institute) [36]. After pre-processing steps, depending on the modality of analyses (e.g., sMRI, fMRI, and dMRI),

statistical analyses are applied to make inferences.

The various pre-processing and analysis steps involved in a neuroimaging experiment are often combined in programs called workflows or pipelines. Pipelines are used to automate data analysis and accelerate the processing of complicated analyses.

1.5 Thesis Outline

This thesis aims to study the numerical stability of neuroimaging pipelines focusing on the effect of operating system variability. For this purpose, we leverage system call interception techniques including the ReproZip tool, and perturbation models such as Monte-Carlo Arithmetic (MCA) [100] as an extension of standard floating-point arithmetic that exploits randomness in basic floating-point operations. The major contributions of my thesis are listed below as the separate chapters that we published or aim to publish.

C.I – File-based localization of numerical perturbations in data analysis pipelines (Chapter 3)

C.II – Accurate simulation of operating system updates in neuroimaging using Monte-Carlo arithmetic (Chapter 4)

C.III – Comparing tool variability and numerical variability in fMRI analyses (Chapter 5)

In Chapter 2, we will review the background material related to this thesis in general. Chapter 3 will introduce Spot, a tool to detect the source of numerical differences in complex pipelines executed in different operating systems. This chapter is completed and published in the GigaScience journal. Chapter 4 will then study whether the MCA method is a good perturbation model for evaluating pipeline stability across operating systems. This chapter is also completed and published in the MICCAI workshop on Uncertainty for Safe Utilization of Machine Learning in Medical Imaging (UNSURE). Chapter 5 will present a comparison of numerical and software variability through Monte-Carlo arithmetic. This chapter is under review, and we aim to submit it to the Human Brain Mapping (HBM) journal by the end of Fall 2021 **From Tristan:** update. The thesis will then follow with a discussion and a conclusion in Chapters 6 and 7, respectively.

1.6 Authors Contributions

I was responsible for software development, data processing, analysis, drafting the manuscript, and designing figures for each manuscript. Tristan Glatard was responsible for supervising

and supporting all of my contributions. The contributions of authors to each publication are described below.

C.I – File-based localization of numerical perturbations in data analysis pipelines

I was responsible for the tool development, data processing, analysis, drafting the manuscript, and designing the figures. Lindsay B. Lewis and Alan C. Evans provided input on the dataset and pipelines, reviewed the results, and approved the final version of the manuscript. Gregory Kiar and Tristan Glatard supported development processes and data visualization. Tristan Glatard edited the manuscript, contributed to the interpretation of results, and supervised the findings of this work.

C.II – Accurate simulation of operating system updates in neuroimaging using Monte-Carlo arithmetic

I was responsible for the software implementation, data processing, analysis, drafting the manuscript, and designing the figures. All authors contributed to the editing of the manuscript, experimental design and discussed the results. Yohan Chatelain helped with Monte-Carlo arithmetic simulations and software testing. Gregory Kiar and Tristan Glatard provided software development support. Tristan Glatard supervised the findings of this work.

C.III – Comparing tool variability and numerical variability in fMRI analyses

I was responsible for reproducing the experiments, data processing, drafting the manuscript, and designing the figures. Gregory Kiar, Yohan Chatelain, and Tristan Glatard contributed to the experimental design and interpretation of results. Tristan Glatard edited the manuscript and supervised the findings of this work.

Chapter 2

Literature Review

In this chapter, we present previous works that investigated the effect of computational environments on scientific results: we provide results that show the magnitude of the effect of computing environment changes such as hardware and software implementations. Next, we review techniques and tools to enhance the reproducibility of the experiments including code and data sharing methods using version control systems, and virtualization techniques to encapsulate computational variability of the analysis. Finally, provenance management tools are described to collect and represent the analysis dependencies.

2.1 Computational Reproducibility

There have been many works investigating the reproducibility of computational pipelines in the past few years. In general, analysis results are not reproducible across small perturbations of the execution environments, including hardware configuration or operating system.

Changes in the computational environment may introduce small numerical errors, subsequently propagated and amplified by pipelines. In this case, the analysis pipelines are said to be numerically unstable. Numerical instability is a characteristic of the pipelines which amplify small numerical errors and then hamper the reproducibility of the analyses depending on the length of the pipeline and magnitude of the errors. In many cases, numerical instability is an important issue for reproducibility.

The following sections will discuss the effect of influential elements on reproducibility, in particular workstation type, parallelization techniques, operating system changes, analysis software variety, and perturbations applied in input data.

2.1.1 Effect of Hardware Resources

The hardware configuration of computers has been detected as an influential source of irreproducibility [56]. Such differences are particularly noticeable across computing processors such as CPUs (Central Processing Units), GPUs (Graphics Processing Units) and APUs (Accelerated Processing Units), mainly due to conflicts of floating-point units (FPU) with the IEEE-754 standard when arithmetic precision of the floating-point values are not specified uniformly.

Even using the same arithmetic precision, it is difficult to achieve bitwise identical results across different hardware resources. Recent studies show that hardware developments to improve computational performance sacrifice numerical reproducibility [33, 26]. For instance, code optimization techniques embedded inside the CPUs, known as out-of-order execution (dynamic scheduling) paradigm, impede the reproducibility. In this paradigm, the processors might execute instructions out of the original order they appear based on the availability of input data and execution units to use resources efficiently [123]. Therefore, it might compute floating-point operations in different order, which often leads to different results. In this case, these papers [33, 26] showed that some operations in particular sum and division are not associative because of different rounding of the intermediate floating-point results.

Furthermore, the study in [65] implements acoustic wave equation to see the effect of processor architecture on results. The authors illustrate irreproducible results across different processors including AMD CPU, NVIDIA GPU, and AMD APU, even using the same IEEE-754 standard. The results numerically vary from one architecture to another, the maximal relative difference between results is of 10^{-1} to 1 and its mean value is 10^{-5}. Such differences often occur due to rounding errors generated by different orders in the sequence of arithmetic operations. Indeed, this is already a challenge on today's platforms.

In neuroimaging, it is important to evaluate the consistency of results when they are executed on a heterogeneous computing system. For this purpose, a number of tests were conducted in [51] to gain insight into the variability of results from neuroimaging packages based on different data processing conditions like different workstation types. In this paper, two different types of workstations are compared: an HP (Hewlett Packard) one using Centos 5.3 and 8 CPU cores, and a Mac one using OSX 10.5.8 and 2 CPU cores. This study shows significant absolute differences among the volumes of anatomical structures obtained on the two different workstations.

2.1.2 Effect of Parallelization

Developers leverage parallelization techniques to accelerate the execution performance at different levels, from multi-threaded programming to high-performance computing (HPC). Within such computations, contrary to sequential implementations, the execution order of the processes may change in different runs. Consequently, several runs of the parallelized code may produce different results, even on the same computer.

To show the existence of such issues, the impact of the number of processors on numerical reproducibility is studied in [31]. This study simulated the process of deformation of metal sheets in the packaging industry to measure local change of the sheet thickness using different number of processors. Results obtained different sheet thicknesses, which shows the amplification of rounding errors in summations after running the same simulation on the same computers with different number of processors. This proved that summation operation is not associative because of different rounding of the intermediate floating-point results, even using the standard IEEE double-precision arithmetics. Therefore, final result of the summation depends on the order in which values are processed which could be changed by the number of processors.

Another statistical simulation showed reproducibility failures in multi-core processing performed on GP-GPUs and multi-core CPUs [116]. Multi-core architectures enable multi-threaded environment for running numerical intensive applications at high speeds. This study showed that the stability of molecular dynamics simulation results is not guaranteed in multi-core processors due to using different order of floating-point operations (e.g., division and square root operations) in ways that these operations lead to different rounding and truncation.

In addition, parallel programming may lead to race conditions that further impede reproducibility. A race condition is a situation in concurrent programming where two concurrent threads or processes have access to the same resources and attempt to change it at the same time. When one thread is performing read on a particular data element, another thread is allowed to modify or delete this element. So, the resulting final state depends on the order of process operations, which is not specified by the application. In addition to race condition, some other problems have been listed as the main sources of numerical differences in many parallelized experiments such as out-of-order execution, and message buffering non-blocking communication operations [107].

Message buffering is a type of communication using send/receive functions in parallel programming, which can be blocking and non-blocking. In contrast to blocking, non-blocking

communication do not block the process if the communication is not finished yet. Non-blocking means that computing and transferring data can happen at the same time for a single process. This allows communication to overlap, which generally can lead to different computing orders and irreproducible results for different runs.

Furthermore, some experiments are reported in [51] to determine the effect of parallelization on neuroimaging pipelines, most precisely in different versions of FreeSurfer. Regarding these experiments, results demonstrate that concurrent running would not make statistically significant differences based on the comparison of voxel volume of specific brain structures for the same conditions. This is an example where Peng's reproducibility is achieved while numerical reproducibility is not.

2.1.3 Effect of Operating System

In this section, we summarize the results of the work in [46], which quantified the reproducibility of computational analyses across operating systems. In particular, the authors determined the reproducibility of three neuroimaging workflow packages, FSL, FreeSurfer, and CIVET between CentOS 5.10 and Fedora 20.

Using FSL package, cortical and subcortical tissue classifications resulted in minor differences between the classified tissues on CentOS and Fedora operating systems. These differences mainly correspond to the mathematical functions implemented in different operating system libraries.

The results of RS-fMRI analysis revealed significant inter-OS differences in the second experiment. This analysis showed that each pre-processing step could introduce small numerical variations, but their accumulation creates important differences. These numerical differences are caused by changing the implementation of mathematical functions like sinf() between operating systems.

Using FreeSurfer and CIVET packages, cortical thickness extraction introduced important differences in some specific brain regions across the operating systems. Figure 1 shows localized regions of these differences for CIVET, which are quantified by the metrics including mean absolute difference, standard deviation of absolute difference, t-statistic and random field theory (RFT).

Additionally, inter-build differences are measured in this study. A static build of a pipeline refers to its compiled version where libraries are statically linked. In this test, the authors used the static builds of FreeSurfer CentOS 4 and CentOS 6 to measure their reproducibility. Results show that building static program improves reproducibility across OSes, but small

differences still remain. The main cause of such differences is dynamic libraries that are loaded by the static executable at run-time.

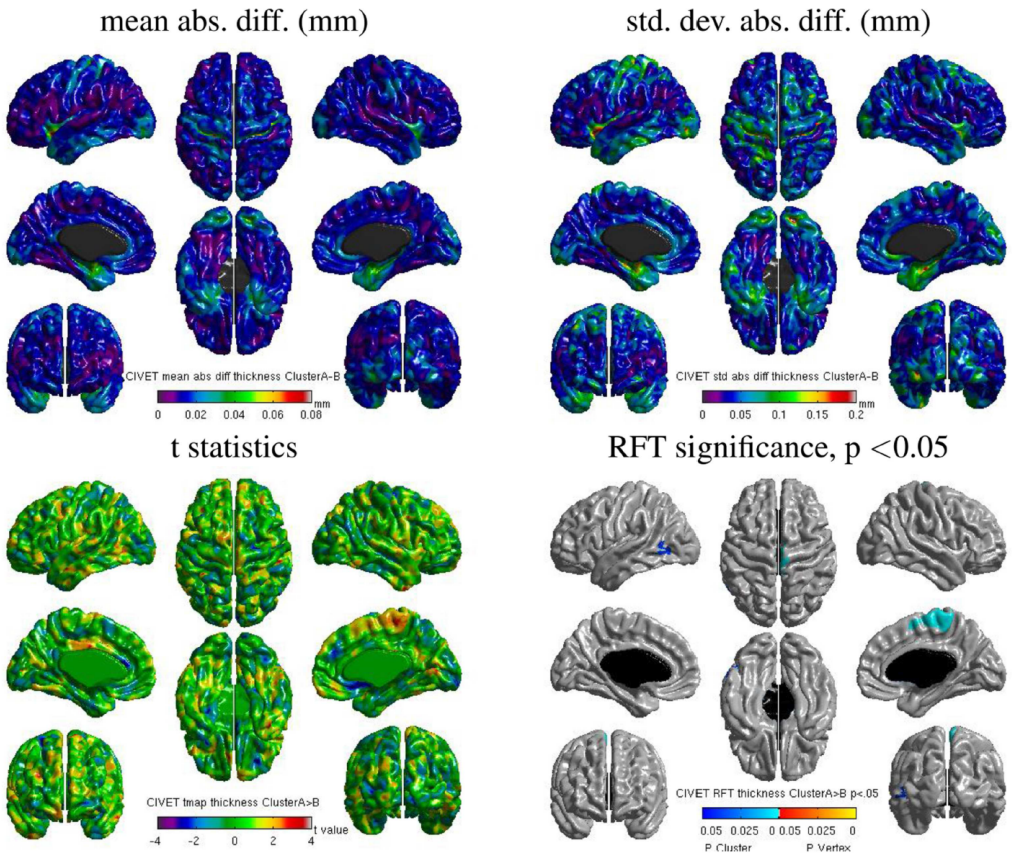


Figure 1: Surface maps of four metrics, standard-deviation and mean absolute differences, t-statistic and RFT significance values, indicate the inter-OS differences for the cortical thickness extracted with CIVET over 146 subjects [46].

In [46], it has been detected that most of the neuroimaging pipelines are sensitive to the operating systems. The effect size of the variations is changed based on the complexity of the analysis pipeline. For instance, shorter analyses like brain extraction have much less significant disagreement compared to the longer ones like subcortical tissue classification and RSfMRI analysis.

Furthermore, the authors expect similar reproducibility issues for the other Linux distributions including Debian and Ubuntu as long as they are based on glibc, the GNU C library, which includes mathematical libraries. In addition, other studies [51, 77] have reported similar issues for non-Linux operating systems.

2.1.4 Effect of Analysis Software

Reproducibility of computations also depends on the executed analysis software, even using the same operating system and hardware resources. Different version of analysis software used in a computation may produce different results. Also, re-implementation of the same experiment through different software packages can introduce discrepancies in results. In this section, we summarize the impact of software variability including different software versions and a wider range of software packages on reproducibility of results.

Effect of software versions

In addition to comparing hardware and operating system variability in [51], the impact of using different pipeline versions is investigated. Significant volume differences are quantified across the FreeSurfer versions for both anatomical brain structures and cortical thickness measures. Thus, it is important for users to be able to reproduce analyses in any future update of these analytic software.

Moreover, the same study [51] showed that the effect sizes of different operating systems or software versions are close to the ones measured in neuropsychiatric diseases. For example, the impact of Alzheimer disease and semantic dementia on Grey Matter volume changes are reported in [80]. These results show similar changes between volumes of specific structures compared to the discrepancies caused by computation environment variability in [51]. In addition, differences in cortical thickness caused by various operating systems, software versions and workstation types were roughly of the same order of magnitude than the findings reported in [78] from patients who suffered from schizophrenia. There are many other proofs in different domains that show the influence of software updates on results [113, 121].

Effect of software packages

In all aforementioned analyses, the choice of the software package remained fixed for carrying out the analyses in each study. To figure out the impact of analysis software variations on task fMRI results, several tests were conducted in [14]. They investigated differences produced across three of the most popular neuroimaging software packages, AFNI, FSL, and SPM. They replicated specific analyses, a number of image processing steps, as closely matched to the original study as possible.

The statistical comparisons show a substantial disagreement between software package

results such as producing different location of activation regions. Figure 2 shows the substantial variation between each main activation area found in the original study and the reanalyses. Results indicate that the precise location of the significantly activated regions is highly dependent on the choice of software package and inference method.

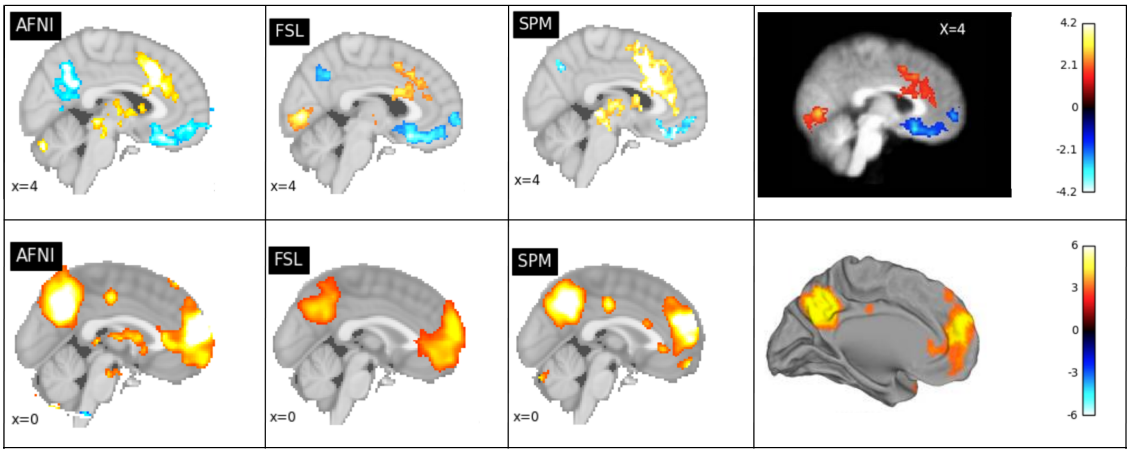


Figure 2: Comparison of the thresholded statistic maps of two different analyses within AFNI, FSL and SPM. Each row shows the results of each reanalyses, and the last column shows the main figure from the original publication. Total 16 subjects and 21 subjects are participated in the first study (first row) and second study (second row) respectively [14].

In addition, from the analyses conducted in [14], it is found that the size of datasets can contribute to the variation of results. For instance, results obtained from analyses that use smaller sample size are less likely to be reproducible than analyses in which more subjects are participated. Therefore, variation in the outcome of an fMRI analysis depends not only on the choice of software package used, but also on the dataset being analyzed.

2.1.5 Effect of Small Data Perturbations

It has been shown in the previous sections that neuroimaging pipelines are sensitive to changes in the computing environment. More precisely, a few studies were conducted to show the instability of some specific steps of MRI analysis through the simulation of minor perturbations in input data. For instance, reproducibility of the cortical surface reconstruction analysis in the presence of small perturbations is measured in [82]. They investigated results of two pipelines, CIVET and FreeSurfer, after applying 1% intensity modification on one voxel located in a non-cortical region. Contrary to expectations, widespread surface changes were observed across the cortex.

Similarly, another study [43] observed substantial variability of motion correction algorithms in fMRI analyses by applying one-voxel perturbation. Results demonstrate significant differences specifically for Niak and FSL packages in this study. These variations may result in wrong activation maps and increase the prevalence of false activations on the subsequent steps of fMRI processing.

Recently, the processing of high-resolution images has been made possible through a new version of pipelines. Therefore, the study in [83] quantifies the variability of analysis results across different image resolutions. The authors investigated the partial volume effects of various image resolutions on the automated cortical surface extraction through CIVET and FreeSurfer pipelines. This study shows significant variability in results for the same analysis using images with different resolutions. For both pipelines, mean absolute error, signed error, and standard deviation are mostly reduced as a function of increasing resolution. Also, comparison of projected distance error maps between histological ground truth surfaces and MRI-derived surfaces confirms that the accuracy of analysis is increased in higher resolutions. Further research is needed to minimize partial volume effects along with magnifying the resolution to get more accurate results.

2.2 Techniques to Improve Reproducibility

Reproducibility is mainly ensured through three properties, including source sharing for both code and data, research portability, and pipeline stability. Source sharing and research portability can be related to the FAIR principles [125] according to which scientific sources have to be findable, accessible, interoperable, and re-usable.

The first step in reproducibility is finding and accessing research products related to Findable and Accessible principles in FAIR. Code and data must be publicly available in a machine-readable structure. Therefore, reproducibility increases the reliability and transparency of experiments. It enables the verification of scientific results by independent investigators.

Through reproducibility, analysis pipelines need to be able to integrate with other execution environments. This is termed research portability and can be achieved using virtual machines and containerization technologies. Portability enables researchers to re-run analyses in a variety of execution conditions. This can be matched with the Interoperability and Re-usability of FAIR principles.

In addition, analysis pipelines must be numerically stable across a variation of the computing environments to be reproducible. Although many solutions currently exist to address analysis sharing and portability, the effect of numerical instability remains largely unexplained. In this section, we will discuss a number of techniques and tools used to enhance sharing, portability, and stability of the analysis.

2.2.1 Code and Data Sharing

To successfully reproduce a computational experiment, analysis sources must be accessible in a machine-readable structure [114, 54]. The importance of a proper structure is clear, specifically when we aim to share codes with others or contribute to a wider group.

One foundation of code sharing is modern software engineering, which includes practices like version control systems (VCS). Version control ensures that the history of the code is available and archived. Git [85], as one of the most popular VCS frameworks, provides a distributed platform to manage project files. Git facilitates the collaboration of developers on the same project using GitHub. GitHub is a web-based service for Git, which hosts repositories.

Sometimes developers tend to share programs instead of source code because of commercial reasons, simplifying its usage, reproducibility, etc. For this purpose, several sharing tools exist that maintain a set of packages. As an example of more generic sharing tools, PyPI (Python Package Index) is a software repository for the Python programming language. PyPI helps to share python packages and allows users to search for packages by keywords. Furthermore, there are more specific sharing tools for neuroimaging programs including Boutiques [45], a system to publish and integrate command-line applications automatically using a JSON descriptor across computational platforms, or NITRC-CE [70] which provides a number of pre-installed neuroimaging tools such as AFNI, FSL, and FreeSurfer into a standardized computational environment.

Furthermore, data sharing is important as it facilitates reproducibility of analyses, and it enables the assessment of future works in comparison with previous analyses. However, there are some challenges associated with data sharing including concerns about the privacy of personal information, lack of incentive by other researchers, and technical issues associated with sharing of large datasets.

Git is a very efficient tool for managing textual information such as code, text, and configuration, but it is inefficient for storing large data. Therefore, extensions of Git named git-annex [66] and Git-LFS (Large File Storage) [6] were developed to address the problem

of sharing and versioning large data collections. git-annex uses Git to store and index files without committing large files into the Git repository. Similarly, Git-LFS reduces the impact of file size in the repository by replacing large files with lightweight pointer files, which refer to the actual file location.

Both Git and git-annex are great for collaboration on a single repository, but sharing code and data between multiple projects can be an issue across these tools. Also, they lack advanced meta-data search capabilities. For instance, they cannot crawl throughout domain-specific repositories. For this purpose, Datalad [61] is an efficient tool particularly for data sharing and versioning across multiple datasets. This tool is built on top of the git-annex and provides unified access to data regardless of its origin. There is a guarantee that the content for the same version would be the same across all clones of a dataset, regardless of where the content was obtained. DataLad supports multiple redundant data providers for each file in a dataset and will transparently attempt to obtain data from an alternative location if a particular data provider is not available. Furthermore, a provenance record is provided by Datalad with all necessary information about input data to reproduce the analysis results.

In addition, higher-level platforms were designed to help scientists share neuroimaging data and make them public on web such as openNeuro [48], LORIS [25], and XNAT [87]. These tools usually have a web-based user-friendly interface, and can integrate with processing platforms. Also, Most of these tools use the Brain Imaging Data Structure (BIDS) [50] to describe and organize neuroimaging data. BIDS specification provides a standard for organizing and representing MRI data that reduces the effort of data sharing.

Moreover, there exists a number of projects that promote open data-sharing initiatives in the field of neuroimaging including the International Neuroimaging Data-Sharing Initiative (INDI) [90, 92], the Alzheimer's Disease Neuroimaging Initiative (ADNI) [59], and Human Connectome Project (HCP) [119]. Researchers who once struggled to access the restricted datasets can now explore thousands of published subjects using data published by these projects. Public access to this amount of brain imaging data has become invaluable for specialists to test a variety of scientific hypotheses and evaluate novel image processing algorithms.

2.2.2 Portability

Given that source code and data used in the original experiment are available, re-executing a computational analysis is still not straightforward. To reproduce computational analyses, information about the computing environment is needed, in particular the operating system

configuration, the hardware system architecture, and specific versions of tools. Therefore, virtual machines and containerization techniques are suggested to ensure that specific computing parameters are completely preserved.

Virtual machines (VMs) can be used to encapsulate the entire context of computations, which provides an exact replica of the computational environment where analyses took place. The most popular implementations of VMs include VMware [124], VirtualBox [122], and KVM [76]. VMs may produce large images because they hold a copy of all the operating system files including the kernel, system libraries, and system configuration files. In addition, VMs bring an extra performance overhead such as I/O, CPU, and memory.

In contrast, containerization tools like Docker [12] and Singularity [79] dramatically reduce the performance overhead and image size of VMs by sharing the kernel of the host system across the containers. These tools have emerged to build lightweight and portable images. Container images can be version controlled in the same way as the analysis code so that the exact same computing environment can be used to re-execute an analysis. However, similar to VMs, users are faced with the burden of ensuring that all necessary dependencies are collected inside the containers. For this purpose, some workflow management systems are developed to record the computational dependencies: we will describe them in the next section.

As an example of a container-based tool, Nextflow [28] is implemented to ensure workflow reproducibility. Nextflow uses Docker to containerize pipeline dependencies including data, code, and the computing environment. Nextflow can be integrated with public repositories in GitHub and cloud computing infrastructures to provide a rapid computation and effective scaling.

It should be noted that containers are excellent technologies to solve portability issues. However, they are not perfect solutions to address the reproducibility of analysis across computing environments because they mostly mask differences instead of fixing them, as explained hereafter.

2.2.3 Numerical Instability

Containers and VMs are good to mask the effect of hardware, parallelization, and operating system. However, this effect is likely due to numerical instabilities in the data analysis pipelines. We believe that these effects are the combined results of 1) the creation of numerical errors between conditions and 2) the amplification of these numerical errors throughout the pipelines.

Creation of Numerical Errors

The main causes of numerically irreproducibility are the limitations of floating-point operations, in particular, using a finite precision in their arithmetic operations like summation [56, 116]. In this section, several solutions are proposed to improve numerical reproducibility related to the floating-point operation, but they are not satisfactory solutions to fix numerical instability.

Floating-point numbers are composed of a mantissa (significand) as the significant digits of the number, a base and an exponent that specifies a finite precision representation, and a sign for both negative and positive values. Floating-point data type represents an approximation of a real number on computers, depending on the size of the mantissa [56]. Each computing system provides standardized math libraries necessary for the floating-point computations with a finite precision. However, finite precision computations create numerical errors mostly due to truncation error and round-off error.

Computers can represent floating-point numbers with limited precision; therefore, they have to round numerical results to the closest number that they can represent, this is leading to rounding error [37]. Truncation error is the difference between actual (analytical) value and truncated (approximated) value of computation with an infinite number of process such as Taylor series and exponential functions [75]. When truncating a number to a limited number of decimal places, say x, the first x digits of the mantissa are reserved, and simply chopping off the remainder. When rounding a number, the computer chooses the closest number that is representable by the computer. This is called round-off error and is caused by the approximate representation of numbers. Although rounding error is in the order of magnitude of $e > 10^{-7}$ for IEEE-754 single-precision and $e > 10^{-16}$ for IEEE-754 double-precision, their accumulation can be significant.

Notably, due to the non-associativity of floating-point addition, rounding errors can lead to different results depending on the order in which operations are performed. For instance, assuming a computer with 4 decimal digits of precision, the following summation in different orders leads to different results.

$$(4.127 \oplus 100.2) \oplus -104.2 = 104.3 \oplus -104.2 = 0.100$$

$$4.127 \oplus (100.2 \oplus -104.2) = 4.127 \oplus -4.000 = 0.127$$

In the first summation, rounding error would be introduced in the truncation of 104.327 to 104.3. This shows rounding errors leading to different results, depending on the finite

precision of floating-point numbers used by the system.

In the following paragraphs, we explain several approaches discussed in [95] to improve these numerical errors, such as using higher precision, deterministic order of operations, arbitrary-precision operations, and fixed-point arithmetic.

Higher precision. Using high-precision numbers, for instance, calculations using double-precision produce more accurate results than single-precision. However, it is not a satisfactory solution for numerical instability because tiny rounding errors still exist for higher precision, which can produce significant bits flip.

Deterministic order of operations. It is possible to make computations deterministic in terms of the order in which floating-point operations are performed. This can lead to more numerically stable results across runs, but this solution may add memory overhead and affect the performance of executions.

Fixed-point arithmetic. Fixed-point numbers reserve a fixed number of digits after the decimal point, assuming that numbers are integer multiple of some common denominator and come from similar orders of magnitude. It is common to use fixed-point arithmetic to represent large fractional numbers. Fixed-point operations are often faster than floating-point ones since they don't depend on the availability of an FPU. Although it helps reduce rounding errors, it limits the range of values, and overflows can occur if the result of an operation is larger or smaller than the numbers in that range.

Arbitrary-precision operations. We can use high-precision or even arbitrary-precision operations to push the precision limitation of floating-point arithmetic. This means that the precision of numbers is limited only by the available memory of the host system. This requires many hardware instructions for each arithmetic operation and the difficulty of handling the variable-width storage.

Amplification of Numerical Errors

There is evidence showing that analyses are not stable to small numerical errors because of the propagation and amplification of these errors. For instance, propagation of rounding errors from the initial value in numerical computations were studied by performing different experiments in [37]. In this paper, several computational experiments are presented to demonstrate the rapid growth of rounding errors in iterative computations like iterative

addition. The accumulation of rounding error from summation operation indicates that analyses may produce different results. Due to similar reasons, the propagation of rounding error when simulating the metal sheet thickness changes in [31] turned to different results.

Another study [43] evaluates the stability of different neuroimaging pipelines in presence of one voxel perturbations. This study showed that iterative initialization schemes in motion correction algorithms lead to the propagation and amplification of numerical errors along the time series.

To address the numerical instability of pipelines, we can use bootstrap technique. In [43], the authors explained that bootstrapping is an efficient technique to improve the robustness of motion estimation. The bootstrap version of the pipelines computed the median transformation results from the 30 samples from the medians of the parameters of the 30 transformations. It is, however, a compute-intensive technique that should be used only when no other solution to the instability is available.

In addition to bootstrapping, it is shown that bagging technique can reduce the effect of perturbations [16, 17]. Bagging, also called bootstrap aggregating, is a simple and powerful ensemble method. It helps reduce both bias and variance in the results. So, we can possibly stabilize pipelines and improve their accuracy using aggregates of results obtained with data perturbations.

2.3 Provenance Capture

We discussed about portability of analyses as a necessary feature for reproducibility, which enables researchers to re-run analyses in a variety of execution conditions. Portability requires comprehensive information about the computational analysis in a machine-executable form. This information can be achieved by provenance capturing tools.

Provenance is defined as the collected information about objects and processes involved in workflow results. This information can be used to verify the reliability and reproducibility of executions [93]. Provenance information can contain the metadata that displays what data processing is undergone [97]. For example, which parameters are used for the analysis, what form of image is used, how the image was registered/aligned to a standard space, how noise was eliminated, how a specific feature has been recognized. Capturing such information is out of the scope of my research. Instead, we are looking for detail of the computing environments provided by the provenance capturing tools. In this section, we will discuss different aspects of provenance capturing such as system-level provenance capturing and

workflow specifications. Finally, we give examples of some specific workflow engines that provide these features.

2.3.1 System-Level Provenance Management Tools

Automated provenance capturing of computational analyses that contain a complicated sequence of dependencies is a challenging issue. There are packaging tools that automate the configuration capturing of an experiment by tracing the executed process using system call interceptions, such as ReproZip [22], CDE (Code, Data, and Experiment) [52], and CARE (Comprehensive Archiver for Reproducible Execution) [60]. These tools support reproducibility of research projects in a system-level provenance capturing.

ReproZip provides a lightweight solution that simplifies the process of making experiments reproducible without forethought. ReproZip creates a self-contained package for experiments by tracking processes and identifying all system dependencies automatically.

ReproZip packs all the necessary information of the experiment in a single package including input/output data, executable programs and steps, and computing environments. Using this provenance receipt, readers/reviewers can then extract the packages and reproduce analysis. In addition, ReproZip generates a workflow specification for experiments that models the processes involved in the workflow. Using this, users can easily explore the reproducibility of experiments or test other configurations.

The ReproZip tool also suffers from limitations as it cannot deal with packing the experiments in different operating systems except a Linux-based OS. Also, packages may not be re-executed if they use absolute path hard-coded in the underlying experiment because it is incompatible with the target environment.

In addition, ReproZip is unable to capture values processed in-memory and not written to disk, and temporary files that are removed during the execution. Therefore, full replication of the experiments may be impossible using ReproZip since these files and variables are not available in the provenance template. Furthermore, ReproZip cannot identify the execution order of files that are written by multiple processes concurrently. So, there is no guarantee to reproduce analysis in this condition as well.

Similar to ReproZip, CARE is a packaging tool that enables users to reproduce Linux-based experiments by making a compressed archive of all the software dependencies. CARE is a portable tool that makes it easy to run because it doesn't need any installation process, neither administrative privileges [60]. With the same purpose, CDE tool relies on system call interception to capture and make an independent package of computing environments [52].

In contrast to CDE, which is able to capture dependencies of simple analyses, CARE is more practical for complex analyses because of tracking the history of processes.

2.3.2 Provenance Formats

All aforementioned provenance capturing tools have a common feature on tracking, bundling, and sharing all the necessary dependencies of a project automatically and systematically. Besides, representation of this data is necessary to have an understandable structure for everyone with different backgrounds. Therefore, a few works have been conducted recently to introduce an integrated and standard provenance specification.

It is important to define a standard data model for representing and exchanging provenance information produced by the workflow engines on the web. Therefore, the World Wide Web Consortium (W3C) designed the PROV data model based on the history of three captured elements including entities, activities, and agents. The PROV model contains a set of documents to define various aspects of provenance information in heterogeneous environments such as web. For instance, PROV can make a relational model of such provenance elements as an XML format. Also, the PROV model is not tailored to any specific application domains [21, 93].

Using PROV model, we can check the reproducibility of the scientific workflows by comparing results of the same workflow on different conditions. Also, this specification can provide information about the processes that lead to execution failures [93]. Similarly, a number of projects specific to the neuroimaging field were proposed to support the reproducibility of research studies. Among them, we will discuss two popular neuroimaging provenance specifications in this section: NIDM-Results and BIDS-Derivatives.

NIDM-Results, as a part of the Neuroimaging Data Model (NIDM) project [1], is a domain-specific extension of PROV based on semantic web technologies. NIDM-Results provides a machine-readable representation of neuroimaging results. This specification aims to encode the provenance results of some specific neuroimaging software such as SPM and FSL. It is also suitable for different neuroimaging modalities including functional MRI, structural MRI, and diffusion MRI.

NIDM-Results uses the same elements introduced in PROV (entities, activities, and agents) to provide an interpretable data provenance across the heterogeneous neuroimaging workflow results. There is a scenario to show the relation between these elements [88]: when a voxel-wise inference (as an activity) is associated with SPM (as an agent) to generate a NIfTI image (as an entity) using the segmentation (as an activity).

BIDS-Derivatives provides a standard data provenance compatible with BIDS [50] raw data format. BIDS-Derivatives simplified both provenance capturing and representing. For the ease of many scientists usage with limited technical knowledge, the specification is created on a JSON file based on a simple file format and folder structure. In this way, researchers can easily share derived data, statistical models, and computational results automatically. However, in addition to the MRI modality, BIDS needs to support other neuroimaging data types.

2.3.3 Neuroimaging-Specific Workflow Engines

Capturing and documenting provenance information in neuroimaging pipelines is a challenging issue for reproducibility. Therefore, workflow engines were developed to address these issues using the specification models and capturing techniques introduced in the previous sections. These engines can facilitate workflow composition procedure and document them in a machine-readable form, which significantly enhance reproducibility. Some of the existing workflow engines in neuroimaging are explained in this section.

Nipype [49] is a Python package that introduces a framework to 1) make uniform access to neuroimaging analysis software and usage information, which allows mixing components from other packages developed in different programming languages through interfaces provided by Nipype; 2) simplify the design of workflows and facilitate the interaction between workflow modules; 3) reduce the training time of how use the packages. Nipype represents the provenance information using the W3C-Prov specification. Furthermore, VisTrails [18] and Taverna [98] perform similar methodology but not specific to neuroimaging, and they are a bit different in the way of data representation and the type of information they capture.

LONI [108] pipeline is a provenance framework for documenting data flows in computational environments without user intervention for describing such processing provenance [86]. Therefore, the relationship between processes can easily be captured in an XML file format and re-executed later similarly. Also, LONI pipeline is a java-based program that facilitates the process of provenance capturing using a graphical user interface. The XML extension provided by LONI can clearly be interpreted across different environments. Additionally, LONI supports the parallel execution and provides a simple mechanism for researchers, particularly in the neuroimaging field, to disseminate their experiments.

ReproNim [69] is an integration of tools to ensure reproducibility of the neuroimaging literature in different stages of the analysis including data acquisition, annotation, processing, publication. ReproNim helps researchers to comprehensively describe data and analysis

workflows in precisely machine-readable form (with ReproIn and Brainverse), manage the computational environments (with NICEMAN), find and share data in a FAIR fashion (with NeuroBlast). This framework facilitates the implementation of analysis in a reproducible fashion.

Chapter 3

File-based localization of numerical perturbations in data analysis pipelines

Ali Salari[1], Gregory Kiar[23], Lindsay Lewis[4], Alan C. Evans[42], Tristan Glatard[1]

Published in:

GigaScience journal

https://doi.org/10.1093/gigascience/giaa106

[1]Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada
[2]McGill University, Montreal, Canada
[3]Montreal Neurological Institute, Montreal, Canada

Abstract

Data analysis pipelines are known to be impacted by computational conditions, presumably due to the creation and propagation of numerical errors. While this process could play a major role in the current reproducibility crisis, the precise causes of such instabilities and the path along which they propagate in pipelines are unclear. We present Spot, a tool to identify which processes in a pipeline create numerical differences when executed in different computational conditions. Spot leverages system-call interception through ReproZip to reconstruct and compare provenance graphs without pipeline instrumentation. By applying Spot to the structural pre-processing pipelines of the Human Connectome Project, we found that linear and non-linear registration are the cause of most numerical instabilities in these pipelines, which confirms previous findings.

3.1 Introduction

Numerical perturbations resulting from variations in computational environments impact data analyses in various fields, but identifying the origin of these perturbations in complex pipelines remains challenging. In some cases, small perturbations resulting from changes in operating system versions [46], hardware [65], or parallelization parameters [30], result in substantially different analysis outcomes, due to the propagation and amplification of floating-point errors. While the existence of such numerical errors is well known [115], their impact on scientific computations has multiplied with the rise of the Big Data era, due to the sustained growth of data sets, the increasing complexity of analysis pipelines, and the diversification of computing infrastructures. To better understand and correct these effects, efficient tools are needed to assist pipeline developers in the comparison of results obtained across different conditions.

In neuroimaging, our primary application field, data analyses often consist of hundreds of computational processes – often coming from multiple toolboxes – that are aggregated to perform a specific function. For instance, the fMRIprep pipeline [35] assembles software blocks from FSL [63], AFNI [24], FreeSurfer [39] and ANTs [7] to provide a state-of-the art functional MRI processing tool with minimal user input. Another example are the pipelines of the Human Connectome Project [42] that combine tools from FSL and FreeSurfer to pre-process structural, functional and diffusion data from their uniquely high-fidelity open dataset. In both cases, pipelines leverage toolboxes that are widely trusted in the community, yet, at the same time substantial variations in results have been observed in these toolboxes resulting from minor data or infrastructure perturbations [51, 46, 82, 69], suggesting that further investigation of their numerical conditioning is required. For such complex pipelines, a lightweight solution has to be found to perform such evaluations with limited code instrumentation.

Numerical evaluations are traditionally performed using techniques such as interval arithmetics [55] that require complete code re-writes and are therefore barely applicable to complex pipelines. Recently, Monte-Carlo Arithmetic [100, 27] provided a practical way to evaluate the uncertainty of numerical results without the need to rewrite the application in a different paradigm. By perturbating floating-point computations, it introduces a controllable amount of noise in the pipelines, effectively sampling results from a random distribution. While this technique is very appealing, it suffers from two main issues that make it impractical at the scale of a complete pipeline. First, it requires that all software components be recompiled for MCA instrumentation, which is not always feasible. Second, it multiplies the

execution time by a factor of 10 to 100, which is impractical when executions already take a few hours to complete.

We present Spot, a tool to identify the source of numerical differences in complex pipelines without instrumentation. Using system-call interception through the ReproZip tool [106], Spot traverses graphs of processes and intermediary files to pinpoint the pipeline components that are unstable across execution conditions. When differences start accumulating, effectively masking any further instability, it restores clean data copies through a set of wrapper scripts. Wrapper scripts are also used to restore temporary data that might have been deleted during the execution, and to disambiguate files that have been written by multiple processes. The remainder of this paper presents the design of Spot, and its application to pre-processing pipelines of the HCP project.

3.2 Tool description

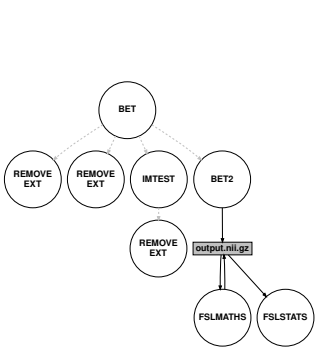
Spot identifies the components in a pipeline, at the resolution level of a system process, that produce different results in different execution conditions. First, a directed bipartite provenance graph is recorded for each pipeline execution, where nodes represent application processes and files, and edges represent read and write file accesses (Figure 3a). Second, transient files, i.e., files that are either deleted during pipeline execution or modified by multiple processes, are identified and disambiguated, resulting in a provenance DAG (Directed Acyclic Graph) in which file nodes have a single parent (in-degree of 1) (Figure 3b). DAGs produced in different conditions are then compared, in a step-by-step execution that prevents the propagation of differences in the pipeline (Figure 3c). The resulting labeled graph identifies the non-reproducible processes in the pipeline.

To ensure that a file can be unambiguously associated with the process that created it, we assume that the pipeline can be transformed such that:

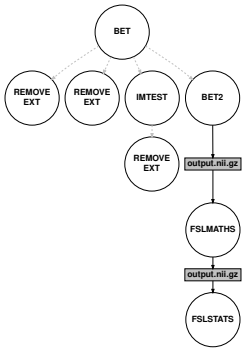
1. Processes don't run concurrently;

2. Each process sequentially reads, computes, and writes.

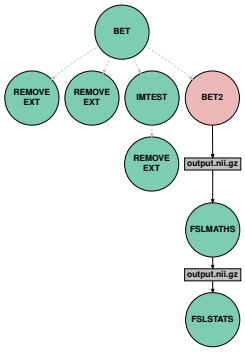
In practice, pipeline processes may still run concurrently provided that they don't write concurrently to the same files. A process may also interleave file writes with computing, for instance when different file blocks are processed sequentially. However, only a single version of the file must eventually be made available to the other processes. In particular, in case a process deletes a file that it had created itself, this file must not be used by any other



(a) Raw provenance graph (ReproZip output), with transient files shown in gray boxes.



(b) Provenance DAG, with disambiguated transient files.



(c) Labeled DAG comparing 2 execution conditions, showing 1 non-reproducible process.

Figure 3: Provenance graphs created from the example pipeline in Listing 1. Processes are represented with circles, files with rectangles, and read/write accesses with plain edges. For convenience, the process tree is also shown, with gray dashed edges. Processes forked by `bet` were captured by ReproZip while they did not appear in Listing 1. Processed associated with executables located in `/usr/bin/` or `/bin/` are not shown.

process. Finally, we also require that processes are associated to a command line (executable and arguments), to facilitate process instrumentation.

3.2.1 Recording provenance graphs

We use ReproZip [106] to capture: (1) the set of processes created by the pipeline, and (2) the set of files read and written by each process, including temporary files. ReproZip collects this information through the `ptrace()` system call, with no required instrumentation of the pipeline. Using the ReproZip trace, Spot reconstructs a provenance graph by creating process and file nodes and by adding directed edges corresponding to file reads and writes (Figure 3a). We assume that provenance graphs are identical for the ReproZip traces obtained from the same subjects in different operating systems.

Provenance graphs are often data-dependent, due to variations in input data that may trigger differing branching or looping patterns across executions, for example. Some of these differences can be neglected: for instance, when a data decompression step is present at the beginning of the execution for some subjects only. Other differences cannot: for instance, when entirely different processing paths are used for different datasets. Spot includes helpers to identify different instances of provenance graphs, such as supporting the clustering of process trees, where nodes are processes and edges are `fork()` or `clone()` system calls,

Listing 1 Example pipeline that computes the volume of the brain from a T1 image.

```
#!/usr/bin/env bash
if [ $# != 1 ]
then
    echo "usage: $0 <input_image.nii.gz>"
    exit 1
fi
# Parse argument, set output file names
input_image=$1
# Run FSL bet, put result in £{bet_output}
bet ${input_image} output.nii.gz
# Create binary mask
fslmaths output.nii.gz -bin output.nii.gz
echo "Voxels / volume in binarized brain mask:"
fslstats output.nii.gz -V > voxels.txt
# Remove temporary file
\rm output.nii.gz
```

using the tree edit distance [128] implemented in Python's `zss` package.

3.2.2 Capturing transient files

We capture temporary files by replacing every process P by a wrapper that first calls P and then saves the produced temporary files to a read-only directory. This process replacement is done by pre-pending to the `PATH` environment variable a directory that contains a wrapper script named after the executable called by P.

Files written by multiple processes are disambiguated using a similar technique. For a file F written by the processes in $\mathbf{P} = \{P_1, \ldots, P_n\}$, we first check that processes in $\mathbf{P}$ do not write concurrently to F, which would violate our assumptions. Then, we replace every process P_i by a `PATH`-based wrapper that first calls P_i and then saves F to a read-only directory. In this way, successive versions of F are preserved for comparison. We finally update the provenance graph accordingly, so that all files in the graph have an in-degree of 1 (Figure 3b). This operation also makes the provenance graph acyclic, since we assumed that a process could only release a single version of a file.

3.2.3 Labeling processes

After capturing transient files in the first condition (i.e. operating system, library version, etc.), we re-run the pipeline step by step in the second one to label processes. The output files created by a process in both conditions are compared: if no differences are found, the process is marked as reproducible; otherwise, the process is marked as non-reproducible, and the output files produced in the first condition are copied to the second one, to ensure that differences do not propagate further in the pipeline. Processes are instrumented transparently through a modification of the `PATH` variable similar to the one described previously. By default, differences in output files are identified by comparing file checksums. Other comparison functions can also be defined for specific file types, for instance to ignore file headers or file sections containing timestamps. Spot finally creates a labeled provenance graph highlighting non-reproducible processes.

Figure 3c illustrates a hypothetical incremental labeling of the example in Listing 1. Process `bet2` is labeled as non-reproducible (red) as it produces files with differences. To prevent the propagation of these differences, the files produced by `bet2` in Condition 2 are replaced with the files produced by `bet2` in Condition 1. Processes `fslmaths` and `fslstats` are then executed and labeled as reproducible (green) as they produce files without differences.

The labeled graph can differ depending on the order of executions in which condition we capture transient files or execute the pipeline to pinpoint the propagation of differences. Therefore, we run the comparison in both condition orders, and we label a process as non-reproducible (red) if it creates different results in at least one condition order.

3.2.4 Implementation

Spot is implemented in Python (¿=3.6). In this work we used Spot version 0.2 and the following version of the Python package dependencies: NumPy v1.19.0 [99] and Pandas v1.0.5 [89], for data manipulations, SciPy v1.5.1 [120] and Scikit-learn v.0.23.1 [101] for the clustering of provenance graphs, Zss v1.2.0 [128] for tree distances, ReproZip v1.0.11 for the capture of provenance traces, Docker v17.05 [91] for the edition of container images, and Boutiques v0.5.25 [44] for uniform pipeline executions.

Software users will mostly have to interact with the Boutiques and ReproZip packages. Boutiques is a flexible description framework for containerized pipelines, required by the pipelines analyzed in Spot. It provides a JSON schema to describe inputs, outputs and their dependencies. Examples, tutorials and usage documentation are available at http:

//boutiques.github.io. ReproZip intercepts system calls to identify the files and processes involved in a pipeline execution. Before using Spot, users have to collect ReproZip traces of their pipeline executions. Examples in the Spot documentation include ReproZip provenance capture. More documentation on ReproZip is available at https://www.reprozip.org.

3.3 Experiments

We applied Spot to the minimal pre-processing pipelines released by the Human Connectome Project (HCP), a leading initiative in neuroimaging.

3.3.1 HCP pipelines and dataset

The HCP developed a set of pre-processing pipelines to process structural, functional, and diffusion MRI data acquired in the project. We focus on HCP pre-processing pipelines for structural data, and particularly on PreFreeSurfer and FreeSurfer. A detailed description of the analyses done by these pipelines is available in [42]. In summary, the PreFreeSurfer pipeline consists of the following steps:

- Gradient Distortion Correction (DC),
- Alignment and Anatomical Average (AAve), T1w(s), T2w(s),
- Anterior/Posterior Commissure Alignment (ACPC-A),
- Brain Extraction (BExt),
- Bias Field Correction (BFC),
- Atlas-Registration (AR).

And the FreeSurfer pipeline consists of the following:

- Image downsampling,
- T1w image registration,
- T1w image segmentation,
- Surface placement,
- Surface registration.

We randomly selected 20 unprocessed subjects from the HCP data release S500 available in the ConnectomDB repository as a subset of the 1200 Subject Release (see Supplementary Table S1). For each subject, available data consisted of 1 or 2 T1-weighted images and 1 or 2 T2-weighted images, with $256 \times 320 \times 320$ voxels of size $0.7 \times 0.7 \times 0.7$ mm. Acquisition protocols and parameters are detailed in [118].

3.3.2 Data processing

We built Docker images for the HCP pre-processing pipelines v3.19.0 (PreFreeSurfer and FreeSurfer) in CentOS 6.9 (Final) and CentOS 7.4 (Core), available on DockerHub. Container images contain the HCP software dependencies, including FSL (version 5.0.6), FreeSurfer (version 5.3.0-HCP, CentOS4 build), and Connectome Workbench (version 1.0).

We processed the 20 subjects with PreFreeSurfer and FreeSurfer, using the 2 CentOS versions. The PreFreesurfer results obtained in CentOS6 were used as the input of FreeSurfer in both conditions. We also used the ReproZip trace file captured in CentOS6 for labeling the processes in both pipelines. Each subject was processed twice on the same operating system to detect within-OS variability coming from pseudo-random operations. We compared pipeline results using FreeSurfer tools `mri_diff`, `mris_diff`, and `lta_diff`, to ignore execution-specific information such as file path or timestamps. To compare segmentations X and Y, we used the Dice coefficient defined as follows:

$$DICE = \frac{2|X \cap Y|}{|X| + |Y|}$$

The Dice coefficient [29] is a commonly used metric to validate medical image segmentation. Dice values range from 0 to 1, with 1 indicating a perfect overlap between two segmentation results and 0 indicating no overlap. Alternatively, the Jaccard coefficient [58] could be used; there is a direct correspondence between both metrics.

3.4 Results

All experiments were run on a machine with a 3.4GHz, 8-core Intel Core i7 processor, 32GB of RAM, CentOS 7.3.1611, and Linux kernel version 3.10. The processing time, output file size, number of file accesses and number of processes observed in PreFreeSurfer and FreeSurfer are shown in Table 5. The scripts and analyses used to create the figures in this section are available at `https://github.com/big-data-lab-team/HCP-reproducibility-paper`.

Table 2: Execution statistics of the pipelines per subject.

	PreFreeSurfer		FreeSurfer	
	Mean	Standard error	Mean	Standard error
Processing time (mins)	106.67	2.68	650.25	8.88
Output file size (GB)	2.8	0.10	4.15	0.15
Number of file accesses	94,089	2,645	62,729	984
Number of processes	8,731	198	4,031	47

Within-OS differences

We did not observe any within-OS difference in PreFreeSurfer. In FreeSurfer, we identified 2 processes leading to within-OS differences due to the use of pseudo-random numbers: image registration with `mri_segreg`, and cortical surface curvature estimations with `mris_curvature`. Fixing the random seed used in FreeSurfer removed these differences.

Between-OS differences in PreFreeSurfer

We identified four types of subjects with different PreFreeSurfer provenance graphs (Table 3). Differences between subject types came from different numbers of T1 and T2 images in the raw data. We verified that the provenance graphs were identical for all subjects of the same type, for both versions of CentOS.

Figure 4 shows the frequency of non-reproducible pipeline processes in PreFreeSurfer. The processes identified as non-reproducible were observed in linear registration with FSL `flirt` (in ACPC-Alignment, Brain Extraction, Distortion Correction, and Atlas Registration), in non-linear registration with FSL `fnirt` (in Brain Extraction and Atlas Registration), and in image warping with FSL `new_invwarp` (in Brain Extraction and Atlas Registration). Differences were also observed in image mean computations with FSL `maths` (in Anatomical Average). Figure 5 shows a complete PreFreeSurfer labeled DAG, localizing the observed differences in the entire pipeline, for a given subject.

Table 3: Types of provenance graphs in PreFreeSurfer.

Type	Number of Subjects	Number of T1w images	Number of T2w images
1	9	2	2
2	8	1	1
3	1	1	2
4	2	2	1

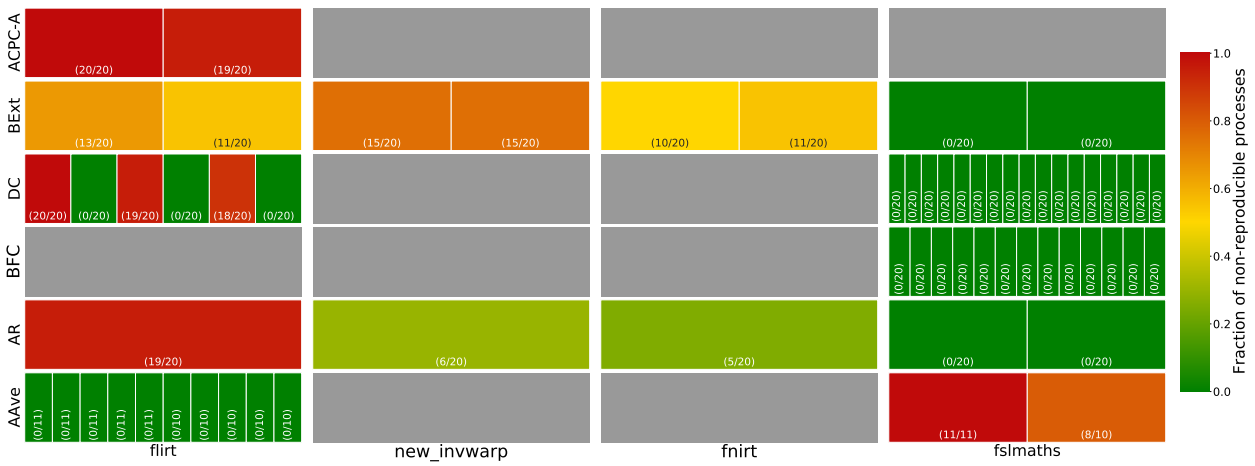


Figure 4: Heatmap of non-reproducible processes across PreFreeSurfer pipeline steps. Each cell represents the occurrence of a particular command line in a pipeline step among Anatomical Average (AAve), Anterior/Posterior Commissure Alignment (ACPC-A), Brain Extraction (BExt), Bias Field Correction (BFC), or Atlas-Registration (AR). Cell labels indicate the fraction of subjects for which the corresponding process wasn't reproducible. For example, the `flirt` tool was invoked 6 times in step DC for each of the 20 subjects: 2 instances weren't reproducible in 19 subjects, 3 instances were always reproducible, and 1 instance wasn't reproducible in 17 subjects. Grey cells indicate that the process did not occur in the corresponding pipeline step.

Figure 6 compares `fnirt` results in Brain Extraction for a particular subject using the checkerboard pattern, a common method to illustrate the magnitude of the differences in registration results. Differences appear to be visually important, in particular in the areas framed in red, to the point that most experimenters would likely reject such a registration following visual quality control.

Between-OS differences in FreeSurfer

The only non-reproducible process identified by Spot in FreeSurfer was `mris_make_surfaces` (cortical and white matter surfaces generation), a dynamically-linked executable that produced different results for 10 out of 20 subjects.

However, FreeSurfer results still differ between conditions, due to the propagation of differences created in PreFreeSurfer. We observed the effect of this propagation in FreeSurfer results, as shown in Figure 7 for whole-brain segmentations. The Dice coefficients associated with the 44 regions segmented by FreeSurfer are shown in Figure 8, showing that Dice coefficients below 0.9 are observed in most regions, and particularly in the smallest ones. However, no significant correlation between the Dice values and the region sizes was found

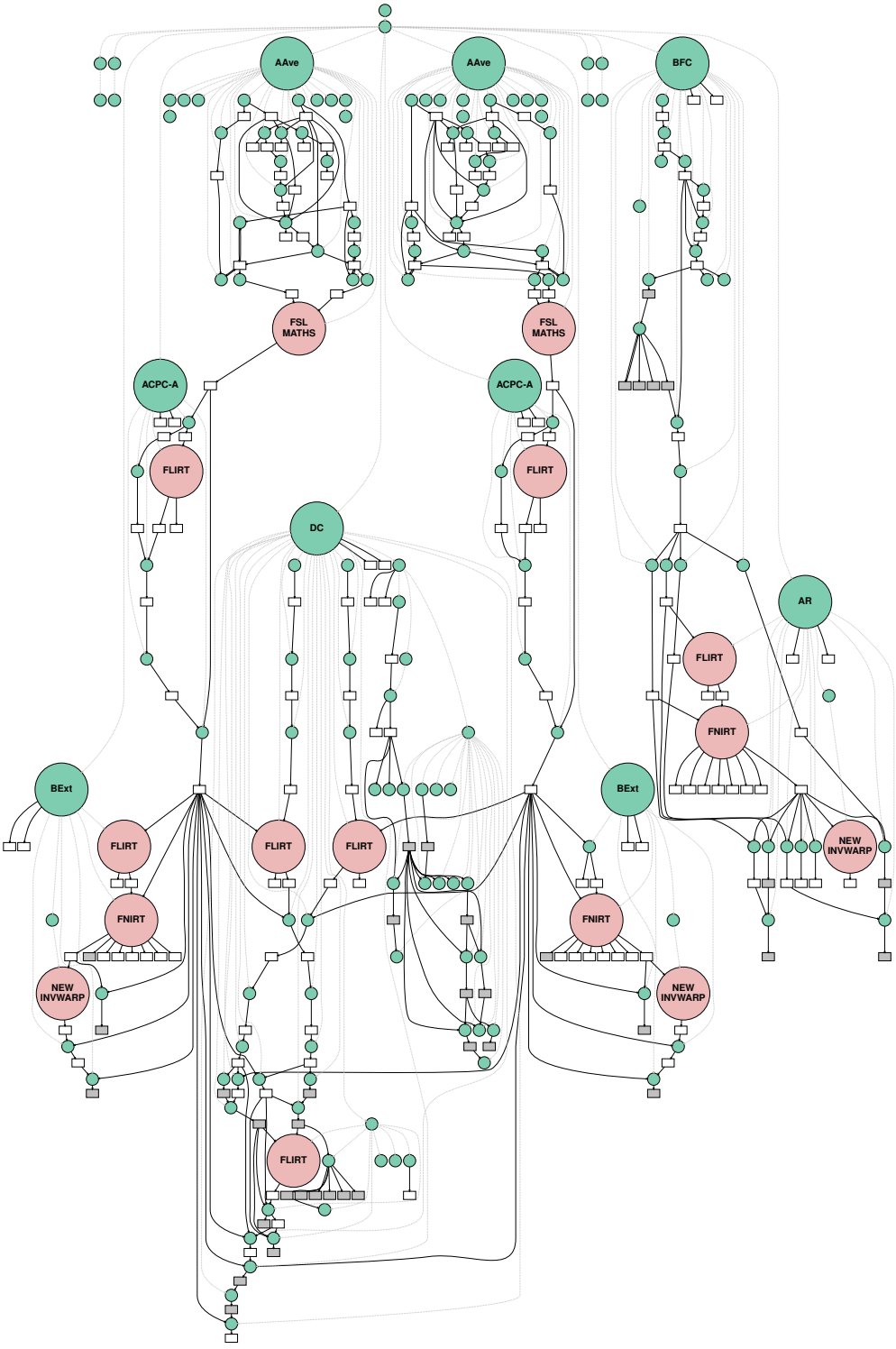


Figure 5: A complete provenance graph from the PreFreesurfer pipeline. Node labels use the same abbreviations as in Figure 4. For better visualization, processes associated with commands in `/bin` or `/usr/bin` were omitted, as well as `imtest`, `imcp`, `remove_ext`, `fslval`, `avscale`, and `fslhd`.

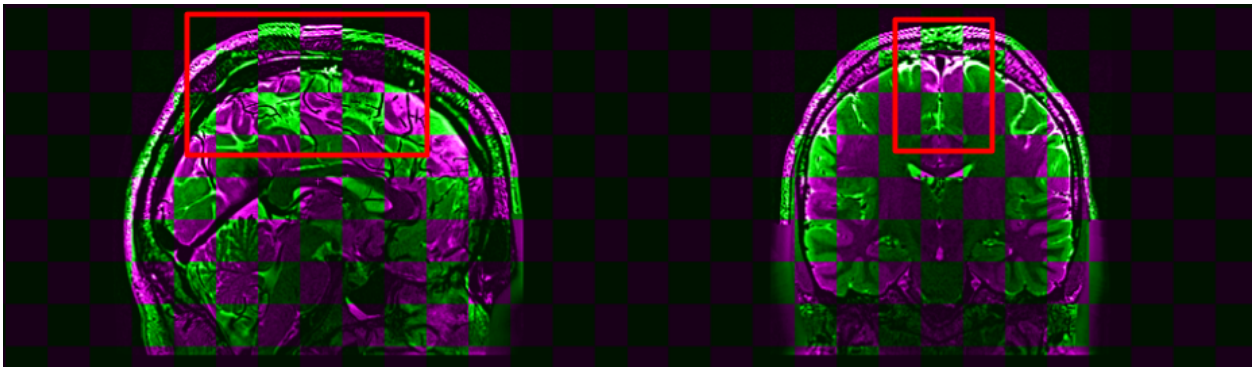


Figure 6: Differences between T2 `fnirt` results in PreFreeSurfer's Brain Extraction (CentOS6 vs CentOS7). The colored squares indicate results obtained with CentOS6 (in purple) and CentOS7 (in green). The red boxes highlight regions with significant differences between the two OSes. An animated version of the comparison is available here for better visualization.

(Pearson's coefficient = 0.12, p-value = 0.43).

3.5 Discussion

Our results provide insights on the reproducibility of neuroimaging pipelines, and on the relevance of the approach implemented in Spot for reproducibility studies.

3.5.1 Key findings

Linear and non-linear registration with FSL were found to frequently lead to differences between results obtained with different operating systems. This does not come as a surprise given the instabilities associated with these processes. It also corroborates our previous findings in [46], where fMRI pre-processing with FSL was found to vary across operating systems starting from the motion correction step, a step that uses FSL's `flirt` tool internally. It would be relevant to investigate if the observed instability of registration processes generalizes to other toolkits, or if it remains specific to FSL. In view of the effect of small data perturbations in a variety of toolboxes and processes, such as cortical surface extraction using FreeSurfer and CIVET [82] or connectome estimation using Dipy [74], it is probable that this observation generalizes widely across toolboxes and requires a deeper investigation of the stability of linear and non-linear registration.

While only a handful or processes were found non-reproducible across the tested operating systems, the effect of such instabilities were found to propagate widely in the pipelines, and

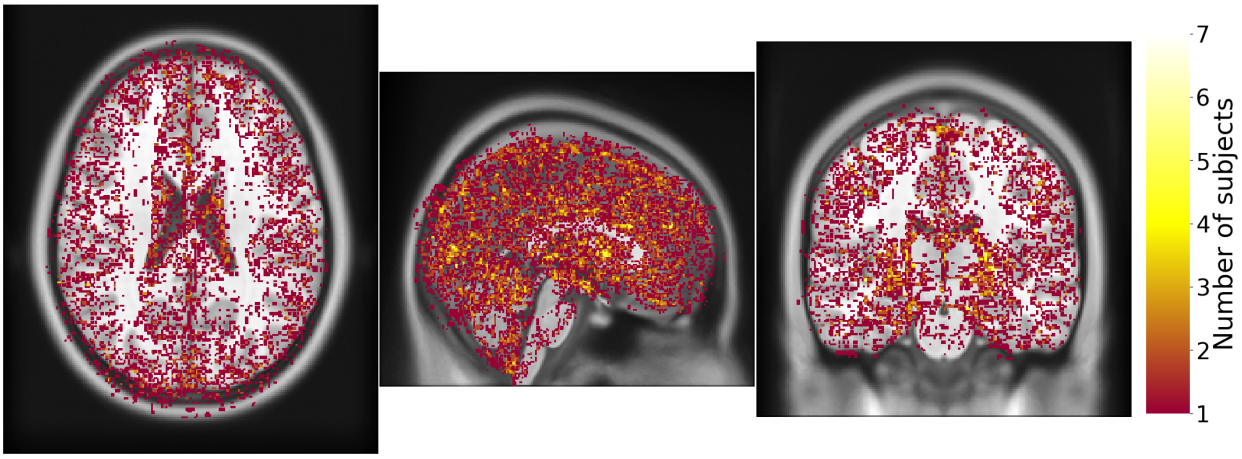


Figure 7: Sum of binarized differences between whole-brain FreeSurfer segmentations obtained from PreFreeSurfer processings in CentOS6 vs CentOS7 (N=20). Segmentations were resampled and overlaid to the MNI152 volume template. Each voxel shows the number of subjects for which different results were observed between CentOS 6 and CentOS 7. An animated comparison of segmentations obtained for a particular subject is available here for better visualization.

to substantially impact the segmentations created by FreeSurfer. This illustrates the need to conduct reproducibility studies on entire pipelines rather than isolated processes. It also highlights the need for a deeper stability analysis of pipeline processes.

As is shown in Figure 4, the reproducibility of a given tool may vary across subjects and across processing parameters. For instance, linear registration with `flirt` seems to be fully reproducible in the Anatomical Average sub-pipeline, while it is highly non-reproducible in ACPC Alignment. In Brain Extraction, the same tool was found reproducible for some subjects only. Therefore, reproducibility studies need to be performed on several subjects. While this is common practice to some extent in neuroimaging, software tests are often executed only on a single dataset to reduce the associated computational load. Our results show that pipeline tests should encompass enough subjects to cover execution paths adequately.

Our results illustrate the type of variability that can be introduced in neuroimaging results due to operating system updates. The numerical noise introduced by operating system updates is realistic, as such updates are likely to occur throughout the time span of a neuroscience study, but it is also uncontrolled, as it originates in updates of low-level libraries by third-party developers. A possible method to study this problem more comprehensively would be to introduce controlled numerical perturbations in pipelines, which could be done by introducing noise either in the data, or in floating-point computations through Monte-Carlo Arithmetic [100]. The work in [74] discusses and compares these two techniques.

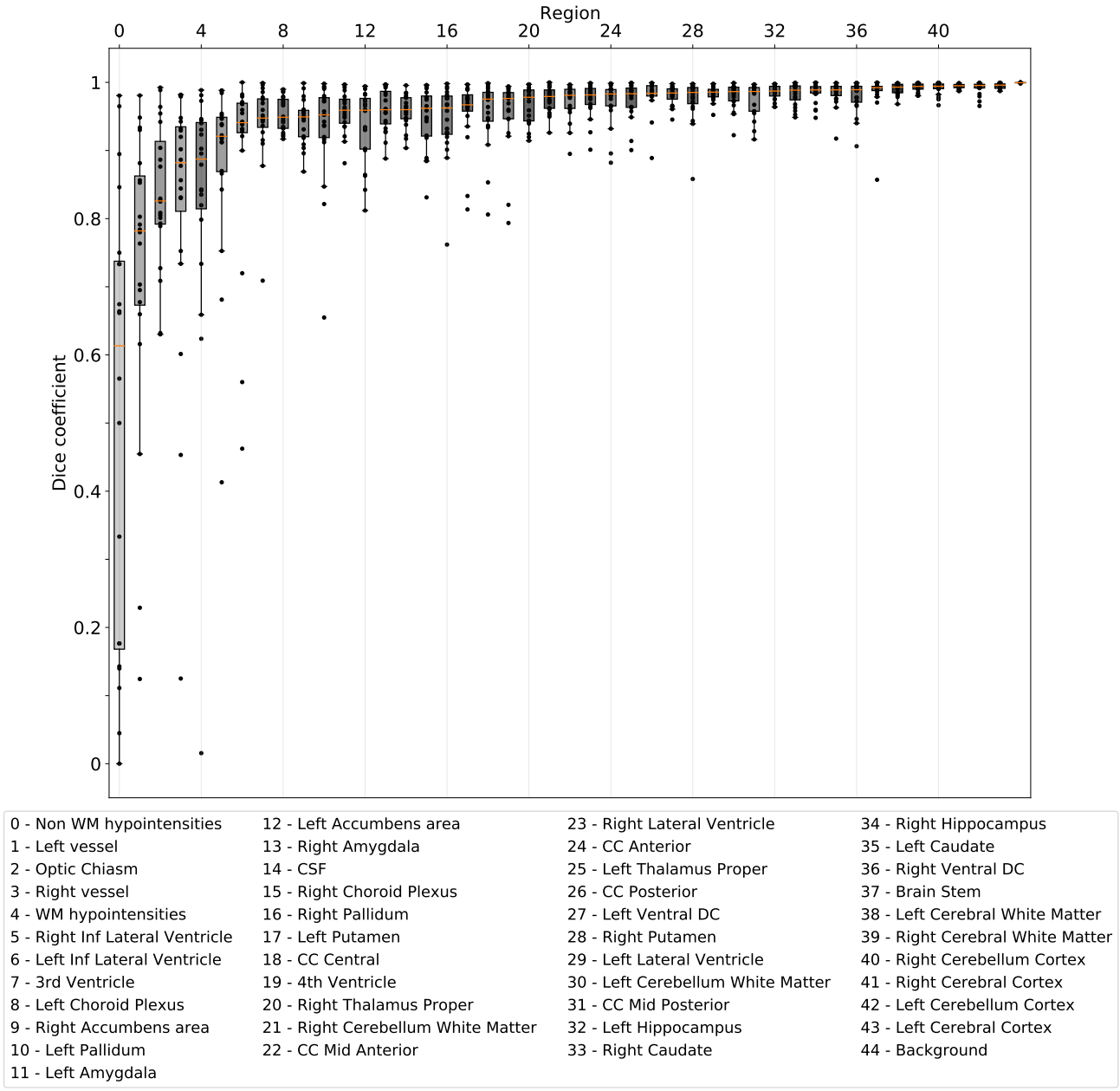


Figure 8: Dice coefficients between regions segmented by FreeSurfer in CentOS6 vs CentOS7 (N=20), ordered by increasing median values. Each point represents the Dice coefficient between segmentations of a particular region obtained in CentOS 6 vs CentOS 7 for a given subject. Boxes brightness is proportional to the logarithm of the corresponding brain region size.

3.5.2 Spot evaluation

The processes identified by Spot as non-reproducible were all associated with dynamically-linked executables. This makes complete sense as statically-linked executables are not impacted by library updates. Moreover, the hypothetical effects of hardware or Linux kernel updates were not measured, as the different operating systems were deployed in Docker containers on the same host, that is, using the same kernel and hardware.

To evaluate the reproducibility of a pipeline, Spot needs to execute it 5 times in order to (1) record a first ReproZip trace, (2) save transient files in the first condition, (3) compare results in the second condition, and repeat steps (2) and (3) for the other order of execution. It might be possible to further reduce this overhead by executing at step (2) only the processes depending on transient files, and capturing the transient files for the second condition simultaneously at step (3).

The target users of the Spot tool are primarily pipeline developers and users who have technical skills for creating Docker containers and Boutiques JSON files. We demonstrated the applicability of our approach by evaluating two of the arguably most complex pipelines in neuroimaging. Technically, these pipelines consist of a mix of tools assembled from different toolboxes through a variety of scripts written in different languages. Our file-based approach, notably enabled by ReproZip, was able to analyze these pipelines without requiring their instrumentation, which saved a very substantial technical effort. The assumptions made on the pipeline structure, related to the absence of concurrent writes, were not violated in our analysis, and are likely to not impede Spot's applicability to the most common neuroimaging pipelines.

Spot only tests pipeline reproducibility in the scope of a particular dataset. However, it is very plausible for pipeline processes to exhibit different reproducibility behaviors when executed on different datasets. Therefore, only the lack of reproducibility of a pipeline process could be guaranteed from an analysis with Spot, since proving reproducibility would require testing the pipeline on all possible datasets, in all possible environments, which is not feasible. Two elements could be considered in future work to address this issue. First, similar to conventional software testing, a code coverage metric could be developed to assess the fraction of the pipeline code involved in the tested dataset and parameters. This would quantify the representativity of the dataset and pipeline parameters used in the evaluation. Second, statistical risk models could be used to estimate the probability for a process to be reproducible, given a set of observations with no numerical differences. For instance, models described in [9] could be leveraged for this purpose.

File-based analyses also have limitations related to the granularity at which they operate. Indeed, differences can only be identified at the level of an entire operating-system process, which can correspond to arbitrary amounts of code. Narrowing down the analysis to particular libraries, functions, or even code sections would require another approach. Similarly, Spot would not be able to detect differences in data not saved in files but instead passed to subsequent processes in memory. A common scenario in neuroimaging pipelines is that tools return results in their standard output, which is parsed by the calling process and passed to subsequent ones through variables.

Computational environments are only one of many factors contributing to the on-going reproducibility crisis. In fact, sample size selection, publication bias, or methodological flexibility in the analysis are likely to have a stronger effect than numerical perturbations, although to our knowledge no evidence of this is available. We refer to the studies in [13, 14, 11, 69] for deeper analyses of the associated effects on neuroimaging analyses. It should also be noted that the effects of computational environments and these other factors manifest at different levels: referring to the terminology used in [102], computational environments are associated with reproducibility, the minimal standard by which identical results should be obtainable from identical data and parameters, while the other aforementioned factors belong to replicability, the ultimate standard by which independent experimenters should be able to draw similar conclusions from similar experiments. In practice, variability resulting from computational environments manifests during software testing (test results depend on execution platform), deployment on HPC systems (results obtained on local vs HPC systems differ), or software version updates (results obtained before vs after the update differ), while factors related to replicability impact the community more broadly. Ultimately, both reproducibility and replicability should be understood and improved.

3.6 Conclusion

We presented Spot, a tool to detect the source of numerical differences in complex pipelines executed in different computational conditions. Spot leverages system-call interception through the ReproZip tool, and therefore can be applied to the most complex pipelines without requiring their instrumentation. It is available at https://github.com/big-data-lab-team/spot under MIT license.

By applying Spot to the pre-processing pipelines of the Human Connectome Project, compared in different operating systems, we showed that between-OS differences are mostly

originating in linear and non-linear image registration tools. Moreover, differences introduced during image registration propagate widely in the pipelines, leading to important variability in whole-brain segmentations.

Future work will investigate in more details the numerical stability of registration algorithms. Additionally, we plan on using Monte-Carlo arithmetic to inject controlled amounts of noise in pipelines and monitor uncertainty propagation and amplification in their results.

3.7 Availability of Source Code and Requirements

- Project name: Spot
- Project home page: `https://github.com/big-data-lab-team/spot`
- Operating system: Linux
- Programming language: Python (3.6 or higher)
- Main dependencies: ReproZip, Docker, and Boutiques
- Other dependencies: see `setup.py`
- License: MIT License
- Biotools identifier: spottool
- SciCrunch ID: RRID:SCR_018915
- DOI: 10.5281/zenodo.3873219

Chapter 4

Accurate simulation of operating system updates in neuroimaging using Monte-Carlo arithmetic

Ali Salari[1], Yohan Chatelain[1], Gregory Kiar[2], Tristan Glatard[1]

Published in:

MICCAI workshop on Uncertainty for Safe Utilization of Machine Learning in Medical Imaging (UNSURE)

https://doi.org/10.1007/978-3-030-87735-4_2

[1]Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada
[2]Center for the Developing Brain, Child Mind Institute, New York, NY, USA

Abstract

Operating system (OS) updates introduce numerical perturbations that impact the reproducibility of computational pipelines. In neuroimaging, this has important practical implications on the validity of computational results, particularly when obtained in systems such as high-performance computing clusters where the experimenter does not control software updates. We present a framework to reproduce the variability induced by OS updates in controlled conditions. We hypothesize that OS updates impact computational pipelines mainly through numerical perturbations originating in mathematical libraries, which we simulate using Monte-Carlo arithmetic in a framework called "fuzzy libmath" (FL). We applied this methodology to pre-processing pipelines of the Human Connectome Project, a flagship open-data project in neuroimaging. We found that FL-perturbed pipelines accurately reproduce the variability induced by OS updates and that this similarity is only mildly dependent on simulation parameters. Importantly, we also found between-subject differences were preserved in both cases, though the between-run variability was of comparable magnitude for both FL and OS perturbations. We found the numerical precision in the HCP pre-processed images to be relatively low, with less than 8 significant bits among the 24 available, which motivates further investigation of the numerical stability of components in the tested pipeline. Overall, our results establish that FL accurately simulates results variability due to OS updates, and is a practical framework to quantify numerical uncertainty in neuroimaging.

4.1 Introduction

Numerical round-off and cancellation errors are ubiquitous in floating-point computations. In neuroimaging, they contribute to results uncertainty along with other sources of variability, including population selection, scanning devices, sequence parameters, acquisition noise, and methodological flexibility [14, 13]. Numerical errors manifest particularly through variations in elementary mathematical libraries resulting from operating system (OS) updates. Indeed, due to implementation differences, mathematical functions available in different OS versions provide slightly different results. The impact of such epsilonesque differences on image analysis depends on the conditioning of the problem and the pipeline's numerical implementation. In neuroimaging, established image processing pipelines have been shown to be substantially impacted: for instance, differences in cortical thicknesses measured by the same Freesurfer version in different execution platforms were shown to reach statistical significance in some brain regions [51], and Dice coefficients as low as 0.6 were observed between FSL or Freesurfer segmentations obtained in different platforms [46, 110]. Such observations threaten the validity of neuroimaging results by revealing systematic instabilities.

Despite its possible implications on results validity, the effect of OS updates remains seldom studied due to (1) the lack of closed-form expressions of condition numbers for complex pipelines and non-differentiable non-linear analyses, (2) the technical challenge associated with experimental studies involving multiple OS distributions and versions, (3) the uncontrolled nature of OS updates. As a result, the effect of OS updates on neuroimaging analyses is generally neglected or handled through the use of software containers (Docker or Singularity), static executable builds, or similar approaches. While such techniques improve experiment portability, they only mask numerical instabilities and do not tackle them. Numerical perturbations are bound to reappear due to security updates [68], obsoleting software [104], or parallelization. Therefore, the mechanisms through which numerical instabilities propagate need to be investigated and eventually addressed.

This paper presents "fuzzy libmath" (FL), a framework to simulate OS updates in controlled conditions, allowing software developers to evaluate the robustness of their tools with respect to likely-to-occur numerical perturbations. As we hypothesize that numerical perturbations resulting from OS updates primarily come from implementation differences in elementary mathematical libraries, we leverage Monte-Carlo arithmetic (MCA) [100] to introduce controlled amounts of noise in these libraries. FL enables MCA in mathematical functions used by existing pipelines without the need to modify or recompile them. To

demonstrate the approach, we study the effect of common OS updates on the numerical precision of structural MRI pre-processing pipelines of the Human Connectome Project [118], a major neuroimaging initiative.

4.2 Simulating OS updates with Monte-Carlo arithmetic

MCA models floating-point roundoff and cancellations errors through random perturbations, allowing for the estimation of error distributions from independent random result samples. MCA simulates computations at a given virtual precision using the following perturbation:

$$inexact(x) = x + 2^{e_x - t}\xi \tag{1}$$

where e_x is the exponent in the floating-point representation of x, t is the virtual precision and ξ is a random uniform variable of $(-\frac{1}{2}, \frac{1}{2})$.

MCA allows for three perturbation modes: Random Rounding (RR) introduces the perturbation in function outputs, simulating roundoff errors; Precision Bounding (PB) introduces the perturbation in function operands, allowing for the detection of catastrophic cancellations; and, Full MCA combines RR and PB, resulting in the following perturbation:

$$mca\_mode(x \circ y) = inexact_{RR}(inexact_{PB}(x) \circ inexact_{PB}(y)) \tag{2}$$

To simulate OS updates, we introduce random perturbations in the GNU mathematical library, the main mathematical library in GNU/Linux systems. Instrumenting mathematical libraries with MCA raises a number of issues as many functions assume deterministic arithmetic. For instance, applying random perturbations around a discontinuity or within piecewise approximations results in large variations and a total loss of significance that are not relevant in our context. Therefore, we have applied MCA to proxy mathematical functions wrapping those in the original library, such that only the outputs of the original functions were perturbed but not their inputs or the implementations themselves. This technique allows us to control the magnitude of the perturbation as perceived by the application.

We instrumented the GNU mathematical library with MCA using Verificarlo [27], a tool that (1) uses the Clang compiler to generate an LLVM (http://llvm.org) Intermediate Representation (IR) of the source code, (2) replaces floating-point operations in the IR by a call to the Verificarlo API, and (3) compiles the modified IR to an executable using LLVM.

The perturbation applied by the Verificarlo API can be configured at runtime, for instance to change the virtual precision applied to single- and double-precision floating-point values.

The resulting MCA-instrumented mathematical library, "fuzzy libmath" (FL), is loaded in the pipeline using `LD_PRELOAD`, a Linux mechanism to force-load a shared library into an executable. As a result, functions defined in fuzzy libmath transparently overload the original ones without the need to modify or recompile the analysis pipeline. Fuzzy libmath functions call the original functions through `dlsym`, a function that returns the memory address of a symbol. To trigger MCA instrumentation, a floating-point zero is added to the output of the original function and the result of this sum is perturbed and returned.

Finally, we measure results precision as the number of significant bits among result samples, as defined in [100]:

$$s = -\log_2 \left| \frac{\sigma}{\mu} \right| \tag{3}$$

where σ and μ are the observed cross-sample standard deviation and average.

4.3 HCP Pipelines & Dataset

We apply the methodology described above to the minimal structural pre-processing pipeline associated with the Human Connectome Project (HCP) dataset [42], entitled "PreFreeSurfer". This pipeline consists of many independent components, including: spatial distortion correction, brain extraction, cross-modal registration, and alignment to standard space. Each high-level component of this pipeline (Fig. 9) consists of several function calls using FSL, the FMRIB Software Library [63]. The pipeline requires T1w and T2w images for each subject. A full description of the pipeline is available at [42].

It should be noted that the PreFreeSurfer pipeline uses both single and double precision functions from the GNU mathematical library. Among the pre-processing steps in the pipeline, it has been shown that linear and non-linear registrations implemented in FSL FLIRT [64, 62] and FNIRT [5] are the most sensitive to numerical instabilities [110].

We selected 20 unprocessed subjects from the HCP data release S500 available in the ConnectomDB repository. We selected these subjects from different subject types to cover execution paths sufficiently. For each, the available data consisted of 1 or 2 T1w and T2w images each, with spatial dimensions of $256 \times 320 \times 320$ and voxel resolution of 0.7 mm. Acquisition protocols and parameters are detailed in [118]. Two distinct experimental configurations were tested:

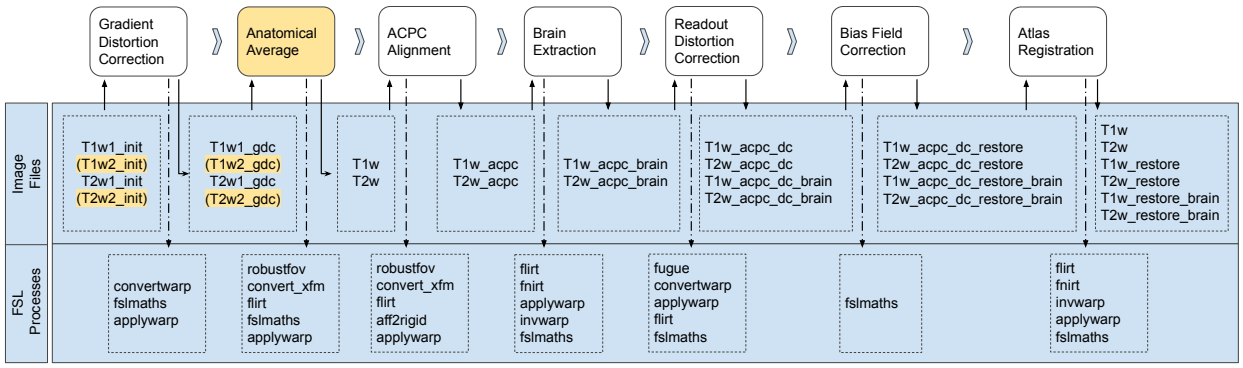


Figure 9: PreFreeSurfer pipeline steps.

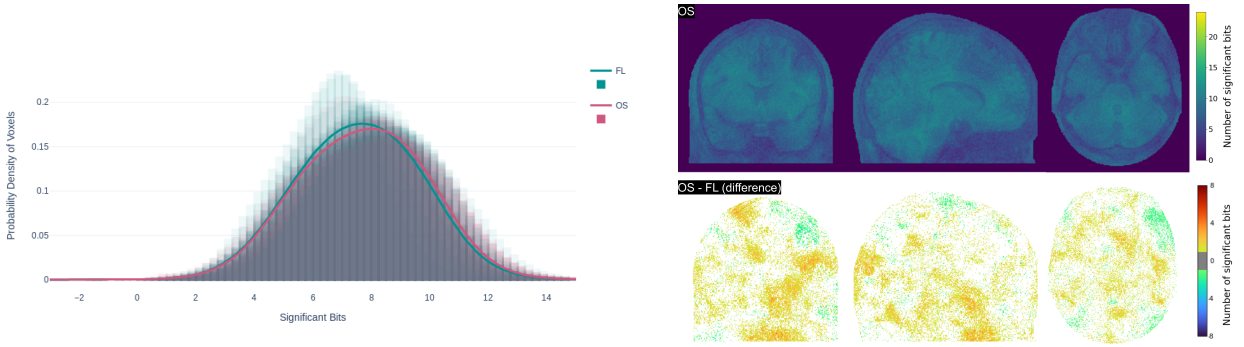
Operating Systems (OS): subjects were processed on three different Linux operating systems inside Docker images: CentOS7 (glibc v.2.17), CentOS8 (glibc v.2.28), and Ubuntu20 (glibc v.2.31).

Fuzzy libmath (FL): the dataset was processed on an Ubuntu20 system using fuzzy libmath. The virtual precision (t) for the perturbations was swept from 53 bits (the full mantissa for double-precision data) down to 1 bit by steps of 2. For $t >= 24$ bits, only double-precision was altered and single-precision was set to 24 bits, and for $t < 24$ bits, both double- and single-precision simultaneously were changed. Three FL-perturbed samples were generated for each subject and virtual precision, to match the number of OS samples.

After conducting both experiments, we selected the virtual precision that most closely simulated the variability observed across OSes via the root-mean-square error (RMSE) between the number of significant bits per voxel in all subjects and conditions. This precision is referred to as the global nearest virtual precision and was used to compare results obtained in both the FL and OS versions.

4.4 Results

The fuzzy libmath source code, Docker image specifications, and analysis code to reproduce the results are available at https://github.com/big-data-lab-team/MCA-libmath-paper. All experiments were conducted on the Béluga HPC computing cluster made available by Compute Canada through Calcul Québec. Béluga is a general-purpose cluster with 872 available nodes. All nodes contain 2× Intel Gold 6148 Skylake @ 2.4 GHz (40 cores/node) CPU, and node memory can range between 92 to 752 GB. The average processing time of



(a) Distribution of significant bits

(b) Significance map (subject average)

Figure 10: Comparison of OS and FL effects on the precision of PreFreeSurfer results for n=20 subjects. FL samples were obtained at the global nearest virtual precision of t=37 bits.

the pipeline without FL instrumentation was 69 minutes (average of 3 executions). The FL perturbation increased it to 93 minutes.

We ensured that the pipeline does not use pseudo-random numbers by processing each subject twice on the same operating system. To validate that FL was correctly instrumented with Verificarlo, we used Veritracer [20], a tool for tracing the numerical quality of variables over time. For one subject, the traces showed that the number of significant bits in the function outputs varied over time, confirming the instrumentation with MCA. Throughout the pipeline execution, Veritracer reported approximately 4 billion calls to FL, with the following ratio of calls: 47.12% `log`, 40.96% `exp`, 6.92% `expf`, 3.39% `logf`, 1.55% `sincosf`, and 0.06% of cumulated calls to `atan2f`, `pow`, `sqrt`, `exp2f`, `powf`, `log10f`, `log10`, `cos`, and `asin`. We also checked that long double types were not used.

4.4.1 Fuzzy libmath accurately simulates the effect of OS updates

Fuzzy libmath accurately reproduced the effect of OS updates, both globally (Fig. 10a) and locally (Fig. 10b). The distributions of significant bits in the atlas registered T1w images were nearly identical ($p > 0.05$, KS test) on the average and individual subject distributions for 15/20 subjects, after correcting for multiple comparisons. Locally, the spatial distribution of significant digits also appeared to be preserved. Losses in significance were observed mainly at the brain-skull interface and between brain lobes, indicating spatial dependency of numerical properties.

The average number of significant bits in either the FL or OS conditions were 7.76 out of 24 available, which corresponds to 2.32 significant (base 10) digits. This relatively low

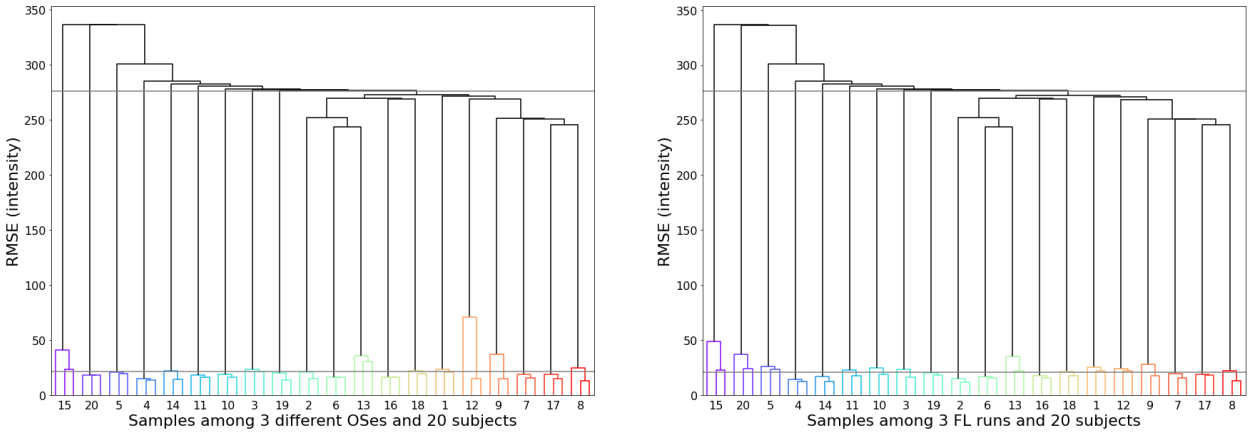


Figure 11: RMSE-based hierarchical clustering of OS (left) and FL (right) samples. Colors identify different subjects, showing that similarities between subjects are preserved by the numerical perturbations. Horizontal gray lines represent average RMSEs between (top line) and within (bottom line) subject clusters.

precision motivates future investigations of the stability of pipeline components, in particular for image registration.

4.4.2 Fuzzy libmath preserves between-subjects image similarity

Numerically-perturbed samples remained primarily clustered by individual subjects (Fig. 11), indicating that neither FL nor OS perturbations were impactful enough to blur the differences between subjects. Notably, the similarity between subjects was also preserved by the numerical perturbation, leading to the same subject ordering in the dendrograms. However, the average RMSE within samples of a given subject was approximately 13× lower than the average RMSE between different subjects. The fact that between-subject variabilities were nearly on the same order of magnitude as OS and FL variability demonstrates the potential severity of these instabilities.

4.4.3 Results are stable across virtual precision

The FL results presented previously were obtained at the global nearest virtual precision of t=37 bits, determined as the precision which minimized the RMSE between FL and OS average maps of significant bits. We varied the virtual precision in steps of 2 between t=1 and t=53 bits (Fig. 12). On average, no noticeable RMSE change was observed between the FL and OS variability for precisions ranging from t=21 to t=53 bits, which shows that FL can robustly approximate OS updates.

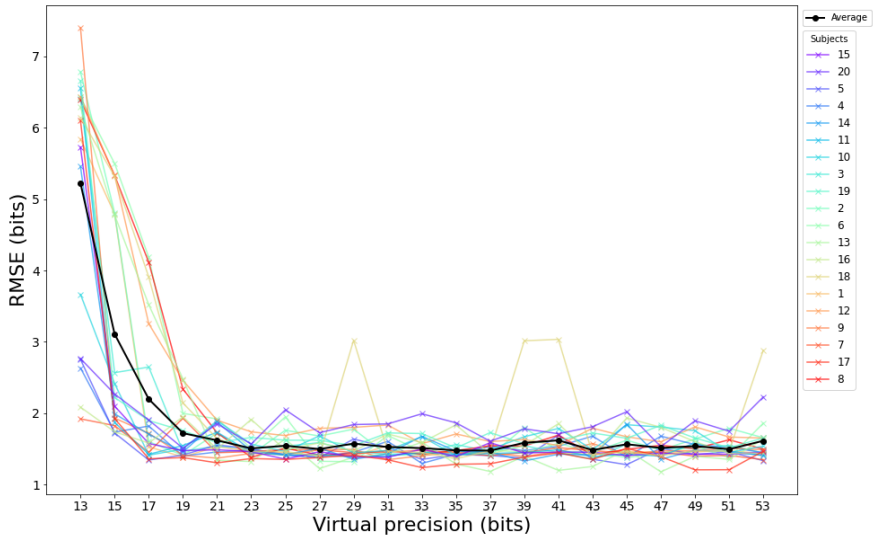


Figure 12: Comparison of RMSE values computed between OS and FL results for different virtual precisions.

The observed plateau suggests the existence of an "intrinsic precision" for the pipeline, above which no improvement in results precision is expected. For the tested pipeline, this intrinsic precision was observed at t=21 bits, which indicates that the pipeline could be implemented exclusively with single-precision floating-point representations (24 bits of mantissa) without loss of results precision. This would substantially decrease the pipeline memory footprint and computational time, as approximately 88% of operations used in this pipeline made use of double-precision data. In addition, the presence of such a plateau suggests that numerical perturbations introduced by OS updates might be in the range of machine error (t=53 bits), although it is also possible that the extent of the plateau results from the numerical conditioning of the tested pipeline. It is possible in contrast that the absence of such a plateau would suggest an unstable pipeline that would benefit either from correction or larger datatypes. The ability to capture stability across a range of precisions importantly demonstrates a key advantage of using FL to simulate OS variability.

The relationship between RMSE of individual subjects was generally consistent with the average line, with the notable exception of subject 18. The observed discrepancies between this subject and potential others might be leveraged for quality control checks and, as a result, inform tool development.

The pipeline failed to complete for at least one subject below the virtual precision of t=13 bits, also referred to as the tolerance of the pipeline. Specifically, 51% of pipeline executions crashed among all subjects for precisions ranging from 1–11 bits, and there was

no relationship between tolerance-level and precision. The error raised was in the Readout Distortion Correction portion of the pipeline, and appears to stem from the FSL FAST tissue segmentation. The specific source of the error within this component is presently unknown, but is an open question for further exploration.

4.5 Conclusion & Discussion

We demonstrated fuzzy libmath as an accurate method to simulate variability in neuroimaging results due to OS updates. Alongside this evaluation, fuzzy libmath can be used by pipeline developers or consumers to evaluate the numerical uncertainty of tools and results. Such evaluations may also help decrease pipeline memory usage and computational time through the controlled use of reduced numerical precision. Fuzzy libmath does not require any modification of the pipeline as it operates on the level of shared libraries. The accuracy of the simulations were shown to be robust across a wide range of virtual precisions, which reinforces the applicability of the method.

The proposed technique is directly applicable to MATLAB code executed with GNU Octave, to Python programs executed on Linux, and to C programs that depend on GNU libmath. Numerical noise can be introduced in other libraries, such as OpenBLAS or NumPy, using our https://github.com/verificarlo/fuzzy environment.

A commonly used approach to address instabilities resulting from OS version updates in practice is to sweep the issue under the rug of software containers or static linking. While such solutions are undoubtedly helpful to improve code portability or strict re-executability, a more honest position is to consider computational results as realizations of random variables depending on numerical error. The presented technique enables estimating result distributions, a first step toward making analyses reproducible across heterogeneous execution environments. While this work did not investigate the precise cause of numerical instabilities by tracing the system function calls, this is a topic for future work.

The tested OS versions span a timeframe of 7 years (2012–2020) and focused on GNU/Linux, a widely-used platform in neuroimaging [53]. Given that our experiments focused on numerical perturbations applied to mathematical functions, which are implemented similarly across OSes, our findings are likely to generalize to OS/X or MS Windows, although future work would be needed to confirm that. The tested pipeline is the official solution of the HCP project to pre-process data, and is considered the state-of-the-art. This pipeline assembles

software components from the FSL toolbox consistent with common practice in neuroimaging, such as in fMRIPrep [35] or the FSL feat workflow [63], to which fuzzy libmath can be directly applied. Efforts are on-going to use fuzzy libmath in fMRIPrep software tests, to guarantee that bug fixes do not perturb results beyond numerical uncertainty.

The fact that the induced numerical variability preserves image similarity between subjects is reassuring and, in fact, exciting. OS updates provide a convenient, practical target to define a virtual precision leading to a detectable but still reasonable numerical perturbation. However, it is also of importance that OS- and FL-induced variability were on a similar order of magnitude as subject-level effects. This suggests that the preservation of relative between-subject differences may not hold in all pipelines, and such a comparison could be used to evaluate the robustness of a pipeline to OS instabilities. The fact that the results observed across OS versions and FL perturbations arise from equally-valid numerical operations also suggests that the observed variability may contain meaningful signal. In particular, signal measured from these perturbations might be leveraged to enhance biomarkers, as suggested in [72] where augmenting a diffusion MRI dataset with numerically-perturbed samples was shown to improve age classification.

Chapter 5

Software variability in fMRI analysis: comparing between-tool and numerical errors

Ali Salari[1], Yohan Chatelain[1], Alexander Bowring[2], Camille Maumet[3], Gregory Kiar[4], Tristan Glatard[1]

Under review process and aim to submit on:
Human Brain Mapping (HBM) journal

[1]Department of Computer-Science and Software Engineering, Concordia University, Montreal, Canada
[2]Li Ka Shing Centre for Health Information and Discovery, Nuffield Department of Population Health, Big Data Institute, University of Oxford, Oxford, UK
[3]Inria, Univ Rennes, CNRS, Inserm, IRISA UMR 6074, Empenn ERL U 1228, Rennes, France
[4]Center for the Developing Brain, Child Mind Institute, New York, NY, USA

Abstract

Software variability broadly affects functional MRI analyses due to differences among analytical tools as well as numerical instabilities. However, the extent to which numerical and between-tool variabilities compare for a given analysis is unclear. We extended a previous comparison between fMRI analysis tools FSL, AFNI, and SPM, to measure numerical variability through Monte-Carlo arithmetic. We found that, in group analyses, between-tool variability was consistently larger than machine error, whereas in subject analyses, machine error approached between-tool variability in some cases. Furthermore, between-tool variability and machine error appeared moderately correlated in all tested conditions. Finally, we also found that numerical error at the precision of 17 bits — a precision level found in commonly-used libraries — had effects of similar magnitude **From Tristan:** check that than those of between-tool variability. Our findings motivate the continued investigation of numerical instability in neuroimaging, and position it as a possible proxy for between-tool variations.

5.1 Introduction

Software variability has been highlighted as an important source of error in fMRI analysis, originating from mainly two factors: between-tool variability denotes differences in the outcomes of different software toolboxes — such as FSL, AFNI and SPM — when analyzing the same dataset using similar methods. Numerical variability, in contrast, denotes differences in outcomes of the same analytical toolbox executed in different environments – such as different operating systems or different hardware platforms. This manuscript investigates the extent to which both types of variability relate to each other.

Numerical variability stems from errors introduced in floating point computations by numerical rounding, cancellation and absorption [96]. In practice, the occurrence of such errors is affected by variations in hardware, software dependencies, compilation, parallelization, and most likely other factors. In general, the magnitude of such errors and their variations is in the range of 1 unit in the last place (ulp) — also known as machine error, the difference between two successive floating-point numbers. However, when computations depend on imprecise libraries, numerical error may vary by several ulps. For instance, the numerical error of the complementary error function (erfc) is larger than 3 ulps in several mathematical libraries [129], implying that an update of this function could introduce numerical perturbations larger than 3 ulps. Due to numerical instability, machine error may amplify during its propagation through the analysis, resulting in considerable loss of precision in final results. For instance, in fMRI, numerical variability produced by Linux updates was shown to result in Dice coefficients ranging from 0.0 to 1.0 for ICA components produced by FSL MELODIC [46].

Between-tool variability is not a surprising concept: different programs executed on the same data may obviously produce different results. However, in fMRI analysis, programs all aim to estimate brain activity from measured variations in the BOLD signal, which is not expected to be measurably impacted by computational artifacts. Therefore, the observation reported in [14] that the three main fMRI analysis toolboxes may result in marked analytical differences (Dice coefficients ranging from 0.000 to 0.684 for thresholded activation maps), has raised substantial concern. In a follow-up study [15], the same authors isolated the analytical components where the pipelines diverge, highlighting a major impact of the signal model and suggesting that the root-cause for between-tool variability depends on the analysis design and task paradigm. Further, the work in [84] found moderate agreement between results produced by five independent fMRI analysis pipelines and analyzed in details the components that led to these variations. Between-tool variability is supposedly an important

determinant of the substantial analytical variability recently observed in fMRI [13].

We investigate the relationship between numerical and between-tool variability through two main questions: (1) how do both types of variability compare in magnitude, and (2) is there an association between them. The first question determines the extent to which each variability source may impact current fMRI results, and which one should be addressed in priority. The second question may provide deeper insights on the origins of software variability. Conceptually, both types of variability may reflect a single solution space shaped by local minima and ill-conditioning, which could be explored through numerical or software perturbations.

5.2 Materials and Methods

5.2.1 fMRI analysis & Dataset

We replicated the analysis described as study 'ds000001' in [111], relying on the data publicly available in OpenNeuro at https://openneuro.org/datasets/ds000001 and using three widely-used software packages for fMRI data processing, namely FMRIB Software Library (FSL) [63], Analysis of Functional NeuroImages (AFNI) [23], and Statistical Parametric Mapping (SPM) [103]. We selected this dataset because comparable analysis pipelines implemented in FSL, AFNI and SPM were already publicly available and extensively described in [14]. Furthermore, the work in [14] already evaluated the effect of tool variability for this dataset, which we intended to extend with the present quantification of numerical variability.

In the selected study, 16 healthy adult subjects participated in the balloon analog risk task [81] to measure risk-taking behavior over three scanning sessions [111]. We reused the preprocessing, first-level, and second-level analyses implemented by [14] consistently across all three software packages. Table 4, adapted from [14], summarizes the analytical steps in each pipeline.

5.2.2 Fuzzy libmath environment

To introduce numerical noise in the analyses, we used Fuzzy Libmath [109], a version of the GNU mathematical library (libmath) instrumented with Monte-Carlo arithmetic. Monte-Carlo arithmetic simulates numerical errors errors by introducing a controlled amount of

		FSL	AFNI	SPM
Preprocessing	Motion Correction	✓	✓	✓
	Segmentation			✓
	Brain Extraction (Anatomical)	✓	✓	✓
	Brain Extraction (Functional)		✓	
	Intra-subject Coregistration	✓	✓	✓
	Inter-subject Registration	✓	✓	✓
	Analysis Voxel Size	✓	✓	✓
	Smoothing	✓	✓	✓
First-level	Model Specification	✓	✓	✓
	Inclusion of 6 Motion Parameters	✓	✓	✓
	Model Estimation			✓
	Contrasts	✓	✓	✓
Second-level	Model Specification	✓	✓	✓
	Model Estimation			✓
	Contrasts		✓	✓
	Second-level Inference	✓	✓	✓

Table 4: Software processing steps (adapted from [14]).

noise in floating-point operations through the following perturbation [100]:

$$inexact(x) = x + 2^{e_x - t}\xi, \tag{4}$$

where e_x is the exponent in the floating-point representation of x, t is the virtual precision (the number of unperturbed bits in the mantissa of x), and ξ is a random uniform variable of $(-\frac{1}{2}, \frac{1}{2})$. We introduced the perturbation using Verificarlo [27], an LLVM compiler supporting Monte-Carlo arithmetic and other types of numerical instrumentations.

We loaded the instrumented libmath functions in the pipeline using LD_PRELOAD, a Linux mechanism to force-load a shared library into an executable. This mechanism allows functions defined in Fuzzy Libmath to transparently overload the original ones without the need to modify or recompile the analysis pipeline.

Fuzzy Libmath introduces numerical perturbations in the values returned by mathematical functions but not in their input values or within their implementation. This is done by wrapping the original functions and applying function `inexact` to their returned values. Listing 5.1 shows an example of this wrapping for the `log` function in single and double precision. In this wrapper, the original function is called through `dlsym`, a function that returns the memory address of a symbol — in our case `RTLD_NEXT`, the address of the next

occurrence of the function in memory. Compiling function wrappers with Verificarlo instruments the result of the addition between the original function output and the floating-point zero.

Listing 5.1: Sample wrapper function (C code)

```c
#include <dlfcn.h>
#include <math.h>

static double (*real_log)(double dbl);
static float (*real_logf)(float dbl);

// Override
double log(double dbl);
{
        real_log = dlsym(RTLD_NEXT, "log");
        return real_log(dbl) + 0.0;
}
float logf(float dbl);
{
        real_logf = dlsym(RTLD_NEXT, "logf");
        return real_logf(dbl) + 0.0f;
}
```

In [109], Fuzzy Libmath was shown to accurately simulate the effect of Linux operating system updates in structural pre-processing pipelines of the Human Connectome Project which are largely based on FSL. To validate our pipeline instrumentations for the present study, we first verified that non-instrumented executions of the same pipeline on the same dataset led to identical results. We also listed the pipeline library dependencies using the `ldd` Linux utility and verified that (1) the tested pipelines were dynamically linked to the GNU libmath library, and (2) there was no alternative implementation of elementary mathematical functions in the pipeline dependencies. Finally, we verified that the use of Fuzzy Libmath affected computational results.

5.2.3 Data processing

We measured between-tool (BT) variability by running the pipelines described in [14] with FSL version 5.0.10, AFNI version 18.1.09, and SPM12 version r7771 executed with GNU/Octave version 5.2. These software versions were identical to the study in [14] except for SPM

for which we used GNU/Octave instead of MATLAB to enable mathematical function instrumentation using Fuzzy Libmath. Indeed, MATLAB uses its own built-in mathematical functions, which prevents the use of Fuzzy Libmath. In AFNI, we set the number of threads to 7 even though AFNI executions in [14] were single-threaded. This was meant to reduce the time overhead resulting from Fuzzy Libmath instrumentation. All the analyses were conducted on the CentOS 7.3 operating system. The computations were performed on Compute Canada's Béluga cluster nodes, each with 2× Intel Gold 6148 Skylake @ 2.4 GHz (40 cores/node) CPU and 8 GB of RAM per core. To facilitate portability and reproducibility, we encapsulated the above-mentioned software packages in Docker container images based on CentOS 7.3.

We measured numerical (within-tool – WT) variability by running the same analyses three times using Fuzzy Libmath with a virtual precision of $t = 53$ bits for double-precision values and $t = 24$ bits for single-precision values. These values were chosen such that the numerical perturbation simulates machine error. The resulting samples are equally plausible estimates of the true numerical result at the precision used by the pipelines. Moreover, to evaluate numerical perturbations at different magnitudes, we repeated the FSL analyses for virtual precisions ranging from $t = 1$ bit to $t = 24$ bits for single-precision and double-precision values. We also repeated the FSL analyses for virtual precisions ranging from $t = 24$ bits to $t = 53$ bits for double-precision values, having set the virtual precision to $t = 24$ bits for single-precision values.

We evaluated WT and BT variability for thresholded as well as unthresholded group-level and subject-level t-statistics maps, by computing the standard deviation of t-statistic maps across pipelines (BT) or Fuzzy Libmath samples (WT). For BT, we computed the standard deviation across a given pair of tools. For WT, we computed the standard deviation across the three Fuzzy Libmath samples of a given tool. Moreover, we computed WT variability for a pair of tools (A, B) as follows:

$$\sigma^2_{WT(A,B)} = \sigma^2_{WT(A)} + \sigma^2_{WT(B)}, \tag{5}$$

where $\sigma_{WT(.)}$ is the numerical variability for a given tool and $\sigma_{WT(.,.)}$ is the numerical variability for a pair of tools.

Further, from the thresholded maps, we determined regional instability between activation clusters in the 360 regions in the Human Connectome Project Multi-Modal Parcellation atlas version 1.0 (HCP-MMP1.0) [41]. For BT, we considered a region unstable for a pair of tools if it contained activated voxels for a tool but not for the other one. For WT, we

considered a region unstable for a pair of tools (A, B) if for tool A or tool B it contained activated voxels only for some Fuzzy Libmath samples.

5.3 Results

The scripts, Docker images, and data to reproduce the results are available in out GitHub repository at https://github.com/big-data-lab-team/fuzzy-neurotools.

5.3.1 Sanity check

We verified the correctness of our analyses by comparing our unperturbed t-statistic group maps with the ones obtained in [14]. For FSL, we found identical checksums. For SPM, the checksums were different but differences were visually unnoticeable. For AFNI, the activation maps were similar overall, however, differences were more noticeable visually. The observed differences remained small (see Supplemental Material S1) and might be due to the use of GNU/Octave vs MATLAB in SPM and of multithreading in AFNI. We performed visual quality control of the AFNI and SPM results for each individual subject and confirmed that T1-weighted images were correctly skull-stripped and registered to the MNI template.

5.3.2 In the group analysis, BT variability was larger than and correlated with machine error

Table 5 presents summary statistics for the group-level t-statistics. For each tool pair (A, B), BT variability was significantly larger than machine error in tool A or B, in both thresholded and unthresholded maps (Wilcoxon signed-rank test and t-test $p < 10^{-5}$ for all tests). Figure 13-**A**,**B**,**C** shows that these global differences were confirmed regionally. In addition, BT and WT variability appeared moderately correlated across voxels (Pearson's R in [0.56, 0.58], p=0.0, Figure 13-**D**). Interestingly, the correlation appears driven by a set of voxels exhibiting comparable BT and WT values located around the identity line in Figure 13D, which, however, did not seem to have any spatial consistency.

		Group map				Subject maps	
		Thresholded		Unthresholded		Unthresholded	
		μ	σ	μ	σ	μ	σ
Between Tools (BT)	FSL vs. SPM	1.282	0.525	0.443	0.344	0.366	0.293
	FSL vs. AFNI	1.548	0.616	0.547	0.441	0.439	0.352
	AFNI vs. SPM	1.475	0.672	0.608	0.477	0.491	0.381
Within Tool (WT)	FSL	0.354	0.491	0.082	0.065	0.077	0.054
	SPM	0.252	0.448	0.054	0.045	0.048	0.037
	AFNI	0.434	0.524	0.128	0.135	0.108	0.131

Table 5: Voxel-wise mean and standard deviation of BT and WT variability in t-statistics maps.

5.3.3 In subject analyses, machine error approached BT variability in some regions

Table 5 also presents summary statistics for subject-level unthresholded t-statistics maps. As for group-level maps, BT variability was consistently larger than WT variability (Wilcoxon signed-rank test and t-test $p < 10^{-5}$ for all tests). However, Figure 14 shows that for some subjects, machine error (WT variability) approached and even surpassed BT in some regions. As for the group maps, BT and WT variability appeared moderately correlated for all subjects (R in [0.442, 0.612], p=0.0 for all subjects, see Supplemental Material S2).

5.3.4 At precision t=17 bits, WT variability approached BT variability in the group analysis

While the previous results were obtained for machine error, we also evaluated numerical variability across different virtual precisions for FSL and found that the virtual precision of t=17 bits minimized the RMSE between BT and WT in unthresholded group maps. FSL produced group maps with $\mu = 0.325$ and $\sigma = 0.287$ at this virtual precision, which indicates a bit lower variability compared to BT variability (Wilcoxon signed-rank test and t-test $p < 10^{-5}$ for all tests). Figure 15 shows BT and WT in the corresponding group maps obtained at t=17 bits, showing that BT and WT reach comparable magnitudes in some regions. BT and WT remained moderately correlated at this precision.

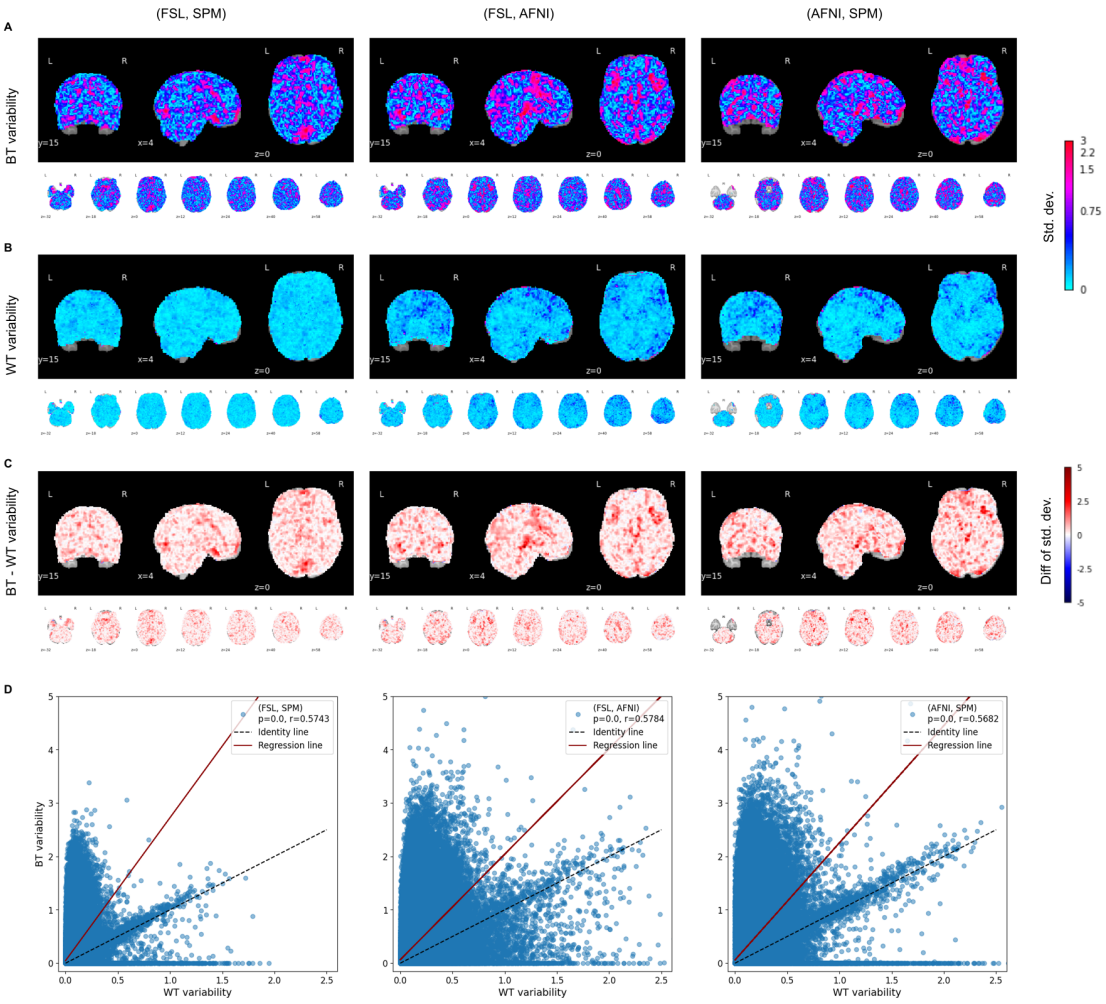


Figure 13: Unthresholded group-level variability computed between tools (**A**), within tools at machine error (**B**), difference between them (**C**), and voxel-wise comparison (**D**).

5.3.5 Previous results were confirmed in thresholded group maps

Figure 16-**A**,**B**,**C** compares BT and WT for thresholded group maps. Thresholding is an unstable operation that introduced variability at the edges of active regions for both BT and WT. Except in these regions, BT remained consistently larger than WT. Moreover, to measure correlation between BT and WT, we measured WT instability and BT instability in each region of the HCP-MMP1.0 parcellation. The confusion matrices in Figure 16-**D** report these instabilities for the 360 tested regions. The average ratio of unstable regions was 26% for BT and 9.2% for WT, which confirmed that BT variability was larger than machine error. The average Cohen's kappa score[5] between WT instability and BT instability was $\kappa = 0.17$,

[5] $\kappa \leq 0$ denotes chance agreement, $-1 \leq \kappa \leq 1$

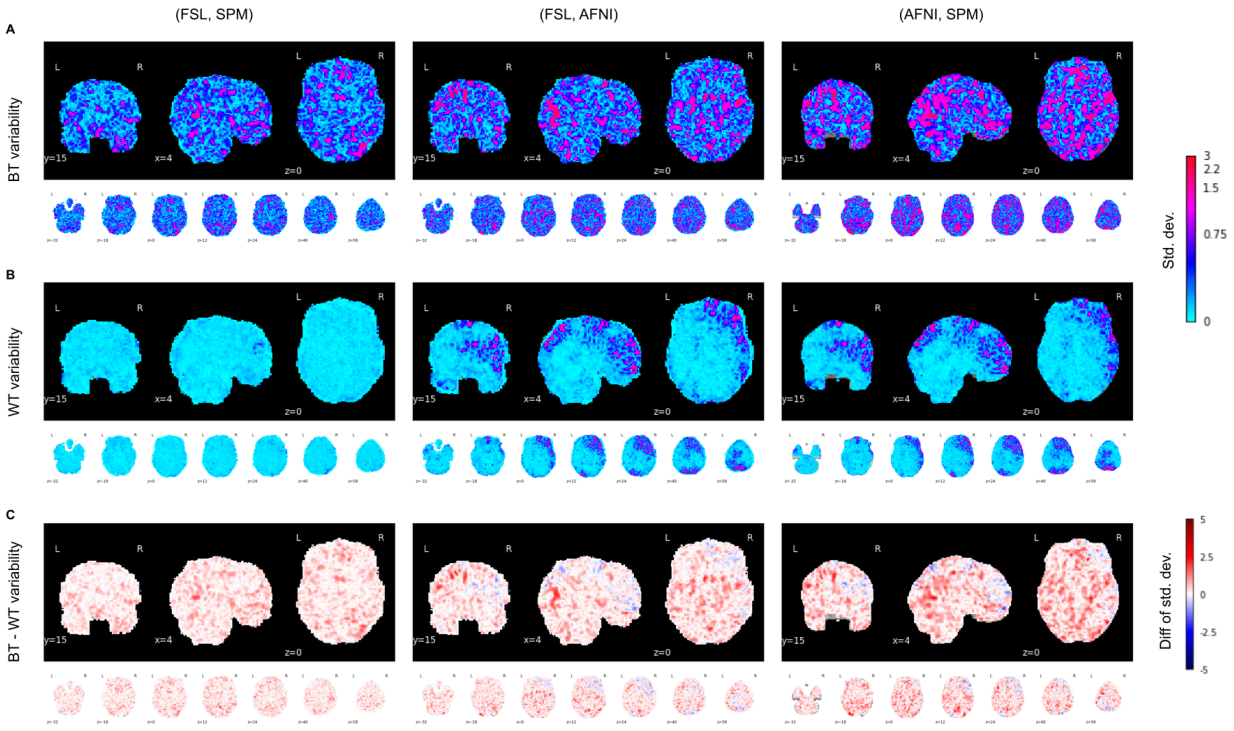


Figure 14: For subject with highest WT variability, unthresholded subject-level variability computed between tools (**A**), within tools at machine error (**B**), and difference between them (**C**).

indicating a moderate agreement between WT instability and BT instability.

5.4 Discussion

In fMRI group analyses, machine error remains an order of magnitude smaller than between-tool variability. Group analyses have a regularization effect toward numerical noise, which is expected to amplify as sample size increases. Therefore, for fMRI studies with large sample sizes, machine error can safely be neglected with respect to between-tool variability. The recommendation made in [13] to rely on "multiverse" analyses where multiple analysis tools are compared is therefore likely to successfully correct for machine error in group studies. Nevertheless, machine error remains substantial in group analyses that are based on a single tool, as is commonly the case in current fMRI studies. In particular, in our study, the inherent instability of thresholding was triggered by machine error in 9% of 360 brain regions, which indicates that machine error might have impacted neuroscientific conclusions related to these regions.

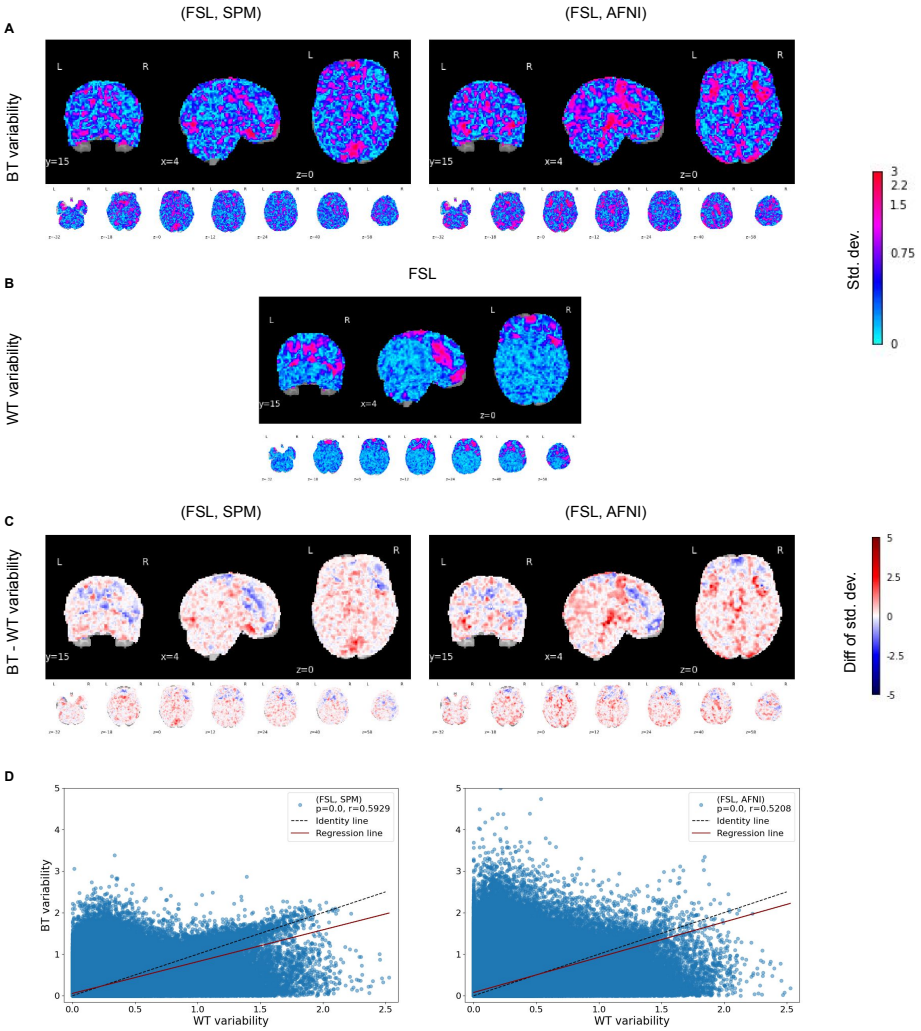


Figure 15: Unthresholded group t-statistics standard deviations computed between tools (**A**), within tools at the virtual precision of t=17 bits (**B**), difference between them (**C**), and voxel-wise comparison (**D**).

In subject analyses, machine error and between-tool variabilty can become of comparable magnitude for some subjects in some regions. This observation is particularly relevant to the development of fMRI-based biomarkers aiming at individualized phenotype predictions. Machine error is expected to play a non-negligible role in such analyses, even when predictions combine results produced by multiple tools. The observed regularization effect of group analyses toward numerical noise is consistent with observations made in [71] from diffusion MRI where connectome graph statistics were found to be substantially unstable at the subject-level while group distributions remained consistent.

For both group and subject analyses, between-tool variability and machine error were

found to be moderately correlated. Even though between-tool variability and numerical variability are different in nature, this result suggests that in some cases both sources of variability may have a common cause that might be related to the conditioning of the BOLD signal estimation problem in a specific dataset. Therefore, in some regions, instabilities of similar magnitude may be triggered by small numerical perturbations, model variations, or implementation differences. For instance, the analysis of datasets with high motion may be unstable both across tools and numerically.

Therefore, numerical stability may be a suitable proxy to study between-tool variability. This speculation might be of practical value to address software variability at large given that numerical stability refers to a consistent mathematical framework whereas between-tool variability remains more empirical. Likewise, useful quality control metrics may be derived from numerical and between-tool stability.

The finding that numerical variability approached tool variability at the virtual precision of t=17 bits is interesting too. Indeed, while machine error generally introduces differences in the order of 1 ulp — or t=53 bits for double-precision values and t=24 bits for single-precision ones — common scientific software dependencies introduce larger errors. For instance, SciPy's 2D spline interpolation was recently found to be precise up to t=10 bits [**?**] **From Tristan:** Fix ref and might therefore introduce numerical perturbations leading to errors in the range of between-tool variability. Such high errors might be triggered by updates in operating systems, Python, MATLAB, and other software dependencies.

Our results are limited by the type of numerical noise introduced in the analyses. We only perturbed the outputs of elementary mathematical functions while numerical noise could creep in any floating-point operation. Our estimation of numerical variability should therefore be considered a lower bound. Our estimation of tool varability is also likely to be underestimated, having tested only 3 analytical pipelines among the thousands available [19].

In conclusion, our results motivate further numerical stability investigations in fMRI analysis pipelines. Pipeline-level analyses could be conducted to identify specific components that contribute to numerical variability, and if possible correct them accordingly. When instability is inherent to the analysis, sampling results distributions through numerical perturbations might improve stability, as explored in [73]. Finally, pipeline-specific statistical corrections might be envisaged to account for numerical variability.

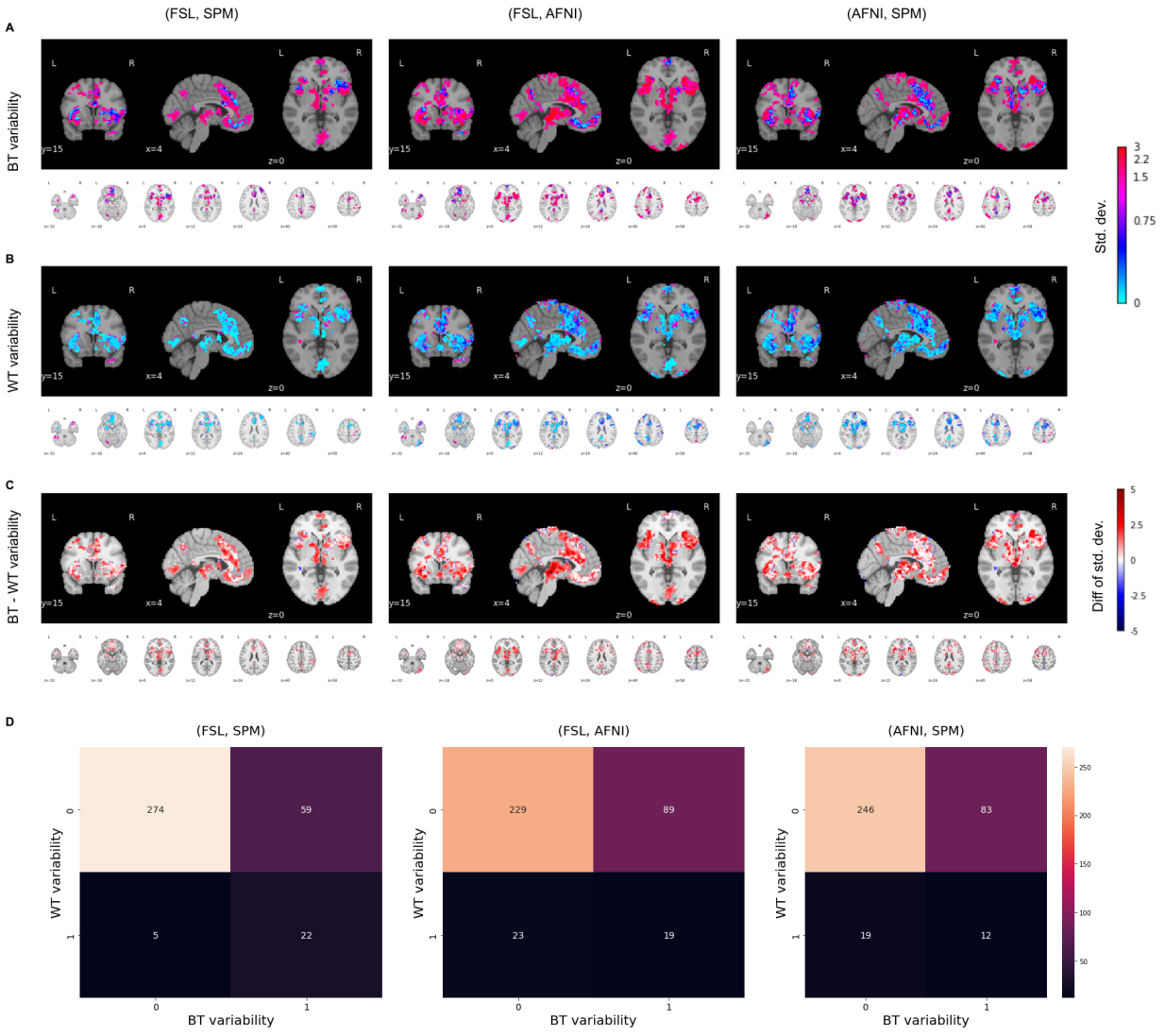


Figure 16: Thresholded group t-statistics standard deviations computed between tools (**A**), within tools at the virtual precision of t=17 bits (**B**), difference between them (**C**), and confusion matrices of activation instability in BT and WT among the 360 regions of the HCP-MMP1.0 parcellation (**D**).

Supplemental Materials

S1 Reproduced results

Figure S1 shows the difference in SPM and AFNI group analyses maps between the results in [14] and our replication. Numerical perturbations of 1 ulp are likely to have been introduced by our replication due to the use of GNU/Octave vs MATLAB for SPM and multithreading for AFNI. However, the group maps remained very similar overall, which led us to conclude that our results correctly reproduced the ones in [14].

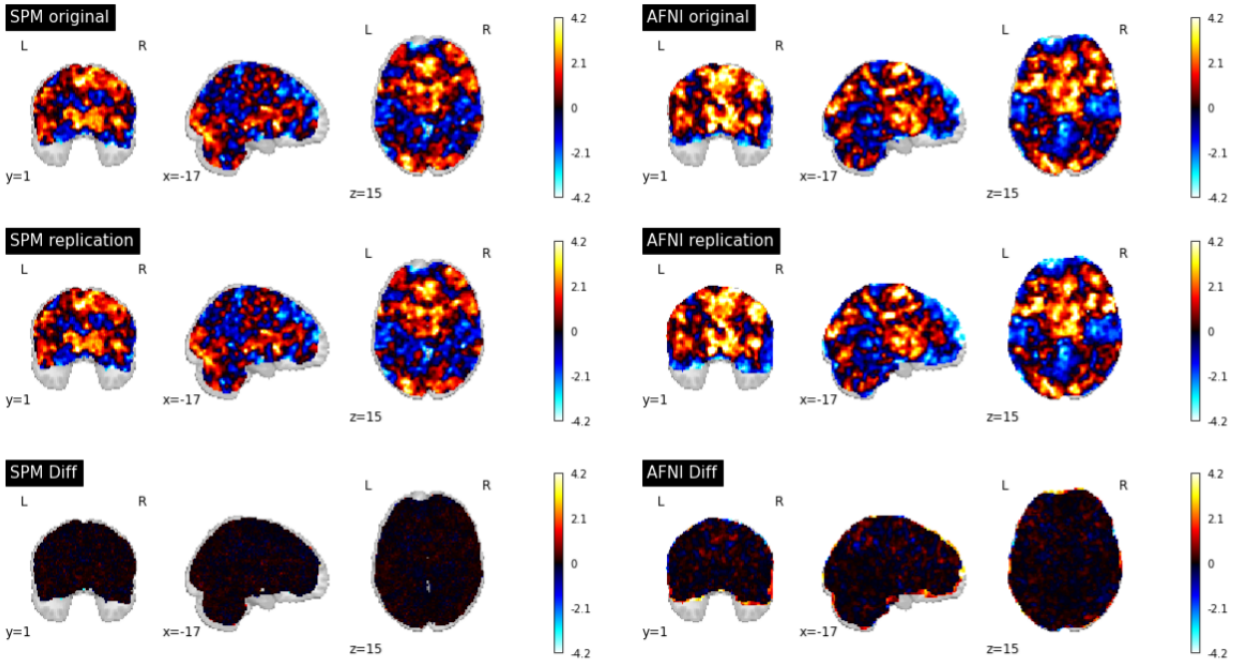


Figure S1: Differences between reproduced and original results obtained in [14] of unthresholded group-level t-statistics for SPM (left) and AFNI (right).

S2 BT and WT correlations for all subjects

Figure S2 plots the relationship between WT and BT for each subject, showing a consistent moderate correlation between BT and WT.

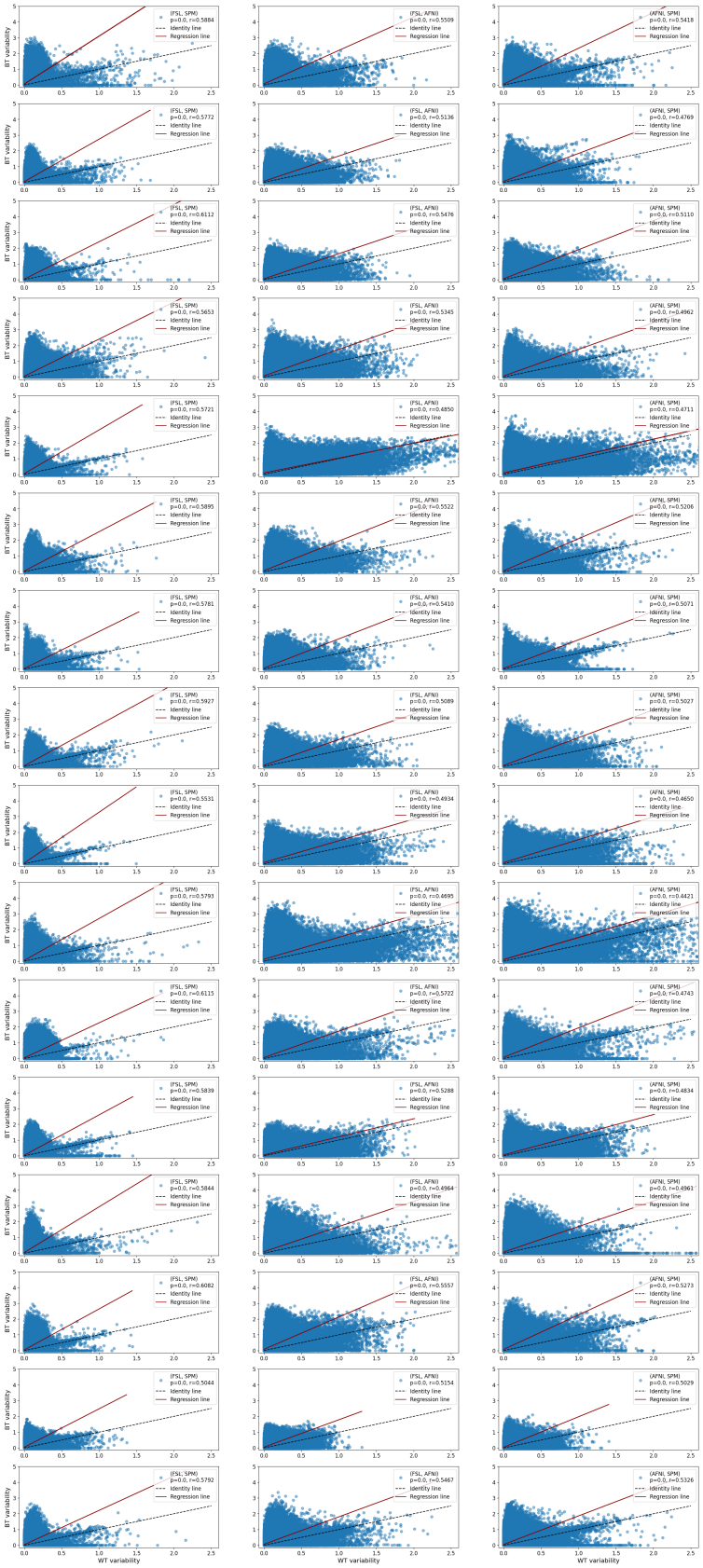


Figure S2: Comparison between BT and WT variabilty for 16 subjects.

Chapter 6

Discussion

The main aim of this thesis was to study the numerical stability of neuroimaging pipelines focusing on the effect of operating system variability. For this purpose, we leveraged system call interception techniques, including the ReproZip tool and perturbation models such as Monte-Carlo arithmetic (MCA) [100] as an extension of standard floating-point arithmetic that exploits randomness in basic floating-point operations. With this in mind, this chapter first considers the discussion of the findings and contributions. We then discuss the implication of the results and limitations of the current methodology in this work. This chapter concludes with recommendations for future studies.

S1 The impact of numerical perturbations

Throughout this thesis, we demonstrated the role of numerical errors in different computational environments in neuroimaging pipelines. In Chapter 3, we introduced Spot, a tool to detect the source of numerical differences in pipelines executed in different operating systems. This work localized the origin of processes that hamper bitwise reproducibility as the effect of OS updates. Among them, we observed registration processes as the highly contributed source of instability in the FSL tool. We found that differences can propagate during the pipeline execution and substantially impact the results of the entire pipeline (e.g., segmentations created by FreeSurfer). We also captured different results between subjects, showed the sensitivity of the analyses to dataset selection. Notably, the irreproducible processes identified by Spot were all associated with dynamically linked executables, showing that library updates do not impact statically linked executables.

The numerical differences were caused by truncation and round-off errors due to the precision limitations of the floating-point arithmetic. This is uncontrolled noise that originated from the updates of the system libraries associated with dynamically linked executables of the pipeline. Thereby, the observed instability across OSes depends on which operating systems are used. This limits the evaluation of the underlying stability of a particular tool.

Thereafter, in Chapter 4, we presented a framework to model the numerical uncertainty induced by OS updates in a control condition using Monte-Carlo arithmetic. We obtained an accurate simulation of OS variability using the FL library in which perturbing mathematical GNU/Linux library (libmath). We found that the pipeline could be implemented exclusively with lower precision of floating-point representations (single-precision) without loss of results precision. This would substantially decrease the pipeline memory footprint and computational time. However, the finding showed a very low number of significant digits, 5 out of 15 available digits. This motivates further investigation of the numerical stability of the main components of the tested pipelines, such as linear and non-linear registrations. Also, it is notable that OS- and FL-induced variability were on a similar order of magnitude as subject-level effects, showing that we applied a reasonable amount of perturbations.

This framework works on the level of the shared libraries, so there is no need for recompilation or modification of the pipeline or any other sources. We, therefore, believe that pipelines that depend on different third-party mathematical libraries could be studied similarly by building fuzzy versions of these dependencies. This motivated us to investigate the numerical quality in more applications like tool variability using the proposed MCA-based method.

Finally, in Chapter 5, we investigated the type of variability introduced in variations between software packages. We compared numerical variability with tool variability across three of the most popular software packages in neuroimaging. We found significant numerical variability that was comparable to tool variability in some brain regions. We also found that numerical instability in individual analyses was attenuated in group analyses. It is notable that we obtained more uncertainty on thresholded results than unthresholded maps, probably due to different thresholds used in different tools.

S2 The importance of numerical instability

This study was a contribution to uncertainty quantification in medical imaging. The numerical error is often neglected or only partially studied due to the associated engineering

challenges among the various sources of uncertainty involved in medical imaging results, including population selection, scanning devices and sequence parameters, acquisition noise, image reconstruction algorithms, and methodological flexibility. Our work proposed the methodologies and implementations to address this issue.

The current approach to address numerical instability resulting from OS updates is mainly to ignore the issue and sweep it under the rug of Docker containers or other types of virtualization. Although building static program and containerization techniques improve reproducibility across OSes, but small differences remained. It remains that computational results should be understood as realizations of a random variable resulting from floating-point arithmetic. The presented techniques in this thesis enable estimating result distributions, the initial step toward making analyses reproducible across execution environments, including HPC systems, GPU accelerators, or merely different workstations.

Results across my thesis showed the significance of numerical instabilities in neuroimaging pipelines and demonstrated numerical analysis techniques such as MCA as valuable methods for evaluating the associated variability. Moreover, in related work [71], it has been suggested that capturing this variability may improve the robustness of scientific findings. This finding importantly highlights how numerical variability may be regarded as a feature that should be taken into considerations by the pipeline developers in the neuroimaging and other scientific domains.

We demonstrated the numerical noise sensitivity of two of the most complex pipelines in neuroimaging, HCP preprocessing pipelines, PreFreeSurfer and FreeSurfer. Technically, these pipelines consist of a mix of tools assembled from different toolboxes through a variety of scripts written in different languages. In addition to the preprocessing steps, we evaluated the numerical stability of a complete fMRI analysis using three of the most popular tools in neuroimaging, including FSL, SPM, and AFNI. We, therefore, believe that the results presented in this thesis would apply to a wide range of other pipelines. However, the current methodology is limited to Linux operating systems. Our findings are likely to generalize to OS/X or MS Windows, although future work would be needed to confirm that.

Moreover, the interposition technique is only applicable to intercept system calls in dynamically linked programs. Further investigations are needed if we aim to evaluate the stability of the pipelines with statically linked executables.

S3 Recommendations for the future research

Among the processes that contribute to the pipeline instability, some of them substantially have a higher effect on results. For instance, iterative computations can accumulate rounding errors in some cases and significantly amplify errors. Therefore, evaluating the stability of pipeline components as well as the entire pipeline helps to identify processes that propagate and amplify errors within pipelines and then substitute them for more stable tools that perform a similar function, ultimately improving the quality of the pipeline. It is important to propose numerical debugging tools to identify such numerical bugs raised during the execution of a complicated pipeline. This is a well-studied problem in the floating-point error characterization research field, which has resulted in the development of debugging tools such as VeriTracer [20], FpDebug [10], Verrou [38]. These tools automatically replace all the floating-point operations with their MCA counterparts or other stochastic arithmetics to evaluate the numerical quality of the computation and pinpoint the parts of the source code. They instrument pipelines at compilation time in the compiler-based tools that need to access the source code, or running time in the Valgrind-based tools that add computation time overhead. Moreover, some of these tools need pre-knowledge of the functions or variables to be traced in the pipeline.

Using a combination of techniques developed in this thesis, in future work, we can create a debugging tool at the level of system call processes without recompiling the source code to identify the processes with the highest impact on results in the pipeline due to the operating system variability. This functionality could leverage the tool developed in Chapter 3 and the interposition technique to inject MCA perturbations described in Chapter 4. Moreover, we can use the principle of minimization by delta-debugging to search through a large number of processes. The delta-debugging algorithm is a general technique that can automatically narrow down the failure caused by changing circumstances that are critical to producing the bug, such as program input, program code, environmental configurations, etc [126, 127]. Instead of working on the program's code, we can apply the delta-debugging methodology to the program history by comparing various versions until the faulty change is found. This enables to identify of those processes in the program that significantly have higher effects on results. Finding these processes could narrow down further investigations toward stabilizing the pipelines.

Improving the instability needs actions different from evaluation and localization of instability. It is not a general approach to stabilize the entire pipeline; we should look at the relevant code section to find an appropriate solution. For example, we found that linear

and non-linear registration processes introduce errors in both FreeSurfer and PreFreeSurfer pipelines. On the other hand, we know that the registration procedure is sensitive to the initialization of the optimization method used [43]. A possible method to address such instabilities is using the bootstrapping technique. In [43], the authors explained that bootstrapping is an efficient technique to improve the robustness of motion estimation. The bootstrap version of the pipelines computed the median transformation results from the 30 samples. In addition to bootstrapping, it is shown that the bagging technique can reduce the effect of the medians of the parameters of the 30 transformations. Bagging, also called bootstrap aggregating, is a simple and powerful ensemble method. It helps reduce both bias and variance in the results. So, we can possibly stabilize pipelines and improve their accuracy using aggregates of results obtained with data perturbations. However, it is a compute-intensive technique that should be used only when no other solution to the instability is available. Further study needs to be conducted into the processes involved in analysis instability.

The main goal of this thesis was to evaluate instabilities within neuroimaging pipelines. An exciting research topic for future work could be finding that the variability arising from numerical perturbations may contain meaningful signals. This suggests that the perturbation model is not only helpful as a method for measuring the stability of analyses but that it can be applied to improve their quality. For this purpose, we can also apply other types of perturbations. Given that FL only perturbs basic mathematical functions, we expect more numerical variability by perturbing operations that rely on the linear algebra libraries BLAS and LAPACK. For future study, this can be evaluated using MCA-instrumented versions of BLAS and LAPACK along with other libraries available in the Fuzzy project in Verificarlo's GitHub repository at `github.com/verificarlo/fuzzy`.

Chapter 7

Conclusion

The numerical stability of the computational analyses plays an important role in the reproducibility of the scientific findings. It has been evaluated that results are sensitive to the computing environment changes such as operating system and analysis toolbox, particularly in computationally intensive domains where results rely on a series of complex computations. Throughout this thesis, we demonstrated the impact of numerical instabilities on neuroimaging results. We implemented Spot tool that localizes irreproducible processes between operating system variations. Moreover, we presented an MCA-based method to apply numerical perturbations in floating-point operations, simulating OS variability and studying the numerical instabilities more comprehensively. This was done using the Linux interception utility LD_PRELOAD, which transparently interposed system calls to an instrumented counterpart. We expanded our findings by capturing numerical variability among different software packages and comparing it with the tool variability. As the successful completion of this thesis, we built the tools that enable software developers and researchers to evaluate the stability of their pipelines and results when dynamically linked mathematical libraries are changed. Furthermore, the findings of this thesis could be used to narrow down further investigations toward stabilizing pipelines.

Bibliography

[1] Neuroimagin data model. nidm-results. http://nidm.nidash.org/specs/nidm-results_130.html. [Online; accessed 6-October-2021].

[2] Spm-statistical parametric mapping. https://www.fil.ion.ucl.ac.uk/spm/. [Online; accessed 6-October-2021].

[3] Association for computing machinery. artifact review and badging., 2021. [Online; accessed 6-October-2021].

[4] Yasser Ad-Dab'bagh, O Lyttelton, JS Muehlboeck, C Lepage, D Einarson, K Mok, O Ivanov, RD Vincent, J Lerch, E Fombonne, et al. The civet image-processing environment: a fully automated comprehensive pipeline for anatomical neuroimaging research. In Proceedings of the 12th annual meeting of the organization for human brain mapping, page 2266. Florence, Italy, 2006.

[5] Jesper LR Andersson, Mark Jenkinson, Stephen Smith, et al. Non-linear registration, aka Spatial normalisation FMRIB. Technical Report TR07JA2, FMRIB Analysis Group of the University of Oxford, 2007.

[6] Atlassian. Git lfs - large file storage — atlassian git tutorial, 2021. [Online; accessed 6-October-2021].

[7] Brian B Avants, Nick Tustison, and Gang Song. Advanced normalization tools (ants). Insight j, 2:1–35, 2009.

[8] Monya Baker. 1,500 scientists lift the lid on reproducibility. Nature News, 533(7604):452, 2016.

[9] Bernard Beauzamy. Méthodes probabilistes pour l'étude des phénomènes réels. Société de Calcul Mathématiques, SA" Algorithmes et optimisation", 2004.

[10] Florian Benz, Andreas Hildebrandt, and Sebastian Hack. A dynamic program analysis to find floating-point accuracy problems. ACM SIGPLAN Notices, 47(6):453–462, 2012.

[11] Nikhil Bhagwat, Amadou Barry, Erin W Dickie, Shawn T Brown, Gabriel A Devenyi, Koji Hatano, Elizabeth DuPre, Alain Dagher, M Mallar Chakravarty, Celia MT Greenwood, et al. Understanding the impact of preprocessing pipelines on neuroimaging cortical surface analyses. bioRxiv, 2020.

[12] Carl Boettiger. An introduction to docker for reproducible research. ACM SIGOPS Operating Systems Review, 49(1):71–79, 2015.

[13] Rotem Botvinik-Nezer, Felix Holzmeister, Colin F Camerer, Anna Dreber, Juergen Huber, Magnus Johannesson, Michael Kirchler, Roni Iwanir, Jeanette A Mumford, R Alison Adcock, et al. Variability in the analysis of a single neuroimaging dataset by many teams. Nature, 582(7810):84–88, 2020.

[14] Alexander Bowring, Camille Maumet, and Thomas E Nichols. Exploring the impact of analysis software on task fMRI results. Human Brain Mapping, 40:1–23, 2019.

[15] Alexander Bowring, Thomas Nichols, and Camille Maumet. Isolating the sources of pipeline-variability in group-level task-fmri results. 2021.

[16] Leo Breiman. Bagging predictors. Machine learning, 24(2):123–140, 1996.

[17] Leo Breiman et al. Heuristics of instability and stabilization in model selection. The annals of statistics, 24(6):2350–2383, 1996.

[18] Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos E Scheidegger, Cláudio T Silva, and Huy T Vo. Vistrails: visualization meets data management. In Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pages 745–747. ACM, 2006.

[19] Joshua Carp. On the plurality of (methodological) worlds: estimating the analytic flexibility of fmri experiments. Frontiers in neuroscience, 6:149, 2012.

[20] Yohan Chatelain, Pablo de Oliveira Castro, Eric Petit, David Defour, Jordan Bieder, and Marc Torrent. Veritracer: Context-enriched tracer for floating-point arithmetic analysis. In 2018 IEEE 25th Symposium on Computer Arithmetic (ARITH), pages 61–68. IEEE, 2018.

[21] James Cheney, Anthony Finkelstein, Bertram Ludäscher, and Stijn Vansummeren. Principles of provenance (dagstuhl seminar 12091). In Dagstuhl Reports, volume 2. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2012.

[22] Fernando Chirigati, Rémi Rampin, Dennis Shasha, and Juliana Freire. Reprozip: Computational reproducibility with ease. In Proceedings of the 2016 International Conference on Management of Data, pages 2085–2088. ACM, 2016.

[23] Robert W Cox. Afni: software for analysis and visualization of functional magnetic resonance neuroimages. Computers and Biomedical research, 29(3):162–173, 1996.

[24] Robert W Cox. Afni: what a long strange trip it's been. Neuroimage, 62(2):743–747, 2012.

[25] Samir Das, Alex P Zijdenbos, Dario Vins, Jonathan Harlap, and Alan C Evans. Loris: a web-based data management system for multi-center studies. Frontiers in neuroinformatics, 5:37, 2012.

[26] James Demmel and Hong Diep Nguyen. Numerical reproducibility and accuracy at exascale. In 2013 IEEE 21st Symposium on Computer Arithmetic, pages 235–237. IEEE, 2013.

[27] Christophe Denis, Pablo de Oliveira Castro, and Eric Petit. Verificarlo: Checking Floating Point Accuracy through Monte Carlo Arithmetic. In 2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH), pages 55–62, 2016.

[28] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. Nature biotechnology, 35(4):316, 2017.

[29] Lee R Dice. Measures of the amount of ecologic association between species. Ecology, 26(3):297–302, 1945.

[30] Kai Diethelm. The limits of reproducibility in numerical simulation. Computing in Science & Engineering, 14(1):64–72, 2011.

[31] Kai Diethelm. The limits of reproducibility in numerical simulation. Computing in Science & Engineering, 14(1):64–72, 2012.

[32] David L Donoho, Arian Maleki, Inam Ur Rahman, Morteza Shahram, and Victoria Stodden. Reproducible research in computational harmonic analysis. Computing in Science & Engineering, 11(1), 2009.

[33] Peter D Düben, Hugh McNamara, and Tim N Palmer. The use of imprecise processing to improve accuracy in weather & climate prediction. Journal of Computational Physics, 271:2–18, 2014.

[34] Anders Eklund, Thomas E Nichols, and Hans Knutsson. Cluster failure: why fmri inferences for spatial extent have inflated false-positive rates. Proceedings of the National Academy of Sciences, page 201602413, 2016.

[35] Oscar Esteban, Christopher J Markiewicz, Ross W Blair, Craig A Moodie, A Ilkay Isik, Asier Erramuzpe, James D Kent, Mathias Goncalves, Elizabeth DuPre, Madeleine Snyder, et al. fMRIPrep: a robust preprocessing pipeline for functional MRI. Nature methods, 16(1):111–116, 2019.

[36] Alan C Evans, Sean Marrett, Peter Neelin, Louis Collins, Keith Worsley, Weiqian Dai, Sylvain Milot, Ernst Meyer, and Daniel Bub. Anatomical mapping of functional activation in stereotactic coordinate space. Neuroimage, 1(1):43–53, 1992.

[37] Suvarna Fadnavis. Some numerical experiments on round-off error growth in finite precision numerical computation. arXiv preprint physics/9807003, 1998.

[38] François Févotte and Bruno Lathuilière. Debugging and optimization of hpc programs with the verrou tool. In 2019 IEEE/ACM 3rd International Workshop on Software Correctness for HPC Applications (Correctness), pages 1–10. IEEE, 2019.

[39] Bruce Fischl. Freesurfer. Neuroimage, 62(2):774–781, 2012.

[40] Eleftherios Garyfallidis, Matthew Brett, Bagrat Amirbekian, Ariel Rokem, Stefan Van Der Walt, Maxime Descoteaux, and Ian Nimmo-Smith. Dipy, a library for the analysis of diffusion mri data. Frontiers in neuroinformatics, 8:8, 2014.

[41] Matthew F Glasser, Timothy S Coalson, Emma C Robinson, Carl D Hacker, John Harwell, Essa Yacoub, Kamil Ugurbil, Jesper Andersson, Christian F Beckmann, Mark Jenkinson, et al. A multi-modal parcellation of human cerebral cortex. Nature, 536(7615):171–178, 2016.

[42] Matthew F Glasser, Stamatios N Sotiropoulos, J Anthony Wilson, Timothy S Coalson, Bruce Fischl, Jesper L Andersson, Junqian Xu, Saad Jbabdi, Matthew Webster, Jonathan R Polimeni, et al. The minimal preprocessing pipelines for the Human Connectome Project. Neuroimage, 80:105–124, 2013.

[43] Tristan Glatard and Pierre Bellec. Numerical stability of motion estimation in fmri time series. In Annual Meeting of the Organization for Human Brain Mapping, 2018.

[44] Tristan Glatard, Gregory Kiar, Tristan Aumentado-Armstrong, Natacha Beck, Pierre Bellec, Rémi Bernard, Axel Bonnet, Shawn T Brown, Sorina Camarasu-Pop, Frédéric Cervenansky, et al. Boutiques: a flexible framework to integrate command-line applications in computing platforms. GigaScience, 7(5):giy016, 2018.

[45] Tristan Glatard, Gregory Kiar, Tristan Aumentado-Armstrong, Natacha Beck, Pierre Bellec, Rémi Bernard, Axel Bonnet, Sorina Camarasu-Pop, Frédéric Cervenansky, Samir Das, et al. Boutiques: a flexible framework for automated application integration in computing platforms. arXiv preprint arXiv:1711.09713, 2017.

[46] Tristan Glatard, Lindsay B. Lewis, Rafael Ferreira da Silva, Reza Adalat, Natacha Beck, Claude Lepage, Pierre Rioux, Marc-Etienne Rousseau, Tarek Sherif, Ewa Deelman, Najmeh Khalili-Mahani, and Alan C. Evans. Reproducibility of neuroimaging analyses across operating systems. Frontiers in Neuroinformatics, 9:12, 2015.

[47] Steven N Goodman, Daniele Fanelli, and John PA Ioannidis. What does research reproducibility mean? Science translational medicine, 8(341):341ps12–341ps12, 2016.

[48] K Gorgolewski, Oscar Esteban, Gunnar Schaefer, B Wandell, and R Poldrack. Openneuro—a free online platform for sharing and analysis of neuroimaging data. Organization for Human Brain Mapping. Vancouver, Canada, 1677, 2017.

[49] Krzysztof Gorgolewski, Christopher D Burns, Cindee Madison, Dav Clark, Yaroslav O Halchenko, Michael L Waskom, and Satrajit S Ghosh. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. Frontiers in neuroinformatics, 5:13, 2011.

[50] Krzysztof J Gorgolewski, Tibor Auer, Vince D Calhoun, R Cameron Craddock, Samir Das, Eugene P Duff, Guillaume Flandin, Satrajit S Ghosh, Tristan Glatard, Yaroslav O Halchenko, et al. The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. Scientific Data, 3:160044, 2016.

[51] Ed H B M Gronenschild, Petra Habets, Heidi I L Jacobs, Ron Mengelers, Nico Rozendaal, Jim van Os, and Machteld Marcelis. The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements. PloS one, 7(6):e38234, January 2012.

[52] Philip Guo. Cde: A tool for creating portable experimental software packages. Computing in Science & Engineering, 14(4):32–35, 2012.

[53] Michael Hanke and Yaroslav O Halchenko. Neuroscience runs on gnu/linux. Frontiers in neuroinformatics, 5:8, 2011.

[54] Khawar Hasham, Kamran Munir, and Richard McClatchey. Cloud infrastructure provenance collection and management to reproduce scientific workflows execution. Future Generation Computer Systems, 86:799–820, 2018.

[55] Timothy Hickey, Qun Ju, and Maarten H Van Emden. Interval arithmetic: From principles to implementation. Journal of the ACM (JACM), 48(5):1038–1068, 2001.

[56] David RC Hill. Numerical reproducibility of parallel and distributed stochastic simulation using high-performance computing. In Computational Frameworks, pages 95–109. Elsevier, 2017.

[57] John PA Ioannidis. Why most published research findings are false. PLoS medicine, 2(8):e124, 2005.

[58] Paul Jaccard. The distribution of the flora in the alpine zone. 1. New phytologist, 11(2):37–50, 1912.

[59] Clifford R Jack Jr, Matt A Bernstein, Nick C Fox, Paul Thompson, Gene Alexander, Danielle Harvey, Bret Borowski, Paula J Britson, Jennifer L. Whitwell, Chadwick Ward, et al. The alzheimer's disease neuroimaging initiative (adni): Mri methods. Journal of Magnetic Resonance Imaging: An Official Journal of the International Society for Magnetic Resonance in Medicine, 27(4):685–691, 2008.

[60] Yves Janin, Cédric Vincent, and Rémi Duraffort. Care, the comprehensive archiver for reproducible execution. In Proceedings of the 1st ACM SIGPLAN Workshop on Reproducible Research Methodologies and New Publication Models in Computer Engineering, page 1. ACM, 2014.

[61] Yaroslav Halchenko; Michael Hanke; Benjamin Poldrack; Debanjum; Gergana Alteva; jason gors; Christian Olaf Häusler; Alex Waite; yetanothertestuser; yarikoptic-private; Horea Christian. datalad: Keep scientific data under control with git and git-annex, 2017.

[62] Mark Jenkinson, Peter Bannister, Michael Brady, and Stephen Smith. Improved optimization for the robust and accurate linear registration and motion correction of brain images. Neuroimage, 17(2):825–841, 2002.

[63] Mark Jenkinson, Christian F Beckmann, Timothy EJ Behrens, Mark W Woolrich, and Stephen M Smith. FSL. Neuroimage, 62(2):782–790, 2012.

[64] Mark Jenkinson and Stephen Smith. A global optimisation method for robust affine registration of brain images. Medical image analysis, 5(2):143–156, 2001.

[65] Fabienne Jézéquel, Jean-Luc Lamotte, and Issam Saïd. Estimation of numerical reproducibility on cpu and gpu. In 2015 Federated Conference on Computer Science and Information Systems (FedCSIS), pages 675–680. IEEE, 2015.

[66] Joey Hess. git-annex: a distributed file synchronization system written in haskell., 2021. [Online; accessed 6-October-2021].

[67] Jorge Jovicich, Silvester Czanner, Xiao Han, David Salat, Andre van der Kouwe, Brian Quinn, Jenni Pacheco, Marilyn Albert, Ronald Killiany, Deborah Blacker, et al. Mri-derived measurements of human subcortical, ventricular and intracranial brain volumes: reliability effects of scan sessions, acquisition sequences, data analyses, scanner upgrade, scanner vendors and field strengths. Neuroimage, 46(1):177–192, 2009.

[68] Bhupinder Kaur, Mathieu Dugré, Aiman Hanna, and Tristan Glatard. An analysis of security vulnerabilities in container images for scientific data analysis. GigaScience, 10(6):giab025, 2021.

[69] David N Kennedy, Sanu A Abraham, Julianna F Bates, Albert Crowley, Satrajit Ghosh, Tom Gillespie, Mathias Goncalves, Jeffrey S Grethe, Yaroslav O Halchenko, Michael Hanke, et al. Everything matters: the ReproNim perspective on reproducible neuroimaging. Frontiers in neuroinformatics, 13:1, 2019.

[70] David N Kennedy, Christian Haselgrove, Jon Riehl, Nina Preuss, and Robert Buccigrossi. The nitrc image repository. Neuroimage, 124:1069–1073, 2016.

[71] Gregory Kiar, Yohan Chatelain, Pablo de Oliveira Castro, Eric Petit, Ariel Rokem, Gael Varoquaux, Bratislav Misic, Alan C Evans, and Tristan Glatard. Numerical instabilities in analytical pipelines lead to large and meaningful variability in brain networks. bioRxiv, 2020.

[72] Gregory Kiar, Yohan Chatelain, Ali Salari, Alan C Evans, and Tristan Glatard. Data augmentation through monte carlo arithmetic leads to more generalizable classification in connectomics. bioRxiv, 2020.

[73] Gregory Kiar, Yohan Chatelain, Ali Salari, Alan C Evans, and Tristan Glatard. Data augmentation through monte carlo arithmetic leads to more generalizable classification in connectomics. Neurons, Behavior, Data analysis, and Theory, 2021.

[74] Gregory Kiar, Pablo de Oliveira Castro, Pierre Rioux, Eric Petit, Shawn T Brown, Alan C Evans, and Tristan Glatard. Comparing perturbation models for evaluating stability of neuroimaging pipelines. The International Journal of High Performance Computing Applications, 34(5):491–501, 2020.

[75] Jaan Kiusalaas. Numerical methods in engineering with Python 3. Cambridge university press, 2013.

[76] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the linux virtual machine monitor. In Proceedings of the Linux symposium, volume 1, pages 225–230. Dttawa, Dntorio, Canada, 2007.

[77] Dagmar Krefting, Michael Scheel, Alina Freing, Svenja Specovius, Friedemann Paul, and Alexander Brandt. Reliability of quantitative neuroimage analysis using freesurfer in distributed environments. In MICCAI Workshop on High-Performance and Distributed Computing for Medical Imaging.(Toronto, ON), 2011.

[78] Gina R Kuperberg, Matthew R Broome, Philip K McGuire, Anthony S David, Marianna Eddy, Fujiro Ozawa, Donald Goff, W Caroline West, Steven CR Williams, Andre JW van der Kouwe, et al. Regionally localized thinning of the cerebral cortex in schizophrenia. Archives of general psychiatry, 60(9):878–888, 2003.

[79] Gregory M Kurtzer, Vanessa Sochat, and Michael W Bauer. Singularity: Scientific containers for mobility of compute. PloS one, 12(5):e0177459, 2017.

[80] Manja Lehmann, Abdel Douiri, Lois G Kim, Marc Modat, Dennis Chan, Sebastien Ourselin, Josephine Barnes, and Nick C Fox. Atrophy patterns in alzheimer's disease and semantic dementia: a comparison of freesurfer and manual volumetric measurements. Neuroimage, 49(3):2264–2274, 2010.

[81] Carl W Lejuez, Jennifer P Read, Christopher W Kahler, Jerry B Richards, Susan E Ramsey, Gregory L Stuart, David R Strong, and Richard A Brown. Evaluation of a behavioral measure of risk taking: the balloon analogue risk task (bart). Journal of Experimental Psychology: Applied, 8(2):75, 2002.

[82] Lindsay Lewis, Claude Lepage, Najmeh Khalili-Mahani, Mona Omidyeganeh, Seun Jeon, Patrick Bermudez, Alex Zijdenbos, Robert Vincent, Reza Adalat, and Alan Evans. Robustness and reliability of cortical surface reconstruction in civet and freesurfer. In Annual Meeting of the Organization for Human Brain Mapping, 2017.

[83] Lindsay B Lewis, Claude Y Lepage, and Alan C Evans. Utilizing the bigbrain as ground truth for evaluation of civet & freesurfer structural mri pipelines. In Annual Meeting of the Organization for Human Brain Mapping, 2018.

[84] Xinhui Li, Lei Ai, Steve Giavasis, Hecheng Jin, Eric Feczko, Ting Xu, Jon Clucas, Alexandre Franco, Anibal Sólon Heinsfeld, Azeez Adebimpe, Joshua T. Vogelstein, Chao-Gan Yan, Oscar Esteban, Russell A. Poldrack, Cameron Craddock, Damien Fair, Theodore Satterthwaite, Gregory Kiar, and Michael P. Milham. Moving beyond processing and analysis-related variation in neuroscience. bioRxiv, 2021.

[85] Linus Torvalds. Git: a free and open source distributed version control system, 2021. [Online; accessed 6-October-2021].

[86] Allan J MacKenzie-Graham, Arash Payan, Ivo D Dinov, John D Van Horn, and Arthur W Toga. Neuroimaging data provenance using the loni pipeline workflow environment. In International Provenance and Annotation Workshop, pages 208–220. Springer, 2008.

[87] Daniel S Marcus, Timothy R Olsen, Mohana Ramaratnam, and Randy L Buckner. The extensible neuroimaging archive toolkit. Neuroinformatics, 5(1):11–33, 2007.

[88] Camille Maumet, Tibor Auer, Alexander Bowring, Gang Chen, Samir Das, Guillaume Flandin, Satrajit Ghosh, Tristan Glatard, Krzysztof J Gorgolewski, Karl G Helmer,

et al. Sharing brain mapping statistical results with the neuroimaging data model. Scientific data, 3:160102, 2016.

[89] Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. Python for High Performance and Scientific Computing, 14(9), 2011.

[90] Maarten Mennes, Bharat B Biswal, F Xavier Castellanos, and Michael P Milham. Making data sharing work: the fcp/indi experience. Neuroimage, 82:683–691, 2013.

[91] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. Linux journal, 2014(239):2, 2014.

[92] Michael Peter Milham. Open neuroscience solutions for the connectome-wide association era. Neuron, 73(2):214–218, 2012.

[93] Paolo Missier, Khalid Belhajjame, and James Cheney. The w3c prov family of specifications for modelling provenance metadata. In Proceedings of the 16th International Conference on Extending Database Technology, pages 773–776. ACM, 2013.

[94] Veronika I M"uller, Edna C Cieslik, Ilinca Serbanescu, Angela R Laird, Peter T Fox, and Simon B Eickhoff. Altered brain activity in unipolar depression revisited: meta- analyses of neuroimaging studies. JAMA psychiatry, 74(1):47–55, 2017.

[95] Ingo Müller, Andrea Arteaga, Torsten Hoefler, and Gustavo Alonso. Reproducible floating-point aggregation in rdbmss. arXiv preprint arXiv:1802.09883, 2018.

[96] Jean-Michel Muller, Nicolas Brisebarre, Florent De Dinechin, Claude-Pierre Jeannerod, Vincent Lefevre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, Serge Torres, et al. Handbook of floating-point arithmetic, volume 1. Springer, 2018.

[97] Thomas E Nichols, Samir Das, Simon B Eickhoff, Alan C Evans, Tristan Glatard, Michael Hanke, Nikolaus Kriegeskorte, Michael P Milham, Russell A Poldrack, Jean-Baptiste Poline, et al. Best practices in data analysis and sharing in neuroimaging using mri. Nature Neuroscience, 20(3):299, 2017.

[98] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R Pocock, Anil Wipat, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics, 20(17):3045–3054, 2004.

[99] Travis E Oliphant. A guide to NumPy, volume 1. Trelgol Publishing USA, 2006.

[100] Douglass Stott Parker. Monte Carlo Arithmetic: exploiting randomness in floating-point arithmetic. University of California (Los Angeles). Computer Science Department, 1997.

[101] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. the Journal of machine Learning research, 12:2825–2830, 2011.

[102] Roger D Peng. Reproducible research in computational science. Science, 334(6060):1226–1227, 2011.

[103] William D Penny, Karl J Friston, John T Ashburner, Stefan J Kiebel, and Thomas E Nichols. Statistical parametric mapping: the analysis of functional brain images. Elsevier, 2011.

[104] Jeffrey M Perkel. Challenge to scientists: does your ten-year-old code still run? Nature, 584(7822):656–658, 2020.

[105] Hans E Plesser. Reproducibility vs. replicability: a brief history of a confused terminology. Frontiers in neuroinformatics, 11:76, 2018.

[106] Rémi Rampin, Fernando Chirigati, Dennis Shasha, Juliana Freire, and Vicky Steeves. Reprozip: The reproducibility packer. Journal of Open Source Software, 1(8):107, 2016.

[107] Nathalie Revol and Philippe Théveny. Numerical reproducibility and parallel computations: Issues for interval algorithms. arXiv preprint arXiv:1312.3300, 2013.

[108] David E Rex, Jeffrey Q Ma, and Arthur W Toga. The loni pipeline processing environment. Neuroimage, 19(3):1033–1048, 2003.

[109] Ali Salari, Yohan Chatelain, Gregory Kiar, and Tristan Glatard. Accurate simulation of operating system updates in neuroimaging using monte-carlo arithmetic. In Uncertainty for Safe Utilization of Machine Learning in Medical Imaging, and Perinatal Imaging, Placental and Preterm Image Analysis, pages 14–23. Springer, 2021.

[110] Ali Salari, Gregory Kiar, Lindsay Lewis, Alan C Evans, and Tristan Glatard. File-based localization of numerical perturbations in data analysis pipelines. GigaScience, 9(12), 12 2020.

[111] Tom Schonberg, Craig R Fox, Jeanette A Mumford, Eliza Congdon, Christopher Trepel, and Russell A Poldrack. Decreasing ventromedial prefrontal cortex activity during sequential risk-taking: an fmri investigation of the balloon analog risk task. Frontiers in neuroscience, 6:80, 2012.

[112] Matthias Schwab, N Karrenbach, and Jon Claerbout. Making scientific computations reproducible. Computing in Science & Engineering, 2(6):61–67, 2000.

[113] Ji Suk Shim, Jin Sook Lee, Jeong Yol Lee, Yeon Jo Choi, Sang Wan Shin, and Jae Jun Ryu. Effect of software version and parameter settings on the marginal and internal adaptation of crowns fabricated with the cad/cam system. Journal of Applied Oral Science, 23(5):515–522, 2015.

[114] Victoria Stodden, Marcia McNutt, David H Bailey, Ewa Deelman, Yolanda Gil, Brooks Hanson, Michael A Heroux, John PA Ioannidis, and Michela Taufer. Enhancing reproducibility for computational methods. Science, 354(6317):1240–1241, 2016.

[115] Josef Stoer and Roland Bulirsch. Introduction to numerical analysis, volume 12. Springer Science & Business Media, 2013.

[116] Michela Taufer, Omar Padron, Philip Saponaro, and Sandeep Patel. Improving numerical reproducibility and stability in large-scale numerical simulations on gpus. In Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on, pages 1–9. IEEE, 2010.

[117] J-Donald Tournier, Fernando Calamante, and Alan Connelly. Mrtrix: diffusion tractography in crossing fiber regions. International Journal of Imaging Systems and Technology, 22(1):53–66, 2012.

[118] David C Van Essen, Stephen M Smith, Deanna M Barch, Timothy EJ Behrens, Essa Yacoub, Kamil Ugurbil, Wu-Minn HCP Consortium, et al. The wu-minn Human Connectome Project: an overview. Neuroimage, 80:62–79, 2013.

[119] David C Van Essen, Kamil Ugurbil, E Auerbach, D Barch, TEJ Behrens, R Bucholz, Acer Chang, Liyong Chen, Maurizio Corbetta, Sandra W Curtiss, et al. The human

connectome project: a data acquisition perspective. Neuroimage, 62(4):2222–2231, 2012.

[120] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17:261–272, 2020.

[121] Lina Wadi, Mona Meyer, Joel Weiser, Lincoln D Stein, and Jüri Reimand. Impact of outdated gene annotations on pathway enrichment analysis. Nature methods, 13(9):705, 2016.

[122] Jon Watson. Virtualbox: bits and bytes masquerading as machines. Linux Journal, 2008(166):1, 2008.

[123] Wikipedia contributors. Out-of-order execution — Wikipedia, the free encyclopedia, 2021. [Online; accessed 6-October-2021].

[124] Wikipedia contributors. Vmware workstation — Wikipedia, the free encyclopedia, 2021. [Online; accessed 6-October-2021].

[125] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The fair guiding principles for scientific data management and stewardship. Scientific data, 3, 2016.

[126] Andreas Zeller. Yesterday, my program worked. today, it does not. why? ACM SIGSOFT Software engineering notes, 24(6):253–267, 1999.

[127] Andreas Zeller. Isolating cause-effect chains from computer programs. ACM SIGSOFT Software Engineering Notes, 27(6):1–10, 2002.

[128] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. SIAM journal on computing, 18(6):1245–1262, 1989.

[129] Paul Zimmermann. Accuracy of Mathematical Functions in Single, Double, Extended Double and Quadruple Precision. working paper or preprint, February 2021.